NIST
National Institute of
Standards and Technology
Technology Administration
U.S. Department of Commerce

Special Publication 500-259

Networking for Pervasive Computing

Research from the National Institute of Standards and Technology

Kevin L. Mills, Editor

NIST Special Publication 500-259 *Networking for Pervasive Computing*

Kevin L. Mills, Editor

C O M P U T E R NETWORKING

Advanced Network Technology Division
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8920

July 2005

U.S. Department of Commerce
Carlos M. Gutierrez, Secretary

Technology Administration
Phillip J. Bond, Under Secretary for Technology

National Institute of Standards and Technology
William A. Jeffrey, Director

Acknowledgements

The editor, Kevin L. Mills of the National Institute of Standards and Technology (NIST), wishes to thank his many colleagues who conducted and documented the research reprinted in this publication. Due to the superior efforts of Nada Golmie, who led a fine team of researchers, we report significant findings regarding interference issues between wireless personal-area network (WPAN) devices and wireless local-area network (WLAN) devices. This research team also investigated technical approaches to mitigate interference between WPAN and WLAN devices. Over the course of the WPAN research reported here, contributing researchers included: Nicolas Chevrollier, Issam ElBakkouri, Frederic Mouveaux, Olivier Rebala, Amir Soltanian, Arnaud Tonnerre, and Robert Van Dyck. A key collaboration between Christopher Dabrowski of the NIST software division and Kevin Mills of the NIST networking division provided focused and study direction for a group of contributing researchers who investigated properties of first-generation service-discovery systems. Thanks to these researchers we provide substantial understanding of the functionality, behavior, performance, and robustness of first-generation service-discovery systems. This research group also investigated adaptive techniques that could enable components in service-discovery systems to regulate their own behavior to achieve desired performance properties. Over the duration of the service-discovery research reported here, contributing researchers included: Kevin Bowers, Mackenzie Britton, Jesse Elder, Steve Quirolgico, Scott Rose, Andrew Rukhin, and Ceryen Tan. Thanks are also due to the many reviewers, both inside and outside of NIST, who provided helpful suggestions to improve the various papers reprinted in this publication. We also must not ignore the significant contributions made by the designers, specification writers, and implementers who developed the proposed technologies for wireless network access and for service discovery, which formed the basis for the investigations reported here. Without the imagination and resources of these industrial contributors, we would not be in a position to provide the knowledge we gained by studying various networking technologies proposed for use in pervasive computing applications. A select few individuals who had the foresight to provide funding to support the research reported here provided a final, but key, contribution to this work. Funding was provided by: Susan Zevin, acting director of the NIST Information Technology Laboratory, Douglas Maughan, manager of the Defense Advanced Research Projects Agency (DARPA) Fault-Tolerant Networks Program, John Salasin, manager of the DARPA program in Dynamic Assembly for System Adaptability, Dependability and Assurance, and James Puffenbarger of the Advanced Research and Development Activity (ARDA).

RESEARCH OVERVIEW

Information technology is undergoing a paradigm shift from desktop computing, where isolated workstations connect to shared servers across a network, to pervasive computing, where myriad portable, embedded, and networked information appliances continuously reconfigure themselves individually and collectively to support the information requirements of mobile workers and work teams. This shift will not occur overnight, nor will it be achieved without solving a range of new technical and social problems. Still, this inexorable change should yield many economic opportunities for the global information technology industry, and for the increasing swath of businesses that depend on information. The potential value of pervasive computing motivated the NIST Information Technology Laboratory (ITL) to establish a five-year program of research to help the information technology industry identify and solve some looming technical roadblocks that seemed likely to slow development and acceptance of the new paradigm. The ITL Pervasive Computing program addressed three general areas: human-computer interaction, programming models, and networking. This special publication provides a compendium of technical papers published by NIST researchers who investigated networking for pervasive computing.

Pervasive computing changes the emphasis of networking from the core (or backbone) to the edge, where many portable devices will move through a wireless environment. Mobile devices can cause unpredictable traffic patterns, or traffic patterns that may be predictable but different from traffic patterns arising with conventional desktop computing. Since the network edge will comprise largely wireless communications, one may expect sudden crowding of the shared wireless spectrum and also surging demands for particular resources, such as wireless access points or configuration servers. Further, network protocols will need to dynamically discover and compose resources and services to support changing application demands associated with user mobility. Finally, it seems likely that user mobility will imply a diverse set of processors, transmission schemes, and protocols, which suggests the need to mediate among incompatible rules and descriptions and to allocate, schedule, and control shared, heterogeneous resources. From among the changes presaged by pervasive computing, NIST researchers elected to investigate two significant areas: (1) wireless personal area networks and (2) service discovery protocols.

Industry has developed a number of technical standards to provide wireless local-area network (WLAN) access and to support wireless personal-area network (WPAN) configurations. Pervasive computing will leverage both WLAN and WPAN technology, which operate in shared, unlicensed bands of wireless spectrum. Given that myriad wireless devices will operate simultaneously in close proximity, it appears possible that interference could compromise the quality of service available to mobile users. This concern motivated NIST researchers to ask two questions. First, can we characterize the performance of WLAN and WPAN protocols operating in the same network area? Second, can we devise technical approaches to mitigate interference and enhance coexistence between competing WLAN and WPAN devices? This special publication reprints six technical papers (Paper #1 through Paper #6) investigating the effects of interference between WLAN and WPAN protocols. This special publication also reprints eight technical papers (Paper #7 through Paper #14) reporting on various techniques to mitigate interference between WLAN and WPAN devices. The techniques captured in these eight papers were also submitted to the Institute for Electrical and Electronic Engineers (IEEE) technical committee developing standards for coexistence among wireless devices sharing unlicensed spectra. Wireless communications will provide the underlying infrastructure through which pervasive-computing devices and services can discover each other and interact, possibly configuring into collections to support application needs. This process of dynamic discovery, configuration, and monitoring provides a second area investigated by NIST researchers.

Over the period from about 1998 to 2000, industry developed a first generation of competing architectures and protocols for device and service discovery. Such a plethora of incompatible approaches seemed likely to impede the interoperability required by a market for pervasive computing. Is the existence of so many different service-discovery systems justified? NIST researchers analyzed the various technical approaches and developed a model to unify the features, functions, and processes provided. The

goal of this modeling effort was threefold: (1) to understand the essential service-discovery functionality defined by the industry, (2) to reveal any technical deficiencies in existing service-discovery specifications, and (3) to define the functional and behavioral bounds achievable from this first-generation of service-discovery systems. The result of this modeling effort is reported in "A Model-based Analysis of First-Generation Service Discovery Systems" a separate NIST special publication – SP 500-260. Here, we reprint (as Paper #21) only the executive summary from SP 500-260. A particular goal of service-discovery systems is to monitor the state of distributed resources in a network so that failed resources can be detected and recovery actions can be initiated. Do the various architectures for service-discovery systems provide different levels of robustness in the face of selected failure types? This question is addressed in part by five research papers (Paper #15 through Paper #19) reprinted in this special publication. One paper, "Understanding Failure Response in Service Discovery Systems" (Paper #20), provides a comprehensive report on the robustness of the three main service-discovery architectures (two-party, three-party, and adaptive two-three party) in the face a various types of failure (node failure, communication failure, message loss, and power failure). While investigating service-discovery protocols, NIST researchers noticed that the performance of some features depended upon adopting appropriate parameter settings, but that the most apt parameter settings depended upon the size of the system. This dependency caused some concern because, while service-discovery systems are intended to support dynamic changes in system composition, none of the protocols investigated provide any requirement to monitor system state and then to adjust selected parameter settings (or behavior) to improve system performance. This observation led NIST researchers to propose and evaluate some self-adaptive techniques that enable service-discovery components to monitor system state and to adjust various parameters and behaviors in real time. The related papers are reprinted in this special publication. One paper (Paper#22) investigates techniques to mitigate a possible implosion of responses to multicast queries. Three papers (Paper #23 through Paper #25) explore algorithms to dynamically adjust the duration assigned to leases (or subscriptions), which are often used by service-discovery protocols to detect failures among remote components. One paper (Paper #26) shows how a particular algorithm for dynamic adjustment of lease periods can be used to support various functions in a number of service-discovery protocols. Finally, one paper (Paper #27) evaluates distributed algorithms that service-discovery clients could use to select replicas to query. These research results regarding service-discovery systems should: (1) help prospective users to understand the functionality, behavior, and robustness of first-generation service-discovery systems, (2) inform implementers about the performance improvements possible through various self-adaptive algorithms, and (3) provide designers with ideas for improving the next generation of service-discovery systems.

Table of Contents

INTRODUCTION **1**

WIRELESS LOCAL AND PERSONAL AREA NETWORKS **3**

PERFORMANCE CHARACTERIZATION UNDER INTERFERENCE 4

Paper #1 Interference in the 2.4 GHz ISM Band: Impact on the Bluetooth Access Control Performance by Nada Golmie and Frederic Mouveaux, published in the *Proceedings of the 18th International Conference on Communications* (ICC'01), June 2001 8

Paper #2 Physical Layer Performance for Coexistence of Bluetooth and IEEE 802.11b by Amir Soltanian and Robert E. Van Dyck, published in the *Proceedings of the 2001 Virginia Tech Symposium* on Wireless Personal Communications, June 2001……..……...14

Paper #3 Performance of the Bluetooth System in Fading Dispersive Channels and Interference by Amir Soltanian and Robert E. Van Dyck, published in the *Proceedings of IEEE Globecom*, November 2001 25

Paper #4 Interference of Bluetooth and IEEE 802.11: Simulation Modeling and Performance Evaluation by Nada Golmie, Robert E. Van Dyck, and Amir Soltanian, published in the *Proceedings of the Fourth International ACM Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, July 2001 30

Paper #5 Interference Evaluation of Bluetooth and IEEE 802.11b Systems by Nada Golmie, Robert E. Van Dyck, Amir Soltanian, Arnaud Tonnerre, and Olivier Rebala, published in the journal *Wireless Networks*, 9, pp. 201-211, Kluwer Academic Publishers, 2003 38

Paper #6 Interference in the 2.4 GHz ISM Band: Challenges and Solutions by Nada Golmie, published in the *Proceedings of the First International Conference on Applications and Services in Wireless Networks*, ASW'2001, pp.187-199, July 2001 48

INTERFERENCE MITIGATION TECHNIQUES 61

Paper #7 Rejection of Bluetooth Interference in 802.11 LANs by Amir Soltanian and Robert E. Van Dyck, published in the *Proceedings of the IEEE Vehicular Technical Conference*, September 2002…..……...65

Paper #8 Techniques to Improve Bluetooth Performance in Interference Environments by Nada Golmie and Nicolas Chevrollier, published in the *Proceedings of Military Communications*, MILCOM, October 2001 …….……...68

Paper #9 Interference Aware Bluetooth Packet Scheduling by Nada Golmie, Nicolas Chevrollier, and Issam ElBakkouri, published in the *Proceedings of the IEEE Global Telecommunications Conference*, GLOBECOM '01, vol. 5, pp. 2857-2863,2001…….……..73

Paper #10 Techniques to Improve the Performance of TCP in a mixed Bluetooth and WLAN Environment by Nada Golmie and Olivier Rebala, published in the *Proceedings of the IEEE International Conference on Communications*, ICC, May 2003……..... 80

Paper #11 Bluetooth Dynamic Scheduling and Interference Mitigation by Nada Golmie, published in the ACM journal *Mobile Networks*, MONET Vol. 9, No. 1, February 2004……..……85
Paper #12 Bluetooth Adaptive Techniques to Mitigate Interference by Nada Golmie and Olivier Rebala, published in the *Proceedings of IEEE GLOBECOM*, December 2003…… 106
Paper #13 Bluetooth Adaptive Frequency Hopping and Scheduling by Nada Golmie, Olivier Rebala, and Nicolas Chevrollier, published in the *Proceedings of Military Communications*, MILCOM, October 2003 111
Paper #14 Bluetooth and WLAN Coexistence: Challenges and Solutions by Nada Golmie, Nicolas Chevrollier, and Olivier Rebala, published in the *IEEE Wireless Communications Magazine*, December 2003 116

SUMMARY OF CONTRIBUTIONS TO WLAN-WPAN TECHNOLOGY 125

FIRST-GENERATION SERVICE-DISCOVERY SYSTEMS ……..……..……126

BEHAVIORAL AND PERFORMANCE CHARACTERIZATION OF DISCOVERY SYSTEMS..... 127
Paper #15 Analyzing Properties and Behavior of Service Discovery Protocols Using an Architecture-Based Approach by Christopher Dabrowski and Kevin Mills, published in the *Proceedings of Working Conference on Complex and Dynamic Systems Architecture*, December 2001 131
Paper #16 Understanding Consistency Maintenance in Service Discovery Architectures during Communication Failure by Christopher Dabrowski, Kevin Mills, and Jesse Elder, published in the *Proceedings of the 3rd International Workshop on Software Performance*, ACM, pp. 168-178, July 2002 ……...144
Paper #17 Understanding Consistency Maintenance in Service Discovery Architectures in Response to Message Loss by Christopher Dabrowski, Kevin Mills, and Jesse Elder, published in the *Proceedings of the 4th International Workshop on Active Middleware Services*, IEEE Computer Society, pp. 51-60, July 2002…….155
Paper #18 Understanding Self-healing in Service-Discovery Systems by Christopher Dabrowski and Kevin Mills, published in the *Proceedings of the First ACM SigSoft Workshop on Self-healing Systems* (WOSS '02), ACM Press, pp. 15-20, November 18-19, 2002 165
Paper #19 Performance of Service-Discovery Architectures in Response to Node Failures by Christopher Dabrowski, Kevin Mills, and Andrew Rukhin, published in the *Proceedings of the 2003 International Conference on Software Engineering Research and Practice* (SERP'03), CSREA Press, pp. 95-101, June 2003…….. 171
Paper #20 Understanding Failure Response in Service Discovery Systems by Kevin Mills, Christopher Dabrowski, and Steve Quirolgico, submitted to the journal *Cluster Computing*, March 2005 ....……..……..178

Paper #21 A Model-based Analysis of First-Generation Service Discovery Systems by Christopher Dabrowski, Kevin Mills, and Steve Quirolgico, NIST Special Publication 500-260…..……..…..…….…198

PEFORMANCE IMPROVEMENT TECHNIQUES FOR DISCOVERY SYSTEMS 200

Paper #22 Adaptive Jitter Control for UPnP M-Search by Kevin Mills and Christopher Dabrowski, published in the Proceedings of International Conference on Communications (ICC), IEEE, May 2003 202

Paper #23 Self-Adaptive Leasing for Jini by Kevin Bowers, Kevin Mills, and Scott Rose, published in the *Proceedings of the Pervasive Computing Conference*, IEEE, pp. 539-542, March 2003…..…….208

Paper #24 Improving Failure Responsiveness in Jini Leasing by Scott Rose, Kevin Bowers, Steve Quirolgico, and Kevin Mills, published in the *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition* (DISCEX-III 2003), IEEE Computer Society, Vol. II, pp. 103-105, April 2003, ...212

Paper #25 Self-Managed Leasing for Distributed Systems by Kevin Bowers, Kevin Mills, Steve Quirolgico, and Scott Rose, published in the *Proceedings of the 1st Workshop on Algorithms and Architectures for Self-Managing Systems*, co-sponsored by ACM SIGMETRICS, June 2003.....215

Paper #26 An Autonomic Failure-Detection Algorithm by Kevin Mills, Scott Rose, Steve Quirolgico, Mackenzie Britton, and Ceryen Tan, published in the *Proceedings of the 4th International Workshop on Software Performance*, ACM, pp. 79-83, January 2004…….. 217

Paper #27 Performance Characterization of Decentralized Algorithms for Replica Selection in Distributed Object Systems by Ceryen Tan and Kevin Mills, published in the *Proceedings of the 5th International Workshop on Software Performance,* Palma de Mallorca, Spain, ACM, July 11-14, 2005 (in press)..……..……..222

SUMMARY OF CONTRIBUTIONS TO SERVICE DISCOVERY TECHNOLOGY 228

INTRODUCTION

Information technology is undergoing a paradigm shift from desktop computing, where isolated workstations connect to shared servers across a network, to pervasive computing, where myriad portable, embedded, and networked information appliances continuously reconfigure themselves individually and collectively to support the information requirements of mobile workers and work teams. This shift will not occur overnight, nor will it be achieved without solving a range of new technical and social problems. Still, this inexorable change should yield many economic opportunities for the global information technology industry, and for the increasing swath of businesses that depend on information. The potential value of pervasive computing motivated the NIST Information Technology Laboratory (ITL) to establish a five-year program of research to help the information technology industry identify and solve some looming technical roadblocks that seemed likely to slow development and acceptance of the new paradigm. The ITL Pervasive Computing program addressed three general areas: human-computer interaction, programming models, and networking. This special publication provides a compendium of technical papers published by NIST researchers who investigated networking for pervasive computing.

Pervasive computing changes the emphasis of networking from the core (or backbone) to the edge, where many portable devices will move through a wireless environment. Mobile devices can cause unpredictable traffic patterns, or traffic patterns that may be predictable but different from traffic patterns arising with conventional desktop computing. Since the network edge will comprise largely wireless communications, one may expect sudden crowding of the shared wireless spectrum and also surging demands for particular resources, such as wireless access points or configuration servers. Further, network protocols will need to dynamically discover and compose resources and services to support changing application demands associated with user mobility. Finally, it seems likely that user mobility will imply a diverse set of processors, transmission schemes, and protocols, which suggests the need to mediate among incompatible rules and descriptions and to allocate, schedule, and control shared, heterogeneous resources. From among the changes presaged by pervasive computing, NIST researchers elected to investigate two significant areas: (1) wireless personal area networks and (2) service discovery protocols.

Industry has developed a number of technical standards to provide wireless local-area network (WLAN) access and to support wireless personal-area network (WPAN) configurations. Pervasive computing will leverage both WLAN and WPAN technology, which operate in shared, unlicensed bands of wireless spectrum. Given that myriad wireless devices will operate simultaneously in close proximity, it appears possible that interference could compromise the quality of service available to mobile users. This concern motivated NIST researchers to ask two questions. First, can we characterize the performance of WLAN and WPAN protocols operating in the same network area? Second, can we devise technical approaches to mitigate interference and enhance coexistence between competing WLAN and WPAN devices? The answers found by NIST researchers are reported below in fourteen papers organized in two sections: performance characterization and interference mitigation techniques. Overall, the research results from NIST regarding mutual interference of WLAN and WPAN devices should: (1) help prospective users to understand the limitations of current wireless devices deployed in the unlicensed 2.4 GHz ISM band, (2) inform standards setters about the performance improvements possible through adoption of various co-existence algorithms, and (3) provide designers with ideas for improving the next generation of wireless devices.

While wireless communications provide the underlying infrastructure through which pervasive-computing devices can discover each other and configure into collections, a similar process of dynamic discovery, configuration, and monitoring will occur at a higher level, where software components and services cooperate to meet application needs. Over the period from about 1998 to 2000, industry developed a first generation of competing architectures and protocols for component and service discovery. Such a plethora of incompatible approaches seemed likely to impede the interoperability required by a market for pervasive computing. Is the existence of so many different service-discovery systems justified?

NIST researchers analyzed the extant technical approaches and developed a model to unify the features, functions, and processes provided. The modeling effort aimed: (1) to understand the essential service-discovery functionality provided by the industry, (2) to reveal any technical deficiencies in existing service-discovery specifications, and (3) to define the technical bounds achievable from this first-generation of service-discovery systems. A particular goal of service-discovery systems is to monitor the state of distributed resources in a network so that failed resources can be detected and recovery actions can be initiated. Do the various architectures for service-discovery systems provide different levels of robustness in the face of selected failure types? NIST researchers address these objectives and questions in a collection of seven papers that characterize the behavior and performance of first-generation service-discovery protocols. Of particular note, one paper, "A Model-based Analysis of First-Generation Service-Discovery Systems", provides a comprehensive model of the service-discovery domain and compares several specific service-discovery systems with the domain model.

While investigating service-discovery protocols, NIST researchers noticed that the performance of some features depends upon adopting appropriate parameter settings, where the most apt parameter settings depend upon the size of the system. This dependency caused some concern because, while service-discovery systems are intended to support dynamic changes in system composition, none of the protocols investigated provide any requirement to monitor system state and then to adjust selected parameter settings (or behavior) to improve system performance. This observation led NIST researchers to propose and evaluate some self-adaptive techniques that enable service-discovery components to monitor system state and to adjust various parameters and behaviors in real time. Six related papers are reprinted here in a section on performance improvement techniques for service-discovery systems. Overall, the research results from NIST regarding service-discovery systems should: (1) help prospective users to understand the functionality, behavior, and robustness of first-generation service-discovery systems, (2) inform implementers about the performance improvements possible through various self-adaptive algorithms, and (3) provide designers with ideas for improving the next generation of service-discovery systems.

WIRELESS LOCAL AND PERSONAL AREA NETWORKS

The ITU-T (the telecommunications standards organization of the International Telecommunications Union) reserved radio spectrum in selected bands (900 MHz, 2.4 GHz, and 5.8 GHz) throughout the world for non-commercial use in support of industrial, scientific, and medical (ISM) endeavors. The free availability of these radio-frequency bands led to a number of innovations in wireless local network communications. Of specific interest in this publication are wireless local-area network technologies (WLAN) that meet the so-called IEEE 802.11 standards and wireless personal-area network (WPAN) technologies that meet the Bluetooth specification (and the related IEEE 802.15 standards). WLAN technologies are designed to operate in the unlicensed 2.4 GHz ISM spectrum band, providing wireless device communication over ranges of tens of meters. WPAN technologies are designed to operate in the same ISM band, providing wireless device communication over a range of 1 to 10 meters. WLAN technologies provide communications support for computational devices such as laptop and notebook computers and personal digital assistants. WPAN technologies aim mainly to provide communications support for input/output devices such as microphones, headphones, and video cameras. Both WLAN and WPAN technologies appear as likely candidates for combination into systems of computation and communication that will support mobile individuals; however, since WLAN and WPAN devices share the same spectrum, the possibility arises for mutual interference when operating in close proximity. More visionary designs also exist for so-called pervasive computing systems to permit mobile individuals to congregate within buildings, conference rooms, and auditoriums that provide wireless access to a rich set of local resources that could be deployed on demand to support collaborative work. In these latter cases, the wireless spectrum could become quite crowded and mutual interference might prevent realization of the pervasive computing vision.

Two key questions arise when considering the potential for WLAN and WPAN technologies to revolutionize workplaces and lifestyles. First, what will be the nature of any mutual interference that might arise among WLAN and WPAN devices? Second, what technical approaches might be developed and deployed to mitigate mutual interference among WLAN and WPAN devices, and how successful will such approaches prove? These questions, which motivated one facet of research in the NIST program on networking for pervasive computing, are addressed in the following set of fourteen papers that document findings by researchers in the Advanced Network Technologies Division at NIST. The papers divide naturally into two sets. Six papers (Papers #1 through #6) assess the nature of mutual interference that can arise when WLAN and WPAN devices operate in close proximity (within 15 meters). Eight papers (Papers #7 through #8) investigate various technical approaches to mitigate mutual interference in order to allow WLAN-WPAN devices to co-existent in close proximity. More detail on these papers appears in the introduction to each set.

PERFORMANCE CHARACTERIZATION UNDER INTERFERENCE

The design for WLANS and WPANS share a similar logical structure. A physical layer (PHY) defines techniques for framing, encoding, protecting, transmitting, receiving, correcting, and decoding bits conveyed over a wireless channel. A media-access control (MAC) layer defines techniques for sharing access to the wireless channel, including avoiding, detecting, and recovering from transmissions that overlap (collide) in space and time. Transport-layer protocols, such as the transmission control protocol (TCP) and the user-datagram protocol (UDP), employ services of the MAC and PHY layers to exchange messages related to a variety of applications, including file transfer, web surfing, and audio-video communication. Particular combinations of application and transport protocol present a wireless channel with data traffic exhibiting different characteristics and requiring different qualities of service. Overall, each protocol layer has a complex set of factors that influence its behavior and performance, and the combination of four layers (PHY, MAC, transport, and application) must be considered to develop any meaningful characterization of system performance. For this reason, while a few papers presented below consider only the PHY and MAC layers, most of the papers consider the larger set of four layers. To provide some background to better understand the papers, more information is needed about each protocol layer.

The wireless channel space has two facets: (1) physical distance and (2) frequency distance. That is, two devices operating near each other, but on different frequencies, do not experience mutual interference, while two devices operating on the same frequencies, but out of range from each other, do not experience mutual interference. (Of course, other factors, such as noise and channel fading, which may cause different frequencies to propagate with different characteristics, can impair a wireless channel even in the absence of other interference.) The WLAN (IEEE 802.11) PHY provides for two different techniques for nearby devices to share frequencies. One technique, direct sequence spread spectrum (DSSS), uses codes that allow each device to transmit over a broad range of frequencies, while placing some information about each bit at selected points within the frequency band. A receiver uses the same code to extract the transmitted bit information from the frequency band. A second technique, frequency hopping spread spectrum (FHSS), configures each device with a set of frequencies and times to transmit on those frequencies. A device then sends bits at the designated times on the assigned frequencies. A receiver uses the same frequency-time schedule to extract the transmitted bits from the channel. Since most deployments of 802.11-compatible WLANS use DSSS, the papers described below model the WLAN PHY layer as a DSSS channel. Bluetooth (and IEEE 802.15) use the frequency-hopping technique to share the wireless channel; thus, the papers included here model the WPAN PHY layer as a FHSS channel.

The MAC layers differ significantly for 802.11 WLANS and Bluetooth WPANS. WLANS use a distributed collision avoidance, detection, and recovery algorithm that requires each device to determine when a channel is free for use, to monitor its own transmission, to detect its own collisions, and to schedule a future attempt to reacquire the channel. WPANS use a centralized polling scheme that designates a single device as a master that organizes and oversees the transmissions of other (slave) devices sharing the same channel. Another difference concerns message sizes. WLANS permit all transmission to take on a variable length within an allowed range, while WPANS require that messages consist of some integral number of fixed-size slots. The papers below model these differences in MAC layer procedures.

Both WLANS and WPANS, as modeled here, employ the same set of transport protocols (TCP and UDP). Though WPAN profiles have been specified and implemented to use other transport protocols, these other protocols are not investigated in this set of papers. Regarding application traffic, the models used in the following papers usually adopt the assumption that WLAN traffic consists of either file transfer or web surfing. This is consistent with the most common current usage today. On the other hand, since WPANS are designed explicitly to support both data and multimedia traffic, the models investigated below use a variety of different application-traffic profiles when simulating WPANS.

In Paper #1, "Interference in the 2.4 GHz ISM Band: Impact on Bluetooth Access Control Performance", Golmie and Mouveaux quantify the packet-loss rate of the Bluetooth MAC layer when the radio operates in close proximity to a (802.11) WLAN. The paper reports results from two simulation experiments. Both experiments model a pair of Bluetooth devices (one master and one slave) separated by one meter. The Bluetooth devices may transmit either symmetric voice traffic (64 Kbps each direction) or data traffic (at 50% of an available 1 Mbps channel capacity). The experiments also model two WLAN devices on a channel, where one device transmits data and the other device receives data and sends acknowledgments. The WLAN devices are separated by about 10 meters, with the WLAN data transmitter being about 10 meters from the Bluetooth devices and the WLAN acknowledgement transmitter within one meter. In the first experiment, the WLAN generates a constant 50% offered load, but with varying packet sizes (packet spacing is varied as necessary to maintain a 50% offered load). In the second experiment, the WLAN generates a variable offered load (ranging between 5% and 70%) by altering spacing of fixed-size packets. Prior to reporting the simulation results, the authors develop an analytical model of packet-loss rate under similar assumptions. The simulations found packet-loss rates for the Bluetooth MAC to reach 25% for voice traffic and 27% for data traffic. The analytical predictions showed similar results. The authors observe that any assessment of interference among wireless devices must consider the characteristics (arrival rate and size) of application traffic. The authors conclude that WLAN devices can create significant performance degradation for WPAN devices operating in close proximity; thus, the cause and properties of such degradation deserve further investigation.

In Paper #2, "Physical Layer Performance for Coexistence of Bluetooth and IEEE 802.11b", Soltanian and Van Dyck study the bit-error rate performance at the physical layer for Bluetooth and 802.11b receivers under various combinations of mutual interference, noise, and fading. The study aims to inform future investigations of physical-layer techniques that might be employed to reduce mutual interference among wireless devices. The paper develops two separate (and detailed) models. One model represents a Bluetooth system that can be parameterized with specific noise and fading assumptions and that can be subjected to interference by either a WLAN system or another Bluetooth system. Because the Bluetooth specification permits multiple Bluetooth networks (called piconets) to share the same channel, mutual interference among piconets is a useful situation to consider. The Bluetooth model includes the inexpensive receiver filter specified in the related technical standard, but also provides a variant with a more sophisticated Viterbi receiver. The second model represents an 802.11b system that can be parameterized with specific noise and fading assumptions and that can be subjected to interference by a Bluetooth system. The WLAN (802.11b) model can consider either the 1-Mbps or 11-Mbps channel capacities supported by the corresponding technical specification. While the models of the physical systems are quite detailed, the models of application traffic are rather simple. The WLAN model assumes constant transmissions and the WPAN model assumes constant transmissions always synchronized on packet boundaries. These amount to worst-case assumptions, where interferers are never idle. The paper presents simulation results showing that mutual interference may damage the physical-layer performance of both WLAN and WPAN devices. The results also suggest that Bluetooth devices would become more resistant to interference and noise if they included (a significantly more expensive) Viterbi receiver.

In Paper #3, "Performance of the Bluetooth System in Fading Dispersive Channels and Interference", Soltanian and Van Dyck continue their investigation of the WPAN physical layer, focusing on the ability of Viterbi receivers to improve the performance of the Bluetooth PHY. Here, the researchers study the bit-error and packet-loss rates under fading and interference in Bluetooth systems with either a standard (non-coherent limited-discriminator) receiver or a Viterbi receiver, finding that the Viterbi receiver provides superior performance. Of course, the Viterbi receiver would add significantly to the cost of Bluetooth devices, which have a cost objective as low as $5 per device. The researchers also suggest that the technical specification for Bluetooth permits too large a range for the transmitter's modulation index.

In Paper #4, "Interference of Bluetooth and IEEE 802.11: Simulation Modeling and Performance Evaluation", Golmie, Van Dyck, and Soltanian introduce simulation models of combined MAC and PHY layers for both WLAN and WPAN, and use those models to study the effects of interference between

Bluetooth and 802.11b wireless networks under several application scenarios. The paper validates that a detailed PHY model can be replaced by a pre-computed table lookup in order to achieve accurate results with a significant savings in compute time. The paper presents details about the WLAN and WPAN simulation models, and then presents four experiments, which all use the same topology of devices. The topology consists of two Bluetooth devices (master and slave) separated by 1 meter, and two WLAN devices: a fixed access point about 15 meters from the Bluetooth devices and a mobile device that changes position, moving within a range of ½ and 5 meters of the Bluetooth devices. One experiment considers Bluetooth voice traffic with interference from the mobile WLAN node sending to the access point, while a second experiment considers Bluetooth data traffic under the same conditions. A third experiment considers WLAN data traffic flowing from access point to mobile device with interference from Bluetooth voice traffic. A fourth experiment subjects WLAN data traffic to interference from Bluetooth data traffic. The study considers WLANS operating at both 1 and 11 Mbps. The paper finds that Bluetooth voice traffic can cause significant packet loss for WLAN devices operating within ½ meter: 65% for 1 Mbps WLANS and 30% for 11 Mbps WLANS. In addition, WLAN devices can cause 8% packet loss for Bluetooth devices operating at the same ½-meter distance. WLANS can also interfere with Bluetooth data traffic, causing as much as 14% in packet losses at close range. Bluetooth data traffic can induce similar packet losses in WLAN devices. The authors conclude that these significant interference problems deserve further study in more complex scenarios involving large topologies and including transport protocols.

In Paper #5, "Interference Evaluation of Bluetooth and IEEE 802.11 Systems", Golmie, Van Dyck, Soltanian, Tonnerre, and Rebala investigate interference effects on Bluetooth and 802.11b wireless networks under a range of parameters, such as transmission power, offered load, and traffic profile. Further, the paper studies performance of WPAN and WLAN devices in a variety of complex scenarios involving multiple Bluetooth piconets and 802.11 devices. The study reveals some interesting findings. First, to overcome Bluetooth interference, WLAN devices would have to increase transmission power by more than 50 times, which appears impractical. Second, limiting transmission power in WLAN devices can help to avoid interference with Bluetooth devices. Third, using a slower frequency-hop rate (than the 1,600 hops per second specified) for Bluetooth devices may reduce interference with WLAN devices. Overall, the authors find that the traffic distribution has the largest affect on mutual interference, which suggests that sophisticated control schemes might best be designed as application-dependent – an impractical step due to complexity and cost. The findings reported in this paper lead the authors to suggest that coexistence mechanisms for WLAN and WPAN devices might be a fruitful direction to investigate.

In Paper #6, "Interference in the 2.4 GHz ISM Band: Challenges and Solutions", Golmie recounts lessons learned from previous studies (see Papers #1 through #5 in this special publication), and suggests the outlines for a coexistence framework that might help WLAN and WPAN devices operate in close proximity with better performance. In explaining existing interference problems and possible solutions for WLAN and WPAN devices, Golmie takes a tutorial approach. The paper gives an overview of several coexistence solutions proposed for various interference scenarios, and suggests that forward-error correction (where packets include redundant bits that might be used to reconstruct damaged bits) has limited benefits in many interference scenarios. The paper also shows that fragmenting packets can reduce packet loss between two devices, at the expense of more interference to other devices. In the end, Golmie suggests that research on coexistence mechanisms should examine both adaptive frequency hopping (AFH) and MAC scheduling. AFH would require Bluetooth masters to detect and map used frequencies and then adjust the hopping sequence for the piconet to select only an effective pattern of unused frequencies. MAC scheduling would require Bluetooth masters to consult the map of used frequencies and postpone transmissions to slaves experiencing used frequencies until an appropriate frequency becomes available. The paper outlines some advantages and disadvantages of the two approaches, and also considers briefly the idea of time-division scheduling when a WLAN and WPAN device operates from the same computer. This approach does not address the general case of interference

between independent WLAN and WPAN devices operating in close proximity. Overall, this paper sets the framework for investigating interference mitigation techniques, as reported in the next set of papers.

Interference in the 2.4 GHz ISM Band: Impact on the Bluetooth Access Control Performance

Nada Golmie and Frederic Mouveaux
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
Email: nada.golmie@nist.gov

Abstract— **Bluetooth is a radio technology operating in the 2.4 GHz ISM frequency band, that is emerging as a low-level and low-power wireless communication protocol used for wireless personal area networks (WPANs) where proximal devices can share information and resources. In this paper, we quantify the performance of the Bluetooth access control layer when the radio is operating in close proximity to a WLAN system. We use a probability analysis approach to derive the packet error for Bluetooth. The analytical results are validated using detailed simulation models for an interference scenario consisting of Bluetooth and WLAN devices. Packet loss is obtained for voice and data traf c for different interference conditions.**

Keywords—**WPANs, Bluetooth, Interference.**

I. Introduction

AN increasingly mobile lifestyle is creating the need for Wireless Personal Area Networks (WPANs) consisting of ad-hoc communications between portable computing devices such as laptops, PDAs, pagers, and cellular telephones. What is emerging today are wireless technologies, including IEEE 802.11 [1], and Bluetooth [2], that promise to out t portable and embedded devices with high bandwidth, localized wireless communication capabilities that can also reach the globally wired Internet.

Due to its almost global availability, the 2.4 GHz Industry Scienti c and Medical (ISM) unlicensed band constitutes a popular frequency band suitable to low cost radios. New proposed solutions for WPANs such as IEEE 802.15 and Bluetooth plan to operate in the 2.4 GHz ISM band while IEEE 802.11 [1] has standards for Wireless Local Area Networks operating in this band and microwave ovens are a primary user of the band at 2.45 GHz. Therefore, it is anticipated that some interference will result from all these technologies operating in the same environment and frequency space. Furthermore, since IEEE 802.11, and Bluetooth devices may likely come together in a laptop or may be close together at a desktop, interference may lead to signi cant performance degradation.

The main goal of this paper is to present our results on the performance of a Bluetooth access control system when its radio is operating in close proximity to an IEEE 802.11 system. The evaluation of interference in the 2.4 GHz band has been receiving more attention lately. Zurbes et. al. simulate the impact of 100 co-located sessions on the Bluetooth radio performance [3]. Kamerman reports on tolerable interference levels between Bluetooth and 802.11 devices for various scenarios and device positions [4]. His analysis is based on a simple path loss model and Signal to Interference (SIR) requirements for Bluetooth and 802.11 receivers. Furthermore, the probability of an 802.11 packet error in the presence of a Bluetooth piconet has been derived by Ennis [5], then extended by Shellhammer [6] and Chiasserini and Rao [7].

In this paper, we rst use a probability analysis approach to capture the interference environment. Our analytical results are then validated against simulation results obtained from detailed simulation models of the Bluetooth and IEEE 802.11 Medium Access Control (MAC) and Physical (PHY) layers. Our goal is to give additional insights on the performance of Bluetooth voice and data traf c under different interference traf c conditions.

This paper is organized as follows. In sections II and III we give some general insights on the Bluetooth and IEEE 802.11 protocol operation respectively. In section IV, we present our interference analysis and the probability that a packet containing error is received at the Bluetooth node. In section V, we evaluate the impact of WLAN interference on the Bluetooth performance and present simulation results. Concluding remarks are offered in section VI.

II. Bluetooth Protocol Overview

In this section, we give a brief overview of the Bluetooth technology [2] and discuss the main functionality of its protocol speci cations which consist of several modules, namely, the Radio Frequency (RF), Baseband (BB) and Link Manager (LM). Bluetooth is a short range (0 m - 10 m) wireless link technology aimed at replacing non-interoperable proprietary cables that connect phones, laptops, PDAs and other portable devices together. Bluetooth operates in the ISM frequency band starting at 2.402 GHz and ending at 2.483 GHz in the USA, and Europe. 79 RF channels of 1 MHz width are de ned. The air interface is based on an antenna power of 1 mW (0 dBi gain). The signal is modulated using binary Gaussian Frequency Shift Keying (GFSK). The raw data rate is de ned at 1 Mbits/s. A Time Division Multiplexing (TDM) technique divides the channel into 625 μs slots. Transmission occurs in packets that occupy an odd number of slots (up to 5). Each packet is transmitted on a different hop frequency with a maximum frequency hopping rate of 1600 hops/s.

Two or more units communicating on the same channel form a piconet, where one unit operates as a master and the others (a maximum of seven active at the same time) act as slaves. A channel is de ned as a unique pseudo-random frequency hopping sequence derived from the master device's 48-bit address and its Bluetooth clock value. Slaves in the piconet synchronize their timing and frequency hopping to the master upon

connection establishment. In the connection mode, the master controls the access to the channel using a polling scheme where master and slave transmissions alternate. A slave packet always follows a master packet transmission as illustrated in Figure 1 that depicts the master's view of the slotted TX/RX channel.

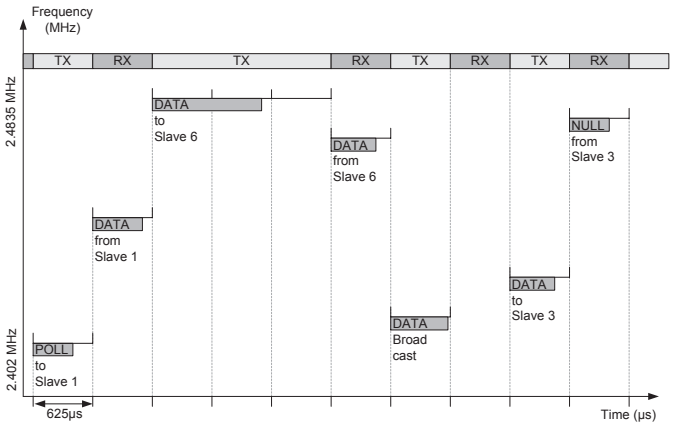


Fig. 1. Master TX/RX Hopping Sequence

There are two types of link connections that can be established between a master and a slave: the Synchronous Connection-Oriented (SCO), and the Asynchronous Connection-Less (ACL) link. The SCO link is a symmetric point-to-point connection between a master and a slave where the master sends an SCO packet in one TX slot at regular time intervals, de ned by T_{SCO} time slots. The slave responds with an SCO packet in the next TX opportunity. T_{SCO} is set to either 2, 4 or 6 time slots for $HV1$, $HV2$, or $HV3$ packet formats respectively. All three formats of SCO packets are de ned to carry 64 Kbits/s of voice traf c and are never retransmitted in case of packet loss or error. The ACL link, is an asymmetric point-to-point connection between a master and active slaves in the piconet. Several packet formats are de ned for ACL, namely $DM1$, $DM2$, and $DM3$ packets that occupy 1, 3, and 5 time slots respectively. An Automatic Repeat Request (ARQ) procedure is applied to ACL packets where packets are retransmitted in case of loss until a positive acknowledgement (ACK) is received at the source. The ACK is piggy-backed in the header of the returned packet where an ARQN bit is set to either 1 or 0 depending on whether the previous packet was successfully received or not. In addition, a sequence number (SEQN) bit is used in the packet header in order to provide a sequential ordering of data packets in a stream and lter out retransmissions at the destination. Forward Error Correction (FEC) is used on some SCO and ACL packets in order to correct errors and reduce the number of ACL retransmissions.

III. IEEE 802.11 Protocol Overview

The IEEE 802.11 standard [1] de nes both the physical (PHY) and medium access control (MAC) layer protocols for WLANs. In this sequel, we will be using WLAN and 802.11 interchangeably.

The IEEE 802.11 standard calls for three different PHY speci cations: frequency hopping (FH) spread spectrum, direct sequence (DS) spread spectrum and infrared (IR). The transmit power for DS and FH devices is de ned at a maximum of 1 W and the receiver sensitivity is set to -80 dBm. Antenna gain is limited to 6 dBi maximum.

Under FH, each station's signal hops from one modulating frequency to another in a predetermined pseudo-random sequence. Both transmitting and receiving stations are synchronized and follow the same frequency sequence. There are 79 channels de ned in the (2.4000 - 2.4835) GHz region spaced 1 MHz apart. The time each radio dwells on each frequency depends on each individual implementation and government regulation. The basic access rates of 1 and 2 Mbits/s use multilevel Gaussian frequency shift keying (GFSK).

A DS transmitter converts the data stream into a symbol stream where each symbol represents a group of multiple bits to spread over a relatively wideband channel of 22 MHz. The basic data rate is 1 Mbits/s encoded with differential binary phase shift keying (DBPSK) or 2 Mbits/s using differential quadrature phase shift keying (DQPSK). Higher rates of 5.5 and 11 Mbits/s are also available with techniques combining pulse-position-modulation (PPM) and quadrature amplitude modulation (QAM).

The IEEE 802.11 MAC layer speci cations common to all PHYs and data rates coordinate the communication between stations and control the behavior of users who want to access the network. The Distributed Coordination Function (DCF) which describes the default MAC protocol operation is based on a scheme known as carrier-sense, multiple access, collision avoidance (CSMA/CA). Both the MAC and PHY layers cooperate in order to implement collision avoidance procedures. The PHY layer samples the received energy over the medium transmitting data and uses a clear channel assessment (CCA) algorithm to determine if the channel is clear. This is accomplished by measuring the RF energy at the antenna and determining the strength of the received signal commonly known as RSSI, or received signal strength indicator. In addition, carrier sense can be used to determine if the channel is available. This technique is more selective since it veri es that the signal is the same carrier type as 802.11 transmitters. A virtual carrier sense mechanism is also provided at the MAC layer. It uses the request-to-send (RTS) and clear-to-send (CTS) message exchange to make predictions of future traf c on the medium and updates the network allocation vector (NAV) available in stations. Communication is established when one of the wireless nodes sends a short RTS frame. The receiving station issues a CTS frame that echoes the sender's address. If the CTS frame is not received, it is assumed that a collision occurred and the RTS process starts over. Regardless of whether the virtual carrier sense routine is used or not, the MAC is required to implement a basic access procedure (depicted in Figure 2) as follows. If a station has data to send, it waits for the channel to be idle through the use of the CSMA/CA algorithm. If the medium is sensed idle for a period greater than a DCF interframe space (DIFS), the station goes into a backoff procedure before it sends its frame. Upon the successful reception of a frame, the destination station returns an ACK frame after a Short interframe space (SIFS). The backoff window is based on a random value uniformly distributed in the interval $[0, CW]$

where CW represents the Contention Window parameter and is varied between CW_{min} and CW_{max}. If the medium is determined busy at any time during the backoff slot, the backoff procedure is suspended. It is resumed after the medium has been idle for the duration of the DIFS period. If an ACK is not received within an ACK timeout interval, the station assumes that either the data frame or the ACK was lost and needs to retransmit its data frame by repeating the basic access procedure.

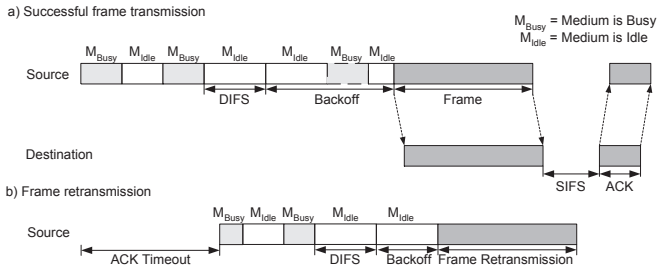


Fig. 2. WLAN Frame Transmission Scheme

IV. Interference Analysis

Since we are mainly concerned with evaluating the Bluetooth performance in an interference environment, we consider a Bluetooth receiver node as our reference and derive the probability that a packet containing errors (at least one error), P(PE), is received at this node. The interfering signal is assumed to be from proximally located WLAN devices.

A collision occurs when both the Bluetooth and the interfering packets overlap in time and frequency. This collision is detected at the Bluetooth receiver in the form of SIR that depends on the power transmitted, the distance traveled, and the path loss model used. The SIR then translates into a Bit Error Rate (BER) according to the GFSK carrier modulation and the Bluetooth receiver implementation used.

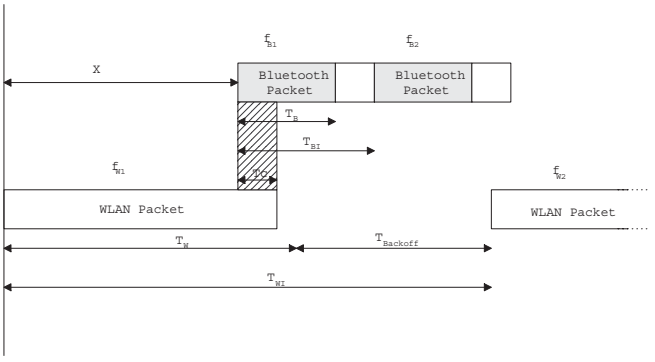


Fig. 3. Collisions at the Bluetooth Receiver Node

Figure 3 illustrates the timing of the Bluetooth packets with respect to WLAN packets. Let f_B and f_W be the frequencies used to transmit the Bluetooth and WLAN packets respectively. We denote by T_B and T_W, the Bluetooth and the WLAN packet transmission periods respectively. In order to determine the position of the Bluetooth packet with respect to the WLAN packet when both systems use the same frequency ($f_B = f_W$), we define a variable X that represent the time offset between a Bluetooth and a WLAN packet. Let T_C represent the time interval when both WLAN and Bluetooth packets overlap. We denote by T_{WI} the interval between two WLAN packets including the packet transmission time T_W and a backoff period, $T_{Backoff}$. $T_{Backoff}$ is the sum of several variables such as SIFS, DIFS, the ACK transmission time, and C_W. Similarly, we denote by T_{BI}, the interval between two Bluetooth packet transmissions. Due to the slotted structure of the Bluetooth channel, a packet transmission occurs at the boundary of a Bluetooth time slot. We assume that X is a random variable that is uniformly distributed between zero and T_{WI}. Note that X is a continuous random variable, however in this analysis it is quanti ed to the resolution of a Bluetooth symbol period at the rate of a symbol (or a bit) per μs.

$$X \sim U(0, T_{WI}) \tag{1}$$

Thus, the probability that a Bluetooth packet overlaps in time and frequency with a WLAN packet depends on:

- The position of the WLAN packet with respect to the Bluetooth packet, i.e. X
- The transmission frequencies, f_B and f_W of the Bluetooth and WLAN systems respectively

The probability mass function of X is equal to $p_X(k) = \frac{1}{T_{WI}}$ where $k = 1, 2, ..T_{WI}$. Both the Bluetooth and WLAN systems have a frequency hopping span of 79 channels. The probability that a WLAN system lands on the same frequency as a Bluetooth system depends on a discrete random variable f_W whose probability mass function is $p_{f_W}(j) = \frac{n}{79}$ where j varies between 1 and 79 and n determines the number of overlapping channels. For FH $n = 1$, while for DS WLAN systems, $n = 22$.

Expressing $P(PE)$ as a joint probability of frequency and packet overlap yields:

$$P(PE) = \sum_{k=0}^{T_{WI}} P(PE \mid X = k; f_W = j) p_X(k) p_{f_W}(j)$$

where $P(PE \mid X = k; f_W = j)$ depends on T_C and BER. Thus, we write:

$$P(PE \mid X = k; f_W = j) = 1 - (1 - BER)^{T_C} \tag{2}$$

Therefore,

$$P(PE) = (\frac{n}{79})(\frac{1}{T_{WI}}) \sum_{k=0}^{T_{WI}} (1 - (1 - BER)^{T_C}) \tag{3}$$

The value of T_C depends on X, T_W, and T_B. We distinguish three cases.

- $T_B \leq T_W$ and $T_B \leq T_{WI}$-T_W

$$T_C = \begin{cases} T_B & \text{if } X \leq T_W - T_B \\ T_W - X & \text{if } T_W - T_B < X < T_W \\ 0 & \text{if } T_W \leq X \leq T_{WI} - T_B \\ X + T_B - T_{WI} & \text{if } T_{WI} - T_B < X \leq T_{WI} \end{cases} \tag{4}$$

- $T_B \leq T_W$ and $T_B > T_{WI}$-T_W

$$T_C = \begin{cases} T_B \\ \qquad \text{if } X < T_W - T_B \\ T_W - X \\ \qquad \text{if } T_W - T_B \leq X < T_{WI} - T_B \\ T_W + T_B - T_{WI} \\ \qquad \text{if } T_{WI} - T_B \leq X \leq T_W \\ X + T_B - T_{WI} \\ \qquad \text{if } T_W < X \leq T_{WI} \end{cases} \tag{5}$$

- $T_B > T_W$;
We let $N(X)$ be the number of WLAN packets that hit a Bluetooth packet.

$$N(X) = \begin{cases} \lceil \frac{T_B}{T_{WI}} \rceil \\ \qquad \text{if } X \leq T_{WI} \lceil \frac{T_B}{T_{WI}} \rceil - T_B \\ \lceil \frac{T_B}{T_{WI}} \rceil + 1 \\ \qquad \text{otherwise} \end{cases} \tag{6}$$

We also de ne T_i as the interval of time overlap with WLAN packet i.

$$T_i = \begin{cases} max(T_W - X, 0) \\ \qquad \text{if } i = 1 \\ T_W \\ \qquad \text{if } i = 2, .., N(X) - 1 \\ min(X + T_B - (N(X) - 1) \times T_{WI}, T_W) \\ \qquad \text{if } i = N(X) \end{cases} \tag{7}$$

In this case T_C is basically the sum of all T_i's over $N(X)$ colliding WLAN packets.

$$T_C = \sum_{i=1}^{N(X)} T_i \tag{8}$$

V. Simulation Results

Our goal in this section is to validate the analytical interference model presented in section IV. We used *OPNET*[1] to develop a simulation model for the Bluetooth protocol. We partially implement the Baseband and L2CAP layer according to the speci cations [2] and use the con guration and system parameters shown in Table I. We assume that a connection is already established between the master and the slave and that the synchronization process is complete. The connection type is either SCO for voice or ACL for data traf c. For WLAN we use the models provided by the OPNET modeler's library.

For the Bluetooth signal we assume a pair of devices; a master and a slave device located at (0,0) and (1,0) meters respectively. Master and slave devices are transmitting either voice or data traf c. For voice traf c, we consider a symmetric stream of 64 kbits/s each way. We use $HV1$ packets that have a total size of 366 bits including a header and an access code of 126 bits. $HV1$ packets are sent every $T_{SCO} = 2$ or 1250 μs. $HV1$ payload bits are corrected with a 1/3 FEC rate.Since the payload does not have a CRC, errors in the payload do not yield to dropping packets. In addition, a 1/3 FEC rate is applied to the header and a Hamming code ($d = 14$) is applied to the access code. Uncorrected errors in either the header or the access code lead to dropping packets. For the data traf c, we consider a LAN access application. Both master and slave devices generate $DM5$ type packets every 0.01250 seconds, thus utilizing 50% of the 1 Mbits/s channel. $DM5$ packets have a total size of 2871 bits, including a 54-bit header and a 72-bit access code and occupy 5 Bluetooth slots. A 2/3 FEC rate is used to correct payload errors, while errors in the header or access code are corrected with a 1/3 FEC and a Hamming code ($d = 14$) respectively. Uncorrected errors in either the packet header or payload lead to dropping packets.

TABLE I
Simulation Parameters

System Parameters	**Values**
Propagation delay	5 μs/km
Length of simulation run	30 seconds
Length of run prior to gathering statistics	10 % of simulated time
Bluetooth Parameters	**Values**
Data Rate	1 Mbits/s
ACL Baseband Packet Encapsulation	DM5
SCO Baseband Packet Encapsulation	HV1
Number of Devices	2 (1 Master, 1 Slave)
Master Coordinates	(1,0) (meters)
Slave Coordinates	(0,0) (meters)
Transmitted Power	1 mW
WLAN Parameters	
Packet Interarrival Time for 1 Mbits/s	10.56 ms
Packet Interarrival Time for 11 Mbits/s	2.52 ms
Transmitted Power	1 mW
Source Coordinates	(0,0.15) (meters)
Sink Coordinates	(0,10) (meters)
Packet Header	224 bits
T_W	includes Packet Header
T_{WI}	includes Backoff and T_W
Slot Time	$2 * 10^{-5}$ seconds
SIFS Time	$1 * 10^{-5}$ seconds
DIFS Time	$5 * 10^{-5}$ seconds
CW_{min}	31
CW_{max}	1023
Fragmentation Threshold	None
RTS Threshold	None
Short Retry Limit	4
Long Retry Limit	7

For the WLAN signal, we use two 802.11 Direct Sequence devices transmitting at 1 Mbits/s. We assume unidirectional traf c; a WLAN source transmits packets to a WLAN sink that returns ACK messages to the source. The WLAN source and sink devices are located at (0,0.15) and (0,10) meters respectively. Traf c sent from the WLAN source constitute the interference signal to the Bluetooth slave device.

We present the results from two different simulation experiments that show the impact of interference on Bluetooth devices for different applications, namely voice and data traf c.

Experiment 1- We vary the WLAN packet length, T_W, and the interarrival packet, T_{WI}, while keeping the WLAN offered load x ed at 50% of the 1 Mbits/s channel capacity. Thus, T_W and T_{WI} are varied from 500 and 1000 μs to 8000 and 16000 μs respectively. Note that T_B and T_W denote the packet length in time and are also equivalent to the packet size in bits assuming a data rate of 1 MBits/s.

Experiment 2- We x T_W at 1000 μs and vary T_{WI} ac-

[1] OPNET is a trademark of OPNET Technologies Inc.

cording to $\frac{T_W}{OL}$ where OL is the offered load as a percentage of the 1 Mbits/s channel capacity.

Table II summarizes the experiments.

TABLE II
VALIDATION EXPERIMENT SUMMARY

Experiment	WLAN Offered Load	WLAN Traf c
Experiment 1	50% of Channel Capacity	T_W and T_{WI} variable
Experiment 2	Variable	$T_W = 1000\ \mu$s, $T_{WI} = \frac{T_W}{OL}$

Given that the WLAN source is at a distance, $d_I = 0.15$m from the Bluetooth slave, while the Bluetooth master is at a distance, $d_M = 1$m, and assuming that both the WLAN source and the Bluetooth master device transmit at 1mW, the SIR at the slave is given by $20log\frac{d_I}{d_M} \approx -16$ dB [2]. The choice of the BER value corresponding to this SIR is based on the PHY results of the Bluetooth receiver used [8]. We note that when the Signal-to-noise ratio (SNR) is above 25 dB and the SIR is below -10 dB, the BER is ~ 0.3 for Bluetooth frequency offsets of 10 MHz from the WLAN DS center frequency. Therefore, we use $BER = 0.3$ and $n = 10$ in our analysis. A summary of the parameters used in the analysis is provided in Table III.

TABLE III
ANALYSIS PARAMETERS

Parameters	Values
T_B	366 for $HV1$ and 2871 for $DM5$
T_W and T_{WI}	Variable
n	20
BER	0.3

All simulations are run for 1000 seconds of simulated time and the rst 10 % of the data is discarded. The performance measurements are logged at the slave device. The metric we use includes the packet loss, P_L, and the packet error, P_E. The packet loss is the number of packets discarded due to uncorrected errors in the packet divided by the total number of packets transmitted. While the packet error is the number of packets received with at least one error (prior to applying error correction on the packet and deciding whether to keep it or drop it). Note that Equation 3 captures the probability that a packet containing at least one error is received at the Bluetooth node. Since different error correcting schemes are applied on different packet types and packet segments, this corresponds to the packet error metric rather than the packet loss.

The simulation model used for this validation assumes the following:

- The WLAN CCA is limited to carrier sense functionality capable of detecting other WLAN devices of the same kind (either FH or DS) but cannot detect the presence of Bluetooth devices.
- The impact of Bluetooth interference on WLAN is disabled in order not to change the WLAN traf c distribution. That is interference from Bluetooth does not cause errors at the WLAN receiver.

[2] Assuming the logarithmic path loss model given in [4]

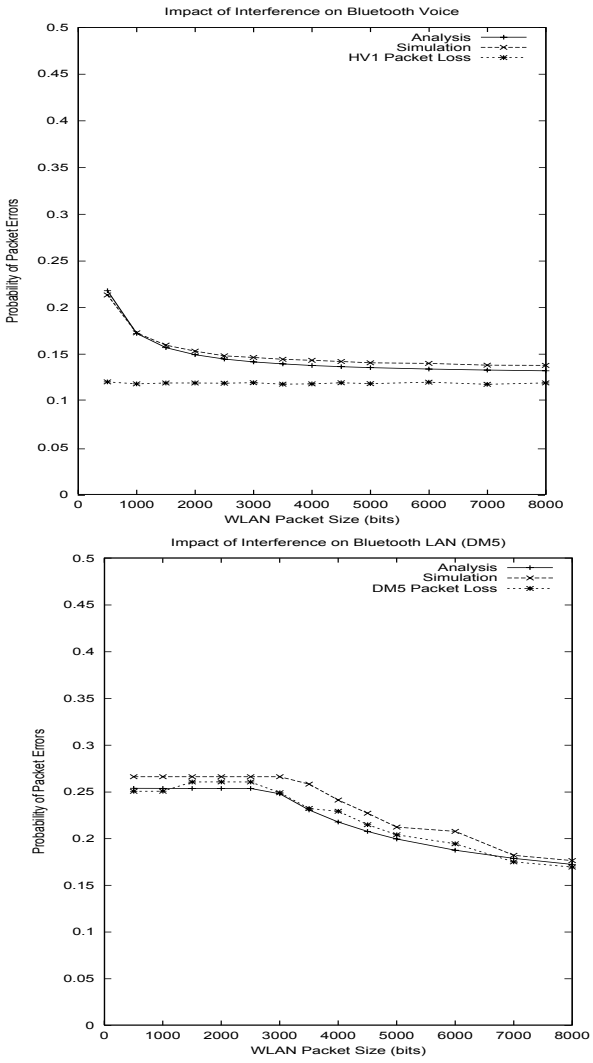


Fig. 4. $\frac{(a)}{(b)}$ Varying T_W and T_{WI} for a 50% WLAN Offered Load (a) Impact of WLAN Interference on Bluetooth Voice (b) Impact of WLAN Interference on Bluetooth LAN

- The BER value used in the Bluetooth receiver is computed according to the receiver's DSP model and varies according to the frequency hop and the signal to interference ratio.

Figure 4 (a) gives the probability of packet error for the Bluetooth voice traf c for different WLAN packet lengths. We note that the analytical results closely approximate the simulation results. The probability of packet error varies between $\sim$ (22% - 13%), while the probability of packet loss remains at $\sim$ 12%. As expected, the packet loss is lower than the packet error due to the use of different error correction schemes applied on different segments of the packet. We note that errors occurring in the payload of $HV1$ packets do not lead to dropping packets. Furthermore, if errors in the header can be corrected the packet is kept, otherwise the packet is dropped. This explains the difference between the packet loss and the packet error.

A similar trend applies to the Bluetooth LAN results given in Figure 4(b). The packet error varies between $\sim$ (25% - 17%). The difference between the packet loss and the packet error is

not as signi cant as in Experiment 1 (a). In fact, the decision to drop $DM5$ packets is based on uncorrected errors in either the header or the payload. Therefore, the packet loss and packet error measures are very close.

Figure 5(a) and (b) illustrate the effect of varying the WLAN offered load on the Bluetooth voice and LAN performance respectively. The probability of voice packet error and packet loss increase proportionally to the WLAN offered load (Figure 5 (a)). We also note that the difference between the packet error and the packet loss is signi cant ($\sim$10%) at high WLAN offered loads (65%). Note that only packet header collisions affect the packet loss. As more interfering packets are transmitted (increase in WLAN offered load), only a small number of them will "hit" the header and cause a collision.

The results for the Bluetooth LAN are given in Figure 5(b)). The increase in packet error levels off at $\sim$ 25% for WLAN offered loads greater than 25%. This "threshold" phenomenon is a direct effect of having reached an error threshold number per packet. Additional errors above that threshold do not yield to more packets being dropped.

VI. Concluding Remarks

We presented results on the performance of Bluetooth in the presence of WLAN interference based on a probability of packet collision in frequency and time overlap at the Bluetooth receiver. We rst observe that the probability of packet error analysis, in the tractable case where mutual interference effects are not considered and only a particular receiver is studied, can provide a close approximation to the packet error and the packet loss measures. Furthermore, the results clearly show that packet loss due to interference may be signi cant (up to 27% for data traf c and 25% for voice applications) and may lead to severe performance degradation. In addition, longer Bluetooth packets (such as $DM5$ packets) are more prone to packet loss than shorter packets ($HV1$). Note that, although the packet loss is lower than the packet error for voice traf c, the quality of the audio channel is likely to be impaired due to the high number of residual errors in the payload.

More generally, the experiments stress the importance of de ning accurate traf c models and distributions in the evaluation of interference. Both the offered load and the packet length are necessary parameters in order to completely specify the interference signal.

Our future work includes investigating simulation scenarios where both WLAN and Bluetooth interference can be studied together. This may unravel various intricate effects about the traf c distribution and the overall system performance of Bluetooth and WLAN operating in the 2.4 GHz frequency band.

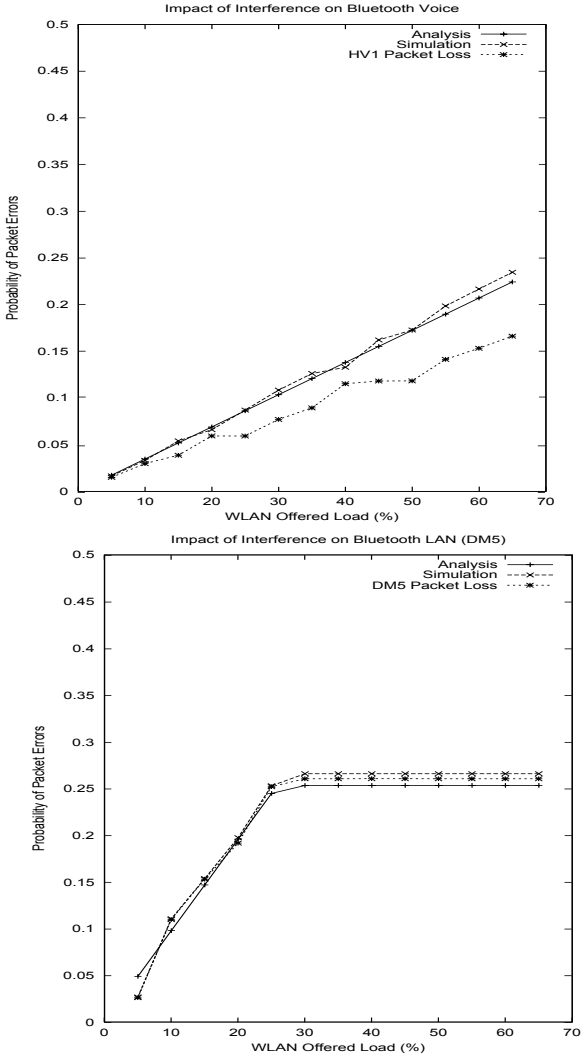


Fig. 5. $\frac{(a)}{(b)}$ Varying WLAN Offered Load (T_W=1000 bits) (a) Impact of WLAN Interference on Bluetooth Voice (b) Impact of WLAN Interference on Bluetooth LAN

References

[1] IEEE Std. 802-11, "IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Speci cation ," June 1997.

[2] Bluetooth Special Interest Group, "Speci cations of the Bluetooth System, vol. 1, v.1.0B 'Core' and vol. 2 v1.0B 'Pro les', " December 1999.

[3] S. Zurbes, W. Stahl, K. Matheus, and J. Haartsen, "Radio network performance of bluetooth ," in *Proceedings of IEEE International Conference on Communications, ICC 2000*, New Orleans, LA, June 2000, vol. 3, pp. 1563–1567.

[4] A. Kamerman, "Coexistence between Bluetooth and IEEE 802.11 CCK: Solutions to avoid mutual interference," in *IEEE P802.11 Working Group Contribution, IEEE P802.11-00/162r0*, July 2000.

[5] G. Ennis, "Impact of Bluetooth on 802.11 Direct Sequence," in *IEEE P802.11 Working Group Contribution, IEEE P802.11-98/319*, September 1998.

[6] S. Shellhammer, "Packet Error Rate of an IEEE 802.11 WLAN in the Presence of Bluetooth," in *IEEE P802.15 Working Group Contribution, IEEE P802.15-00/133r0*, Seattle, Washington, May 2000.

[7] C.F. Chiasserini, R. Rao, "Performance of IEEE 802.11 WLANs in a Bluetooth Environment," in *IEEE Wireless Communications and Networking Conference, WCNC 2000*, Chicago, IL, September 2000.

[8] A. Soltanian and R. E. Van Dyck, "Physical layer performance for coexistence of Bluetooth and IEEE 802.11b," in *to appear in Virginia Tech Symposium on Wireless Personal Communications*, June 2001.

Physical Layer Performance for Coexistence of Bluetooth and IEEE 802.11b

Amir Soltanian and Robert E. Van Dyck
National Institute of Standards and Technology
Gaithersburg, MD 20899
{amirs, vandyck}@antd.nist.gov

Abstract - Simulations are employed to evaluate the physical layer performance of Bluetooth and IEEE 802.11b receivers, mainly but not exclusively, in interference-limited environments. For Bluetooth, a limiter-discriminator with post-detection integrate and dump filtering (LDI) is used. Bit error rate curves are obtained for co-channel and adjacent channel(s) interference in AWGN and flat fading channels. The interference in this case may be a different Bluetooth piconet or an 802.11b transmitter. For 802.11b, differentially coherent 1 Mbit/sec and coherent 11 Mbits/sec receivers are studied with Bluetooth interference, again obtaining BER curves for both AWGN and fading channels. Finally, the LDI is replaced with a coherent Viterbi receiver, and the performance in the interference-limited environment is measured.

1 Introduction

With the coming deployment of Bluetooth wireless personal area networks (WPANs) [1] in the 2.4-GHz ISM band, there is a growing concern about coexistence with other existing systems, especially the IEEE 802.11b wireless local area network (WLAN). The Bluetooth system employs frequency hopping to mitigate the effect of interference and fading channel impairments. A direct sequence spread spectrum (DSSS) 802.11b system occupies approximately 22 MHz in the same band. Therefore, the Bluetooth system will consistently hop into the 802.11b spectrum, causing interference to both. Additionally, the frequency hopping pattern for different Bluetooth networks, called piconets, are not coordinated, so that multiple piconets operating in the same geographic area will interfere with each other.

This paper addresses the coexistence issue by applying baseband models for the physical layers of the two systems. Using appropriate channel models based on recent measurements, we determine the performance degradation in a prototype environment.

The Bluetooth system operates at a channel bit rate of 1 Mbit/sec [2]. The modulation is Gaussian frequency shift keying (GFSK) with a nominal modulation index of $h_f = 0.32$ and a normalized bandwidth of $B_bT = 0.5$, where B_b is the 3 dB Bandwidth of the transmitter's Gaussian low pass filter, and T is the bit period. The Bluetooth radio employs a frequency hopping scheme in which the carrier frequency is changed on a packet by packet basis. There are up to 79 different channels each with 1 MHz separation. The primary communication range is 10 m, but it can be extended up to 100 m. The entire structure of the simulated system is presented in Fig. 1. It includes the transmitter, the channel noise, the receiver and the interference source. The interference source can be selected as either Bluetooth or 802.11b type interference. Note that the interferer can be set to have a different carrier frequency and a random phase offset.

The IEEE 802.11b standard describes four modulation methods providing bit rates of 1, 2, 5.5, and 11 Mbits/sec [3]. The first rate is achieved by differential BPSK (DBPSK) with DSSS using an 11 chip Barker code; the chip rate is 11 Mchips/sec. The last rate is obtained using complementary code keying (CCK), also at

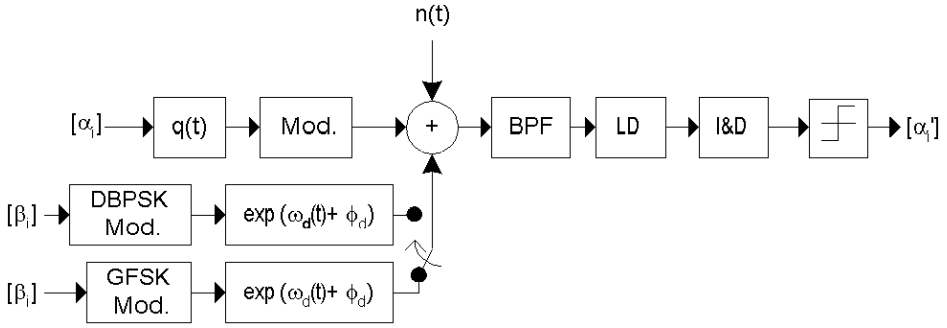


Figure 1: Bluetooth system model.

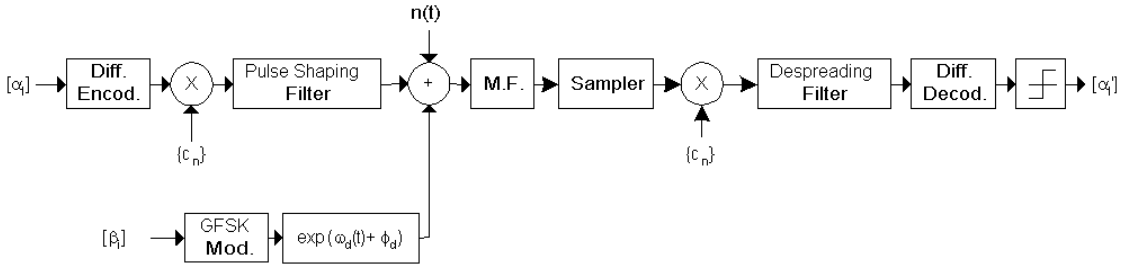


Figure 2: 802.11b DSSS system model.

11 Mchips/sec. In this study, we consider the 1 and 11 Mbits/sec bit rates. The communications system model for the 1 Mbit/sec bit rate is presented in Fig. 2, again consisting of the transmitter, the channel noise, the receiver and the Bluetooth interference source. We explain the details of this model in the following sections.

2 Bluetooth System Model

2-1 The GFSK signal

The GFSK signal can be represented by [4]

$$s(t,\mathbf{a}) = A\cos(2\pi f_c t + \phi(t,\mathbf{a})), \qquad (1)$$

where $A = \sqrt{\frac{2E_b}{T}}$, E_b is the energy per data bit, and f_c is the carrier frequency. $\mathbf{a}$ is the random input stream, comprised of the data bits α_i; $\phi(t,\mathbf{a})$ is the output phase deviation, given by

$$\phi(t,\mathbf{a}) = 2\pi h_f \int_{-\infty}^{t} \sum_{i=-\infty}^{n} \alpha_i g(\tau - iT)d\tau. \qquad (2)$$

One of the key ideas in GFSK is that a single bit is transmitted over multiple symbols, which is done by using a pulse shaping filter with impulse response $g(t)$ given by

$$g(t) = \frac{1}{2T}[Q(2\pi B_b \frac{t-\frac{T}{2}}{\sqrt{ln2}}) - Q(2\pi B_b \frac{t+\frac{T}{2}}{\sqrt{ln2}})], \qquad (3)$$

where $Q(t)$ is the standard Q-function $Q(t) = \int_t^\infty \frac{1}{\sqrt{2\pi}} e^{-\tau^2/2}$. By introducing controlled intersymbol interference, the spectral occupancy of the signal is substantially reduced.

Eq. (2) can also be written as

$$\phi(t,\mathbf{a}) = 2\pi h_f \sum_{i=n-L+1}^{n} \alpha_i q(t-iT) + \pi h_f \sum_{i=-\infty}^{n-L} \alpha_i, \qquad (4)$$

where L is the length of $g(t)$, and

$$q(t) = \int_{-\infty}^{t} g(\tau)d\tau. \qquad (5)$$

For Bluetooth with $B_bT = 0.5$, we have $L = 2$, which means that a single data bit is spread over two consecutive symbol intervals.

2-2 Channel Model

Recent measurements in the ISM band show that the root-mean-square (rms) average of the excess delay for the multipath component is around 30 nsec in an office environment [5, 6]. In another study, Kim *et al.* [7] found that the rms values were below 70 nsec, with an average value of approximately 50 nsec. For a line-of-sight (LOS) path, the power spectral density of the faded amplitude is close to a Rician distribution [7]. Therefore, we chose a flat fading channel model with a Rician distribution. This model assumes that a direct path exists between the transmitter and the receiver, and there are also other low-level scattered paths. The probability density function (pdf) of the Rician distribution is

$$P_R(r) = \frac{r}{\sigma^2} e^{-\frac{-r^2+\nu^2}{2\sigma^2}} I_0(\frac{r\nu}{\sigma^2}) \; r \geq 0, \qquad (6)$$

where I_0 is the zeroth-order modified Bessel function of the first kind, ν is the envelope of the strong component, and σ^2 is proportional to the power of the "scattered" Rayleigh component.

The Rician factor K is the ratio of the power in the direct path P_S to the power in the diffuse path P_d,

$$K = \frac{P_S}{P_d}. \qquad (7)$$

As K approaches zero, the channel behaves as Raleigh fading, whereas as K goes to infinity, the channel is Gaussian.

Two important parameters associated with the receiver are the average carrier-to-noise ratio $\overline{CNR}$, and the average carrier-to-interference ratio $\overline{CIR}$ defined as

$$\overline{CIR} = \frac{P_S + P_d}{P_I}, \qquad \overline{CNR} = \frac{P_S + P_d}{P_n}; \qquad (8)$$

P_I is the interference power, while P_n is the noise power in the receiver's frequency band.

In this simulation, we first consider the AWGN channel model, and then we apply the Rician fading. Also, because of the static behavior of the indoor channels, the Doppler shift frequency is ignored.

2-3 Interference Model

Either a Bluetooth or an 802.11b interference signal can be represented as

$$S_I(t, \mathbf{b}) = B\cos(2\pi(f_c + f_d)t + \phi_2(t, \mathbf{b})), \qquad (9)$$

where $\mathbf{b}$ is the random input data, which is independent of $\mathbf{a}$, and ϕ_2 depends on the type of the interferer. f_d is the frequency difference between the desired signal and the interference. The Bluetooth radio channels are 1 MHz apart, so f_d can take values of $0, 1, 2 \cdots$ MHz. The bandwidth of the 802.11b system is 22 MHz, so we carried out simulations for $f_d \leq 11$ MHz. The sampling rate is $N_s = 44$ samples/bit, which equals 4 samples/chip for the 802.11 DSSS system. This sampling rate is appropriate for f_d up to 22 MHz.

When studying interference, we want to reduce the effects of the traffic patterns and concentrate on the effects of time/frequency overlap between the two systems. For Bluetooth performance, we make these two assumptions: (1.) that the 802.11b WLAN is constantly transmitting and (2.) that any other Bluetooth piconets are synchronized to packet boundaries and are also transmitting. These correspond, in some sense, to worst case scenarios. In a real system, there will be times when the interferer is off. For the 802.11b system, we assume that the Bluetooth interferer is also always transmitting.

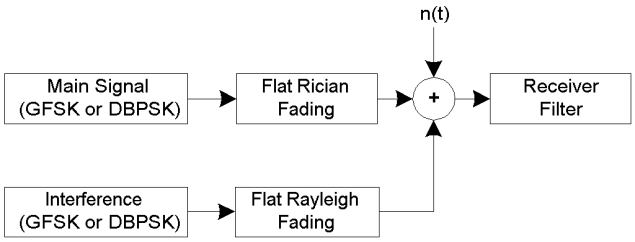


Figure 3: Interference model for the fading channels.

In the AWGN channel, a uniform random delay $t_d \in [0,\ T)$ and a random phase $\phi_d \in [0,\ 2\pi)$ are applied to the interferer signal for each packet. For the fading channels, it is assumed that the intended signal is subjected to Rician fading, whereas the interference undergoes Rayleigh fading. Fig. 3 show a block diagram of this configuration. We chose $K = 5$ ($K \simeq 7$ dB) for the

desired signal, which is close to the profile recommended in [7].

2-4 LDI Receiver

This receiver consists of a pre-detection bandpass filter, a limiter-discriminator, and an integrate and dump filter, as shown in Fig. 1. The final block is the hard limiter, which compares the output phase with a decision level. The pre-detection bandpass filter is a Gaussian filter with an equivalent lowpass impulse response, $h_r(t)$, given by [8]

$$h_r(t) = \sqrt{\frac{2\pi}{ln2}} B_r \, e^{-(\frac{2\pi^2}{ln2})(B_r t)^2}, \qquad (10)$$

where B_r is the 3 dB bandwidth. According to Simon and Wang [8], the optimum bandwidth for this filter is $B_{IF} = 2B_r = 1.1/T$. The discrete impulse response of this filter was obtained by sampling and truncating $h_r(t)$.

The output of the receiver pre-detection filter can be represented using its inphase and quadrature components, $X(t)$ and $Y(t)$, respectively, as

$$\begin{aligned} e(t) &= X(t)\cos(2\pi f_c t) - Y(t)\sin(2\pi f_c t) \\ &= R(t)\cos[2\pi f_c t + \psi(t)]. \end{aligned} \qquad (11)$$

The limiter-discriminator output is thus

$$\psi'(t) = \frac{d\psi(t)}{dt} = \frac{X(t)Y'(t) - X'(t)Y(t)}{X^2(t) + Y^2(t)}. \qquad (12)$$

The discrete impulse response of an ideal differentiator is [9]

$$h_{diff}[n] = \frac{\cos(\pi(n - \frac{M}{2}))}{(n - \frac{M}{2})} - \frac{\sin(\pi(n - \frac{M}{2}))}{(n - \frac{M}{2})^2}. \qquad (13)$$

We truncate this impulse response using a Kaiser window with $M = 5$ and $\beta = 2.4$, and then we use it to approximate the derivatives of the quadrature components required in computing Eq. (12). Another approach to implement this filter is to use a simple difference equation.

The integrate-and-dump filter is simply a rectangular filter with impulse response

$$h_{ID}(t) = \begin{cases} \frac{1}{T} & 0 \le t < T \\ 0 & \text{otherwise.} \end{cases} \qquad (14)$$

The discrete-time filter is obtained by sampling $h_{ID}(t)$. The amplitude of the filter were normalized to 1. The appropriate sampling time for the system is chosen at the maximum eye opening.

3 802.11b System Model

3-1 1 Mbit/sec DSSS

The basic 1 Mbit/sec rate is encoded using DBPSK; thus, it is not necessary to have a coherent phase reference in the receiver to demodulate the received signal.

This system utilizes a spread spectrum scheme to mitigate the effect of a jammer. The Barker sequence with code length $P = 11$ is employed to spread the signal. The bit duration, T, is exactly 11 chip periods, T_c, long. The processing gain (PG) of this system is [10] $PG = \frac{R_c}{R_b} = 11$, where $R_b = \frac{1}{T}$ is the bit rate, and $R_c = \frac{1}{T_c}$ is the chip rate. If we calculate the power spectrum of the Barker codes, we get [11]

$$\begin{aligned} S(f) &= \sum_{\substack{k=-\infty \\ k \neq 0}}^{\infty} (\frac{P+1}{P^2}) sinc^2(\frac{k}{P}) \delta(f - \frac{k}{PT_c}) \\ &\quad + \frac{1}{P^2}\delta(f). \end{aligned} \qquad (15)$$

The function, $S(f)$, is illustrated in Fig. 4 for $P = 11$. We see that a narrowband interference signal -like Bluetooth- located at the middle of the spectrum will be more attenuated than an interferer located 1 MHz away from the middle of the spectrum.

As shown in Fig. 2, the input data bits are first differentially encoded. The resulting sequence is spread by the Barker code. The output of the spreader is fed to a Square-Root Raised-Cosine (SR-RC) pulse-shaping filter. The impulse response of the SR-RC filter with a roll-off factor α is [11]

$$\begin{aligned} p(t) &= \frac{sin[(1-\alpha)\pi t/T_c]}{(\pi t/T_c)[1 - (4\alpha t/T_c)^2]} \\ &\quad + \frac{4\alpha(t/T_c)cos[(1+\alpha)\pi t/T_c]}{(\pi t/T_c)[1 - (4\alpha t/T_c)^2]}. \end{aligned} \qquad (16)$$

The discrete time impulse response of this filter is obtained by sampling $p(t)$.

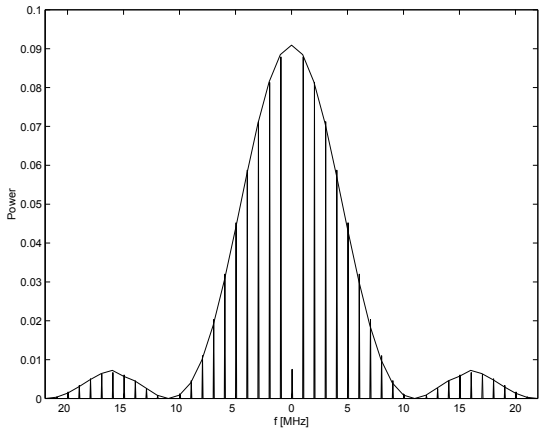


Figure 4: Power Spectrum of the Barker Code.

At the receiver, the input samples are first passed through the SR-RC matched filter. The despreading filter is a rectangular filter that integrates the output of the multiplier during a bit period. The differential decoder compares the phase angle of the received symbol and the previous one to generate the output bit stream. It is assumed that the chip timing of the receiver is synchronized to the transmitter.

3-2 11 Mbits/sec CCK

Complementary codes were originally conceived by M. J. Golay for infrared multislit spectrometry [12]. These codes can be considered block codes over the field of complex numbers. Let the kth code word be given by $\mathbf{s}_k = [s_{k1} s_{k2} \cdots s_{kN}]^T$, where N is the length of the code word, and $k = 1, 2, \cdots, K$. The autocorrelation of the code word is given by [13]

$$R_{kk}[j] = \sum_{i=1}^{N-j} s_{ki}\, s^*_{k(i+j)}. \qquad (17)$$

A set of K codes is considered complementary if and only if it satisfies the following equation

$$\sum_{k=1}^{K} R_{kk}[j] = \begin{cases} 0 & \text{for } j \neq 0 \\ KN & \text{for } j = 0. \end{cases} \qquad (18)$$

The complementary codes in the 802.11b standards are defined by a set of 256 8-chip code words. They are specified by

$$\begin{aligned} c = [&e^{j(\phi_1+\phi_2+\phi_3+\phi_4)}, \quad e^{j(\phi_1+\phi_3+\phi_4)}, \\ &e^{j(\phi_1+\phi_2+\phi_4)}, \quad -e^{j(\phi_1+\phi_4)}, \\ &e^{j(\phi_1+\phi_2+\phi_3)}, \quad e^{j(\phi_1+\phi_3)}, \\ &-e^{j(\phi_1+\phi_2)}, \quad e^{j(\phi_1)}], \end{aligned} \qquad (19)$$

where

$$\phi_i \in \{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\} \quad \text{for } i = 1, 2, 3, 4. \qquad (20)$$

Note that each element of a code word is complex, and so can be transmitted using QPSK modulation as discussed below.

At 11 Mbits/sec, 8 bits (d0 to d7; d0 first in time) are transmitted per code word. The first dibit (d0, d1) encodes ϕ_1 based on DQPSK, which provides the possibility of employing differentially-coherent detection. In this study, we employ a coherent receiver, assuming that the initial phase of the signal is known. The dibits, (d2, d3), (d4, d5), and (d6, d7) encode ϕ_2, ϕ_3, and ϕ_4, respectively, as specified in Table 1.

Dibit Pattern (d_i, d_{i+1})	Phase
00	0
01	$\frac{\pi}{2}$
10	π
11	$\frac{3\pi}{2}$

Table 1: QPSK Encoding.

The system model is presented in Fig. 5. Only an AWGN channel is considered in this case.

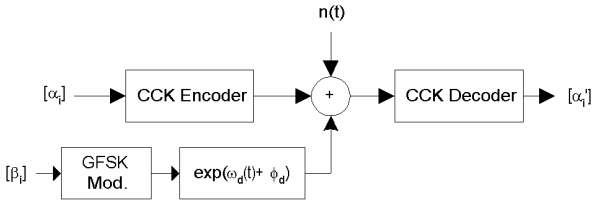


Figure 5: CCK System Model.

The decoder determines the valid code word that is closest to the received signal, and it maps that code word back to data bits. It is well known that in an AWGN channel, a code set which has the largest minimum Euclidean distance between code words yields the lowest bit error rate. Thus, an optimal code set for an AWGN channel would maximize the following minimum distance

$$\begin{aligned} d_{min} &= \min \|\mathbf{s}_k - \mathbf{s}_l\|^2 \\ k, l &\in \{1, 2, \cdots, K\}. \end{aligned} \qquad (21)$$

For complementary codes with length N and M possible phases, it can be shown that the minimum Euclidean distance is equal to [14]

$$d_{min} = \sqrt{\frac{N}{2}\|1 - exp(j\frac{2\pi}{M})\|}. \quad (22)$$

For CCK with $N = 8$ and $M = 4$, the minimum distance is 2.82, which is 3 dB better than the distance of uncoded QPSK whose $d_{min} = 1.4$.

The maximum likelihood method described above needs a bank of 256 correlators in the receiver. Although optimum, this method may be considered too complex for some implementations. There are also less complex sub-optimum algorithms. By looking at the code words of CCK, one can write these equations for the decoded phases [14]

$$\begin{aligned} \phi_2 &= arg\{r_1 r_2^* + r_3 r_4^* + r_5 r_6^* + r_7 r_8^*\} \quad (23)\\ \phi_3 &= arg\{r_1 r_3^* + r_2 r_4^* + r_5 r_7^* + r_6 r_8^*\} \\ \phi_4 &= arg\{r_1 r_5^* + r_2 r_6^* + r_3 r_7^* + r_4 r_8^*\} \\ \phi_1 &= arg\{r_4 e^{-j\phi_4} + r_6 e^{-j\phi_3} + r_7 e^{-j\phi_2} + r_8\}, \end{aligned}$$

where

$$\mathbf{r} = [r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8] \quad (24)$$

is the received vector. We employ the above sub-optimal receiver to measure the performance in the presence of interference.

4 Link Budget

For a transmitter at a power level of P_T, the extracted power by the receiver's antenna may be expressed in decibels as

$$P_R = P_T + G_T + G_R - L_p - L_a, \quad (25)$$

where G_T and G_R are the antenna gains of the transmitter and the receiver, respectively, L_p is the path loss, and L_a accounts for any additional system loss. For an indoor channel, we apply a simple propagation model: line-of-sight propagation (free space) for the first 8 m, thereafter a propagation exponent of 3.3. The path loss can be expressed as [15]

$$L_p = \begin{cases} 40 + 20\log(d) & \text{for } d < 8 \text{ m} \\ 58.3 + 33\log(d/8) & \text{for } d \geq 8 \text{ m.} \end{cases} \quad (26)$$

Assuming 0 dB gain for the transmitter and the receiver antennas and ignoring additional loss, Eq. (26) can be written as

$$P_R = P_T - L_p. \quad (27)$$

The sensitivity of the receiver, R_{sens}, which is the minimum amount of signal power required to achieve a certain bit error rate (BER), may be expressed as

$$R_{sens} = P_n + CNR_{req}. \quad (28)$$

Here, P_n is the receiver noise power and CNR_{req} is the required carrier-to-noise ratio to achieve a $BER = 10^{-3}$ in an AWGN channel. The CNR is defined as

$$CNR = \frac{E_b}{N_0}\frac{1}{BT}. \quad (29)$$

For Bluetooth, our simulation shows that at $CNR \simeq 16$ dB, we get a $BER = 10^{-3}$. Also, R_{sens} may be typically selected as R_{sens}=−80 dbm. By inserting these values, the receiver noise is then

$$P_n = R_{sens} - CNR_{req.} = \text{-96 dBm.} \quad (30)$$

Using Eq. (27) and given P_n, we can calculate Bluetooth's CNR and CIR values for a chosen topology. Table 2 shows the CNR values for $P_T = 1$ mW.

Distance [m]	1	3	5	10	15
CNR [dB]	56	46	42	35	29

Table 2: CNR values for Bluetooth.

The above calculations can be repeated for the 802.11b receiver assuming $R_{sens} = -80$ dBm, and $CNR_{req} = 8$ dB at $BER = 10^{-3}$. Table 3 contains the results for $P_T = 25$ mW.

Distance [m]	1	3	5	10	15
CNR [dB]	62	52	48	40	35

Table 3: CNR values for 802.11b.

For distances less than 15 meters, the CNR is high enough so the errors are mainly caused by the interference signal.

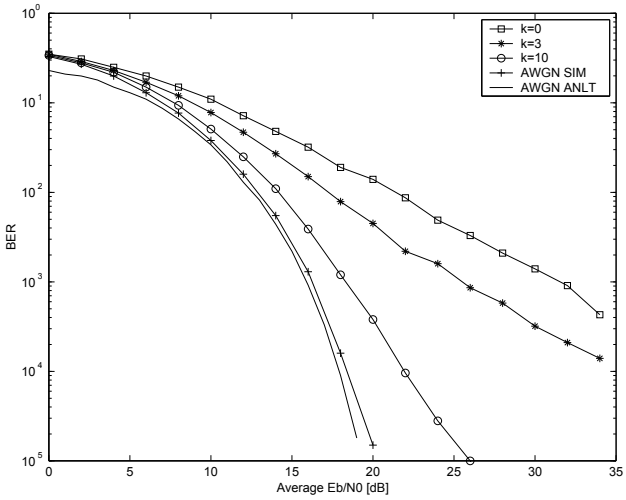


Figure 6: Bluetooth performance in the AWGN and Rician fading channels. LDI receiver.

5 Performance Results

5-1 Bluetooth

Simulation results for the LDI receiver in the AWGN and Rician channels are presented in Fig. 6. We see that for the AWGN case at $E_b/N_0 = 16$ dB, the $BER = 10^{-3}$. For the worst case Rayleigh fading condition ($K = 0$), an $E_b/N_0 = 30$ dB is required for the same BER. Also in this figure, we present the analytical BER curve using the method described in [16, 8] for the AWGN case.

Interference Type	Ratio (dB)
$C/I_{co-channel}$	11
C/I_{1MHZ}	0
C/I_{2MHZ}	-30
$C/I_{\geq 3MHZ}$	-40

Table 4: Specified CIR values for Bluetooth interference.

For Bluetooth interference on a Bluetooth signal, there is a specific requirement according to the standard, *i.e.* the BER shall be less than 0.1% in any of the conditions given in Table 4. The performance is measured with the desired signal 10 dB over the reference sensitivity level. Fig. 7(a) shows the performance in this case. We observe that the LDI receiver can meet the requirement set by the standard. Fig. 7(b) presents the performance with Bluetooth co-channel interference

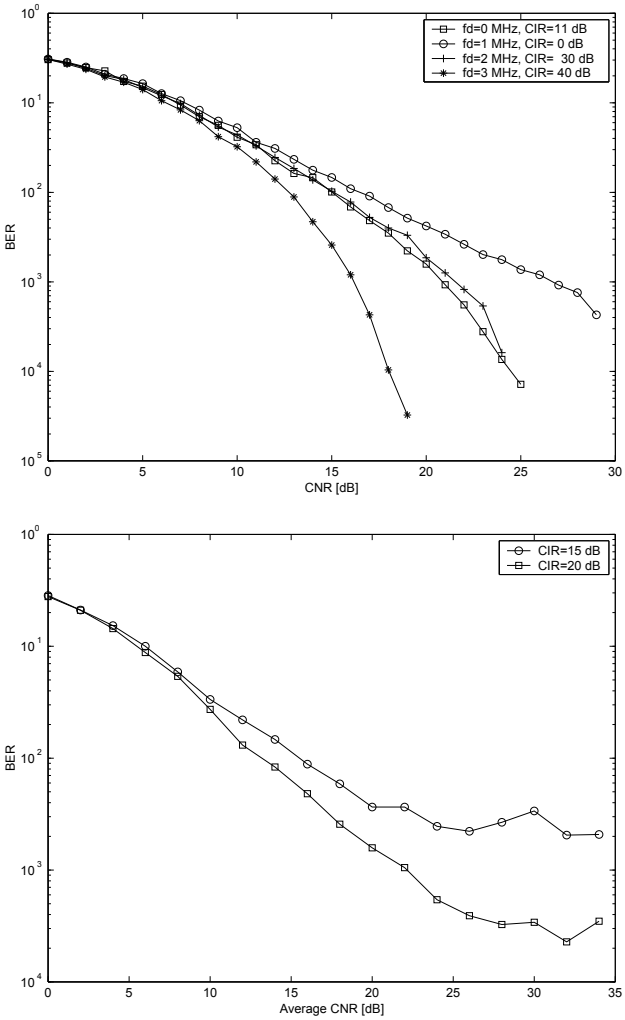


Figure 7: $\boxed{\frac{(a)}{(b)}}$ Bluetooth performance with Bluetooth interference. (a) AWGN channel. (b) Rician channel. LDI receiver.

in the Rician fading channel. As mentioned in Section 2-3, $K = 5$ for the signal, and $K = 0$ (Rayleigh) for the interference. The $\overline{CIR}$ value should be at least 20 dB in order to get low BER for the co-channel interference.

Next, we study the performance of Bluetooth with 802.11b interference. The curves in Figs. 8(a) and (b) are for an interference-limited environment with $CNR = 30$ dB. The 802.11b signal looks like broadband noise at the input to the Bluetooth receiver. The performance degradation for carrier frequency differences up to 4 MHz is almost the same, and so we plot the results for $f_d = 0$ as a representative case. The null in the Barker code spectrum does not improve the performance here, but it does for the

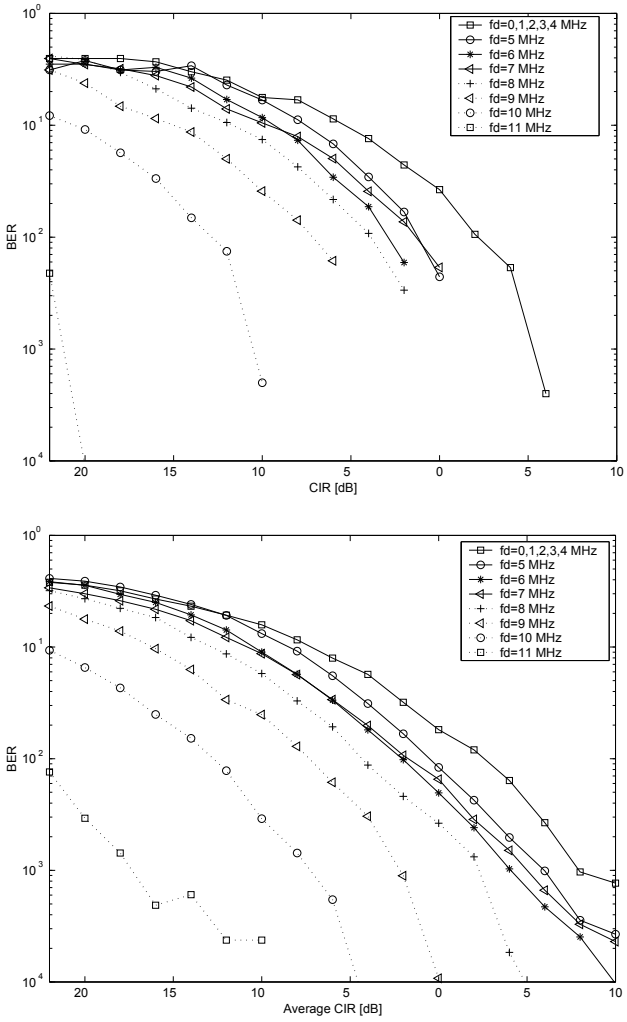


Figure 8: $\boxed{\frac{(a)}{(b)}}$ Bluetooth performance with 802.11b interference. (a) AWGN channel. (b) Rician channel. LDI receiver.

802.11b DSSS system. After 4 MHz, one gradually sees the effect of the pulse shaping filter of the 802.11b transmitter, which has a null at $f_d = 11$ MHz. In fact, the CIR value at $f_d = 11$ MHz has to be very low in order to cause high BER.

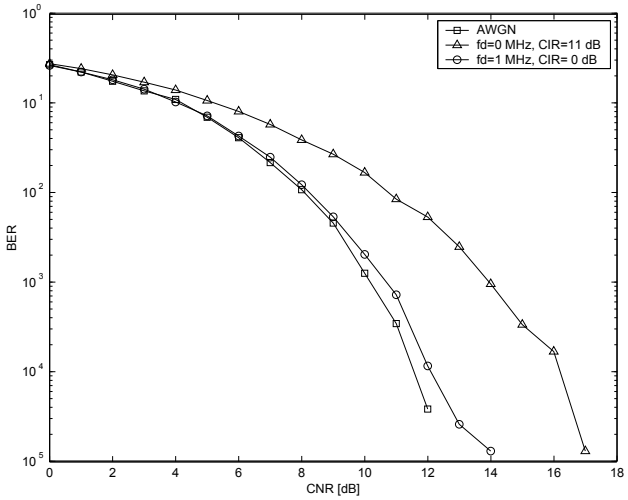
To relate the CIR values to the transmitter powers, consider the topology where the Bluetooth transmitter and the 802.11b interference are both positioned one meter away from the Bluetooth receiver; the Bluetooth transmitter power is 1 mW, while the 802.11b's is 25 mW. Using Eq. (27), $CIR \simeq -14$ dB. So, when a Bluetooth packet hops on a frequency that is less than 10 MHz away from the center of the 802.11 interference, that packet is usually subjected to errors. The roll-off factor α of the 802.11b transmitter determines the range of frequency offsets over which high BERs are observed. In this simulation, we chose $\alpha = 1$, so the interference signal will occupy the maximum available bandwidth. Another observation from Fig. 8(a) is that if the CIR value is always greater than 6 dB, the BER for all frequency offsets is less than 10^{-3}.

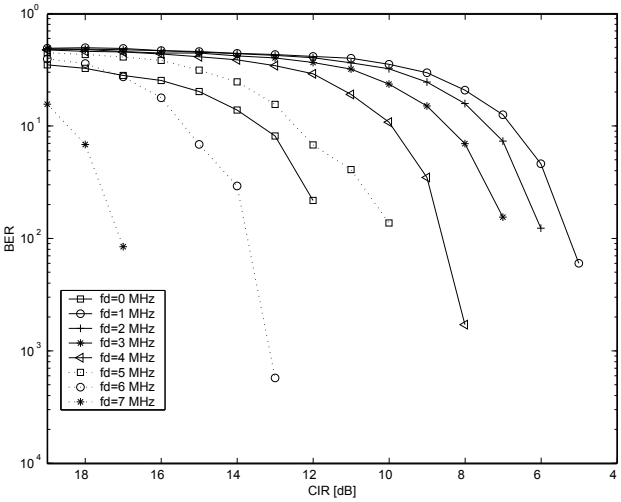
The performance in the Rician channel ($K = 5$ for the signal and $K = 0$ for the interference) is shown in Fig. 8(b). We see that $\overline{CIR} = 10$ dB is the minimum tolerable. In comparison to scenarios where the desired signal undergoes Rayleigh fading or the interference has a LOS path, these results may be considered optimistic. On the other hand, we see that since the interference acts as wideband noise, there is not a great difference in the CIR requirement between the AWGN and Rician channel models.

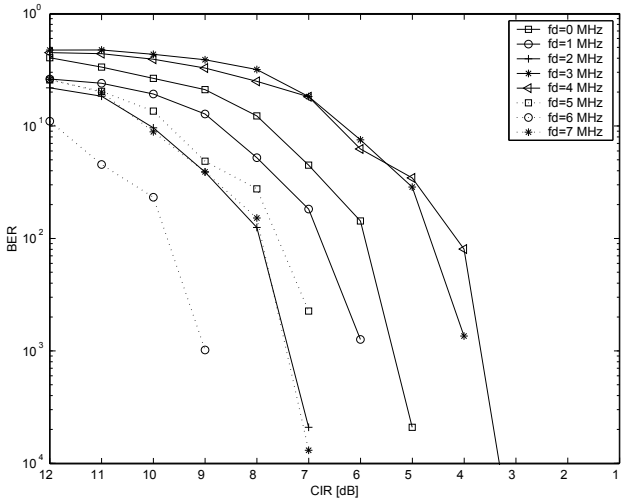
As a solution to mitigate the effect of interference, we use a simple two-state Viterbi receiver for Bluetooth. Again, we assume that the phase of the transmitted signal is known to the receiver. The performances for Bluetooth and for 802.11 interferences are shown in Figs. 9(a) and (b), respectively. A dramatic enhancement is observed in these figures, evidently at a cost of having a more complicated receiver. This improvement is particularly considerable for 802.11b interference, which acts as broadband noise in the Bluetooth receiver's bandwidth.

5-2 802.11b

Now, we consider the performance of the 1 Mbit/sec 802.11b system, again in an interference-limited environment with $CNR = 35$ dB. Since the system takes advantage of DSSS, one observes in Fig. 10(a) that for co-channel interference, $CIR = -11$ dB is adequate to suppress the effect of interference ($BER \leq 10^{-2}$). The most disturbing interference is located at $f_d = 1$ MHz, which needs a minimum CIR of -5 dB. This difference stems from the null at the middle of the spectrum of the Barker code as described before. For frequency offsets grater than 8 MHz, the CIR value must be very low in order to get a high BER. This fact is due







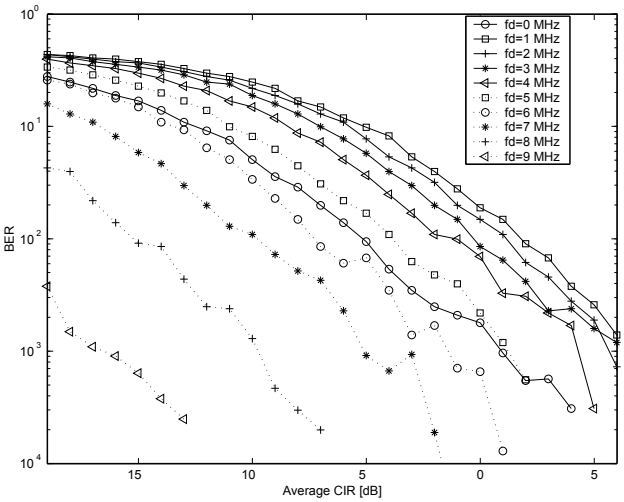


Figure 9: $\boxed{\frac{(a)}{(b)}}$ Bluetooth Viterbi receiver performance. (a) Bluetooth interference. (b) 802.11b interference. AWGN channel.

Figure 10: $\boxed{\frac{(a)}{(b)}}$ 802.11b DSSS performance with Bluetooth interference. (a) AWGN channel. (b) Rician channel.

to the bandpass filter in the 802.11b receiver having high attenuation at frequencies near 11 MHz. The results of this figure are comparable to the analytic method proposed in [17, 18].

Fig. 10(b) indicates the performance in the Rician channel ($K = 5$), where the Bluetooth interference is subjected to Rayleigh fading. The minimum $\overline{CIR}$ in this case is $\overline{CIR} = 2$ dB ($BER = 10^{-2}$). For the 802.11b DSSS system, there is a difference of 7 dB in CIR required to achieve acceptable performance in AWGN and Rician channels, respectively. This difference is greater than the equivalent (4 dB) for the Bluetooth system, using the LDI receiver.

Fig. 11(a) shows the performance of the 11 Mbits/sec 802.11b CCK receiver in the AWGN channel. The optimum receiver performs about 2 dB better than QPSK, and the sub-optimum method is nearly the same as QPSK. The sub-optimal system provides a BER of 10^{-3} for an $E_b/N_o = 8$ dB. It must be noted that CCK was designed explicitly for fading channels, where its gain over QPSK is much more significant.

Fig. 11(b) illustrates the performance of 11 Mbits/sec IEEE 802.11b receiver with Bluetooth co-channel interference. This figure indicates that the CCK modulation is more vulnerable to the interference signal than the 1 Mbits/sec DSSS. A minimum CIR of 3 dB must be achieved to get $BER = 10^{-2}$ for all frequency offsets. This result is not surprising, since the CCK provides a higher bit rate but occupies the same 22

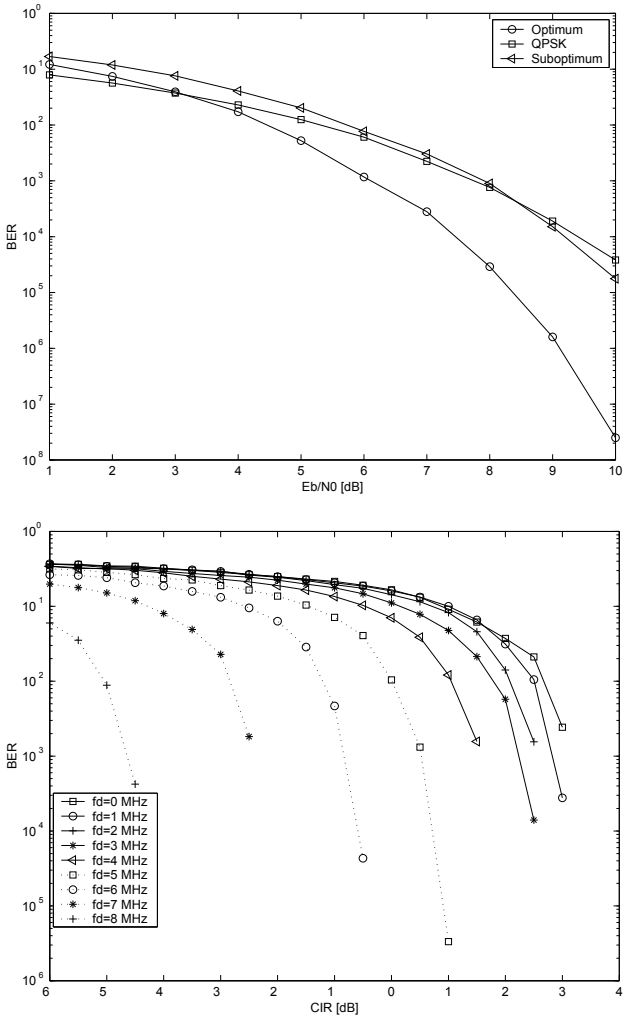


Figure 11: $\boxed{\frac{(a)}{(b)}}$ 802.11b CCK performance. (a) AWGN channel. (b) Bluetooth interference.

MHz bandwidth, thereby having less of a coding gain. Generally, the receivers used for both 1 Mbit/sec and 11 Mbits/sec are fairly simple, and improved performance can most likely be obtained using more complicated signal processing. This fact is especially true for the 11 Mbits/sec CCK system.

6 Conclusions and Present Work

In this work, we have investigated the performance of Bluetooth and 802.11b WLANs in interference-limited environments. We have established preliminary performance results for both the AWGN and fading channel models. The simulations strongly suggest that the interference may severely damage the operation of both systems in some applications. The AWGN results for the effect of interference on Bluetooth are not very far from the fading results, so this model may be adequate for studying the effect of interference in LOS conditions. Moreover, the AWGN channel can be used to evaluate mechanisms designed to improve coexistence.

While most of the study used the simple LDI receiver for Bluetooth, the simulation results suggest that substantially better performance can be achieved using a coherent Viterbi receiver for interference-limited channels. Presently, we are developing a more sophisticated Viterbi receiver, including channel estimation based on the Bluetooth access code.

For the 802.11b DSSS receiver, the results in a fading channel are more degraded than in an AWGN channel, compared to Bluetooth. Therefore, coexistence studies need to chose a more realistic channel model, instead of assuming an AWGN one. Furthermore, multipath fading is even more of a concern for the 11 Mbits/s CCK system, given its relatively short symbol time. Thus, a RAKE-based CCK receiver, which exploits the frequency-selective properties of the channel, should probably be used.

This paper considers the physical layer of the Bluetooth and 802.11b systems, including the design of the radio receivers. However, it does not consider the medium access control (MAC) layer. In Bluetooth, this layer contains the forward error detection and correction, the automatic repeat request (ARQ) protocols, and the frequency hopping. For 802.11b, the MAC is more complex, containing carrier sense, multiple access with collision avoidance, as well as a number of other features. Our current goal is to determine the interference-limited performance of the combined physical and MAC layers for each system with realistic traffic and topologies. Metrics of interest include packet loss probabilities, number of residual errors in a packet, delay, and throughput.

7 Acknowledgment

The authors wish to thank Dr. L. E. Miller for his helpful comments during the course of this study.

References

[1] J. C. Haartsen and S. Mattisson, "Bluetooth - A new low-power radio interface providing short-range connectivity," *Proc. of the IEEE*, vol. 88, no. 10, pp. 1651-1661, Oct. 2000.

[2] Bluetooth Special Interest Group, *Specifications of the Bluetooth System, vol. 1, v.1.0B*, Dec. 1999. Available : http:/www.bluetooth.com.

[3] IEEE Std. 802-11, *IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, 2001 Edition.

[4] R. Steele (Ed.), *Mobile Radio Communications*, John Wiley & Sons Inc., 1996.

[5] G. J. M. Janssen, P. A. Stigter and R. Prasad, "Wideband indoor channel measurements and BER analysis of frequency selective multipath channels at 2.4, 4.75, and 11.5 GHz," *IEEE Trans. Comm.*, pp. 1272-1288, Oct. 1996.

[6] H. J. Zepernick and T. A. Wysocki, "Multipath channel parameters for the indoor radio at 2.4 GHz ISM band," *49th IEEE Veh. Tech. Conf.*, vol. 1, pp. 190-193, 1999.

[7] S. C. Kim, H. L. Bertoni and M. Stern, "Pulse propagation characteristics at 2.4 GHz inside buildings," *IEEE Trans. Veh. Tech.*, vol. 45, pp. 579-592, Aug. 1996.

[8] M. K. Simon and C. C. Wang, "Differential detection of Gaussian MSK in a mobile radio environment, " *IEEE Trans. Veh. Tech.*, pp. 307-320, Nov. 1984.

[9] A. L. Oppenheim and R. W. Schafer, *Discrete-Time Signal processing*, Prentice Hall, 1989.

[10] J. G. Proakis, *Digital Communications*, McGraw-Hill, 1995.

[11] J. S. Lee and L. E. Miller, *CDMA Engineering Handbook*, Artech House, 1998.

[12] M. J. E. Golay,"Complementary series," *IRE Trans. Information Theory*, vol. IT-7, pp. 82-87, Apr. 1961.

[13] S. Halford, K. Halford and M. Webster, "Complementary code keying for RAKE-based indoor wireless communications," *IEEE Int. Conf. on Circuits and Systems*, pp. 427-430, May 1999.

[14] R. D. J. Van Nee, "OFDM codes for peak-to-average power reduction and error correction," *IEEE Global Telecommun. Conf.*, London vol. 1, pp. 740-744, Nov. 1996.

[15] A. Kamerman, "Coexistence between Bluetooth and IEEE 802.11 CCK solutions to avoid mutual interference interference," *Lucent Technologies Bell Laboratories technical report*, Jan. 1999

[16] M. K. Simon and C. C. Wang, "Differential versus limiter discriminator detection of narrow-band FM," *IEEE Trans. Comm.*, pp. 1227-1234, Nov. 1983.

[17] D. L. Schilling, L. B. Milstein, R. L. Pickholtz and R. W. Brown, "Optimization of the processing gain of an M-ary direct sequence spread spectrum communication system," *IEEE Trans. Comm.*, pp. 1389-1398, Aug. 1980.

[18] L. B. Milstein, S. Davidovici, and D. L. Schilling, "The effect of multiple-tone interfering signals on a direct sequence spread spectrum communication system," *IEEE Trans. Comm.*, vol. 30, pp. 436-446, Mar. 1982.

Performance of the Bluetooth System in Fading Dispersive Channels and Interference

A. Soltanian and R. E. Van Dyck
National Institute of Standards and Technology
Gaithersburg, Maryland 20899

Abstract— **A noncoherent limiter-discriminator receiver is often considered for the Bluetooth system because of its simplicity and low cost. While its performance is more than adequate for some channels, the results are signi cantly degraded in either an interference-limited environment or a frequency selective channel. In this paper, we compare the performance of the traditional limiter-discriminator with integrate and dump lter to a more sophisticated Viterbi receiver. We nd that the Bluetooth access code is suf cient to be used for channel estimation in the Viterbi receiver. A comparison is carried out in a Rayleigh fading channel and in the presence of interference either from another Bluetooth piconet or an IEEE 802.11b wireless local area network. Performance metrics include bit error rate and packet loss rate.**

I. Introduction

Bluetooth (BT) works in the 2.4 GHz unlicensed ISM band, which is also shared by other communication systems including 802.11 wireless local area networks (WLANs). The primary range of operation is 10 meters, but it can be extended up to 100 meters. In typical indoor applications where the channel exhibits low delay spread and there is a strong signal path between the transmitter and the receiver, the noncoherent limiter-discriminator with integrate and dump lter (LDI) receiver achieves reasonable performance [1]. However, it would be useful to make the radio system more robust so as to maximize the quality of service in outdoor and large indoor applications.

Some experiments have been conducted [2], [3], [4] to evaluate the power delay pro le of indoor channels at 2.4 GHz. The channel is roughly categorized into two major classes: (1) channels with a line-of-sight (LOS) path and (2) channels with an obscured path. For an LOS path, Kim *et al.* [2] nd that it can be reasonably approximated by a Rician distribution with $K = 5$, where K is the ratio of the power of the dominant path to the power of the scattered paths. For a path with obstructions, the probability density function (pdf) of the amplitude of the fading signal is Rician with $K = 2$, which is close to the Rayleigh distribution. The root-mean-square (rms) average of the delay spread varies between 75 nsec to 90 nsec. Zhang and Hwang [4] report an rms delay spread as large as 217 nsec. Wilkinson [5] studied the channel for the DECT system and considered a worst case rms delay of 200 and 300 nsec for indoor and outdoor channels, respectively. Also in this report, a Rayleigh fading distribution was considered.

Another challenging issue for the Bluetooth system is the coexistence with other Bluetooth piconets and/or with IEEE 802.11 WLANs. The interference emitted by these radios may severely degrade the operation of a Bluetooth radio. The Viterbi receiver may also be a promising substitute for the LDI receiver in this case.

This paper's main contribution is to evaluate the Bluetooth performance in hostile environments. Two scenarios are considered: (1) a multipath Rayleigh fading channel, and (2) an interference-limited environment. We show the bit error rate performance in these scenarios as well as system layer performance for Bluetooth voice packets.

II. Bluetooth

Bluetooth operates at a channel bit rate of 1 Mbit/sec [6]. The modulation is Gaussian frequency shift keying (GFSK) with a nominal modulation index of $h_f = 0.33$ and a normalized bandwidth of $B_bT = 0.5$, where B_b is the 3 dB Bandwidth of the transmitter's Gaussian low pass lter , and T is the bit period. The Bluetooth radio employs a frequency hopping scheme in order to mitigate the effect of interference and fading. There are a total of 79 hopping channels, each separated by 1 MHz, and the hopping frequency is changed on a packet by packet basis.

A. The GFSK Signal

The GFSK signal can be represented by [7]

$$s(t, \mathbf{a}) = A\cos(2\pi f_c t + \phi(t, \mathbf{a})), \tag{1}$$

where $A = \sqrt{\frac{2E_b}{T}}$, E_b is the energy per data bit, f_c is the carrier frequency, and $\mathbf{a}$ is the random input stream, comprised of the data bits α_i; $\phi(t, \mathbf{a})$ is the output phase deviation, given by [7]

$$\phi(t, \mathbf{a}) = 2\pi h_f \sum_{i=n-L+1}^{n} \alpha_i q(t - iT) + \pi h_f \sum_{i=-\infty}^{n-L} \alpha_i. \tag{2}$$

The second sum is the accumulated phase of all previous symbols, and it is called the phase state. $q(t) = \int_{-\infty}^{t} g(\tau)d\tau$, where $g(t)$ is the impulse response of a Gaussian lter , and L is the length of $g(t)$ in bit periods. For Bluetooth with $B_bT = 0.5$, we have $L = 2$.

B. LDI Receiver

This receiver consists of a pre-detection bandpass lter , a limiter-discriminator, and an integrate and dump lter as shown in Fig. 1. Details on the design of the receiver, including parameter choices, are given in [1].

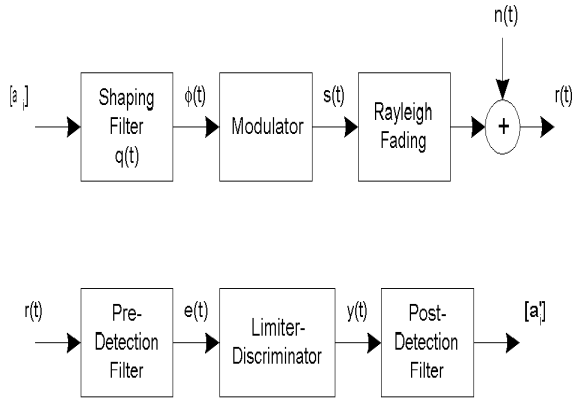


Fig. 1. Simulation model for the LDI receiver.

The bandpass lter has a Gaussian shape with impulse response

$$h_r(t) = \sqrt{\frac{2\pi}{ln2}} B_r \, e^{-(\frac{2\pi^2}{ln\,2})(B_r t)^2}. \qquad (3)$$

In an AWGN channel, the optimum value for $B_{IF} = 2B_r$ is chosen as 1.1 MHz [8],where B_r is the 3 dB bandwidth. The integrate and dump lter has a rectangular impulse response with a length of T. The appropriate sampling time is chosen at the maximum eye opening.

C. Viterbi Receiver With Equalizer

The Viterbi receiver takes advantage of the phase trellis created by the transmitter. For GFSK with modulation index $h_f = \frac{2k}{p}$, $p2^{L-1}$ states are required for the Viterbi receiver [7]. Given $h_f = 1/3$ and $L = 2$, the total number of phase states is $p = 6$, which includes $\{0, \frac{\pi}{3}, \frac{2\pi}{3}, \pi, \frac{4\pi}{3}, \frac{5\pi}{3}\}$. Consequently, the total number of states for the Bluetooth Viterbi receiver is $6 \times 2 = 12$. This receiver may be too complex for low cost implementations since it requires a lot of signal processing hardware.

One way to simplify the receiver is to remove the effect of the additional phase states in the decoding trellis. This action can be done by not only passing the cumulative metrics from a node to all its successor nodes, but also by passing the information about the phase state. In this way, after selecting the metric with minimum value, the phase state of that metric is also recorded at the new trellis node. This architecture change requires adding a little complexity to branch metric calculations, but it reduces the total number of trellis states from 12 to 2. We do not add any additional states to account for channel multipath delay. However, if more signal processing is permitted in the receiver design, the memory of the channel could also be considered as additional states.

Because no equalization is intended in Bluetooth, no training sequence is explicitly de ned in the standard. We found that the 64 bit access codes, which are sent in every packet, show good correlation properties, and so can be used for the estimation of the channel. This estimation is then used to compensate for the effect of fading and phase rotation in the received signal. Also, the correlation function can be used for the purpose of synchronization. In order to have a fair comparison with the LDI receiver, the Viterbi receiver front end contains the same Gaussian lter to reject out of band interference and noise. Results for this receiver appear in Section IV.

III. Channel and Interference

Our channel model is a simple Rayleigh fading two ray model, with variable delay between the two equal average power paths. If the time delay between the paths is equal to τ_1, the rms of the delay spread is, $\sigma = 0.5\tau_1$. This model is a good approximation for indoor channels, especially for low rms delay spreads $\sigma \leq 100$ nsec, but the results for higher delay spreads $\sigma \geq 200$ nsec are optimistic in comparison to more accurate models [5]. The fading is assumed to be static for the duration of the packet length, and the channel coef cients are sampled at the packet rate. This is a weak assumption, since the coherence bandwidth of the indoor channels is usually greater than the frequency separation of the hops [2], [9], and the fading statistics may not vary for several consecutive packets.

For the second scenario, we consider the performance of Bluetooth in the presence of interference. The channel is AWGN in this case, and the interference may be another Bluetooth piconet or an 802.11b system. The 802.11b WLAN can use either direct sequence spread spectrum (DSSS) at 1 or 2 Mbits/sec, or it can use complementary code keying (CCK) [10] at 5.5 or 11 Mbits/sec. Here, we consider 1 Mbit/sec DSSS. At this bit rate, data bits are spread by a Barker code with 11 chips per bit, which leads to a rate of 11 Mchips/sec. The modulation is differential BPSK (DBPSK), which facilitates noncoherent detection. A pulse shaping lter may be employed to reduce the out of band emissions, thereby giving an interference bandwidth of 22 MHz.

Either a Bluetooth or an 802.11b type interference signal can be represented as

$$S_I(t, \mathbf{b}) = B\cos(2\pi(f_c + f_d)t + \phi_2(t, \mathbf{b})), \qquad (4)$$

where **b** is the random input data that is independent of **a**, and ϕ_2 depends on the type of the interferer. f_d is the frequency difference between the desired signal and the interference. We assume that the interference signal is always on and exists for the entire length of the Bluetooth packet. Also, for a pure physical layer simulation, there is no error correction and retransmission in the channel. The Bluetooth radio channels are 1 MHz apart, so f_d can take values of $0, 1, 2 \cdots$ MHz. The bandwidth of the 802.11b system is 22 MHz, so we carried out simulations for $f_d \leq 11$ MHz. There are $Ns = 44$ samples/bit, which

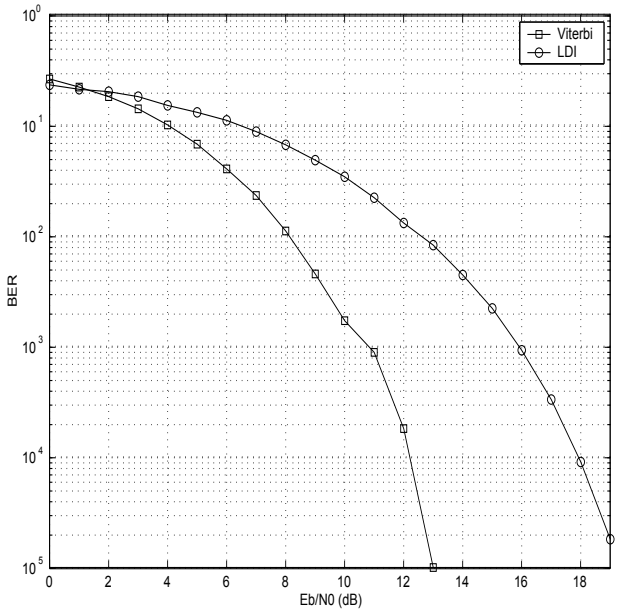


Fig. 2. Performance in the AWGN channel.

equals 4 samples/chip for the 802.11b system. This sampling rate is appropriate for f_d up to 22 MHz. A uniform random delay $t_d \in [0\ T)$ and a random phase $\phi_d \in [0\ 2\pi)$ are applied to the interferer signal for each packet.

IV. Performance Results

A. Physical Layer Performance

As a baseline for the performance comparisons of the two receivers, we rst consider the AWGN channel. Fig. 2 shows that the Viterbi receiver has a gain of 4 dB over the LDI receiver at a BER of 10^{-2}. The gain increases to about 5 dB at 10^{-3} and nearly 6 dB at 10^{-5}. Because of the short ranges involved, even for a transmit power of 1 mW, the received E_b/N_o is typically very high. Consequently, if one considers only this channel, there is no need for the more complex Viterbi receiver.

Simulation results for the LDI receiver in the two ray channel are presented in Fig. 3(a). For very low delay spreads where the channel exhibits at fading, an average E_b/N_0 level of 30 dB is required to achieve a BER close to 10^{-3}. This performance is not maintained as σ gets higher, and for $\sigma \geq 100$ nsec, even for high values of E_b/N_0, the performance is poor. The Viterbi receiver performance in Fig. 3(b) indicates that this receiver can tolerate more delay spread, and it achieves $BER = 10^{-2}$ for $\sigma \simeq 300$ nsec. Also, this receiver is insensitive to the sampling time of the signal.

BER measurements for an interference-limited environment are presented in Figs. 4 and 5; in all cases, the carrier-to-noise ratio, $CNR = 30$ dB.

In these gures, f_d is the absolute frequency offset between the Bluetooth signal and the interference. The carrier-to-interference ratio (CIR) is de ned as the ratio of the received signal power to the received interference power, and it is mea-

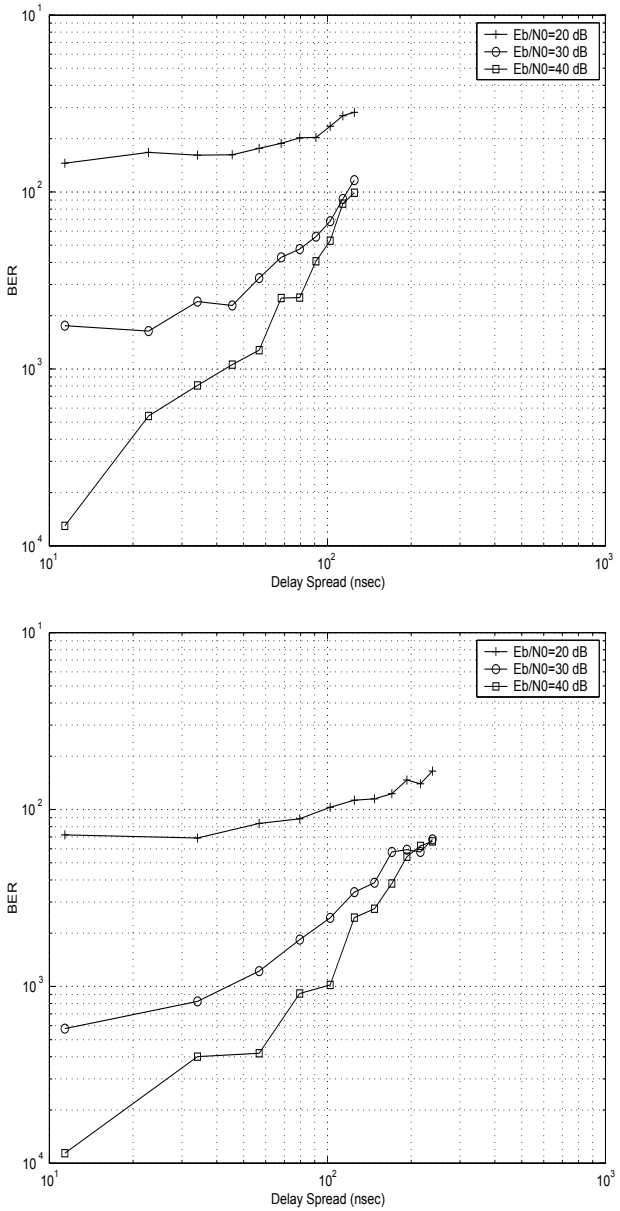


Fig. 3. (a) (b) Performance as a function of channel delay spread. (a) LDI receiver. (b) Viterbi receiver. Rayleigh two path channel.

sured at the input to the bandpass lter . Fig. 4 contains the results for both Viterbi and LDI receivers experiencing Bluetooth interference. For the Viterbi receiver, there is a 2 dB improvement for co-channel interference, and about 3 dB improvement for the adjacent channel. The gure also shows that the Viterbi receiver produces more errors than the LDI receiver in the presence of a strong interferer (low CIR). The main reason is that the interference reduces the effectiveness of the channel estimator used in the Viterbi receiver. However as the CIR increases, the channel estimator performs better and the overall BER improves. Other narrowband interference signals with $f_d \geq 2$ MHz are strongly attenuated by the bandpass lter , and

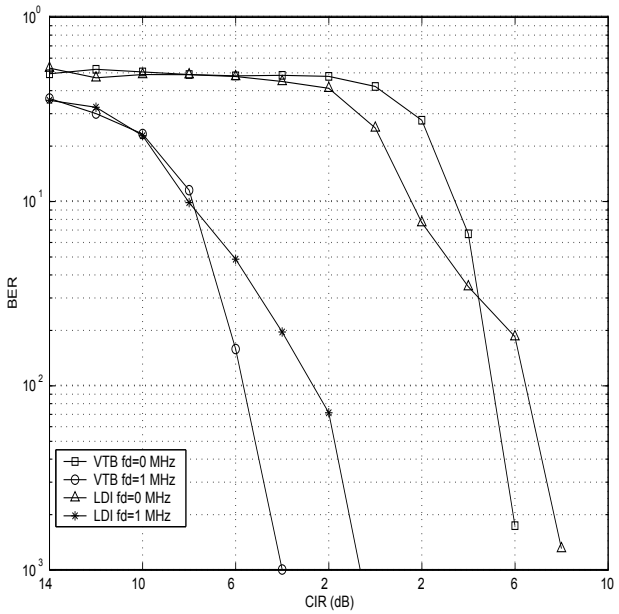


Fig. 4. Performance with Bluetooth interference.

they do not produce errors for this range of CIR.

For the 802.11b interference, Figs. 5(a) and (b) show that for frequency offsets up to 10 MHz, the system is still interference-limited. This result stems from the fact that the two-sided bandwidth of the 802.11b WLAN is 22 MHz, which is much wider than that of Bluetooth.

The LDI receiver needs at least $CIR = 4$ dB in order to get $BER \leq 10^{-2}$ for all frequencies. The degradation for $f_d \leq 4$ MHz is the same, since the 802.11b spectrum is at at these offsets. In Fig. 5(b), we observe a dramatic enhancement in performance for the Viterbi receiver over the LDI receiver. The minimum required CIR is about -4 dB in this case. Since the 802.11b interferer is more like uncorrelated noise at the input of this receiver, this level for CIR can also be concluded by looking at the performance of the Viterbi receiver in the AWGN channel (Fig. 2). This receiver requires $E_b/N_0 = 8$ dB for $BER = 10^{-2}$. The bandpass lter has about 12 dB out-of-band rejection. So, the maximum tolerable CIR at the input of the receiver is about -4 dB.

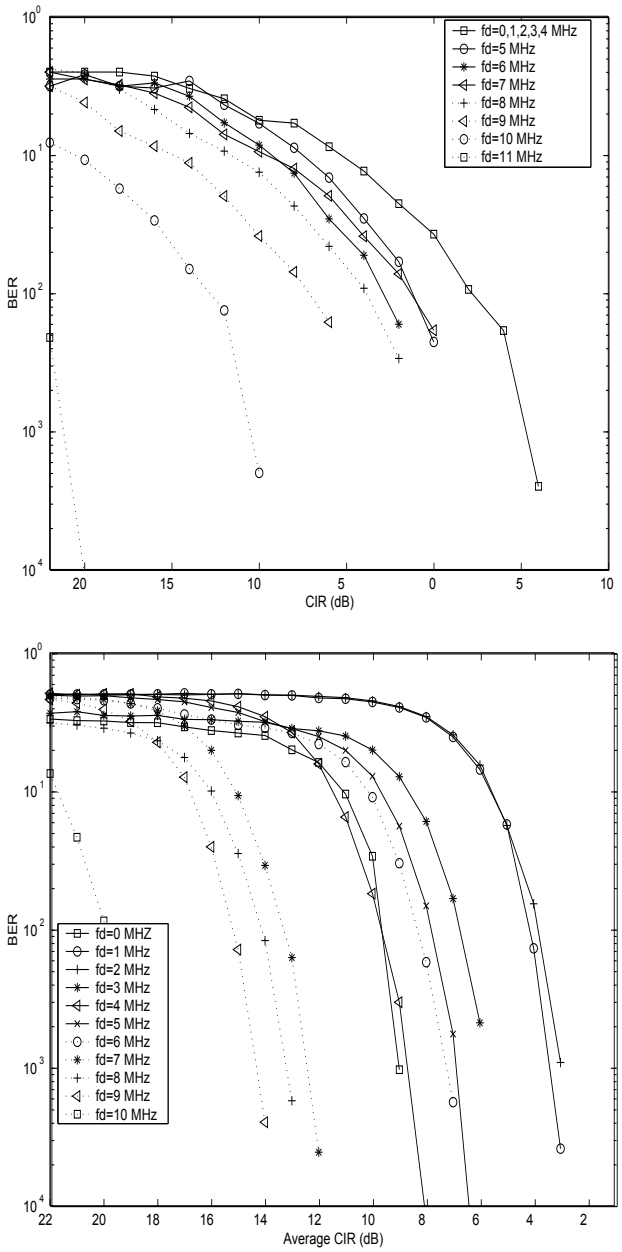


Fig. 5. $\boxed{\frac{(a)}{(b)}}$ Performance with 802.11b interference. (a) LDI receiver. (b) Viterbi receiver.

B. System Layer Performance

While the results of the previous section strongly suggest that the Viterbi receiver provides substantially better physical layer performance, the main question is how does this advantage translate into better system level performance. Four factors affect this mapping: (1) the frequency hopping pattern of the BT system, (2) the error detection and correction in the BT medium access control layer, (3) the BT traf c pattern, and (4) the traf c pattern of the interferer. These issues are discussed in much greater detail in [11], where performance results are provided for a number of scenarios, all using the LDI receiver.

The frequency hopping implies that the probability a BT packet falls within the interference bandwidth is approximately 22/79. Even then, the BER will depend on the frequency offset between the two received signals and whether the interferer is actually transmitting.

We consider a two-way communication between a Bluetooth master and slave, where each is sending 64 Kbits/sec of HV1 voice packets. These packets contain the BT access code, the packet header, and the payload. The access code words have large Hamming distances between each pair, while both the header and payload are protected by 1/3 rate repetition codes. The overall packet length is 366 bits. An uncorrected error in

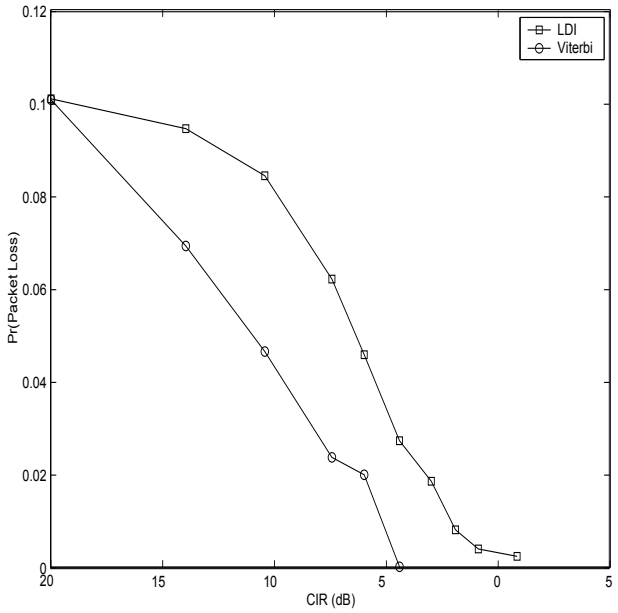


Fig. 6. Bluetooth voice packer loss with 802.11b interference.

either the access code or the header leads to the packet being dropped.

Fig. 6 shows the probability of packet loss versus CIR for both the LDI and the Viterbi receivers. For the LDI receiver, a $CIR = 0$ dB is necessary to get low packet loss. However, this value decreases to $CIR = -5$ dB for the Viterbi receiver. In both cases, we use exponentially distributed packet inter-arrival times for the WLAN, with an offered load of 50%. The packet length for the WLAN interference is x ed and equal to $8,000$ bits.

V. Conclusions

We have investigated the performance of the Bluetooth radio by employing two different types of receivers: (1) a low cost LDI and (2) a more sophisticated Viterbi receiver. From the physical layer simulation results, we conclude that the Viterbi receiver is superior in both the multipath Rayleigh fading channel and in interference. This superiority is particularly considerable in the latter case, especially when the interference comes from an 802.11b WLAN. We have also shown system level performance for Bluetooth voice packets in an interference-limited environment. Even though the frequency hopping and error correction help both receivers, thereby reducing the differences in performance due to the physical layer, the Viterbi receiver still provides a substantial improvement.

One issue of present concern is the large allowed deviation in a Bluetooth transmitter's modulation index. While the nominal value is 0.33, the range is 0.28 to 0.35. For a Viterbi receiver designed to use this nominal value, we nd that it is robust to variations of about ± 0.01. Although there are methods that allow one to estimate the modulation index [12], the receiver architecture, including the number of states, would have to be changed. Therefore, we suggest that the deviation allowed in the standard be reduced.

Acknowledgments

The authors would like to thank Nada Golmie and Oliver Rebala for many useful discussions and for providing the system layer simulation results.

References

[1] A. Soltanian and R. E. Van Dyck, "Physical layer performance for coexistence of Bluetooth and IEEE 802.11b," *Proc. Virginia Tech. Symposium on Wireless Personal Communications*, Blacksburg, VA, June, 2001.

[2] S. C. Kim, H. L. Bertoni, and M. Stern, "Pulse propagation characteristics at 2.4 GHz inside buildings," *IEEE Trans. Veh. Tech.*, vol. 45, pp. 579-592, Aug. 1996.

[3] G. J. M. Janssen, P. A. Stigter, and R. Prasad, "Wideband indoor channel measurements and BER analysis of frequency selective multipath channels at 2.4, 4.75, and 11.5 GHz," *IEEE Trans. Comm.*, vol. 44, pp. 1272-1288, Oct. 1996.

[4] Y. P. Zhang and Y. Hwang, "Time delay characteristics of 2.4 GHz band radio propagation channels in room environments," *IEEE Int. Symp. Personal, Indoor and Mobile Radio Communications,* vol. 1, pp. 28-32, 1994.

[5] T. A. Wilkinson, "Channel modelling and link simulation studies for the DECT test bed program," *6th Int. Conf. Mobile Radio and Personal Comm.*, pp. 293-299, 1991.

[6] Bluetooth Special Interest Group, *Speci cations of the Bluetooth System, vol. 1, v.1.0B 'Core'*, Dec. 1999. Available: http:/www.bluetooth.com.

[7] R. Steele (Ed.), *Mobile Radio Communications*, John Wiley & Sons Inc., 1996.

[8] M. K. Simon and C. C. Wang, "Differential detection of Gaussian MSK in a mobile radio environment," *IEEE Trans. Veh. Tech.*, pp. 307-320, Nov. 1984.

[9] G. F. Pedersen and P. Eggers "Initial investigations of the Bluetooth link," *IEEE Veh. Tech. Conf.*, vol. 1, pp. 64-69, Fall 2000.

[10] IEEE Std. 802-11, *IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Speci cation* , 2001 Edition.

[11] N. Golmie, R. E. Van Dyck, and A. Soltanian, "Interference of Bluetooth and IEEE 802.11: simulation modeling and performance evaluation," *Proc. ACM Int. Workshop on Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, Rome, Italy, July 2001.

[12] F. J. Casajús-Quirós and J. M. Páez-Borrallo, "Improving DECT performance with band-pass equalization," *Proc. of VTC'97*, pp. 1084-1088, May 1997.

Interference of Bluetooth and IEEE 802.11: Simulation Modeling and Performance Evaluation

N. Golmie, R. E. Van Dyck, and A. Soltanian
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
Email: nada.golmie@nist.gov, robert.vandyck@nist.gov

ABSTRACT

The emergence of several radio technologies such as Bluetooth, and IEEE 802.11 operating in the 2.4 GHz unlicensed ISM frequency band may lead to signal interference and result in significant performance degradation when devices are co-located in the same environment. The main goal of this paper is to present a simulation environment for modeling interference based on detailed MAC and PHY models. This framework is then used to evaluate the impact of interference on the performance of Bluetooth and IEEE 802.11. We use several simulation scenarios and measure performance in terms of packet loss, residual number of errors, and access delay.

Keywords

WPANs, Bluetooth, IEEE 802.11, Interference.

1. INTRODUCTION

The proliferation of mobile computing devices including laptops, personal digital assistants (PDAs), and wearable computers has created a demand for wireless personal area networks (WPANs). WPANs allow closely located devices to share information and resources. A key challenge in the design of WPANs is, perhaps, the adaptivity to a hostile radio environment that includes noise, time-varying channels, and abundant electromagnetic interference. Today, most radio technologies considered by WPANs (Bluetooth Special Interest Group [1], and IEEE 802.15) employ the 2.4 GHz ISM frequency band. In addition, both WPANs and Wireless Local Area Networks (WLANs) devices implementing the IEEE 802.11 standard specifications [2] will be sharing the same frequency band. It is anticipated that some interference will result from all these technologies operating in the same environment. WLAN devices operating in proximity to WPAN devices may significantly impact the performance of WPAN and vice versa.

The main goal of this paper is to present a tool for modeling the interference of Bluetooth and IEEE 802.11. In addition, we discuss our findings on the performance of these systems when operating in close proximity to each other. Our results are based on detailed models for the MAC, PHY, and wireless channel.

Previous performance results on Bluetooth and IEEE 802.11 interference include experimental measurements obtained by Kamerman [3]. Furthermore, the probability of an 802.11 packet error in the presence of a Bluetooth piconet has been derived by Zyren [4] and extended by Shellhammer [5]. In addition, Golmie and Mouveaux [6] study the effect of 802.11 on Bluetooth, using a probability analysis approach and validate the analysis with simulation results. They show that significant packet loss can occur and that access delays for data traffic double. Similar results have been obtained by Lansford *et. al.* [7] who use simulation and experimental measurements to quantify the interference resulting from Bluetooth and IEEE 802.11.

This paper is organized as follows. In section 2, we describe in great detail our modeling approach for the MAC, PHY and wireless channel. In section 3, we discuss the accuracy of our model implementation. In section 4, we evaluate the impact of interference on both Bluetooth and WLAN performance and present simulation results. Concluding remarks are offered in section 5.

2. INTEGRATED SIMULATION MODEL

In this section, we describe the methodology and tools used to conduct the performance evaluation. The simulation environment consists of detailed models for the RF channel, the PHY, and MAC layers developed in C and OPNET (for the MAC layer). These detailed simulation models will constitute an evaluation framework that is critical to studying the various intricate effects between the MAC and PHY layers. Although interference is typically associated with the RF channel modeling and measured at the PHY layer, it can significantly impact the performance of higher layer applications including the MAC layer. Similarly, changes in the behavior of the MAC layer protocol and the associated data traffic distribution could play an important factor in the interference scenario and affect the overall system performance.

2.1 Channel Model

The channel model consists of a geometry-based propagation model for the signals, as well as a noise model. For the indoor channel, we apply a propagation model consisting of two parts: 1.) line-of-sight propagation (free-space) for the first 8 meters, and 2.) a propagation exponent of 3.3 for distances over 8 meters. Consequently, the path loss in dB is given by

$$L_p = \begin{cases} 32.45 + 20\log(f \cdot d) & \text{if } d < 8 \text{ m} \\ 58.3 \ + 33\log(d/8) & \text{otherwise,} \end{cases} \quad (1)$$

where f is the frequency in GHz, and d is the distance in meters. This model is similar to the one used by Kamerman [3]. Assuming unit gain for the transmitter and receiver antennas and ignoring additional losses, the received power in dBmW is

$$P_R = P_T - L_p, \quad (2)$$

where P_T is the transmitted power also in dBmW. Eq. (2) is used for calculating the power received at a given point due to either a Bluetooth or an 802.11 transmitter, since this equation does not depend on the modulation method.

Additive white Gaussian noise (AWGN) is used to model the noise at the receivers. At 1 Mb/s any fading would be frequency non-selective; in this case, the SNR is sufficiently high so that the system is interference limited. This flat fading assumption is less true for the 11 Mb/s 802.11 data rate, although the system is still interference limited.

The transmitters, channel, and receivers are implemented at complex baseband. For a given transmitter, inphase and quadrature samples are generated at a sampling rate of 44×10^6 per second. This rate provides four samples/symbol for the 11 Mb/s 802.11 mode, enough to implement a good receiver. It is also high enough to allow digital modulation of the Bluetooth signal to account for its frequency hopping. Specifically, since the Bluetooth signal is approximately 1 MHz wide, it can be modulated up to almost 22 MHz, which is more than enough to cover the 11 MHz bandwidth (one-sided) of the 802.11 signal. The received complex samples from both the desired transmitter and the interferer(s) are added together at the receiver.

To complete the channel model, the noise must be added to the received samples. Consider a fixed transmitter power and no interference. Then, Eqs (1) and (2) allow one to compute the received signal power for a given distance. The SNR is calculated in dB according to

$$SNR = P_R - S_R, \quad (3)$$

where S_R is the receiver's sensitivity in dBmW. In an actual receiver, the sensitivity is determined primarily by the amount of thermal noise in the electronics; within limits imposed by physics, a better design can lead to a higher sensitivity. For our modeling purposes, the situation is somewhat reversed. One assumes a specific (achievable) sensitivity and uses Eq. (3) to compute the SNR. This quantity is used to set the variance of the random number generator that provides the AWGN noise for each inphase and quadrature sample. Please note that the transmitter and interferer powers can be changed on a packet by packet basis.

A few comments should be made about the relationship among the received signal power, the received interference power, the noise power, and the resulting performance. Analogously to SNR, one can define the signal-to-interference ratio (SIR) in dB as

$$SIR = P_R - P_I, \quad (4)$$

where P_I is the interference power at the receiver. In the absence of interference, the bit error rate (BER) for either the Bluetooth or WLAN system is almost negligible for the transmitter powers and ranges under consideration.

2.2 PHY Model

The PHY layer includes detailed models of the signal processing in the Bluetooth and the 802.11 transmitters and receivers. As mentioned before, complex baseband implementations are used.

Bluetooth The GFSK modulation used in the Bluetooth system is a type of binary partial response continuous phase modulation. It is a slight generalization of the GMSK modulation [8] used in the GSM cellular system, which uses a modulation index of 0.5; instead, a modulation index of approximately 0.3 is used in Bluetooth. Because of the Gaussian-shaped filter in the transmitter, every data bit is transmitted over two symbol intervals, causing intersymbol interference but reducing the required bandwidth. The information carrying phase is denoted by $\theta(t, \tilde{\alpha})$, where t designates time, and $\tilde{\alpha}$ represents the data bit vector. The cosine and sine of $\theta(t, \tilde{\alpha})$, sampled 44 times per data bit (symbol), give the inphase and quadrature samples.

While there are a number of possible receiver designs, we chose to implement the noncoherent limiter-discriminator (LD) receiver [9] [10]. Its simplicity and relatively low cost should make it the most common type for many consumer applications. Details of the actual design are given in [11].

802.11b The 1 Mb/s [1] 802.11b system transmits data using differential binary phase shift keying. With DBPSK modulation, the information is conveyed by the phase difference between adjacent transmitted symbols. Thus, it is not necessary to have a coherent phase reference in the receiver. To provide some interference protection, the modulated signal is spread using a Barker sequence with code length equal to eleven [2]. That is, each bit duration is divided into eleven consecutive segments called chips. During each chip, the transmitted signal is multiplied by either ± 1, depending on the code [12]. Because the chip rate is 11×10^6 per second, the two-sided bandwidth of this signal is approximately 22 MHz. After spreading, the signal is fed into a pulse-shaping filter that provides further control on the spectral shape.

To achieve 11 Mb/s in an environment with fading and interference, a more sophisticated modulation scheme is required if the bandwidth is to be kept constant. This is done using a type of coded modulation. The basic idea is that uncoded quadrature phase shift keying provides two bits per symbol. If the symbol rate is kept constant at 11×10^6 per second then a maximum data rate of 22 Mb/s is possible. However, half of these bits are used to provide a coding gain using

[1] The symbol rate is the same as the bit rate, since this is a binary modulation scheme.

complementary code keying (CCK) [13].

2.3 MAC Model

We used *OPNET* to develop a simulation model for the Bluetooth and IEEE 802.11 protocols. For the IEEE 802.11 protocol, we used the model available in the OPNET library. For Bluetooth, we partially implemented the Baseband and L2CAP layers according to the specifications [1]. We assume that a connection is already established between the master and the slave and that the synchronization process is complete. The connection type is either SCO for voice or ACL for data traffic.

A MAC protocol generally consists of a collection of components, each performing a special function, such as the support of higher layer traffic, the synchronization process, the bandwidth allocation, and contention resolution mechanism. In this sequel, we highlight the features that are the most relevant to our work on interference, namely, we give a brief description of the frequency hopping, the interface to the physical layer, and the error detection and correction schemes.

Frequency Hopping Frequency usage constitutes another major component of the protocol model. Bluetooth uses a frequency hopping mechanism that sweeps 79 channels of the frequency band available at a maximum rate of 1600 hops/s depending on the packet size. Both master and slave devices are synchronized and follow the same random frequency hopping sequence. This frequency sequence is derived at the master and slave devices and depends on the master's clock and its Bluetooth address. The algorithm for generating the sequence works as follows. Given a window of 32 contiguous frequencies in the 2.4-2.479 GHz range, a sequence of 32 frequencies is chosen randomly. Once all 32 frequencies in that set have been visited once, a new window of 32 frequencies is selected. This new window includes 16 of the frequencies previously visited and 16 new frequencies. For the IEEE 802.11, we focus in this study on the IEEE 802.11 Direct Sequence mode which uses a fixed frequency that occupies 22 MHz of the frequency band. The center frequency is selected among 11 available channels.

Error Detection and Correction Error detection and correction is an essential component in the interference study. For IEEE 802.11, errors are detected by checking the Frame Check Sequence (FCS) that is appended to the packet payload. In case an error is found, the packet is dropped and is then later retransmitted. Otherwise, a positive ACK notifies the source of a correct reception. For Bluetooth, the device first applies the error correction algorithm corresponding to the packet encapsulation used. The encapsulation of voice packets such as $HV1$ and $DM5$ is shown in Figure 1. $HV1$ packets have a total size packet length of 366 bits including a header and an access code of 126 bits. $HV1$ packets use a payload of 80 information bits, a 1/3 FEC rate and are sent every $T_{SCO} = 2$ or 1250 μs. In case of an error occurrence in the payload, the packet is never dropped. A 1/3 FEC is applied to the packet header while a Hamming code ($d = 14$) is applied to the access code. Uncorrected errors in the header and access code lead to a packet drop. In addition, errors in the payload are corrected using a 1/3 FEC rate.

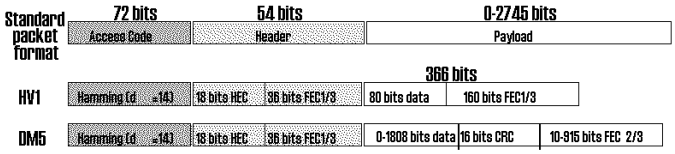


Figure 1: Bluetooth Packet Format

On the other hand, $DM5$ packets use a 2/3 rate FEC to correct payload errors as shown in Figure 1. Errors in the header or access code are corrected by a 1/3 FEC and a Hamming code, respectively. Uncorrected errors lead to dropping packets and the application of the ARQ and SEQN schemes.

Statistics Collection At the MAC layer, a set of performance metrics are defined to include access delay, probability of packet loss, and residual number of errors in the Bluetooth voice packets. The access delay measures the time it takes to transmit a packet from the time it is passed to the MAC layer until it is successfully received at the destination. The access delay for the Bluetooth LAN traffic is measured at the L2CAP layer in order to account for retransmission delays. Packet loss measures the number of packets discarded at the MAC layer due to errors in the bit stream. This measure is calculated after performing error correction. The residual number of errors in the Bluetooth voice packets measures the number of errors that remain in the packet payload after error correction is performed.

2.4 MAC Layer to PHY Layer Interface

The OPNET MAC models were interfaced to the physical layer models described in the previous section in order to simulate the overall system.

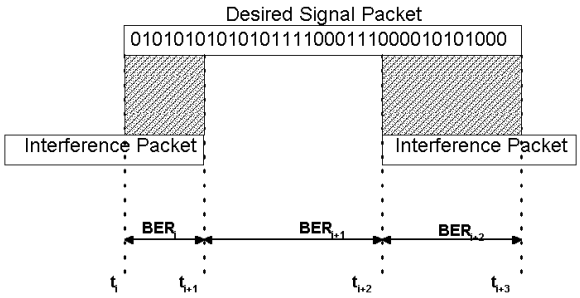


Figure 2: Packet Collision and Placement of Errors

This interface module is required to capture all changes in the channel state (mainly in the energy level). Consider the Bluetooth transmitter-channel-receiver chain of processes. For a given packet, the transmitter creates a set of signal samples that are corrupted by the channel and input to the receiver; interference may be present for all or only specific segments of the packet, as shown in Figure 2. A similar chain of processing occurs for an 802.11b packet. The interface module is designed to process a packet at a time.

At the end of each packet transmission, the MAC layer generates a data structure that contains all the information required to process the packet. This structure includes a list

of all the interfering packets with their respective duration, timing offset, frequency, and transmitted power. The topology of the scenario is also included. The data structure is then passed to the physical layer along with a stream of bits representing the packet being transmitted. The physical layer returns the bit stream after placing the errors resulting from the interference.

3. SIMULATION MODEL VALIDATION

In order to speed up the simulation process, we replace each transmitter-channel-receiver process with a table-based approach combined with a binary symmetric channel. BER tables for different values of SIR and for different frequency offsets were derived. For a segment of a packet where the interference is stationary, the SNR and SIR are computed using the transmitters' powers, the topology, and the path loss model. Thus, using the calculated SIR and the given frequency offset of the intended signal with respect to the interference signal, the average BER can be extracted by a simple table lookup operation. Errors are then generated for each bit of the packet segment using the binary symmetric channel with crossover probability equal to the average BER of the segment. The SNR in these tables is assumed to be very high (greater than 30 dB), which is the case for interference-limited environments. Still, the software can check this assumption by comparing the SIR to this value.

Using tabulated BER values, as opposed to running the detailed signal processing receiver and channel simulation models in real-time, gives a speed up factor of about 120. The main question is the accuracy of this approach; this topic is discussed below.

3.1 Results Accuracy

Since the implementation of the PHY layer required choosing a number of design parameters, the first step in the validation process is comparing the PHY results against theoretical results. Complete BER curves of the Bluetooth and 802.11b systems are given in [11]; for the AWGN and flat Rician channels without interference, all the results match very closely to analytical bounds and other simulation results. Also, the simulation results for both the MAC and PHY models were compared and validated against analytical results for packet loss given different traffic scenarios [6].

3.2 Table Implementation Accuracy

Figure 3 gives the BER in terms of the SNR for varying SIR and for co-channel interference. To create the table, the curves are sampled every 0.5 dB in both SNR and SIR. A couple of points need to be made: (1) For a fixed level of SIR, one notices that the change in BER for a 0.5 dB step in SNR is quite small, even at low SNR. For example, a change in BER from 0.25 to 0.2 is not particularly important, since it is still so high that a packet will most likely be lost. (2) For a fixed SNR, a 0.5 step in SIR also gives a small relative change in BER, especially for SIRs below 2 dB. As the SIR goes above 2 dB, the BER drops below 10^{-2}, and the system performance becomes increasingly good. For the overall system performance, it does not really matter if the BER is 10^{-5}, 10^{-6}, or smaller.

The table implementation does not impact the MAC performance results. A sanity check experiment was conducted

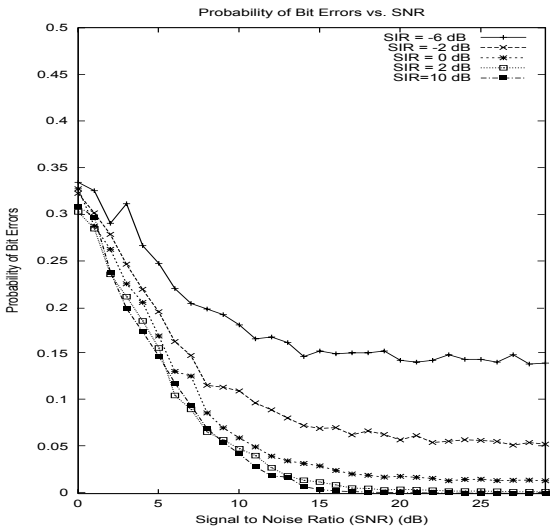


Figure 3: Impact of WLAN Interference on BT; Co-Channel Interference

to validate the simulation results obtained with the BER tables and compare them to the results obtained using the signal processing simulation model. Therefore, we run a set of two experiments, one with the BER tabulated values and one with the integrated DSP models, keeping all other simulation parameters the same. Using tabulated BER values instead of the simulation model for the DSP receiver does not affect the packet loss metric.

4. SIMULATION RESULTS

We present simulation results to evaluate the performance of Bluetooth in the presence of WLAN interference and vice versa. All simulations are run for 30 seconds of simulated time. The performance measurements are logged at the slave device for Bluetooth and at the Mobile device for the WLAN. The mean access delay result is normalized by the mean delay when no interference is present. We use the configuration and system parameters shown in Table 1.

For Bluetooth, we consider two types of application, namely voice and internet traffic. For voice, we assume a symmetric stream of 64 kbits/s each way using $HV1$ packet encapsulation. For modeling internet traffic, we consider a LAN access application. This is typically a connection between a PC and an Access Point or between two PCs, and it allows for exchanging TCP/IP or UDP-like traffic. Both slave and master devices generate IP packets according to the distribution presented in Table 2. The packet interarrival time is exponentially distributed with a mean equal to 29.16ms, which corresponds to a load of 30 % of the channel capacity (248 kbits/s for both directions). Packets are encapsulated with $DM5$ Baseband packets after the corresponding PPP, RFCOMM, and L2CAP packet overheads totaling 17 bytes are added.

For the WLAN, we use the IP traffic distribution presented in Table 2. We set the offered load to 30% of the channel capacity, which corresponds to mean packet interarrival times of 2.52 ms and 10.56 ms for the 11 Mbits/s and the 1

Table 1: Simulation Parameters

Simulation Parameters	**Values**
Propagation delay	5 μs/km
Length of simulation run	30 seconds
Bluetooth Parameters	**Values**
LAN Packet Interarrival Time	29.16 ms
ACL Baseband Packet Encapsulation	DM5
SCO Baseband Packet Encapsulation	HV1
Transmitted Power	1 mW
Slave Coordinates	(0,0)
Master Coordinates	(1,0)
WLAN Parameters	
Packet Interarrival Time for 1 Mbits/s	10.56 ms
Packet Interarrival Time for 11 Mbits/s	2.52 ms
Transmitted Power	25 mW
AP Coordinates	(0,15)
Mobile Coordinates	(0,d)
Packet Header	224 bits
Slot Time	$2 * 10^{-5}$ seconds
SIFS Time	$1 * 10^{-5}$ seconds
DIFS Time	$5 * 10^{-5}$ seconds
CW_{min}	31
CW_{max}	1023
Fragmentation Threshold	None
RTS Threshold	None
Short Retry Limit	4
Long Retry Limit	7

Table 2: IP Traffic: Message Size Distribution

Message Size (bytes)	64	128	256	512	1024	1518
Probability	0.6	0.06	0.04	0.02	0.25	0.03

Mbits/s systems, respectively.

We present the results from four different simulation experiments that show the impact of WLAN interference on Bluetooth devices and vice versa for different applications, namely voice and data traffic. Table 3 provides a summary of these four cases, while Figure 4 shows the experimental topology. Please note that the WLAN access point (AP) is fixed at (0,15), while the WLAN mobile is free to move along the vertical axis, *i.e.* its coordinates are (0,d). The Bluetooth devices are fixed at the given locations. In the first two experiments, the mobile is the generator of the 802.11 data, while the AP is the sink. In the last two experiments the traffic is generated at the AP.

Table 3: Summary of the Experiments

Experiment	Desired Signal	Interferer	WLAN AP	WLAN Mobile
1	BT Voice	802.11	Sink	Source
2	BT LAN	802.11	Sink	Source
3	802.11	BT Voice	Source	Sink
4	802.11	BT LAN	Source	Sink

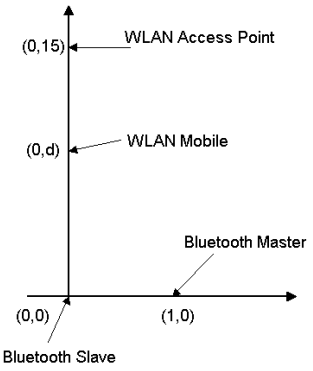


Figure 4: Experiment Topology

Experiment 1 - We study a voice application generating a symmetric stream of 64 kbits/s each way between the Bluetooth master and slave. The interference is from the mobile sending data packets to the AP and receiving acknowledgments (ACKs) from it. Since most of the WLAN traffic is originating close to the Bluetooth slave, the slave may suffer from serious interference. Figure 5(a) shows the probability of Bluetooth voice packet loss at the slave as a function of the distance to the mobile for interference from both 1 Mb/s and 11 Mb/s 802.11 WLANs.

Consider the 1 Mb/s case first. At one meter, approximately eight percent of the packets are dropped, due to an error in either the access code or the packet header. Even when the packet is accepted, it may still contain a significant number of residual payload errors as shown in Figure 5(b). These errors are measured *after* the FEC decoding is applied. While six errors may not seem to be many, in an eighty bit payload they will lead to poor voice quality. The packet loss is still significant even up to a distance of three meters.

The average length of a 1 Mb/s WLAN packet is 3,168 bits. Thus, its transmission time is on the order of five Bluetooth slots. HV1 packets are being transmitted in every Bluetooth slot, but on different frequencies. Since the direct sequence spreading requires a bandwidth of 22 MHz, there is a significant probability that a WLAN packet may cause interference to multiple Bluetooth packets. In other words, although Bluetooth is hopping to a new frequency for each slot, the 802.11 interference is present in roughly 22 of the 79 channels. Yet with an average interarrival time for the WLAN packets of 10.56 ms, many HV1 packets are successfully received between the transmissions of the WLAN packets.

For the 11 Mb/s case, the general trends are similar. However, the probability of packet loss is slightly lower. Because both the 1 and 11 Mb/s 802.11 modulations use the same bandwidth, the time overlap, not the frequency overlap, is the main factor affecting performance. At 11 Mb/s, it takes only 491 μs, on average, to transmit a packet [2]; therefore, the Bluetooth and WLAN packets are about the same length. Thus, a WLAN packet will usually only interfere with a single Bluetooth one.

[2]Including the packet header transmitted at 1 Mb/s.

Experiment 2 - We focus on a LAN access application. Bluetooth is being used to send data from the master to the slave, and the mobile is still the source of WLAN packets. Figure 6(a) shows the probability of Bluetooth LAN packet loss versus the distance to the mobile, again for both WLAN data rates. While up to almost fourteen percent of the packets may be lost, the use of ARQ still allows the system to be useful. Since a packet sent by the master is acknowledged (positively or negatively) in the next slot, the access delays remain quite small, as seen in Figure 6(b). Even at half a meter with the 11 Mb/s WLAN interference, the access delay is just doubled.

One observation is that for Bluetooth LAN packets, the effect of the different 802.11 data rates is reversed. The probability of packet loss is now higher when the 11 Mb/s system is the interferer. This result is also due to the traffic distributions. The Bluetooth LAN packets have a distribution with an average length that needs two DM5 packets, where each packet requires 2,871 μs for transmission. Now, the Bluetooth and 1 Mb/s WLAN packets are approximately the same length, so it is most likely that a WLAN packet corrupts no more than one Bluetooth packet. The 11 Mb/s WLAN packets are much shorter, and so a number of them can occur during the transmission of the Bluetooth packet. If the Bluetooth LAN packet is on the same frequency as any of these WLAN packets, it will probably be corrupted.

Experiment 3 - Next, we are interested in the effect of the Bluetooth voice packets on the 802.11 system. Let the AP be the source of WLAN data packets and the mobile be the receiver. Because the data packets are generally longer then the ACKs, this is a more critical scenario then when the mobile is the source. Figure 7(a) shows the probability of WLAN packet loss as a function of distance to the Bluetooth slave.

For a half meter distance, about sixty five percent of the 1 Mb/s packets are lost. This phenomenon occurs despite the frequency hopping of Bluetooth. The loss rate is so high due to the relatively long length of an 802.11 packet compared to a Bluetooth one. Since 802.11 does not have any error correction, all it takes is a single bit error to effectively erase the packet. When transmitting HV1 voice packets, the Bluetooth system sends many packets during the transmission time of an 802.11 packet. While there is approximately a 22/79 chance that a single packet is in the 802.11 band, this probability must be multiplied by the number of Bluetooth slots occurring doing the WLAN packet transmission. Also, note that the access delay is increased by almost three orders of magnitude due to the interference, as shown in Figure 7(b).

Still considering the 1 Mb/s mode, one sees that the performance significantly improves as the distance exceeds two meters. There appears to be almost a strong "threshold effect." The cause of this phenomenon is the direct sequence spreading, which is reasonably robust to a narrow-band interferer such as Bluetooth. Below two meters, the received interference power, based on the topology and transmitter powers, is so much that the 802.11 receiver makes many bit errors. Above this distance, the Barker code correlation effectively spreads the Bluetooth interference while de-

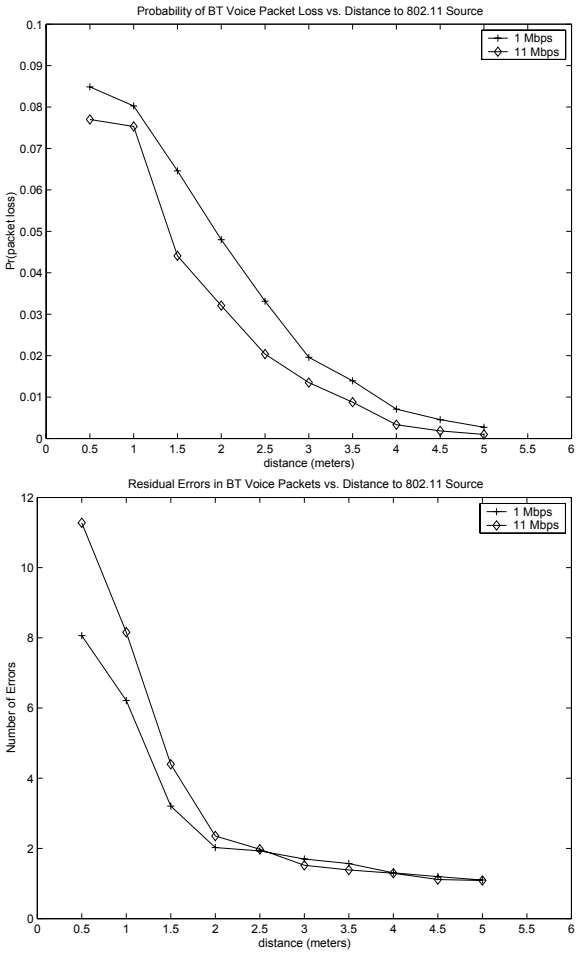


Figure 5: $\boxed{\frac{(a)}{(b)}}$ **Experiment 1. Bluetooth voice packets with 802.11 interference. (a) Probability of packet loss. (b) Residual errors.**

spreading the desired signal. Then, the performance of the 1 Mb/s system is better than the 11 Mb/s system.

The 11 Mb/s system has a 0.3 probability of packet loss at a range of half a meter. This probability drops almost linearly to a value near 0.1 for a range of 3.5 meters; the slope does not increase until after this distance. Thus, there is not as clear a threshold. Since the 11 Mb/s WLAN packets are more than six times shorter than the 1 Mb/s ones, there is a lower probability of overlap in time with the Bluetooth packet. This accounts for the lower packet loss probabilities at distances under two meters. However, the CCK modulation is not as robust at the direct sequence spreading. So, it is unable to provide as low a bit error rate as the DS modulation for distances in the approximate range of 2.5 to 4.5 meters.

Experiment 4 - Let the mobile be the receiver of the WLAN packets from the AP, and consider how the Bluetooth data packets degrade the WLAN performance. Figure 8(a) shows that for both data rates, the probability of an 802.11 packet being lost is much smaller for Bluetooth

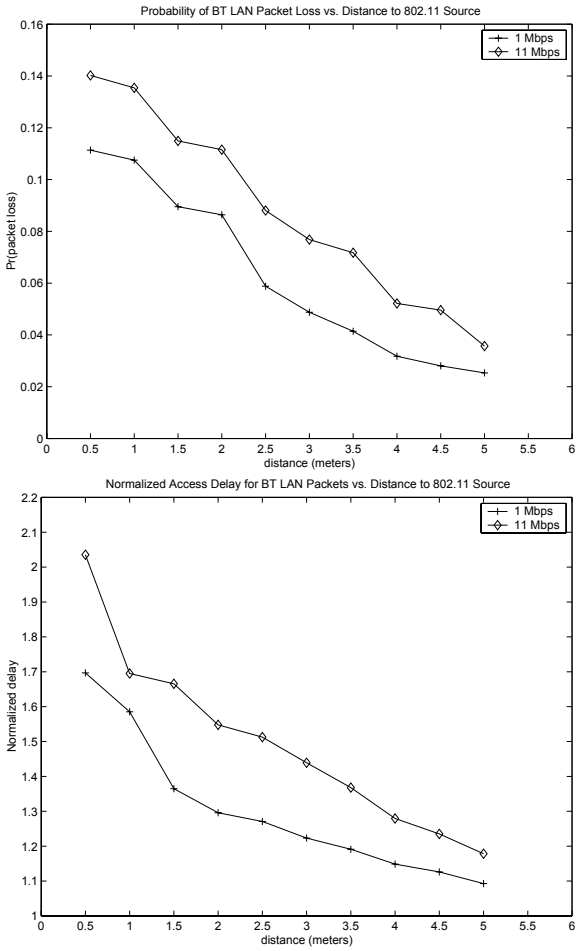


Figure 6: (a)/(b) Experiment 2. Bluetooth data packets with 802.11 interference. (a) Probability of packet loss. (b) Access delay.

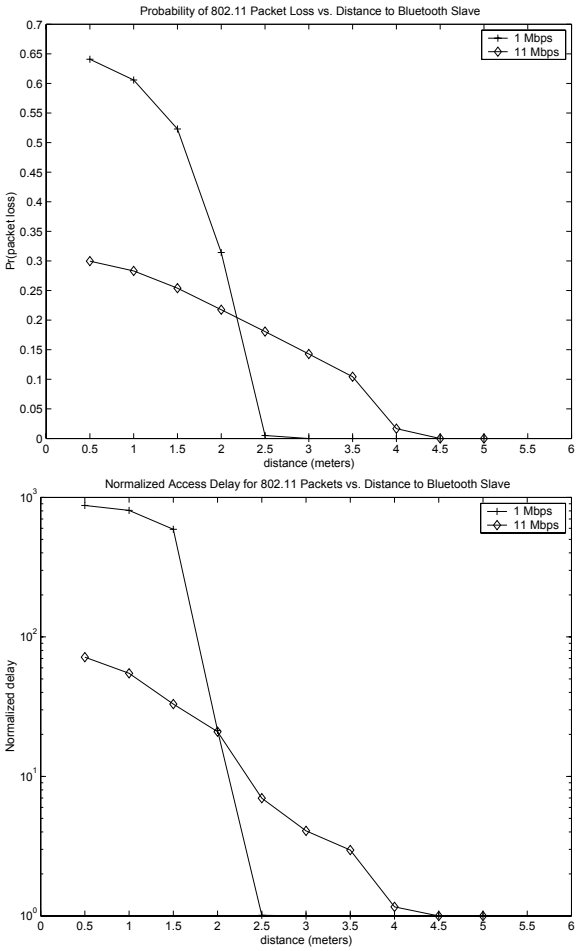


Figure 7: (a)/(b) Experiment 3. 802.11 packets with Bluetooth voice packets as interference. (a) Probability of packet loss. (b) Delay.

LAN interference than for Bluetooth voice interference (Experiment 3). The main reason for this difference is that the average interarrival time of the Bluetooth packets is now 29.16 ms. Again, we see a distance where the performance of the 1 Mb/s system becomes better than the 11 Mb/s system, both in terms of probability of packet loss and delay. Beyond four meters, both systems show very little effects from interference, and the higher speed system again becomes the preferred choice. It should be noted that depending on the topology and the transmitter powers, the exact distance where one data rate becomes better than another will change. Yet, it is conjectured that these results will hold for very general scenarios.

Figure 8(b) shows the access delays. At a distance of half a meter, the 1 Mb/s case requires a delay less than three times the delay with no interference, while the 11 Mb/s case has a delay about 1.5 times its optimal value. Please compare this to the previous experiment, where the delay for the 1 Mb/s case is about 900 times greater, and the delay for the 11 Mb/s is approximately 70 times. Not surprisingly, the streaming voice packets cause substantially more interference.

5. CONCLUDING REMARKS

We presented results on the performance of Bluetooth and WLAN operating in the 2.4 GHz ISM band based on detailed channel, MAC, and PHY layer models for both systems. The evaluation framework used allows us to study the impact of interference in a closed loop environment where two systems are affecting each other, and explore the MAC and PHY layer interactions in each system.

Our results indicate that scenarios using Bluetooth voice traffic may be the worst of all interference cases (65% of packet loss for the WLAN 1 Mbits/s system). Also, we note that Blutooth voice may be severely impacted by interference with packet loss of $\approx 8\%$. Moreover, the results suggest that the data rate in the WLAN system may be a factor in the performance, and, the recommended rate for WLAN depends on the topology and the parameters used. Therefore, one may want to exploit the data rate scaling algorithm available in the WLAN system for improving performance. Additionally, results could be obtained with the

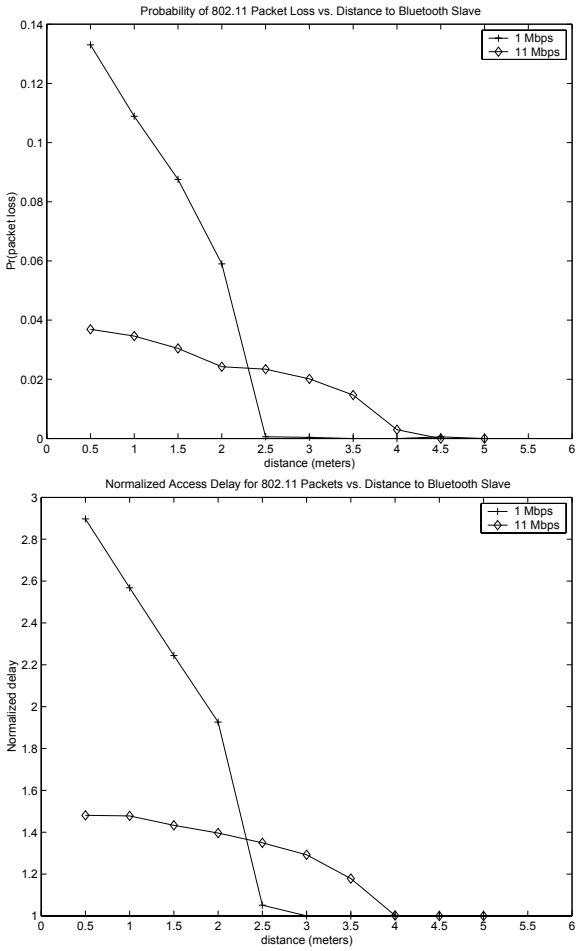


Figure 8: $\frac{(a)}{(b)}$ Experiment 4. 802.11 packets with Bluetooth LAN packets as interference. (a) Probability of packet loss. (b) Delay.

WLAN Frequency Hopping systems and compared to the Direct Sequence system presented here.

Although the results depend on a number of parameters including traffic distribution, we believe that similar trends should apply for most practical scenarios. Still, there may be some benefit in looking at more complicated scenarios with more than two devices of each type and in studying higher layer traffic such as TCP/IP. Other future directions include exploring acquisition mechanisms for WLAN and Bluetooth and their respective performance in an interference-limited environment. Finally, we hope that the work presented here could represent a first step in the development of coexistence mechanisms.

6. REFERENCES

[1] Bluetooth Special Interest Group, "Specifications of the Bluetooth System, vol. 1, v.1.0B 'Core' and vol. 2 v1.0B 'Profiles'," December 1999.

[2] IEEE Std. 802-11, "IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification ," June 1997.

[3] A. Kamerman, "Coexistence between Bluetooth and IEEE 802.11 CCK: Solutions to avoid mutual interference," in *IEEE P802.11 Working Group Contribution, IEEE P802.11-00/162r0*, July 2000.

[4] J. Zyren, "Reliability of IEEE 802.11 WLANs in Presence of Bluetooth Radios," in *IEEE P802.11 Working Group Contribution, IEEE P802.15-99/073r0*, Santa Rosa, California, September 1999.

[5] S. Shellhammer, "Packet Error Rate of an IEEE 802.11 WLAN in the Presence of Bluetooth," in *IEEE P802.15 Working Group Contribution, IEEE P802.15-00/133r0*, Seattle, Washington, May 2000.

[6] N. Golmie and F. Mouveaux, "Interference in the 2.4 GHz ISM band: Impact on the Bluetooth access control performance," in *Proceedings of IEEE ICC'01*, Helsinki, Finland, June 2001.

[7] J. Lansford, R. Nevo, and B. Monello, "Wi-Fi (802.11b) and Bluetooth Simultaneous Operation: Characterizing the Problem," in *Mobilian White Paper*, www.mobilian.com, September 2000.

[8] P. Varshney and S. Kumar, "Performance of GMSK in a land mobile radio channel," in *IEEE Transactions on Vehicular Technology*, Aug. 1991, vol. 40, pp. 607–614.

[9] M. K. Simon and C. C. Wang, "Differential versus limiter-discriminator detection of narrow-band FM," in *IEEE Transactions on Communications*, Nov. 1983, vol. COM-31, pp. 1227–1234.

[10] T. Ekvetchavit and Z. Zvonar, "Performance of Phase-locked Loop Receiver in Digital FM Systems," in *Ninth IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 1998, vol. 1, pp. 381–385.

[11] A. Soltanian and R. E. Van Dyck, "Physical layer performance for coexistence of Bluetooth and IEEE 802.11b," in *Virginia Tech Symposium on Wireless Personal Communications*, June 2001.

[12] D. L. Schilling, L. B. Milstein, R. L. Pickholtz, R. W. Brown , "Optimization of the processing Gain of an M-ary Direct Sequence Spread Spectrum Communication System," in *IEEE Trans. on Communications*, Aug 1980, pp. 1389–1398.

[13] K. Halford, S. Halford, M. Webster, and C. Andren, "Complementary code keying for rake-based wireless communication," in *Proceedings of the 1999 International Symposium on Circuits and Systems*, 1999, vol. 4, pp. 427–430.

Interference Evaluation of Bluetooth and IEEE 802.11 Systems

N. Golmie, R. E. Van Dyck, A. Soltanian, A. Tonnerre, and O. Rebala
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
Email: nada.golmie@nist.gov, robert.vandyck@nist.gov

ABSTRACT

The emergence of several radio technologies such as Bluetooth, and IEEE 802.11, operating in the 2.4 GHz unlicensed ISM frequency band, may lead to signal interference and result in significant performance degradation when devices are co-located in the same environment. The main goal of this paper is to evaluate the impact of interference on the performance of Bluetooth and IEEE 802.11b systems. We develop a simulation framework for modeling interference based on detailed MAC and PHY models. First, we use a simple simulation scenario to highlight the effects of parameters, such as transmission power, offered load, and traffic type. We then, turn to more complex scenarios involving multiple Bluetooth piconets and WLAN devices.

Keywords

WPANs, Bluetooth, IEEE 802.11, Interference.

1. INTRODUCTION

The proliferation of mobile computing devices including laptops, personal digital assistants (PDAs), and wearable computers has created a demand for wireless personal area networks (WPANs). WPANs allow closely located devices to share information and resources. A key challenge in the design of WPANs is adapting to a hostile radio environment that includes noise, time-varying channels, and abundant electromagnetic interference. Today, most radio technologies considered by WPANs (Bluetooth Special Interest Group [1], and IEEE 802.15) employ the 2.4 GHz ISM frequency band, which is also used by Local Area Network (WLAN) devices implementing the IEEE 802.11 standard specifications [2]. It is anticipated that some interference will result from all these technologies operating in the same environment. WLAN devices operating in proximity to WPAN devices may significantly impact the performance of WPAN and vice versa.

The main goal of this paper is to present our findings on the performance of these systems when operating in close proximity to each other. Our results are based on detailed models for the MAC, PHY, and wireless channel. Recently, a number of research activities has led to the development of tools for wireless network simulation [3] [4]. While some of these tools include a PHY layer implementation, it is often abstracted to a discrete channel model that does not implement interference per se. Therefore, in order to model interference and capture the time and frequency collisions, we chose to implement an integrated MAC-PHY module.

Efforts to study interference in the 2.4 GHz band are relatively recent. For example, interference caused by microwave ovens operating in the vicinity of a WLAN network has been investigated [5] and requirements on the signal-to-noise ratio (SNR) are presented by Kamerman *et al.* [6]. In addition, there has been several attempts at quantifying the impact of interference on both the WLAN and the Bluetooth performance. Published results can be classified into at least three categories depending on whether they rely on analysis, simulation, or experimental measurements.

Analytical results based on probability of packet collision were obtained by Shellhammer [7], Ennis [8], and Zyren [9] for the WLAN packet error and by Golmie *et al.* [10] for the Bluetooth packet error. In all these cases, the probability of packet error is computed based on the probability of packet collision in time and frequency. Although these analytical results can often give a first order approximation on the impact of interference and the resulting performance degradation, they often make assumptions concerning the traffic distributions and the operation of the media access protocol, which can make them less realistic. More importantly, in order for the analysis to be tractable, mutual interference that can change the traffic distribution for each system is often ignored.

On the other hand, experimental results such as the ones obtained by Kamerman [11], Howittt *et al.* [12], and Fumolari [13] for a two node WLAN system and a two node Bluetooth piconet, can be considered more accurate at the cost of being too specific to the implementation tested. Thus, a third alternative consists of using modeling and simulation to evaluate the impact of interference. This third approach can provide a more flexible framework. Zurbes *et al.* [14] present simulation results for a number of Bluetooth devices located in a single large room. They show that for 100 concurrent web sessions, performance is degraded by only five

percent. Golmie *et al.* [15] use a detailed MAC and PHY simulation framework to evaluate the impact of interference for a pair of WLAN devices and a pair of Bluetooth devices. Similar results have been obtained by Lansford *et al.* [16] for the case of co-located WLAN and Bluetooth devices on the same laptop. Their simulation models are based on a link budget analysis and a theoretical calculation of the BER (Q function calculation). The work in this paper is an extension of [15].

This paper is organized as follows. In section 2, we give some general insights on the Bluetooth and IEEE 802.11 protocol operation. In section 3, we describe in great detail our modeling approach for the MAC, PHY and wireless channel. In section 4, we evaluate the impact of interference on both Bluetooth and WLAN performance and present simulation results. Concluding remarks are offered in section 5.

2. PROTOCOL OVERVIEW

2.1 Bluetooth

In this section, we give a brief overview of the Bluetooth technology [1] and discuss the main functionality of its protocol specifications. Bluetooth is a short range (0 m - 10 m) wireless link technology aimed at replacing non-interoperable proprietary cables that connect phones, laptops, PDAs and other portable devices together. Bluetooth operates in the ISM frequency band starting at 2.402 GHz and ending at 2.483 GHz in the USA and Europe. 79 RF channels of 1 MHz width are defined. The air interface is based on an antenna power of 1 mW with an antenna gain of 0 dB. The signal is modulated using binary Gaussian Frequency Shift Keying (GFSK). The raw data rate is defined at 1 Mbits/s. A Time Division Multiplexing (TDM) technique divides the channel into 625 μs slots. Transmission occurs in packets that occupy an odd number of slots (up to 5). Each packet is transmitted on a different hop frequency with a maximum frequency hopping rate of 1600 hops/s.

Two or more units communicating on the same channel form a piconet, where one unit operates as a master and the others (a maximum of seven active at the same time) act as slaves. A channel is defined as a unique pseudo-random frequency hopping sequence derived from the master device's 48-bit address and its Bluetooth clock value. Slaves in the piconet synchronize their timing and frequency hopping to the master upon connection establishment. In the connection mode, the master controls the access to the channel using a polling scheme where master and slave transmissions alternate. A slave packet always follows a master packet transmission as illustrated in Figure 1, which depicts the master's view of the slotted TX/RX channel.

There are two types of link connections that can be established between a master and a slave: the Synchronous Connection-Oriented (SCO), and the Asynchronous Connection-Less (ACL) link. The SCO link is a symmetric point-to-point connection between a master and a slave where the master sends an SCO packet in one TX slot at regular time intervals, defined by T_{SCO} time slots. The slave responds with an SCO packet in the next TX opportunity. T_{SCO} is set to either 2, 4 or 6 time slots for $HV1$, $HV2$, or $HV3$ packet formats, respectively. All three formats of SCO packets are defined to carry 64 Kbits/s of voice traffic and are never retransmitted in case of packet loss or error.

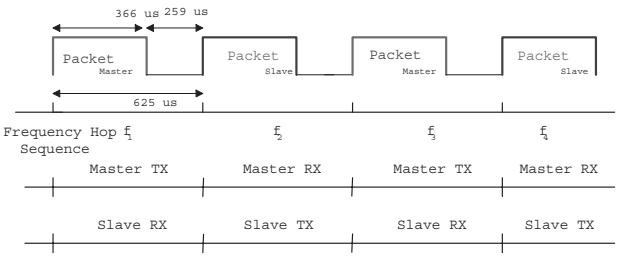


Figure 1: Master TX/RX Hopping Sequence

The ACL link is an asymmetric point-to-point connection between a master and active slaves in the piconet. An Automatic Repeat Request (ARQ) procedure is applied to ACL packets where packets are retransmitted in case of loss until a positive acknowledgement (ACK) is received at the source. The ACK is piggy-backed in the header of the returned packet where an ARQN bit is set to either 1 or 0 depending on whether or not the previous packet was successfully received. In addition, a sequence number (SEQN) bit is used in the packet header in order to provide a sequential ordering of data packets in a stream and filter out retransmissions at the destination. Forward Error Correction (FEC) is used on some SCO and ACL packets in order to correct errors and reduce the number of ACL retransmissions.

Both ACL and SCO packets have the same packet format. It consists of a 72-bit access code used for message identification and synchronization, a 54-bit header and a variable length payload that contains either a voice or a data packet depending on the type of link connection that is established between a master and a slave.

A repetition code of rate 1/3 is applied to the header, and a block code with minimum distance, d_{min}, equal to 14, is applied to the access code so that up to 13 errors are detected and $\lfloor (d_{min}-1)/2 \rfloor = 6$ can be corrected. Note that uncorrected errors in the header and the access code, lead to a packet drop. Voice packets have a total packet length of 366 bits including the access code and header. A repetition code of 1/3 is used for $HV1$ packet payload. On the other hand, DM and $HV2$ packet payloads use a 2/3 block code where every 10 bits of information are encoded with 15 bits. DH and $HV3$ packets do not have any encoding on their payload. HV packets do not have a CRC in the payload. In case of an error occurrence in the payload, the packet is never dropped. Uncorrected errors for DM and DH packets lead to dropped packets and the application of the ARQ and SEQN schemes. Table 1 summarizes the error occurrences in the packet and the actions taken by the protocol.

2.2 IEEE 802.11

The IEEE 802.11 standard [2] defines both the physical (PHY) and medium access control (MAC) layer protocols for WLANs. In this sequel, we shall be using WLAN and 802.11 interchangeably.

Table 1: Summary of error occurrences in the packet and actions taken in case errors are not corrected.

Error Location	Error Correction	Action Taken
Access Code	$d_{min} = 14$	Packet dropped
Packet Header	1/3 Repetition	Packet dropped
HV1 payload	1/3 Repetition	Packet accepted
HV2 payload	2/3 Block Code	Packet accepted
HV3 payload	No FEC	Packet accepted
DM1, DM3, DM5 payload	2/3 Block Code	Packet dropped
DH1, DH3, DH5 payload	No FEC	Packet accepted

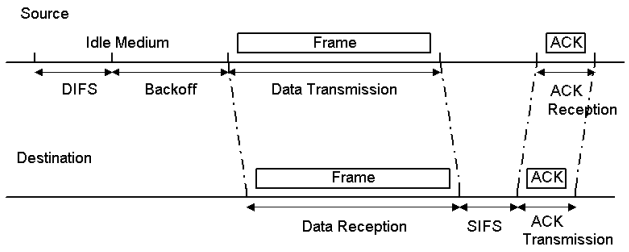


Figure 2: WLAN Frame Transmission Scheme

The IEEE 802.11 standard calls for three different PHY specifications: frequency hopping (FH) spread spectrum, direct sequence (DS) spread spectrum, and infrared (IR). The transmit power for DS and FH devices is defined at a maximum of 1 W and the receiver sensitivity is set to -80 dBmW. Antenna gain is limited to 6 dB maximum. In this work, we focus on the 802.11b specification (DS spread spectrum) since it is in the same frequency band as Bluetooth and the most commonly deployed.

The basic data rate for the DS system is 1 Mbits/s encoded with differential binary phase shift keying (DBPSK). Similarly, a 2 Mbits/s rate is provided using differential quadrature phase shift keying (DQPSK) at the same chip rate of 11×10^6 chips/sec. Higher rates of 5.5 and 11 Mbits/s are also available using techniques combining quadrature phase shift keying and complementary code keying (CCK); all of these systems use 22 MHz channels.

The IEEE 802.11 MAC layer specifications, common to all PHYs and data rates, coordinate the communication between stations and control the behavior of users who want to access the network. The Distributed Coordination Function (DCF), which describes the default MAC protocol operation, is based on a scheme known as carrier-sense, multiple access, collision avoidance (CSMA/CA). Both the MAC and PHY layers cooperate in order to implement collision avoidance procedures. The PHY layer samples the received energy over the medium transmitting data and uses a clear channel assessment (CCA) algorithm to determine if the channel is clear. This is accomplished by measuring the RF energy at the antenna and determining the strength of the received signal commonly known as RSSI, or received signal strength indicator. In addition, carrier sense can be used to determine if the channel is available. This technique is more selective since it verifies that the signal is the same carrier type as 802.11 transmitters. In all of our simulations, we use carrier sense and not RSSI to determine if the channel is busy. Thus, a Bluetooth signal will corrupt WLAN packets, but it will not cause the WLAN to defer transmission.

A virtual carrier sense mechanism is also provided at the MAC layer. It uses the request-to-send (RTS) and clear-to-send (CTS) message exchange to make predictions of future traffic on the medium and updates the network allocation vector (NAV) available in stations. Communication is established when one of the wireless nodes sends a short RTS frame. The receiving station issues a CTS frame that echoes the sender's address. If the CTS frame is not received, it is assumed that a collision occurred and the RTS process starts over. Regardless of whether the virtual carrier sense routine is used or not, the MAC is required to implement a basic access procedure (depicted in Figure 2) as follows. If a station has data to send, it waits for the channel to be idle through the use of the CSMA/CA algorithm. If the medium is sensed idle for a period greater than a DCF interframe space (DIFS), the station goes into a backoff procedure before it sends its frame. Upon the successful reception of a frame, the destination station returns an ACK frame after a Short interframe space (SIFS). The backoff window is based on a random value uniformly distributed in the interval $[CW_{min}, CW_{max}]$, where CW_{min} and CW_{max} represent the Contention Window parameters. If the medium is determined busy at any time during the backoff slot, the backoff procedure is suspended. It is resumed after the medium has been idle for the duration of the DIFS period. If an ACK is not received within an ACK timeout interval, the station assumes that either the data frame or the ACK was lost and needs to retransmit its data frame by repeating the basic access procedure.

Errors are detected by checking the Frame Check Sequence (FCS) that is appended to the packet payload. In case an error is found, the packet is dropped and is then later retransmitted.

3. INTEGRATED SIMULATION MODEL

In this section, we describe the methodology and platform used to conduct the performance evaluation. The simulation environment consists of detailed models for the RF channel, the PHY, and MAC layers developed in C and OPNET (for the MAC layer). These detailed simulation models constitute an evaluation framework that is critical to studying the various intricate effects between the MAC and PHY layers. Although interference is typically associated with the RF channel modeling and measured at the PHY layer, it can significantly impact the performance of higher layer applications including the MAC layer. Similarly, changes in the behavior of the MAC layer protocol and the associated data traffic distribution can play an important factor in the interference scenario and affect the overall system performance.

Figure 3 shows a packet being potentially corrupted by two

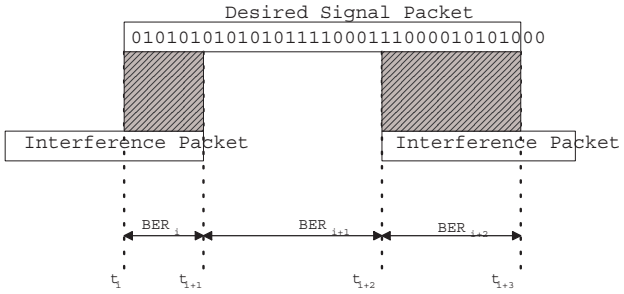


Figure 3: Packet Collision and Placement of Errors. The bit error rate (BER) is roughly constant during each of the three indicated periods.

interference packets. Consider that the desired packet is from the WLAN and the interference packets are Bluetooth (the figure is equally valid if the roles are reversed, except that the frequencies of the packets will be different). For interference to occur, the packets must overlap in both time and frequency. That is, the interference packets must be within the 22 MHz bandwidth of the WLAN. In a system with many Bluetooth piconets, there may be interference from more than one packet at any given time. We define a period of stationarity (POS) as the time during which the interference is constant. For example, $t_i \leq t \leq t_{i+1}$ is such a period, as is $t_{i+1} \leq t \leq t_{i+2}$.

Even during a POS where there is one or more interferers, the number and location of bit errors in the desired packet depends on a number of factors: (1) the signal-to-interference ratio (SIR) and the signal-to-noise ratio at the receiver, (2) the type of modulation used by the transmitter and the interferer, and (3) the channel model. For this reason, it is essential to use accurate models of the PHY and channel, as described below. Just because two packets overlap in time and frequency does not necessary lead to bit errors and the consequent packet loss. While one can use (semi-)analytic models instead, such as approximating Bluetooth interference on WLAN as a narrowband tone jammer, the use of detailed signal processing-based models better allows one to handle multiple simultaneous interferers.

In order to simulate the overall system, an interface module was created that allows the MAC models to use the physical layer and channel models. This interface module captures all changes in the channel state (mainly in the energy level). Consider the Bluetooth transmitter-channel-receiver chain of processes. For a given packet, the transmitter creates a set of signal samples that are corrupted by the channel and input to the receiver; interference may be present for all or only specific periods of stationarity, as shown in Figure 3. A similar chain of processing occurs for an 802.11b packet. The interface module is designed to process a packet at a time.

At the end of each packet transmission, the MAC layer generates a data structure that contains all the information required to process the packet. This structure includes a list of all the interfering packets with their respective duration, timing offset, frequency, and transmitted power. The topology of the scenario is also included. The data structure is then passed to the physical layer along with a stream of bits representing the packet being transmitted. The physical layer returns the bit stream after placing the errors resulting from the interference.

3.1 MAC Model

We used *OPNET* to develop a simulation model for the Bluetooth and IEEE 802.11 protocols. For Bluetooth, we implemented the access protocol according to the specifications [1]. We assume that a connection is already established between the master and the slave and that the synchronization process is complete. The Bluetooth hopping pattern algorithm is implemented. Details of the algorithm are provided in section 2.1. A pseudo-random number generator is used instead of the implementation specific circuitry that uses the master's clock and 48-bit address to derive a random number.

For the IEEE 802.11 protocol, we used the model available in the OPNET library and modified it to bypass the OPNET radio model and to use our MAC/PHY interface module. We focus in this study on the Direct Sequence mode which uses a fixed frequency that occupies 22 MHz of the frequency band. The center frequency is set to 2,437 GHz.

At the MAC layer, a set of performance metrics are defined including probability of packet loss. Packet loss measures the number of packets discarded at the MAC layer due to errors in the bit stream. This measure is calculated after performing error correction.

3.2 PHY Model

The transmitters, channel, and receivers are implemented at complex baseband. For a given transmitter, inphase and quadrature samples are generated at a sampling rate of 44×10^6 per second. This rate provides four samples/symbol for the 11 Mbits/s 802.11 mode, enough to implement a good receiver. It is also high enough to allow digital modulation of the Bluetooth signal to account for its frequency hopping. Specifically, since the Bluetooth signal is approximately 1 MHz wide, it can be modulated up to almost 22 MHz, which is more than enough to cover the 11 MHz bandwidth (one-sided) of the 802.11 signal. The received complex samples from both the desired transmitter and the interferer(s) are added together at the receiver.

While there are a number of possible Bluetooth receiver designs, we chose to implement the noncoherent limiter-discriminator (LD) receiver [17] [18]. Its simplicity and relatively low cost should make it the most common type for many consumer applications. Details of the actual design are given in [19].

In the 802.11b CCK receiver, each group of eight information bits chooses a sequence of eight consecutive chips that forms a symbol. As before, the inphase and quadrature components of these chips are transmitted. The receiver looks at the received symbol and decides which was the most likely transmitted one. While one can implement this decoding procedure by correlating against all 256 possible symbols, we chose a slightly sub-optimal, but considerably faster architecture similar to the Walsh-Hadamard transform; again

details can be found in [19].

3.3 Channel Model

The channel model consists of a geometry-based propagation model for the signals, as well as a noise model. For the indoor channel, we apply a propagation model consisting of two parts: (1) line-of-sight propagation (free-space) for the first 8 meters, and (2) a propagation exponent of 3.3 for distances over 8 meters. Consequently, the path loss in dB is given by

$$L_p = \begin{cases} 32.45 + 20\log(f \cdot d) & \text{if } d < 8 \text{ m} \\ 58.3 \ + 33\log(d/8) & \text{otherwise,} \end{cases} \quad (1)$$

where f is the frequency in GHz, and d is the distance in meters. This model is similar to the one used by Kamerman [11]. Assuming unit gain for the transmitter and receiver antennas and ignoring additional losses, the received power in dBmW is

$$P_R = P_T - L_p, \quad (2)$$

where P_T is the transmitted power also in dBmW. Eq. (2) is used for calculating the power received at a given point due to either a Bluetooth or an 802.11 transmitter, since this equation does not depend on the modulation method.

The main parameter that drives the PHY layer performance is the signal-to-interference ratio between the desired signal and the interfering signal. This ratio is given in dB by

$$SIR = P_R - P_I, \quad (3)$$

where P_I is the interference power at the receiver. In the absence of interference, the bit error rate for either the Bluetooth or WLAN system is almost negligible for the transmitter powers and ranges under consideration.

To complete the channel model, noise is added to the received samples, according to the specified SNR. In decibels, the signal-to-noise ratio is defined by $SNR = P_R - S_R$, where P_R is the received signal power, and S_R is the receiver's sensitivity in dBmW; this latter value is dependent on the receiver model and so is an input parameter. Additive white Gaussian noise (AWGN) is used to model the noise at the receivers.

3.4 Model Validation

4. SIMULATION RESULTS

We present simulation results to evaluate the performance of Bluetooth in the presence of WLAN interference and vice versa. First, we consider the effects of parameters such as transmitted power, offered load, hop rate, and traffic type on interference. Second, we look at two realistic interference scenarios to quantify the severity of the performance degradation for the Bluetooth and WLAN systems.

4.1 Factors Effecting Interference

We first consider a four node topology consisting of two WLAN devices and two Bluetooth devices (one master and one slave) as shown in Figure 4. The WLAN access point (AP) is located at (0,15) meters, and the WLAN mobile is fixed at (0,1) meters. The Bluetooth slave device is fixed at (0,0) meters and the master is fixed at (1,0) meters.

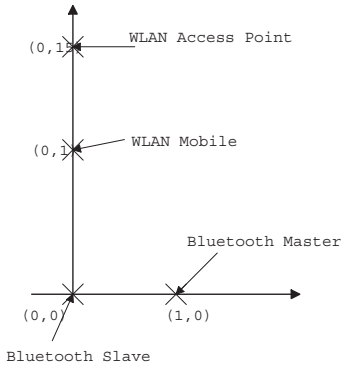


Figure 4: Topology 1 - Two WLAN devices and one Bluetooth piconet

In an effort to control the interference on Bluetooth and WLAN, we define two scenarios. In the first scenario, we let the mobile be the generator of 802.11 data, while the AP is the sink. In this case, the interference is from the mobile sending data packets to the AP and receiving acknowledgments (ACKs) from it. Since most of the WLAN traffic is originating close to the Bluetooth piconet, both the master and the slave may suffer from serious interference. In the second scenario, the traffic is generated at the AP and received at the WLAN mobile. Because the data packets are generally longer then the ACKs, this is a more critical scenario for the WLAN then when the mobile is the source. Table 2 summarizes the two scenarios.

Table 2: Summary of the Scenarios

Scenario	Desired Signal	Interferer Signal	WLAN AP	WLAN Mobile
1	Bluetooth	WLAN	Sink	Source
2	WLAN	Bluetooth	Source	Sink

For Bluetooth, we consider two types of applications, voice and data. For voice, we assume a symmetric stream of 64 Kbits/s each way using $HV1$ packet encapsulation. For data traffic, we consider a source that generates $DM5$ packets. The packet interarrival time is exponentially distributed, and its mean in seconds is computed according to

$$t_B = 2 \times n_s \times T_s/\lambda, \quad (4)$$

where λ is the offered load; n_s is the number of slots occupied by a packet. For $DM5$, $n_s = 5$. T_s is the slot size equal to 625 μs.

For WLAN, we use the 11 Mbits/s mode and consider a data application. Typical applications for WLAN could be ftp or http. However, since we are mainly interested in the MAC layer performance, we abstract the parameters for the application model to packet size and offered load and do not model the entire TCP/IP stack. We fix the packet payload to 12,000 bits which is the maximum size for the MAC payload data unit, and vary λ. The packet interarrival time in seconds, t_W, is exponentially distributed, and its mean is

computed according to

$$t_W = (\frac{192}{1,000,000} + \frac{12,224}{11,000,000})/\lambda, \quad (5)$$

where the 192-bit PLCP header is sent at 1 Mbits/s and the payload at 11 Mbits/s. Unless specified otherwise, we use the configuration and system parameters shown in Table 3.

For scenarios 1 and 2, we run 15 trials using a different random seed for each data point. In addition to plotting the mean value, confidence intervals, showing plus and minus two standard deviations, are also included. From Figures 5 and 6, one sees that the statistical variation around the mean values are very small. In addition to the comparisons with analytical and experimental results described in Section 3.4, this fact provides further validation for the results.

Table 3: Simulation Parameters

Simulation Parameters	**Values**
Propagation delay	5 μs/km
Length of simulation run	10 seconds
Bluetooth Parameters	**Values**
ACL Baseband Packet Encapsulation	DM5
SCO Baseband Packet Encapsulation	HV1
Transmitted Power	1 mW
WLAN Parameters	
Transmitted Power	25 mW
Packet Header	224 bits
Packet Payload	12,000 bits

WLAN Transmission Power

First, we look at the effect on Bluetooth of increasing the WLAN transmission power in scenario 1; that is, increasing the interferer transmission power on the victim signal. Since power control algorithms exist in many WLAN implementations, it is important to consider how varying the transmitted power changes the interference. However, since Bluetooth was designed as a low power device, we fix its transmitter power at 1 mW for all simulations.

We fix WLAN λ to 60% for different Bluetooth traffic types and values of λ. In Figure 5(a), we note a saturation effect around 10 mW. A threshold, which is close to 22/79, corresponds to the probability that Bluetooth is hopping in the WLAN occupied band. Thus, increasing the WLAN transmission power beyond 10 mW does not affect the Bluetooth packet loss. Between 1 and 5 mW, a small change in the WLAN transmitted power triples the Bluetooth packet loss. Please note the relative positions of the packet loss curves for different values of λ between 1 and 5 mW; as λ increases, the packet loss is higher. Also, note that Bluetooth voice has the lowest packet loss, parly due to its short packet size. A second reason for the low loss probability is that voice packets are rejected only if there are errors in the access code or packet headers, *cf.* Table 1. A packet may be accepted with a relatively large number of bit errors in the payload, which may lead to a substantial reduction in subjective voice quality.

Figure 5(b) shows the probability of packet loss for the WLAN mobile device. This corresponds to ACKs being dropped at the WLAN source. The general trend is that the packet loss decreases as the WLAN transmitted power increases. However, we notice a slight "bump" between 1 and 5mW for λ equals 30% and 60%. This is due to the effect of closed-loop interference. The WLAN source increases its transmitted power and causes more interference on the Bluetooth devices; as a result, there are more retransmissions within the Bluetooth piconet, which causes more lost ACKs at the WLAN source. As expected, this effect is not present for $\lambda = 100\%$ and voice traffic since the traffic distribution is not altered by packet retransmission.

Next, we consider the effect of increasing the WLAN transmission power on the WLAN performance in scenario 2. From Figure 5(c), we observe that even if the WLAN transmission power is fifty times more than the Bluetooth transmission power (fixed at 1mW), the packet loss for the WLAN is only decreased by at most half. This leads us to an interesting observation on power control. Basically, we note that increasing the transmission power does not necessarily improve the performance. However, decreasing the transmission power is usually a "good neighbor" strategy that may help reduce the interference on other devices.

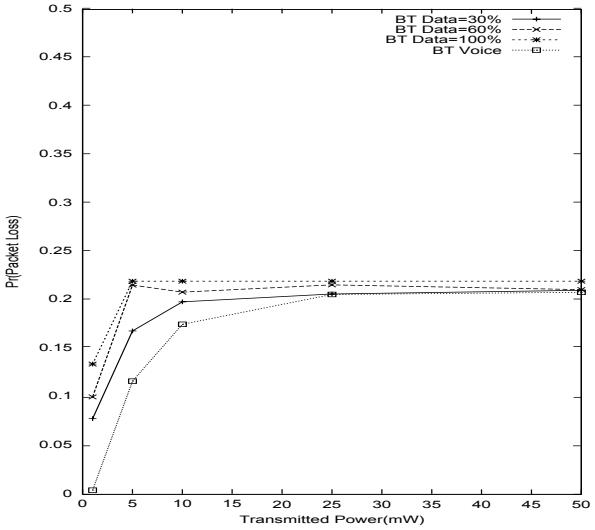
Offered Load

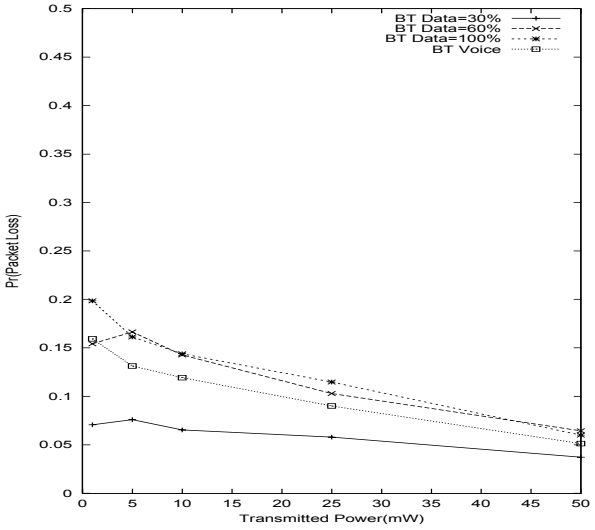
The offered load, also referred to in some cases as duty cycle, is an interesting parameter to track. Consider scenario 1 where Bluetooth is the interferer and fix the WLAN transmission power to 25 mW. We observe that for the WLAN, the packet loss is proportional to the Bluetooth offered load as shown in Figure 6. For λ equals 20%, 50%, and 100% the packet loss is 15%, 35%, and 45%, respectively. This observation has been confirmed analytically in [10], where the packet error is shown to depend not only on the offered load of the interferer system but also on the packet sizes of both systems.

The significance of the packet size is apparent in Figures 5 (a) and (c), where short Bluetooth voice packets lead to less packet loss for Bluetooth but cause more interference for WLAN. However, for the WLAN 11 Mbits/s rate, the effect of changing the WLAN packet size over the range 1,000 to 12,000 bits has very little effect on the performance of both the WLAN and Bluetooth, and that is due to the relatively short transmission time of the WLAN packet. At the 1 Mbits/s rate, WLAN packets of the same bit lengths take considerably longer to transmit, and the effect of packet size is somewhat more pronounced. For a further discussion of the 1 Mbits/s case, please see [15].

Bluetooth Hop Rate

In order to highlight the effect of the Bluetooth hop rate on WLAN, we use different packet types, DM1, DM3, and DM5; these packets occupy 1, 3, and 5 time slots, respectively. The Bluetooth hop rate is determined by the number of time slots occupied by a packet. Thus, the hop rate is 1600, 533, and 320 hops/s for $DM1$, $DM3$, and $DM5$ packets, respectively. The offered load for Bluetooth is set to 100%. The results in Table 4 clearly indicate that a faster hop rate leads to higher packet losses (89%, 59%, and 45%





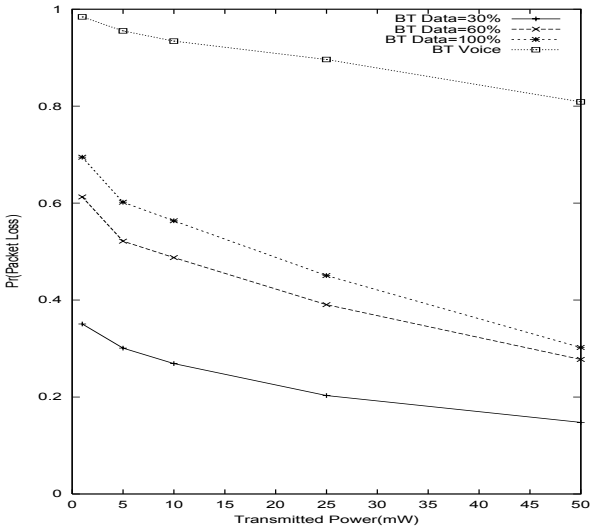


Figure 5: $\frac{(a)}{(b)}$ **WLAN $\lambda = 60\%$. (a) Scenario 1. Probability of packet loss for the Bluetooth slave. (b) Scenario 1. Probability of packet loss for the WLAN mobile. (c) Scenario 2. Probability of packet loss for the WLAN mobile.**

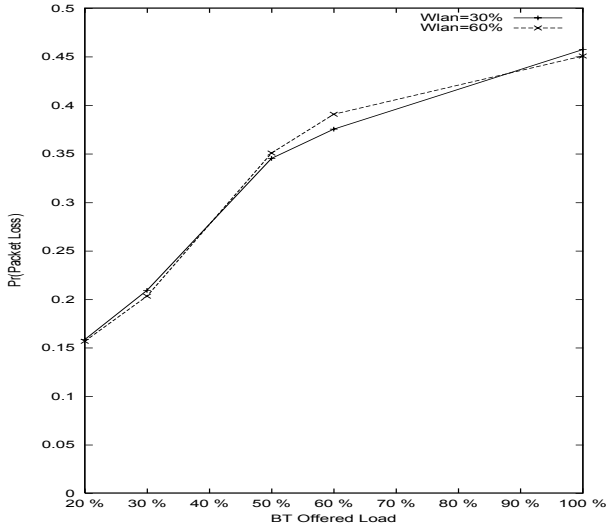


Figure 6: Scenario 2. Probability of packet loss for the WLAN mobile.

for $DM1$, $DM3$ and $DM5$, respectively). Note that the results are insensitive to the WLAN offered load.

Table 4: Scenario 2. Probability of WLAN packet loss versus Bluetooth hop rate.

BT	WLAN $\lambda = 30\%$	WLAN $\lambda = 60\%$
DM1	0.8947	0.8962
DM3	0.5955	0.5954
DM5	0.4572	0.4505

Bluetooth Traffic Type

The question here is, whether Bluetooth voice impacts WLAN more than Bluetooth data, and vice versa. We use three types of packets for voice encapsulation, namely, $HV1$, $HV2$, and $HV3$. $HV1$ represents the worst case of interference for WLAN as shown in Table 5 with 89% packet loss. $HV2$ and $HV3$, which contain less error correction and more user information, are sent less often and therefore interfer less with WLAN (71% and 56% for $HV2$ and $HV3$, respectively). The WLAN packet loss with Bluetooth data interference is 37%. Please note, that the results do not depend on the WLAN offered load.

Table 5: Scenario 2. Probability of WLAN packet loss versus Bluetooth traffic type.

BT		WLAN $\lambda = 30\%$	WLAN $\lambda = 60\%$
Voice	HV1	0.8947	0.8962
	HV2	0.7180	0.7193
	HV3	0.5668	0.5603
Data, $\lambda = 60\%$		0.3752	0.3904

On the other hand, both Bluetooth voice and data are equally affected by the WLAN interference (20%) as shown in Ta-

ble 6. Note that at lower WLAN transmission powers, the Bluetooth voice has less packet loss (Figure 5(a)). Also, since all three types of voice packets suffer the same packet loss, it is preferable to use $HV3$, which causes less interference on the WLAN. The error correction coding in $HV1$ and $HV2$ packets may provide greater range in a noise-limited environment, but this coding is far too weak to protect the packets from interference. Instead, it is the frequency hopping ability of Bluetooth that limits the damage done by the WLAN.

Table 6: Scenario 1. Probability of Bluetooth packet loss versus Bluetooth traffic type.

BT		WLAN $\lambda = 30\%$	WLAN $\lambda = 60\%$
Voice	HV1	0.2045	0.2062
	HV2	0.2016	0.2015
	HV3	0.1992	0.1981
Data, $\lambda = 60\%$		0.2089	0.2159

Bluetooth Transmission Power

While most Bluetooth devices will be operating at 1 mW, the specification also allows higher transmitter powers. Table 7 shows the probability of packet loss for both Bluetooth and the WLAN for three values of the BT transmitter power and two types of Bluetooth traffic. As expected, higher transmitter powers lead to more lost WLAN packets, regardless of the BT traffic type. Increasing the power from 1 to 10 mW leads to approximately a fifty percent increase in WLAN loss. Conversely, the Bluetooth packet error rate decreases. It still not clear how beneficial this decrease is for Bluetooth; even a loss probability of 0.0335 may lead to unacceptable voice quality.

Table 7: Scenario 2. Probability of packet loss versus Bluetooth transmission power. WLAN $\lambda = 60$.

BT Traffic	BT Power	BT Loss Prob.	WLAN Loss Prob.
$\lambda = 60\%$	1	0.2125	0.0961
	2.5	0.2085	0.1227
	10	0.1733	0.1358
Voice	1	0.1417	0.1253
	2.5	0.1179	0.1609
	10	0.0335	0.1977

4.2 Realistic Interference Topologies

In this section, we consider two practical interference topologies. While they appear to be somewhat different, they actually complement each other. The first one has the WLAN device, in the midst of the Bluetooth piconets, acting at the source, while the second one has the WLAN access point acting as the source.

Topology 2

We first look at the topology illustrated in Figure 7. It consists of one WLAN AP located at (0,15) meters, and one

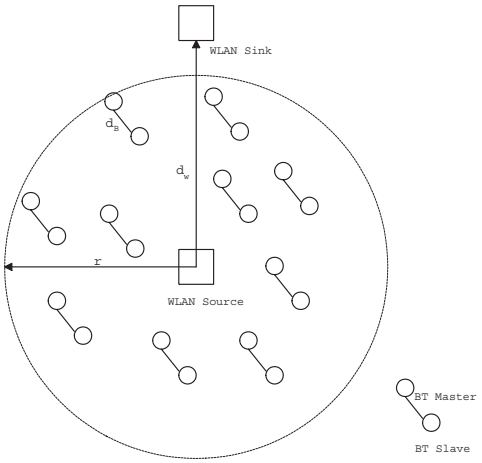


Figure 7: Topology 2 - Two WLAN devices and ten Bluetooth piconets

WLAN mobile at (0,0) meters. The WLAN traffic is generated at the mobile, while the AP returns acknowledgments. The distance between the WLAN AP and mobile is $d_W = 15$ meters. There are ten Bluetooth piconets randomly placed, covering a disk. The center of the disk is located at (0,0) and its radius is $r = 10$ meters. We define d_B as the distance between a Bluetooth master and slave pair. $d_B = 1$ meter for half of the master and slave pairs, while $d_B = 2$ meters for the other half of the master and slave pairs.

In this case, the main interference on Bluetooth is caused by the WLAN source located in the center of the disk; the aggregation of the ten piconets affects the WLAN source. We found that when the WLAN system is not operating, the Bluetooth packet loss is negligible (less than one percent). Table 8 gives the packet loss for the Bluetooth and WLAN devices. The packet loss for the Bluetooth devices is averaged over the master and slave devices and split into two groups: piconets with $d_B = 1$ meter and piconets with $d_B = 2$ meters. For WLAN, the packet loss is measured at the source. It is effectively zero at the sink.

We observe that the WLAN packet loss depends on the Bluetooth traffic load value, λ. As λ is varied from 30% to 60%, the WLAN packet loss is significantly changed from 28.85% to 45.68%. However, the WLAN packet loss is insensitive to the WLAN offered load. Unlike the previous results, Bluetooth voice does not represent the worst case interference scenario for WLAN. This is mainly due to the fact that Bluetooth voice packets are short, and the hopping rate of the piconet is 1600 hops/s, which is fast. Therefore, having ten piconets instead of one does not cause more interference on the WLAN connection.

In general, the Bluetooth packet loss for $d_B = 1$ meters is less than for $d_B = 2$ meters. The reason is that when the Bluetooth signal is stronger (over a shorter distance), the impact of interference is less significant. We also note that the Bluetooth packet loss depends on both WLAN and Bluetooth offered loads. It is higher for higher offered loads.

Topology 3

Table 8: Experiment 3 Results

BT Traffic	WLAN λ	BT Loss		WLAN Loss
		$d_B = 1$ m	$d_B = 2$ m	
$\lambda = 30\%$	30%	0.1868	0.2631	0.2885
	60%	0.2129	0.2957	0.2917
$\lambda = 60\%$	30%	0.2464	0.3796	0.4568
	60%	0.2656	0.3901	0.4557
Voice	30%	0.1025	0.1385	0.3629
	60%	0.1218	0.1659	0.3623

Table 9: Experiment 4 Results

BT Traffic	WLAN λ	BT Loss	WLAN Loss
$\lambda = 30\%$	30%	0.0130	0.3313
	60%	0.0133	0.3005
$\lambda = 60\%$	30%	0.0232	0.5184
	60%	0.0261	0.5196
Voice	30%	0.0011	0.8318
	60%	0.0010	0.8438

We next consider the topology given in Figure 8. It includes one WLAN AP and four WLAN mobile devices. The WLAN AP is located at (0,15) meters, and it is the source of the traffic generation. The four WLAN mobile devices are placed on a two-dimensional grid at (-1,1), (1,1), (-1, -1), and (1,-1) meters. In this topology, there are four Bluetooth piconets, each consisting of a master-slave device pair. The placement of the Bluetooth devices is as shown in the figure.

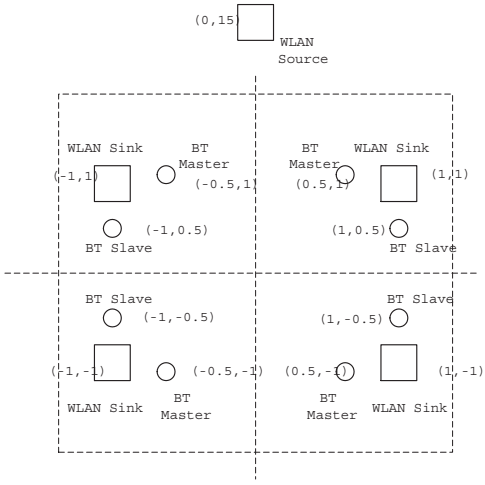


Figure 8: Topology 3 - Five WLAN devices and four Bluetooth piconets

In this case, we are looking at the effect of Bluetooth piconets on the four WLAN sink devices. The packet loss measure for WLAN is averaged over the four devices. As shown in Table 9 the impact of WLAN interference on Bluetooth is minimal, given that the WLAN source is far from the Bluetooth piconets. As expected, the WLAN packet loss depends on the Bluetooth traffic conditions, and it is rather insensitive to the WLAN traffic activity. With Bluetooth voice, the WLAN packet loss is close to 84%. It is 51% and 30% for Bluetooth with loads of $\lambda = 60\%$, and $\lambda = 30\%$, respectively.

5. CONCLUDING REMARKS

We presented results on the performance of Bluetooth and WLAN operating in the 2.4 GHz ISM band based on detailed channel, MAC, and PHY layer models for both systems. The evaluation framework used allows us to study the impact of interference in a closed loop environment where two systems are affecting each other, and explore the MAC and PHY layer interactions in each system.

We are able to draw some useful conclusions based on our results. First, we note that power control may have limited benefits in this environment. In order to reduce the WLAN packet loss to negligible levels, the transmission power has to be significantly increased (even fifty times the power of Bluetooth is not sufficient). On the other hand, limiting the WLAN power, may help avoid interference to Bluetooth. Second, using a slower hop rate for Bluetooth (i.e. longer packet sizes) may cause less interference to WLAN.

Overall, the results are dependent on the traffic distribution. Yet, there may be little room for parameter optimization especially for the practical scenarios. Not only does the complexity of the interactions and the number of parameters to adjust make the optimization problem intractable, but choosing an objective function is very dependent on the applications and the scenario. Thus, achieving acceptable performance for a particular system comes at the expense of the other system's throughput. Therefore, we believe that the primary solutions to this problem lie in the development of coexistence mechanisms.

6. REFERENCES

[1] Bluetooth Special Interest Group, "Specifications of the Bluetooth System, vol. 1, v.1.0B 'Core' and vol. 2 v1.0B 'Profiles'," December 1999.

[2] IEEE Std. 802-11, "IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification ," June 1997.

[3] "BlueHoc: Bluetooth Performance Evaluation Tool," in *Open-Source*, http://oss.software.ibm.com/deverloperworks/opensource/bluehoc, 2001.

[4] M. Takai, R, Bagrodia, A. Lee, and M. Gerla, "Impact of Channel Models on Simulation of Large Scale Wireless Networks," in *Proceedings of ACM/IEEE MSWIM'99*, Seattle, WA, August 1999.

[5] S. Unawong, S. Miyamoto, and N. Morinaga, "Techniques to improve the performance of wireless LAN under ISM interference environments ," in *Fifth Asia-Pacific Conference on Communications, 1999 and Fourth Optoelectronics and Communications Conference*, 1999, vol. 1, pp. 802–805.

[6] A. Kamerman and N. Erkocevic, "Microwave oven interference on wireless LANs operating in the 2.4

GHz ISM band," in *Proceedings of the 8th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 1997, number 3, pp. 1221–1227.

[7] S. Shellhammer, "Packet Error Rate of an IEEE 802.11 WLAN in the Presence of Bluetooth," in *IEEE P802.15 Working Group Contribution, IEEE P802.15-00/133r0*, Seattle, Washington, May 2000.

[8] G. Ennis, "Impact of Bluetooth on 802.11 Direct Sequence," in *IEEE P802.11 Working Group Contribution, IEEE P802.11-98/319*, September 1998.

[9] J. Zyren, "Reliability of IEEE 802.11 WLANs in Presence of Bluetooth Radios," in *IEEE P802.11 Working Group Contribution, IEEE P802.15-99/073r0*, Santa Rosa, California, September 1999.

[10] N. Golmie and F. Mouveaux, "Interference in the 2.4 GHz ISM band: Impact on the Bluetooth access control performance," in *Proceedings of IEEE ICC'01*, 2001.

[11] A. Kamerman, "Coexistence between Bluetooth and IEEE 802.11 CCK: Solutions to avoid mutual interference," in *IEEE P802.11 Working Group Contribution, IEEE P802.11-00/162r0*, July 2000.

[12] I. Howitt, V. Mitter, and J. Gutierrez, "Empirical Study for IEEE 802.11 and Bluetooth Interoperability," in *in IEEE Vehicular Technology Conference (VTC), Spring 2001*, May 2001.

[13] D. Fumolari, "Link Performance of an Embedded Bluetooth Personal Area Network," in *Proceedings of IEEE ICC'01*, Helsinki, Finland, June 2001.

[14] S. Zurbes, W. Stahl, K. Matheus, and J. Haartsen, "Radio network performance of bluetooth ," in *Proceedings of IEEE International Conference on Communications, ICC 2000*, New Orleans, LA, June 2000, vol. 3, pp. 1563–1567.

[15] N. Golmie, R.E. Van Dyck, and A. Soltanian, "Interference of Bluetooth and IEEE 802.11: Simulation Modeling and Performance Evaluation," in *Proceedings of the Fourth ACM International Workshop on Modeling, Analysis, and Simulation of Wireless and Mobile Systems, MSWIM'01*, Rome, Italy, July 2001.

[16] J. Lansford, A. Stephens, and R. Nevo, "Wi-Fi (802.11b) and Bluetooth: Enabling Coexistence," in *IEEE Network Magazine*, Sept/Oct. 2001.

[17] M. K. Simon and C. C. Wang, "Differential versus limiter-discriminator detection of narrow-band FM," in *IEEE Transactions on Communications*, Nov. 1983, vol. COM-31, pp. 1227–1234.

[18] T. Ekvetchavit and Z. Zvonar, "Performance of Phase-locked Loop Receiver in Digital FM Systems," in *Ninth IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 1998, vol. 1, pp. 381–385.

[19] A. Soltanian and R. E. Van Dyck, "Physical layer performance for coexistence of Bluetooth and IEEE 802.11b," in *Virginia Tech Symposium on Wireless Personal Communications*, June 2001.

Interference in the 2.4 GHz ISM Band: Challenges and Solutions

N. Golmie

National Institute of Standards and Technology
Gaithersburg, Maryland 20899
Email: nada.golmie@nist.gov

RÉSUMÉ.

ABSTRACT. Most emerging radio technologies for Wireless Personal Area Networks such as the Bluetooth protocol are designed to operate in the 2.4 GHz ISM band. Since both Bluetooth and IEEE 802.11 devices use the same frequency band and may likely come together in a laptop or may be close together at a desktop, interference may lead to signi cant performance degradation. The main goal of this paper is to describe the interference problem and to highlight a coexistence framework for these technologies to operate in a proximal environment. We give an overview of several coexistence solutions proposed for various interference scenarios. We study several factors that may impact interference such as fragmentation and the choice of packet encapsulation and give simulation results for selected scenarios and con gur ations of interest.

MOTS-CLÉS :

KEYWORDS: WPAN, Bluetooth, WLAN, Interference, Coexistence

1. Introduction

Increasingly people work and live on the move. To support this mobile lifestyle, especially as work becomes more intensely information-based, companies are producing various portable and embedded information devices including PDAs, pagers, cellular telephones and active badges. At the same time, recent advances in sensor integration and electronic miniaturization are making it possible to produce sensing devices equipped with signi cant processing memory and wireless communication capabilities to create smart environments where scattered sensors could coordinate to establish a communication network. These wearable computing devices and ad-hoc smart environments impose unique requirements on the communication protocol design such as low power consumption, frequent make and break connections, resource discovery and utilization and have created the need for Wireless Personal Area Networks (WPANs).

A WPAN is a wireless ad hoc data communications system that allows a number of independent devices to communicate. WPAN is distinguished from other types of wireless networks in both size and scope. Communications in WPAN are normally con ned to a person or object and extend up to 10 meters in all directions.

This is in contrast to Wireless Local Area Networks (WLANs) that typically cover a moderately sized geographic area such as a single building, or campus. WLANs operate in the 100 meter range and are intended to augment rather than replace traditional wired LANs. They are often used to provide the nal few feet of connectivity between the main network and the user. Users can plug into the network without having to look for a place to link their computer, or having to install expensive components and wiring.

What is emerging today are wireless technologies, including IEEE 802.11 [802], Bluetooth [GRO a], IrDa [ASS], and HomeRF [GRO b][K. 00], that promise to out t portable and embedded devices with high bandwidth, localized wireless communication capabilities that can also reach the globally wired Internet.

Due to its almost global availability, the 2.4 GHz Industry Scienti c and Medical (ISM) unlicensed band constitutes a popular frequency band suitable to low cost radio solutions such as the ones proposed for WPANs and WLANs. This sharing of the spectrum among various wireless devices that can operate in the same environment may lead to severe interference and result in signi cant performance degradation.

The main goal of this paper is to describe the interference problem. We give several interference scenario examples and provide a qualitative discussion of the performance degradation resulting from interference based on several published results in the literature. We also give an overview of the coexistence framework adopted by the IEEE 802.15.2 Task Group and discuss some of the coexistence solutions proposed.

The rest of the paper is structured as follows. In section 2, we give some general insights on the Bluetooth and WLAN device operation. In section 3, we describe the interference problem and give several interfence scenarios as example. In section 4,

we present a coexistence framework and in section 5 give some insights on factors that might impact interference such as the use of Forward Error Correction (FEC), the choice of the packet size and encapsulation. Our observations are accompanied with simulation results obtained for an example scenario. Concluding remarks are offered in section 6.

2. Wireless Technologies in the 2.4 GHz Band

In this section we give an overview of the various radio technologies operating in the 2.4 GHz unlicensed ISM band. We focus on the Bluetooth and IEEE 802.11 protocols.

2.1. *The Bluetooth Specifications*

In this section, we give a brief overview of the Bluetooth technology [GRO a] and discuss the main functionality of its protocol speci cations which consist of several modules, namely, the Radio Frequency (RF), Baseband (BB) and Link Manager (LM). Bluetooth is a short range (0 m - 10 m) wireless link technology aimed at replacing non-interoperable proprietary cables that connect phones, laptops, PDAs and other portable devices together. Bluetooth operates in the ISM frequency band starting at 2.402 GHz and ending at 2.483 GHz in the USA, and Europe. 79 RF channels of 1 MHz width are de ned. The air interface is based on an antenna power of 1 mW (0 dBi gain). The signal is modulated using binary Gaussian Frequency Shift Keying (GFSK). The raw data rate is de ned at 1 Mbits/s. A Time Division Multiplexing (TDM) technique divides the channel into 625 μs slots. Transmission occurs in packets that occupy an odd number of slots (up to 5). Each packet is transmitted on a different hop frequency with a maximum frequency hopping rate of 1600 hops/s.

Two or more units communicating on the same channel form a piconet, where one unit operates as a master and the others (a maximum of seven active at the same time) act as slaves. A channel is de ned as a unique pseudo-random frequency hopping sequence derived from the master device's 48-bit address and its Bluetooth clock value. Slaves in the piconet synchronize their timing and frequency hopping to the master upon connection establishment. In the connection mode, the master controls the access to the channel using a polling scheme where master and slave transmissions alternate. A slave packet always follows a master packet transmission.

There are two types of link connections that can be established between a master and a slave : the Synchronous Connection-Oriented (SCO), and the Asynchronous Connection-Less (ACL) link. The SCO link is a symmetric point-to-point connection between a master and a slave where the master sends an SCO packet in one TX slot at regular time intervals, de ned by T_{SCO} time slots. The slave responds with an SCO packet in the next TX opportunity. T_{SCO} is set to either 2, 4 or 6 time slots for $HV1$, $HV2$, or $HV3$ packet formats respectively. All three formats of SCO packets are de -

ned to carry 64 Kbits/s of voice traf c and are never retransmitted in case of packet loss or error. The ACL link, is an asymmetric point-to-point connection between a master and active slaves in the piconet. Several packet formats are de ned for ACL, namely $DM1$, $DM2$, and $DM3$ packets that occupy 1, 3, and 5 time slots respectively. An Automatic Repeat Request (ARQ) procedure is applied to ACL packets where packets are retransmitted in case of loss until a positive acknowledgement (ACK) is received at the source. The ACK is piggy-backed in the header of the returned packet where an ARQN bit is set to either 1 or 0 depending on whether the previous packet was successfully received or not. In addition, a sequence number (SEQN) bit is used in the packet header in order to provide a sequential ordering of data packets in a stream and lter out retransmissions at the destination. Forward Error Correction (FEC) is used on some SCO and ACL packets in order to correct errors and reduce the number of ACL retransmissions.

2.2. *The IEEE 802.11 Specifications*

The IEEE 802.11 standard [802] de nes both the physical (PHY) and medium access control (MAC) layer protocols for WLANs. In this sequel, we shall be using WLAN and 802.11 interchangeably.

The IEEE 802.11 standard calls for three different PHY speci cations : frequency hopping (FH) spread spectrum, direct sequence (DS) spread spectrum, and infrared (IR). The transmit power for DS and FH devices is de ned at a maximum of 1 W and the receiver sensitivity is set to -80 dBmW. Antenna gain is limited to 6 dB maximum. In this work, we focus on the 802.11b speci cation (DS spread spectrum) since it is in the same frequency band as Bluetooth and the most commonly deployed.

The basic data rate for the DS system is 1 Mbits/s encoded with differential binary phase shift keying (DBPSK). Similarly, a 2 Mbits/s rate is provided using differential quadrature phase shift keying (DQPSK) at the same chip rate. Higher rates of 5.5 and 11 Mbits/s are also available using techniques combining quadrature phase shift keying and complementary code keying (CCK) ; all of these systems use 22 MHz channels. Details of the modulation methods are provided in Section III.

The IEEE 802.11 MAC layer speci cations, common to all PHYs and data rates, coordinate the communication between stations and control the behavior of users who want to access the network. The Distributed Coordination Function (DCF), which describes the default MAC protocol operation, is based on a scheme known as carrier-sense, multiple access, collision avoidance (CSMA/CA). Both the MAC and PHY layers cooperate in order to implement collision avoidance procedures. The PHY layer samples the received energy over the medium transmitting data and uses a clear channel assessment (CCA) algorithm to determine if the channel is clear. This is accomplished by measuring the RF energy at the antenna and determining the strength of the received signal commonly known as RSSI, or received signal strength indicator. In addition, carrier sense can be used to determine if the channel is available. This

technique is more selective since it veri es that the signal is the same carrier type as 802.11 transmitters. A virtual carrier sense mechanism is also provided at the MAC layer. It uses the request-to-send (RTS) and clear-to-send (CTS) message exchange to make predictions of future traf c on the medium and updates the network allocation vector (NAV) available in stations. Communication is established when one of the wireless nodes sends a short RTS frame. The receiving station issues a CTS frame that echoes the sender's address. If the CTS frame is not received, it is assumed that a collision occurred and the RTS process starts over. Regardless of whether the virtual carrier sense routine is used or not, the MAC is required to implement a basic access procedure as follows. If a station has data to send, it waits for the channel to be idle through the use of the CSMA/CA algorithm. If the medium is sensed idle for a period greater than a DCF interframe space (DIFS), the station goes into a backoff procedure before it sends its frame. Upon the successful reception of a frame, the destination station returns an ACK frame after a Short interframe space (SIFS). The backoff window is based on a random value uniformly distributed in the interval $[CW_{min}, CW_{max}]$, where CW_{min} and CW_{max} represents the Contention Window parameters. If the medium is determined busy at any time during the backoff slot, the backoff procedure is suspended. It is resumed after the medium has been idle for the duration of the DIFS period. If an ACK is not received within an ACK timeout interval, the station assumes that either the data frame or the ACK was lost and needs to retransmit its data frame by repeating the basic access procedure.

3. Interference in the 2.4 GHz Band

The 2.4 GHz ISM band allows for primary and secondary uses. Secondary uses are unlicensed but must follow rules de ned in the Federal Communications Commission Title 47 of the Code for Federal Regulations Part 15 [COM] relating to total radiated power and the use of the spread spectrum modulation schemes. Interference among the various uses is not addressed as long as the rules are followed. Thus, the major down side of the unlicensed ISM band is that frequencies must be shared and potential interference tolerated. While the spread spectrum and power rules are fairly effective in dealing with multiple users in the band, provided the radios are physically separated, the same is not true for close proximity radios. Multiple users, including self-interference of multiple users of the same application, have the effect of raising the noise oor in the band resulting in a degradation of performance. The impact of interference may be even more severe, when radios of different applications use the same band while located in close proximity.

Thus, the interference problem is characterized by a time and frequency overlap as depicted in Figure 1. In this case, a Bluetooth frequency hopping system occupying 1 MHz of the spectrum is shown to overlap with a WLAN Direct Sequence Spread Spectrum signal occupying a 22 MHz channel. Note that, the collision overlap time depends on the frequency hopping pattern and the traf c distribution of both the Bluetooth and WLAN systems.

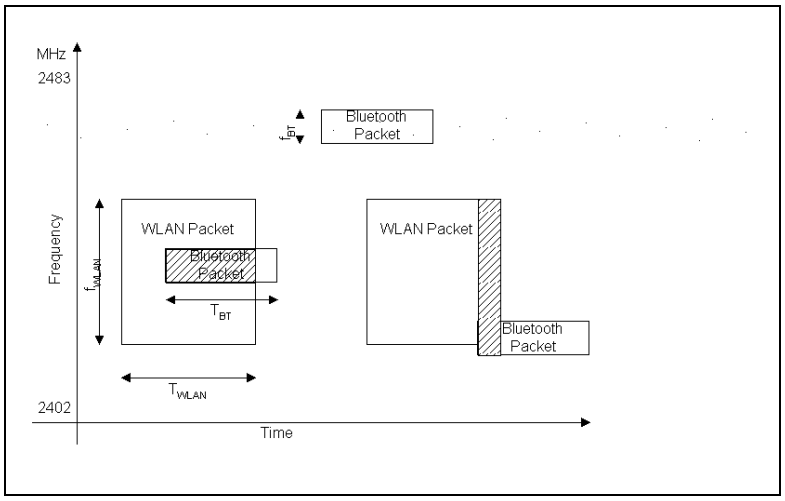


FIG. 1. *Time and Frequency Collisions in the 2.4 GHz Band*

Moreover, we can classify interferers into two classes based on their usage of the spectrum. Devices implementing the Direct Sequence Spread Spectrum (DSSS) technique constitute one class of interferer that utilize a fixed channel in the band. Typically this channel is 22 MHz wide, although the width of the signal depends on the transmitter's implementation. The second class of interferers is represented by devices implementing a type of Frequency Hopping (FH) mechanism. Note that the IEEE 802.11 specifications include a frequency hopping technique that uses a deterministic frequency pattern. On the other hand, the Bluetooth specifications define a pseudo-random frequency sequence based on the Bluetooth device address and its internal clock. While interference among systems from the same type such as Bluetooth on Bluetooth, or IEEE 802.11 on IEEE 802.11 interference can be significant, it is usually considered early on in the design stages of the protocol. Therefore, the worst realistic interference scenario consists of a mix of heterogeneous devices (i.e. devices belonging to different classes). Thus, most results published in the literature today focus on this worst case scenario.

Recently, there has been several attempts at quantifying the impact of interference on both the WLAN and the Bluetooth performance. Published results can be classified into at least three categories depending on whether they rely on analysis, simulation, or experimental measurements. Analytical results based on probability of packet collision were obtained by Shellhammer [S. 00a], Ennis [G. 98], and Zyren [J. 99] for the WLAN packet loss and by Golmie *et. al.* [N. 01b] for the Blutooth packet error. Although, these analytical results can often give a first order approximation on the impact of interference and the performance degradation (up to 25 % for Bluetooth packet loss and close to 70% for WLAN packet loss), they often make a number of assumptions concerning the traffic distributions and the operation of the media access protocol

which can make them less realistic. More importantly, in order for the analysis to be tractable, mutual interference that can change the traf c distribution for each system is often ignored. On the other hand, experimental results such as the ones obtained by Kamerman [A. 00], Howittt *et. al* [I. 01], and Fumolari [D. 01] can be considered more accurate at the cost of being too speci c to the implementation tested. Thus, a third alternative consists of using modeling and simulation to evaluate the impact of interference. This third approach can provide a more e xible framework. However, the accuracy of the results depends on the modeling assumptions made. Zurbes *et. al.* [S. 00b] present simulation results for a number of Bluetooth devices located in a single large room. They show that for 100 concurrent web sessions, performance is degraded by only ve percent. Golmie *et. al.* [N. 01c] use a detailed MAC and PHY simulation framework to evaluate the impact of interference. Similar results have been obtained by Lansford *et. al.* [J. 00a] who use simulation and experimental measurements to quantify the interference resulting from Bluetooth and IEEE 802.11. Their simulation models are based on a link budget analysis and a Q function calculation for the channel and PHY models respectively, in addition to the MAC layer behavior.

4. Coexistence Framework

Wireless system designers have always had to contend with interference from both natural sources and other users of the medium. Thus, the classical wireless communication design cycle has consisted of measuring or predicting channel impairments, choosing a modulation method, signal pre-conditioning at the transmitter, and processing at the receiver to reliably construct the transmitted information. However,in contrast to classical techniques to suppress interference such as modulation, channel coding, interleaving and equalization, most of the techniques proposed for solving the problem of interference in the 2.4 GHz band focus on adaptive non signal processing control strategies including power and frequency hopping control, and MAC parameter adjustments and scheduling.

In fact, the are a number of industry led activities focused on coexistence in the 2.4 GHz band. The IEEE 802.15.2 Coexistence Task Group was formed in order to evaluate the performance of Bluetooth devices interfering with WLAN devices and develop a model for coexistence which will consist of a set of recommended practices and possibly modi cations to the Bluetooth and the IEEE 802.11 standard speci cations [802] that allow the proper operation of these protocols in a cooperating way. At the same time, the Bluetooth Special Interest Group (SIG) formed its own task group on Coexistence. Both the Bluetooth and the IEEE working groups maintain liaison relations and are looking at similar techniques for alleviating the impact of interference. The proposals considered by the groups range from collaborative schemes intended for Bluetooth and IEEE 802.11 protocols to be implemented in the same device to fully independent solutions that rely on interference detection and estimation.

Collaborative Mechanisms-
Mechanisms for collaborative schemes have been proposed to the IEEE 802.15 Co-

existence Task Group and are based on a MAC time domain solution that alternates the transmission of Bluetooth and WLAN packets (assuming both protocols are implemented in the same device and use a common transmitter) [J. 00b]. A priority of access is given to Bluetooth for transmitting voice packets, while WLAN is given priority for transmitting data.

Non-Collaborative Mechanisms-
The non-collaborative mechanisms considered range from adaptive frequency hopping [B. 01] to packet scheduling and traf c control [N. 01a]. They all use similar techniques for detecting the presence of other devices in the band such measuring the bit or frame error rate, the signal strength or the signal to interference ratio (often implemented as the Received Signal Indicator Strength (RSSI)). For example, each device can maintain a bit error rate measurement per frequency used. Frequency hopping devices can then know which frequencies are occupied by other users of the band and thus modify their frequency hopping pattern. They can even choose not to transmit on a certain frequency if that frequency is occupied. The rst technique is known as adaptive frequency hopping, while the second technique is known as MAC scheduling. Each technique has advantages and disadvantages. One of the advantages in using a scheduling policy is that it does not require any changes in the FCC rules. In fact, title 47, part 15 of the FCC rules on radio frequency devices [COM], allows a frequency hopping system to recognize the presence of other users within the same spectrum band so that it adapts its hopsets to avoid hopping on occupied channels. Furthermore, scheduling in the Bluetooth speci cations is vendor implementation speci c. Therefore, one can easily implement a scheduling policy with the currently available Bluetooth chip set. On the other hand, adaptive frequency hopping requires changes to the Bluetooth hopping pattern and therefore a new Bluetooth chip set design. While both techniques can reduce the Bluetooth packet loss and the impact of interference on the other system, only the adaptive frequency hopping technique can increase the Bluetooth throughput by maximizing the spectrum usage.

Figure 2 illustrates the coexistence mechanisms space with respect to the duty cycle or the device activity and frequency band occupancy. As the number of interferers increase, each system is forced to transmit less often in order to avoid collisions. Thus, as the band occupancy increases, the duty cycle is reduced imposing time domain solutions. Frequency domain solutions such as adaptive frequency hopping can only be effective when the band occupancy is low.

5. Factors Impacting Interference

In this section we discuss different factors that may impact interference. Our discussion is based on performance results obtained from our detailed simulation modeling tool [N. 01c]. The example scenario that we use is based on a four node topology including two WLAN nodes (1 access point (AP) and one mobile device) and two Bluetooth nodes (1 master and 1 slave). Data is transmitted from the mobile WLAN node to the AP that responds with acknowledgement messages upon the successful

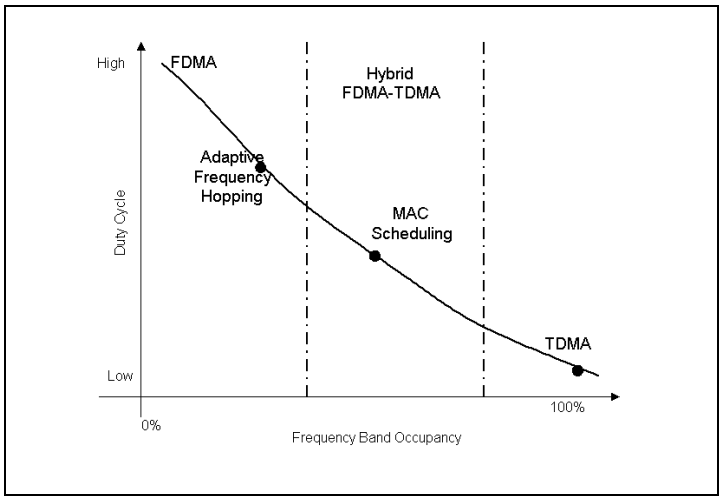


FIG. 2. *Coexistence Solution Space*

receipt of data packets. In order to better visualize the topology we can think of the placement of the four wireless devices on a two dimensional grid. The WLAN devices are located at (0,15) and (0,d) meters for the AP and mobile device respectively. The Bluetooth devices are placed at (0,0) and (1,0) meters for the slave and master device respectively. The transmitting power is set to 25 mW and 1 mW for WLAN and Bluetooth respectively. Statistics are collected at the Bluetooth slave device and the WLAN mobile node. Note that the distance between the WLAN mobile node and the Bluetooth slave is varied along the "y" coordinate axis. The WLAN traf c distribution is set as follows. The offered load is set to 50% of the channel capacity. The packet size is 8000 bits and the packet interarrival time is set to 1.86 ms. The con guration and system parameters are summarized in Table 1.

Choice of Bluetooth Voice Encapsulation-
Figure 3 illustrates the effect of chosing different packet encapsulation schemes for transmitting Bluetooth voice packets in an interference environment. The encapsulation varies from $HV1$ that use a 1/3 FEC rate and a $T_{SCO} = 2$, to $HV2$ that use a 2/3 FEC rate and a $T_{SCO} = 4$, and $HV3$ that use no FEC and a $T_{SCO} = 6$. Note that there is no difference in the total packet length between the different HV packets. From Figure 3(a), we observe that the choice of packet encapsulation does not impact the performance of Bluetooth, in other words the use of additional error correction does not improve performance. On the other hand, we note from Figure 3(b) that $HV3$ is "friendlier" to WLAN due to a longer T_{SCO} period.

FEC Ef ciency -
We use three types of Bluetooth packet encapsulations, namely, $DM1$, $DM3$, and $DM5$, that occupy 1, 3 and 5 slots respectively. The offered load for Bluetooth is set to 30% of the channel capacity which corresponds to a packet interarrival of 2.91 ms,

Simulation Parameters	**Values**
Propagation delay	5 μs/km
Length of simulation run	30 seconds
Bluetooth Parameters	**Values**
Transmitted Power	1 mW
Slave Coordinates	(0,0) meters
Master Coordinates	(1,0) meters
WLAN Parameters	
Packet Length	8000 bits
Packet Interarrival Time for 11 Mbits/s	1.86 ms
Transmitted Power	25 mW
AP Coordinates	(0,15) meters
Mobile Coordinates	(0,d) meters
Packet Header	224 bits
Slot Time	$2 * 10^{-5}$ seconds
SIFS Time	$1 * 10^{-5}$ seconds
DIFS Time	$5 * 10^{-5}$ seconds
CW_{min}	31
CW_{max}	1023
Fragmentation Threshold	None
RTS Threshold	None
Short Retry Limit	4
Long Retry Limit	7

TAB. 1. *Simulation Parameters*

8.75 ms and 14.58 ms for $DM1$, $DM3$ and $DM5$ packets respectively. In this case we note from Figure 4 that the use of FEC has limited bene ts and can only improve the performance of Bluetooth for low interference scenarios (i.e. for distances greater than 3 meters).

Effect of Fragmentation on the Interfering System-
Fragmentation or the transmission of short packets is a well documented technique to alleviate the impact of interference since a shorter packet has a lower probability of collision with an interfering system. However, Figure 5 shows that fragmentation may degrade the performance of the interfering system.

6. Concluding Remarks

In this paper we focus on the problem of interference in the 2.4 GHz unlicensed band. We rst de ne the problem and discuss some of the results previously published in the literature on the evaluation of interference. We then give an overview of

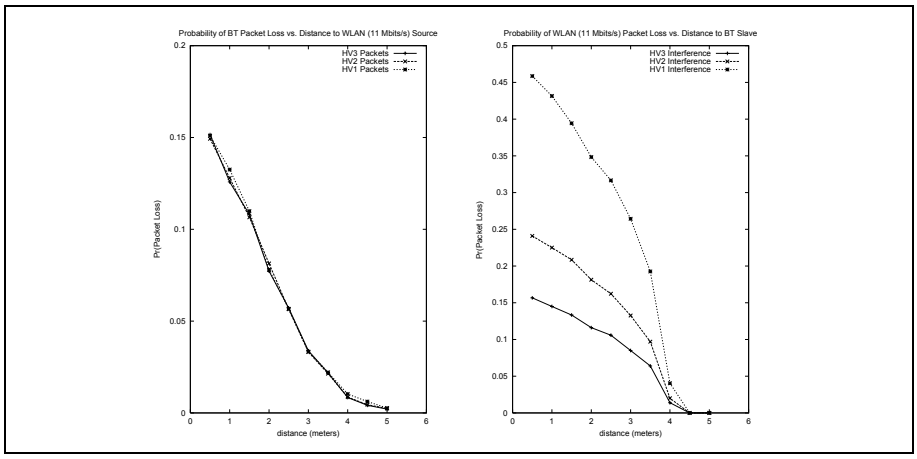


FIG. 3. (a) (b) *Bluetooth voice packets with 802.11 interference. (a) Probability of BT packet loss vs. distance to WLAN Source. (b) Probability of WLAN packet loss vs. distance to BT Slave*

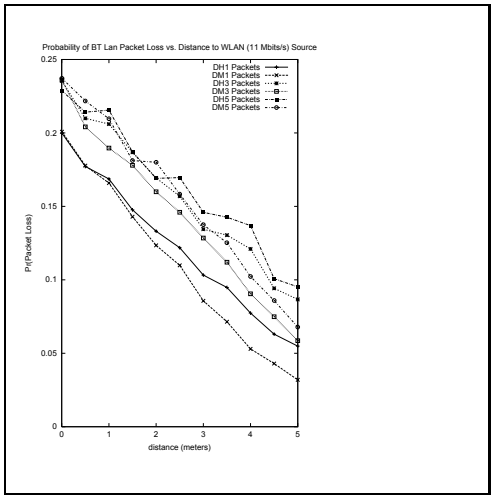


FIG. 4. *Probability of BT packet loss vs. distance to WLAN source*

the coexistence framework consisting of several techniques proposed to alleviate the impact of interference. Several factors that can impact the performance of Bluetooth and WLAN in an interfering environment are explored. We make several observations regarding the use of FEC, the choice of packet encapsulation and fragmentation and the effect on performance. Our results indicate that the use of FEC has limited bene- t for many interfering scenarios. In addition, applying fragmentation can reduce the

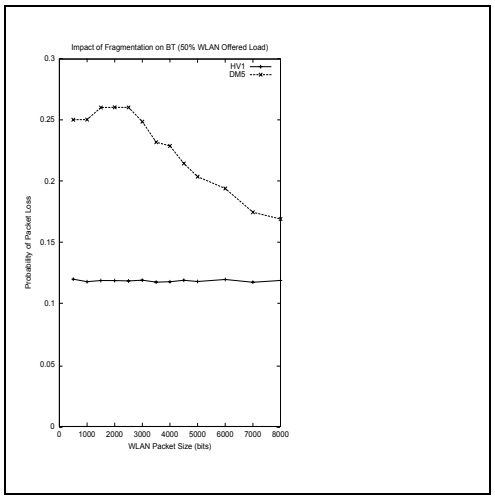


FIG. 5. *Probability of BT packet loss vs. distance to WLAN source*

probability of packet loss at the expense of causing more interference to the "other" system.

7. Bibliographie

[802] 802-11 I. S., « IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Speci cation », June 1997.

[A. 00] A. KAMERMAN, « Coexistence between Bluetooth and IEEE 802.11 CCK : Solutions to avoid mutual interference », *IEEE P802.11 Working Group Contribution, IEEE P802.11-00/162r0*, July 2000.

[ASS] ASSOCIATION I. D., « IrDA Advanced Infrared Physical Layer Speci cation, v. 1.0 », September 1998.

[B. 01] B. TREISTER, A. BATRA, K.C. CHEN, O. ELIEZER, « Adapative Frequency Hopping : A Non-Collaborative Coexistence Mechanism », *IEEE P802.15 Working Group Contribution, IEEE P802.15-01/252r0*, Orlando, FL, May 2001.

[COM] COMMISSION F. C., « Title 47, Code for Federal Regulations, Part 15 », October 1998.

[D. 01] D. FUMOLARI, « Link Performance of an Embedded Bluetooth Personal Area Network », *Proceedings of IEEE ICC'01*, Helsinki, Finland, June 2001.

[G. 98] G. ENNIS, « Impact of Bluetooth on 802.11 Direct Sequence », *IEEE P802.11 Working Group Contribution, IEEE P802.11-98/319*, September 1998.

[GRO a] GROUP B. S. I., « Speci cations of the Bluetooth System, vol. 1, v.1.0B 'Core' and vol. 2 v1.0B 'Pro les' », December 1999.

[GRO b] GROUP H. W., « HomeRF Shared Wireless Access Protocol Cordless Access (SWAP-CA) Speci cations », May 2000.

[I. 01] I. HOWITT, V. MITTER, J. GUTIERREZ, « Empirical Study for IEEE 802.11 and Bluetooth Interoperability », *in IEEE Vehicular Technology Conference (VTC), Spring 2001*, May 2001.

[J. 99] J. ZYREN, « Reliability of IEEE 802.11 WLANs in Presence of Bluetooth Radios », *IEEE P802.15 Working Group Contribution, IEEE P802.15-99/073r0*, Santa Rosa, California, September 1999.

[J. 00a] J. LANSFORD, R. NEVO, AND B. MONELLO, « Wi-Fi (802.11b) and Bluetooth Simultaneous Operation : Characterizing the Problem », *Mobilian White Paper*, www.mobilian.com, September 2000.

[J. 00b] J. LANSFORD, R. NEVO, E. ZEHAVI, « MEHTA : A method for coexistence between co-located 802.11b and Bluetooth systems », *IEEE P802.15 Working Group Contribution, IEEE P802.15-00/360r0*, November 2000.

[K. 00] K. J. NEGUS, A. P. STEPHENS, AND J. LANSFORD, « HomeRF : Wireless Networking for the Connected Home », *IEEE Personal Communications*, February 2000, p. 20-27.

[N. 01a] N. GOLMIE, « Interference Aware Bluetooth Scheduling Techniques », *IEEE P802.15 Working Group Contribution, IEEE P802.15-01/143r0*, Hilton Head, NC, March 2001.

[N. 01b] N. GOLMIE AND F. MOUVEAUX, « Interference in the 2.4 GHz ISM band : Impact on the Bluetooth access control performance », *Proceedings of IEEE ICC*, Helsinki, Finland, June 2001.

[N. 01c] N. GOLMIE, R. E. VAN DYCK, A. SOLTANIAN, « Interference of Bluetooth and IEEE 802.11 : Simulation Modeling and Performance Evaluation », *Proceedings of the Fourth ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWIM'01*, Rome, Italy, July 2001.

[S. 00a] S. SHELLHAMMER, « Packet Error Rate of an IEEE 802.11 WLAN in the Presence of Bluetooth », *IEEE P802.15 Working Group Contribution, IEEE P802.15-00/133r0*, Seattle, Washington, May 2000.

[S. 00b] S. ZURBES, W. STAHL, K. MATHEUS, AND J. HAARTSEN, « Radio network performance of bluetooth », *Proceedings of IEEE International Conference on Communications, ICC 2000*, vol. 3, New Orleans, LA, June 2000, p. 1563-1567.

INTERFERENCE MITIGATION TECHNIQUES

In preliminary work (recall Papers #5 and #6), NIST researchers eliminated from further consideration four possible interference-mitigation techniques. First, increasing WLAN power when needed to overcome periods of Bluetooth interference was found to be impractical because the necessary power increase would be too large. Second, using forward-error correction was found to have only limited benefits in selected interference scenarios. Third, fragmenting packets was found to create additional interference for other devices. Fourth, co-scheduling of WPAN and WLAN transmissions was found applicable only in the special circumstance where Bluetooth and 802.11 devices operate within the same computer node. On the other hand, several potential interference-mitigation techniques appeared worthy of further investigation. For WLAN devices, one might use rate scaling, where WLAN devices lower the transmission rate (for example, from 11 Mbps to 1 Mbps) during periods of Bluetooth interference. WLAN devices might also be re-engineered to reject Bluetooth interference at the PHY layer. For Bluetooth devices, one might adaptively adjust transmission power to a larger value during periods of WLAN interference, or adjust transmission patterns to avoid WLAN interference. In adjusting transmission patterns, one might develop a scheme that permits Bluetooth masters to periodically adapt the frequency-hopping schedule for a piconet to account for interference. In another approach to adjust transmission patterns, one might develop a scheme to delay selected transmissions during periods of interference, and then transmit the information when the interference subsides. These are the techniques studied in the papers contained in this section of the special publication.

In Paper #7, "Rejection of Bluetooth Interference in 802.11 LANs", Soltanian and Van Dyck investigate the use of complex adaptive filters for interference suppression in selected 802.11 systems. The fundamental approach studied would use recursive least-squares lattice filters to estimate and cancel Bluetooth interference. The paper focuses solely on the 1 Mbps version of 802.11 WLAN transmissions; however, argues that the technique might also be applicable in the 2 Mbps model. The simulation results presented in the paper show that WLAN packet-loss rates in range of 10-12% (for the standard 802.11 PHY) could be improved to a range of 1-4% (with the addition of a complex adaptive filter) even in the presence of two interfering Bluetooth piconets. The paper leaves for further study the design of a receiver for fading channels and for WLAN transmissions at higher speeds (5.5 and 11 Mbps). Adopting an adaptive filter would require redesign and deployment of PHY components for existing WLAN receivers. Changing the deployed base of WLAN devices would not occur quickly; thus, it makes sense to explore techniques that could be adopted more easily, perhaps with software or firmware upgrades.

In Paper #8, "Techniques to Improve Bluetooth Performance in Interference Environments", Golmie and Chevrollier investigate the possibility of overcoming interference to Bluetooth devices using either of two techniques: (1) Bluetooth power control or (2) Bluetooth scheduling. The paper presents simulation results based on a topology where a Bluetooth master and slave, separated by 1 meter, operate in proximity of a WLAN with a fixed access point about 15 meters from the Bluetooth devices and a mobile WLAN device (the interferer) that moves within a range of 0 to 5 meters from the Bluetooth devices. The proposed Bluetooth power-control algorithm periodically adjusts transmission power within a bounded range to achieve a target signal-to-interference ratio. The paper shows that adaptive power control (updating power levels after every 300 packets) reduces packet-loss rate from 18% to 4% when the WLAN interferer operates a ½ meter from the Bluetooth devices. Beyond ½ meter, the adaptive-power control algorithm loses no packets, while the standard Bluetooth transmission algorithm is still losing 3% of packets when the WLAN interferer is 5 meters away. The paper also shows that increasing Bluetooth transmission power increases interference for the proximal WLAN, at least within the range of 0 to 2 meters, where packet-loss rate for the WLAN ranges from 17% to 9%; thus, power control has limited benefits and is rarely a friendly-neighbor solution. These results suggest that increasing Bluetooth transmission power to overcome WLAN interference might have deleterious and unacceptable consequences for some WLAN transmissions; thus, the paper also studies a Bluetooth scheduling technique to avoid WLAN interference. The scheduling technique requires a Bluetooth master to acquire

and maintain (during an interference-estimation phase) a table estimating likely error rate on each of the 79 frequencies available in a Bluetooth channel. The Bluetooth master then delays initiating a specific master-slave interaction when a needed frequency pair is not available. While this technique seems likely to increase access delays, the paper finds this to be the case only for small (single-slot) packets, which experienced increased delays in the range of 1.6 ms to 2.6 ms. For longer (two- or three-slot) packets, access delays actually decreased by as much as 2.6 ms. The paper finds that using a Bluetooth scheduling algorithm reduces packet loss to zero for Bluetooth and WLAN devices. Though Bluetooth scheduling applies only to data traffic (voice traffic is time sensitive), the potential advantages of the approach warrant further investigation.

In Paper #9, "Interference Aware Bluetooth Packet Scheduling", Golmie, Chevrollier, and ElBakkouri further investigate the performance of the promising Bluetooth interference-aware scheduling (BIAS) algorithm first conceived in the previous paper. Here, the researchers provide a rigorous and complete description of the BIAS algorithm, including an explanation of how the approach could be implemented within the Bluetooth specification. The paper also introduces and proves two properties of BIAS: (1) error-free connections will be served at the negotiated rate and (2) error-free channels will be shared among error-free and error-prone devices proportional to their assigned rate. The researchers use simulation to compare the effectiveness of BIAS against a round-robin scheduling algorithm, given a specific mix of traffic: 50% offered load on an 11-Mpbs WLAN and 25% offered load on a competing Bluetooth WPAN. The simulation experiments model a topology where a Bluetooth master interacts with three Bluetooth slaves, while an interfering WLAN mobile device interacts with a fixed WLAN access point. The mobile WLAN device moves within a range of 0 to 11 meters of two of the Bluetooth slaves (the master is 3 meters further from the WLAN mobile). The third Bluetooth slave is only ½ meter from the master (in a direction away from the WLAN mobile). The paper shows that BIAS yields no packet losses for any Bluetooth slave at any distance from the WLAN interferer. BIAS does increase access delay (to ≈ 4 ms) over round robin scheduling (at ≈ 2 ms). The simulation results also show that BIAS provides fairer access than round robin, which cedes more access to the Bluetooth slave closest to the master (and farthest from the interfering WLAN). These promising results motivated the NIST researchers to investigate BIAS more fully.

In Paper #10, "Techniques to Improve the Performance of TCP in a mixed Bluetooth and WLAN Environment", Golmie and Rebala investigate the potential to mitigate the effects of mutual interference between a Bluetooth WPAN and 802.11 WLAN using either of two techniques: (1) WLAN rate scaling combined with adaptive filtering or (2) BIAS. The researchers also consider combining BIAS and rate scaling. The investigation simulates a topology where a Bluetooth master and slave (1-meter apart) interact, while a WLAN mobile (ranging over 0 to 10 meters from the Bluetooth devices) communicates with a fixed WLAN access point (located 15 meters from the Bluetooth devices). The paper reports three scenarios: (1) both WLAN and WPAN devices conduct file transfers, (2) WLAN devices conduct file transfers while WPAN devices surf the web, and (3) WLAN devices surf the web while WPAN devices conduct file transfers. All devices use TCP as the transport protocol. Packet-loss probability is reported for all scenarios, while TCP throughput and delay are reported for file transfer and web surfing, respectively. The simulation results indicate that BIAS improves performance (reduces packet loss to zero, increases throughput, and decreases delay) for both WPAN and WLAN systems. The improvement shown with rate scaling is much lower because the WLAN reduces transmission rate by an order of magnitude. Further, adaptive filtering is shown to improve performance for WLAN devices at the expense of degrading performance for WPAN devices.

In Paper #11, "Bluetooth Dynamic Scheduling and Interference Mitigation", Golmie more fully studies BIAS with larger topologies, including multiple Bluetooth piconets, with more diverse traffic types, including electronic mail, remote printing, video transmission, and file transfers, and with asymmetric packet lengths. The study also considers BIAS performance in reaction to dynamic changes in the wireless environment. In addition, the paper investigates two possible extensions: priority scheduling in BIAS and mapping quality of service to BIAS parameters. The paper reports results from four simulation experiments, where all of the experiments compare round robin scheduling for Bluetooth

devices against BIAS in the presence of one or more interfering WLAN systems, each operating at 11 Mbps with a 60% offered load. One experiment simulates a topology with a Bluetooth master and slave separated by 2 meters surrounded by three source-sink pairs of WLAN devices, where each source is about 1 meter from the Bluetooth devices and transmits to a sink about 14 meters past the Bluetooth devices. The topology is developed incrementally, starting with only the Bluetooth devices and adding one WLAN source-sink pair at a time; this presents the Bluetooth devices with dynamically increasing interference. Simulation with file transfers between the Bluetooth devices show no packet loss with BIAS as the Bluetooth offered load increases from 5% to 80%; this holds across all simulated configurations of competing WLAN devices. Round robin scheduling shows packet losses from 10% to 50% as the number of competing WLAN devices increases from one to three pairs. BIAS also shows significantly lower access delays than round robin when compared in similar interference environments. In a second experiment, Golmie studies the effects of dynamically changing interference. Here, the topology is drawn from the first experiment, but WLAN interference is limited to a singled source-sink pair that exhibits on-off periods. Under these conditions, BIAS yields packet-loss rates under 1/10% even with a 100% offered Bluetooth load, though access delay increases steeply after 50% offered load for small packets and after 70% offered load for large packets. In a third experiment, Golmie illustrates how BIAS can be used to enforce quality of service requirements for various Bluetooth applications. Here, the topology consists of a single Bluetooth master acting in a client role communicating with three Bluetooth slaves: a mail server (about ½ meter past the master), a video server, and a print server. The video and print servers are 2 meters closer than the master to a source of WLAN interference. The WLAN network consists of a fixed access point (file-transfer client) about 20 meters from the Bluetooth master and a mobile WLAN device (file-transfer server) that ranges from 2 to 13 meters from the Bluetooth video and print servers. BIAS keeps the packet-loss rate below ½% for all Bluetooth slaves under all circumstances, while round robin yields packet-loss rates ranging from 2% to 15% for the print and video servers. BIAS also provides Bluetooth slaves with access delays significantly superior to round robin scheduling. In a final experiment, Golmie investigates the use of BIAS given 10 Bluetooth piconets (half with master and slave separation of 1 meter and half 2 meters) randomly placed within 15 meters of an interfering WLAN, which can convey either file-transfer or web-surfing traffic. Some piconets carry voice traffic, some transfer files, and some transmit web surfing traffic. Simulation results show that BIAS (when compared to round robin scheduling) leads to the same or lower packet-loss rates for Bluetooth devices under all circumstances, with the advantage markedly increased when the interfering WLAN carries file-transfer traffic. Further, when Bluetooth devices use BIAS (instead of round robin scheduling), WLAN devices also see substantially lowered packet-loss rates. After extensively studying BIAS, NIST researchers decided to investigate the performance of BIAS against adaptive frequency hopping (AFH).

In Paper #12, “Bluetooth Adaptive Techniques to Mitigate Interference”, Golmie and Rebala investigate the relative performance of two interference-mitigation techniques (BIAS and AFH) for Bluetooth devices as interference from WLAN devices varies and user traffic changes over time. While many AFH algorithms can be conceived, Golmie and Rebala propose an algorithm that defines a frequency-hop window comprising a fraction of the 79 available Bluetooth frequencies. When a frequency in the hop window is “bad” then a “good” frequency is selected from among the remaining Bluetooth frequencies. The frequency-hop window is recomputed at some interval, which can be configured. The paper compares the performance of AFH against BIAS in four experiments. Two experiments simulate a (canonical) topology of a Bluetooth master and slave separated by 1 meter and a WLAN (11 Mbps) access point (15 meters from the Bluetooth devices) that communicates with a WLAN device that is 1 meter from the Bluetooth devices. The interfering WLAN operates with on-off periods and an offered load of 60%, while the Bluetooth offered load varies from 10% to 100%. BIAS provides significantly lower packet-loss rates compared to AFH. The researchers repeated the experiment, but this time including the TCP as the transport protocol and defining web-surfing and file-transfer traffic profiles. Again, BIAS showed significantly lower packet-loss rates and comparable access delays to AFH. In a third experiment, the researchers expand the topology to surround a Bluetooth master-slave piconet with three WLAN source-sink pairs. In this case, BIAS again provides lower packet-loss rates, but AFH

provides lower access delay. In a final experiment, the researchers expand the topology again to add two additional Bluetooth master-slave piconets amidst the WLAN interferers. BIAS again out performs AFH in terms of packet-loss rate, and also yields lower access delays for large packets. Overall, for the experiments reported here, AFH yields lower access delays for short (one-slot) packets, while BIAS provides better access delays for long (five-slot packets) and gives much lower packet-loss rates. AFH also shows improved throughput over BIAS in selected situations. The mixed results regarding BIAS and AFH stimulated NIST researchers to further investigate and compare the two interference-mitigation schemes.

In Paper #13, "Bluetooth Adaptive Frequency Hopping and Scheduling", Golmie, Rebala, and Chevrollier investigate the conditions (interference levels, topologies, scenarios, and applications) under which it should prove practical to use AFH and BIAS. Of special interest is studying how fast each algorithm can adjust to changes in the inference environment. The paper investigates AFH and BIAS in support of four applications: voice streaming, video streaming, web surfing, and file transfer. The researchers consider both the AFH defined for the IEEE 802.15 standard (AFH-IEEE) and the AFH that they defined (recall Paper #12). The first experiment considers the canonical topology described in Paper #12 under conditions where a WPAN file transfer competes with a WLAN file transfer. Simulation results show that BIAS provides much lower packet-loss rate, better channel efficiency, and similar access delay when compared to either AFH or AFH-IEEE. In a second experiment, the researchers expand the topology to include an additional competing WLAN source-sink pair, and consider Bluetooth file transfers, web surfing, and video and audio streaming competing against WLAN file transfers. Simulation results again show BIAS provides lower packet-loss rate, better channel efficiency, and similar access delay for all scenarios when compared with AFH and AFH-IEEE. Given these additional results, NIST researchers neared a final set of recommendations regarding interference-mitigation techniques for WLAN and WPAN systems.

In Paper #14, "Bluetooth and WLAN Coexistence: Challenges and Solutions", Golmie, Chevrollier, and Rebala discuss two possible solutions (AFH and BIAS) to mitigate WLAN and WPAN interference, providing some conclusions about the relative merits of each approach. AFH requires a Bluetooth master to collect from each slave a picture of relative interference, to compute a new hopping schedule, and to disseminate that schedule to each slave. This cyclic task might prove very difficult to achieve in dynamic environments with fast changing interference patterns. On the other hand, if AFH is practical for a given environment (where interference is not too volatile), then a number of performance benefits can be obtained. AFH can provide maximum throughput for bandwidth hungry applications, such as file transfers, and can give lower access delays for short packets, such as often used in voice applications. AFH does not provide much benefit for applications with delay-jitter requirements. BIAS provides superior performance for applications that are sensitive to packet loss. Overall, no single technique appears to yield the best performance for all interference scenarios and applications. The researchers conclude that combining BIAS and AFH might provide the flexibility to address a wider range of situations on a case-by-case basis.

Rejection of Bluetooth Interference in 802.11 WLANs

Amir Soltanian and Robert E. Van Dyck
National Institute of Standards and Technology
Gaithersburg, MD 20899
{amirs, vandyck}@antd.nist.gov

Abstract - We investigate the use of complex coefficient adaptive filters for interference suppression in the direct sequence spread spectrum 802.11 system. Extensive simulations are carried out to optimize the parameters of a recursive least-squares lattice filter, which mitigates the effect of a hopping narrow-band interferer such as Bluetooth. The BER curves, as well as the probability of packet loss, are presented. Moreover, the limitations of the adaptive filter approach, in the suppression of multiple jammers, is determined. It is shown that with a low order filter, the hopping jammer can be successfully suppressed.

I. Introduction

With the coming deployment of new wireless personal area networks in the 2.4 GHz ISM band, there is a growing concern about coexistence with existing systems, especially the IEEE 802.11b wireless local area network (WLAN). These systems can seriously damage the operation of the WLAN system [1]. For instance, the BT system occupies a 1 MHz bandwidth, and it has 79 hopping channels [2]. A direct sequence spread spectrum (DSSS) 802.11b system occupies approximately 22 MHz in the same band. Therefore, the BT system will consistently hop into the 802.11b spectrum, causing unintentional interference to both. Given the importance of coexistence between BT and 802.11, there has been considerable research on this topic. An overview of some of the proposed approaches is given in [3]. Most of these methods concentrate on changing the MAC layer behavior, such as by rescheduling packets or otherwise altering the traffic. Some of the approaches are collaborative, requiring a dual Bluetooth and 802.11 receiver.

This paper suggests a non-collaborative approach, based on signal processing in the physical layer. Specifically, we propose interference rejection for the DSSS WLAN using recursive least-squares lattice filters. Since the WLAN has no *a priori* knowledge of the timing or frequency used by the Bluetooth interference, it uses the adaptive filter to estimate and cancel the Bluetooth interference. While this paper focuses on the 1 Mb/s 802.11 mode, the suppression method is also applicable to the 2 Mb/sec QPSK DSSS mode. Moreover, the results can be extended from Bluetooth interference to other narrow-band networks such as HomeRF and the frequency-hopping IEEE 802.11 WLAN.

There has been much research in the area of interference suppression filters; for a recent review please see [4], *cf.* [5] [6] [7]. These papers, to our best knowledge, prominently consider a strong fixed frequency interferer in the bandwidth of the desired signal. There is not as much information in the open literature about the performance of these filters for a hopping jammer. Consequently, this paper studies a new application for these filters in a dynamic environment with hopping interference. In a wireless network environment, a long WLAN packet may be subject to multiple BT interferers, each with a different frequency offset and amplitude. For each interference, the adaptive filter coefficients have to change to compensate for the new conditions. For multiple simultaneous interferers, this is a challenging task. The lack of error correction in the WLAN packets adds to the difficulty, since even one bit error will lead to packet loss. Therefore, the interference suppression algorithm has to detect and cancel the interference in less than one bit interval. On the other hand, for short distance indoor applications, one can assume that the carrier-to-noise ratio (CNR) is high. This helps to speed up the convergence rate for some particular algorithms.

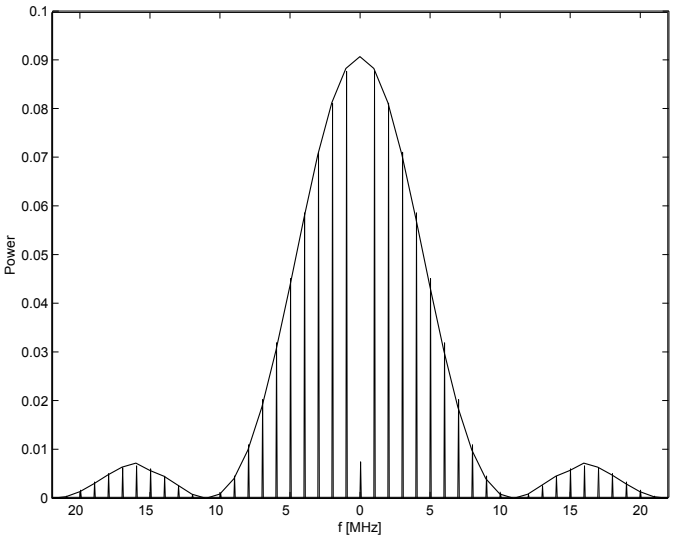


Fig. 1. Frequency spectrum of the Barker code; the frequencies are relative to the center frequency, which is labeled as 0 MHz. Note the notch of approximately 10 dB at the carrier frequency.

II. WLAN System Model

The 1 Mb/sec DSSS 802.11 WLAN employs DBPSK modulation, and each bit is spread by an 11 chip Barker code [8]. Fig. 1 shows the transmitted signal spectrum, which has a substantial notch at the carrier frequency; an analytical derivation of the spectrum is given by Lee and Miller [9]. As will be shown below, this notch is the reason

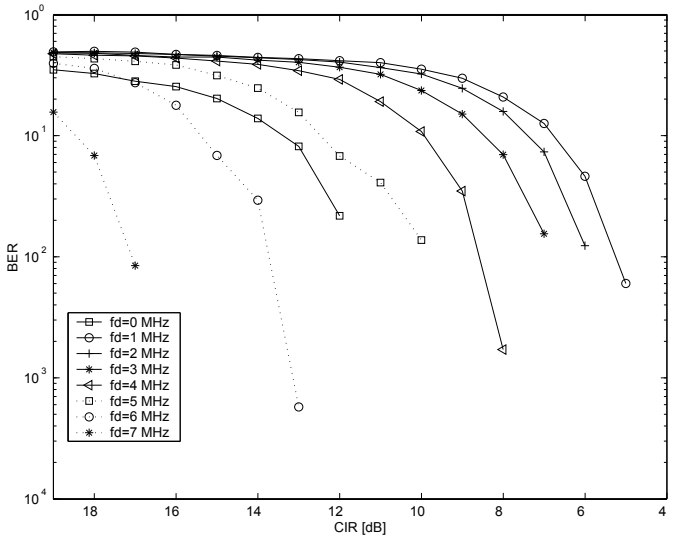


Fig. 2. Bit error rate performance of the 802.11 receiver in the presence of Bluetooth interference. CNR = 35 dB.

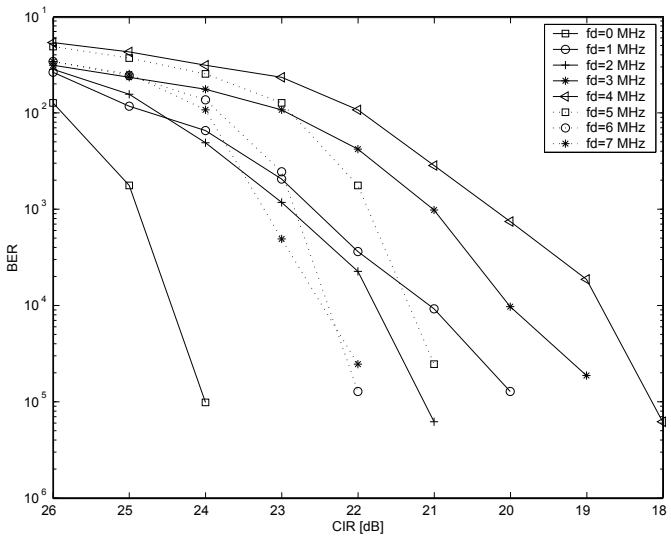


Fig. 3. RLSL algorithm. One BT interferer. CNR = 35 dB.

that the performance of the 802.11 receiver with Bluetooth interference at zero frequency offset is better than the performance with the interference at 1, 2, 3, or 4 MHz offsets.

The discrete input signal to the WLAN receiver, sampled once per chip period, T_c, can be expressed as

$$x(k) = s(k) + i(k) + n(k), \quad k = 1, 2, \cdots. \tag{1}$$

Here, $s(k)$ is the desired 1 Mb/sec DSSS WLAN signal with DBPSK modulation, $i(k)$ is the BT interference, and $n(k)$ is the additive white Gaussian noise. There are 11 complex samples/bit in the baseband signal, because of the 11 chip Barker code [1].

Fig. 2 depicts the BER curves for the above system, obtained using Monte Carlo simulation. We assume an interference-limited environment with $CNR = 35$ dB. In these plots, f_d is the frequency offset between the WLAN carrier and the BT interference. We suppose that the interference is always on and that it exists for the entire length of the WLAN packet. The carrier-to-interference ratio (CIR) is defined before the bandpass filter at the receiver. One sees that a CIR value less than -4 dB is capable of producing errors in the WLAN packet. The details of the simulation can be found in [1].

III. Adaptive Filter Results

While we have obtained good results using an adaptive filter based on the least-mean square (LMS) algorithm, we choose to focus on the recursive least-squares lattice (RLSL) filter. The RLSL algorithm requires more computational complexity than the LMS algorithm, but its fast convergence rate makes it attractive for interference suppression. This algorithm is described in [10], and for the sake of the brevity, it is not repeated here. The parameters of the algorithm are M, which is the order of the lattice filter, and λ, which is the forgetting factor. λ represents the memory of the algorithm, with $\lambda = 1$ corresponding to infinite memory. For one interferer, it was observed that even $M = 1$ would be enough to cancel the interference. However for multiple simultaneous interferers, increasing M to some extent will decrease the BER. We employ $M = 3$ and $\lambda = .97$ for these simulations.

Fig. 3 presents the results for one BT interferer. For $CNR = 35$ dB, one sees that the RLSL algorithm is capable of reducing the BER to below 10^{-2} for CIRs as low as almost -22 dB. It is worth noting that the interference at 3 and 4 MHz offsets is the most difficult to suppress, because the Bluetooth signal is right in the middle of the main lobe. A CIR of -20 dB gives a BER of 10^{-3}, a gain of almost 15 dB compared to the baseline case of Fig. 2. Even when the CNR is only 15 dB, the RLSL filter brings the BER below 10^{-3} for CIRs of -15 to -16 dB (no shown).

IV. Multiple BT Interferences

Fig. 4 illustrates the BER performance for two BT interferers. The abscissa shows the CIR of the second interferer, and the three curves are for different CIRS of the first interferer. Even if the CIR compared to the first interferer is -20 dB, a BER of 10^{-3} is achieved if the CIR compared to the second interferer is -10 dB.

If the number of time-overlapping BT interferers is more than two, it is still possible for adaptive algorithms to suppress all the interference at certain combinations of frequency offsets. For instance, for three equal power jammers, our simulations showed that at some frequency offsets the RLSL filter could achieve $BER \leq 10^{-3}$ at CIR

[1] In the simulations, 4 samples/chip are used to allow a frequency offset between the WLAN and BT without aliasing. However, the sampling rate is properly reduced by the receiver.

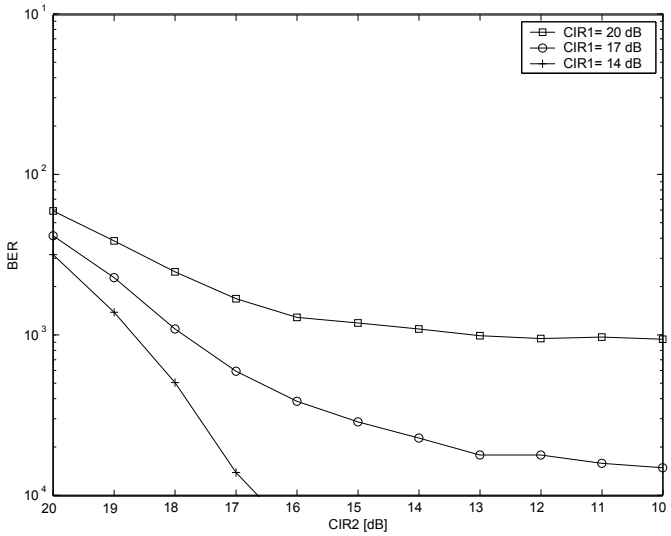


Fig. 4. BER for multiple Bluetooth interferers.

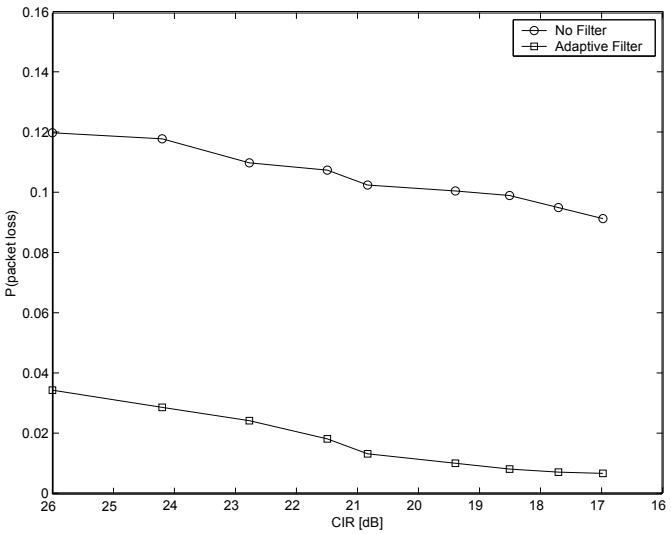


Fig. 5. Packet loss probabilities with and without the use of the adaptive interference suppression filter.

values as low as $CIR1 = CIR2 = CIR3 = -10$ dB. However, for all possible combinations of frequency offsets, the filter does not increase the performance that much unless the CIR is high enough, ($CIR = -4$ dB). This value is close to the minimum tolerable CIR value for the WLAN receiver with no interference rejection filter.

V. Packet Loss Probability

Perhaps of more interest is the effect that the interference suppression filter has on the probability of packet error. This MAC layer statistic is important, since packets with errors need to be retransmitted. Fig. 5 shows the loss probabilities for a Bluetooth interferer for two cases. The first case is for the baseline system, while the second case uses the RLSL adaptive filter. Over a large range of CIRs, the adaptive filter is able to reduce the loss rate from 10 to 12 percent to 1 to 3 percent, significantly improving the system throughput.

VI. Conclusions and Present Work

In this study, we show that an RLSL adaptive filter can be used in order to suppress a hopping Bluetooth interferer. For a single jammer, the algorithm performs quite well, and it can effectively cancel a hopping jammer at different frequency offsets. The RLSL algorithm substantially reduces the BER for two simultaneous jammers, and depending on the actual time and frequency overlap, the packet may be error free. Three simultaneous interferers is a more difficult problem, but even here the adaptive filter substantially improves the performance. Further results for these cases will be shown in the final paper.

There are two extensions to this work. The first one is to develop a receiver architecture, including the adaptive filter, that performs well for multi-path fading channels. Initial results, using a non-coherent combining RAKE receiver with an adaptive filter, are promising. The next extension of this work is to use an adaptive interference cancellation mechanism to mitigate the effect of Bluetooth on the 5.5 and 11 Mb/s 802.11b modes. Since these modes use complementary code keying (CCK), instead of DSSS, one must consider the combined effects of the interference suppression filter and the equalizer in the receiver design.

Acknowledgement

The authors would like to thank O. Rebala for running the packet loss simulations and N. Golmie for discussions about the MAC layer protocols.

References

[1] A. Soltanian and R. E. Van Dyck, "Physical layer performance for coexistence of Bluetooth and IEEE 802.11b ," *Proc. Virginia Tech. Symposium on Wireless Personal Communications*, pp. 31-41, Blacksburg, VA, June, 2001.

[2] Bluetooth Special Interest Group, *Specifications of the Bluetooth System, vol. 1, v.1.0B*, Dec. 1999. Available : http:/www.bluetooth.com.

[3] J. Lansford, A. Stephens, and R. Neve, "Wi-Fi (802.11b) and Bluetooth: Enabling Coexistence," *IEEE Network Mag.*, vol. 15, pp. 20-27, Sept.-Oct. 2001.

[4] J. D. Laster and J. H. Reed, "Interference rejection in digital wireless comm.," *IEEE Sig. Proc. Mag.*, pp. 37-67, May 1997.

[5] J. R. Zeidler, E. H. Satorious, and D. M. Chabries, "Adaptive enhancement of multiple sinusoids in uncorrelated noise," *IEEE Trans. Acoust., Speech, Signal Proc.*, pp. 240-254, June 1978.

[6] L. Li and L. B. Milstein, "Rejection of narrow-band interference in PN spread-spectrum systems using transversal filters," *IEEE Trans. Comm.*, pp. 925-928, May 1982.

[7] L. Li and L. B. Milstein, "Rejection of pulsed CW interference in PN spread-spectrum systems using complex adaptive filters," *IEEE Trans. Comm.*, pp. 10-20, Jan. 1983.

[8] IEEE Std. 802-11, *IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, 2001 Edition.

[9] J. S. Lee and L. E. Miller, *CDMA Engineering Handbook*, Artech House, 1998.

[10] S. Haykin, *Adaptive Filter Theory*, Prentice Hall, 1986.

Techniques to Improve Blueooth Performance in Interference Environments

Nada Golmie and Nicolas Chevrollier
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
Email: nada.golmie@nist.gov

Abstract—*Bluetooth is a radio technology for Wireless Personal Area Networks operating in the 2.4 GHz ISM band. Since both Bluetooth and IEEE 802.11 devices use the same frequency band and may likely come together in a laptop or may be close together at a desktop, interference may lead to signi cant performance degradation. The main goal of this paper is to propose solutions to the interference problem consisting of power control adjustments and scheduling policies to be implemented by the Bluetooth device. Simulation results are given for selected scenarios and con gur ations of interest.*

Keywords—**Bluetooth, Interference, Power Control, MAC scheduling**

I. Introduction

The Bluetooth [1] technology is an emerging short range cable replacement protocol operating in the 2.4 GHz ISM band. Since both the Bluetooth and the IEEE 802.11 [2] protocols operate in the 2.4 GHz, it is anticipated that interference may severely degrade the performance of both systems.

Our goal is to propose solutions to the interference problem pertaining to the Bluetooth radio operating in proximity to an IEEE 802.11 network. We assume that the source of interference to the Bluetooth system is an IEEE 802.11 system operating in a direct sequence spread spectrum (DSSS) mode. In the rest of this sequel, the terms IEEE 802.11 DSSS and WLAN will be used interchangeably.

We investigate two techniques aimed at alleviating the interference problem for Bluetooth. One technique is based on controlling the transmitted power and keeping it proportional to the signal-to-interference ratio (SIR) measured at the receiver. The other technique takes advantage of the frequency hopping sequence of Bluetooth and uses scheduling with the aim of avoiding interference. Simulation results for scenarios of interest are discussed. Performance is measured in terms of the mean access delay, the probability of packet loss, and the transmitted power.

This paper is organized as follows. In sections II and III, we describe the distributed power control algorithm and the scheduling mechanism respectively and give numerical results. Concluding remarks are offered in section IV.

II. Power Control

Given that some devices provide the ability to dynamically modify their transmission power, we would like to investigate the dynamics of a power control (PC) strategy as a means of alleviating the impact of interference.

We use a distributed algorithm to implement a PC procedure. The basic idea is to adjust the interference level in the system to no more than what is needed. We assume that the receiver does not have any knowledge of other systems except for the system it is communicating with. Interference from other systems is measured in terms of the SIR level at the receiver. Note that SIR is a wide-spread link quality measure and has been used in many previous studies for power control and dynamic channel allocation for interference limited systems [3] [4] [5]. The power update algorithm works as follows. Initially, $P_0 = P_{max}$, then every update interval U, the power at the transmitter, $P(t+1)$ is updated as follows:

$$P(t+1) = \min(P_{max}, \max(P_{min}, \frac{\tau_t}{SIR(t)} \times P(t)) \quad (1)$$

where $\tau(t)$ is the target SIR and SIR(t) is based on an average value over many measurements. The power update rule takes into consideration the SIR(t) statistic measured at the receiver side. The receiver can then relay this information to the transmitter every update interval U.

Implementation Considerations Although the exact details of a power control algorithm have been left unde ned for the most part, the Bluetooth speci cations have included the necessary hooks in the protocol in order to implement a power control algorithm. Furthermore, the Bluetooth speci cations classi es devices into three power classes as summarized in Table I

TABLE I
Bluetooth Device Power Classes

Power Class	Maximum Output Power	Minimum Output Power
1	100 mW (20 dBm)	1 mW (0 dBm)
2	2.5 mW (4 dBm)	0.25 mW (-6 dBm)
3	1 mW (0 dBm)	N/A

Class 1 requires power control limiting the transmitted power over 0 dBm, while power control is optional for classes 2 and 3. The speci cations suggest that the transmitted power should be adjusted based on the received signal strength indicator (RSSI) measurements at the receiver. Note that in an interference-limited environment, RSSI corresponds to the the SIR (assuming that noise can be neglected). Furthermore, the speci cations de ne Link Management Protocol (LMP) messages for adjusting the power control as shown in Table II. The general format of a Link Manager Protocol (LMP) message is illustrated in Figure 1.

Both LMP messages, $LMP\_incr\_pow\_req$ and $LMP\_decr\_pow\_req$, include one byte of contents reserved for future use. We propose using this byte to transmit the measured

TABLE II
LMP POWER CONTROL MESSAGES

Message	Op_code	Contents
LMP_incr_pow_req	31	1 byte- future use
LMP_decr_pow_req	32	1 byte- future use
LMP_max_power	33	1 byte
LMP_min_power	34	1 byte

SIR at the receiver in order for the transmitter to implement the update rule given by Equation 1.

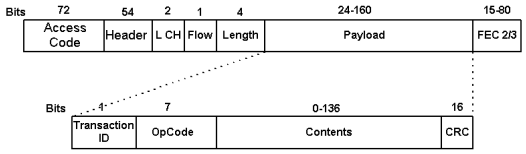


Fig. 1. LMP Message Format

Annother implementation issue to consider is the value of the update interval, U. Andersin *et al.* [6] demonstrate that for a system such as GSM, the SIR can be accurately estimated within 0.1 to 0.3 seconds. The values are for heavily interfered system with an interference level 20 dB above the noise oor . In our case, the value of SIR depends on the main signal and the interference spectral shape (i.e. whether the main signal falls inside or outside of the interfering signal band). Therefore, given 79 frequency channels, U can be chosen proportionally to 4 or 5 times 79. There is a trade-off between the value of U and the amount of signaling traf c required. A small value for U allows the system to be perhaps more responsive at the cost of having to exchange additional signaling information.

Numerical Results

We present simulation results to evaluate the effect of the power control algorithm. We use a 4-node topology as illustrated in Figure 2, and the simulation parameters presented in Table III. We vary the traf c distributions for WLAN and Bluetooth as follows.

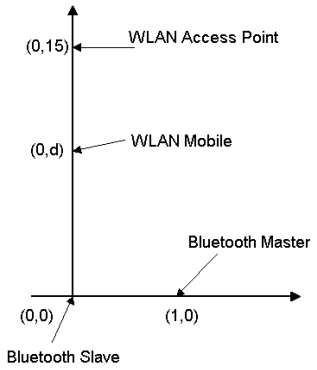


Fig. 2. Experiment Topology

We assume that the WLAN Mobile device is transmitting data packets to the AP device which is responding with ACKs. The WLAN packet payload is set to 7776 bits transmitted at 11 Mbits/s, while the packet header is set to 224 bits transmitted at 1 Mbits/s. We assume that the WLAN packet interarrival rate is exponentially distributed with a mean of 1.86 ms corresponding to 50% of the offered load. For Bluetooth, we assume that both master and slave devices are transmitting DM1 packets with a mean arrival rate of λ where $\lambda = \frac{2*0.000625}{l} - 2 * 0.000625$ seconds, and $l = 30$ is the offered load in percent of the channel capacity. Our setup parameters are summarized in Table III. We measure the probability of packet loss and the mean access delay measured at the Bluetooth slave device.

TABLE III
ADAPTIVE POWER SIMULATION PARAMETERS

Simulation Parameters	**Values**
Update Interval (U)	300 packets
Bluetooth Parameters	**Values**
ACL Baseband Packet Encapsulation	DM1, DM3, DM5
Packet Interarrival Time for DM1	2.91 ms
Packet Interarrival Time for DM3	8.75 ms
Packet Interarrival Time for DM5	15.58 ms
P_{min}	1 mW
P_{max}	100 mW
Slave Coordinates	(0,0)
Master Coordinates	(1,0)
WLAN Parameters	
Packet Interarrival Time	1.86 ms
Offered Load	50 % of Channel Capacity
Transmitted Power	25 mW
Data Rate	11 Mbits/s
AP Coordinates	(0,15)
Mobile Coordinates	(0,d)
Packet Header	224 bits
Payload Size	7776 bits

The power update rule given by Equation 1 was implemented at the Bluetooth master and slave devices. Initially, the power was set to $P_{max} = 100$ mW, then updated according to the rule. SIR was measured over an update interval, U, equal to 300 packets. Figure 3 shows the transmitted power (after 5 U) for the Bluetooth master device versus the distance of Bluetooth slave from the source of interference. Note that if there is no change in the interference signal, the transmitted power should converge to its nal value in one step, i.e. 1 U.

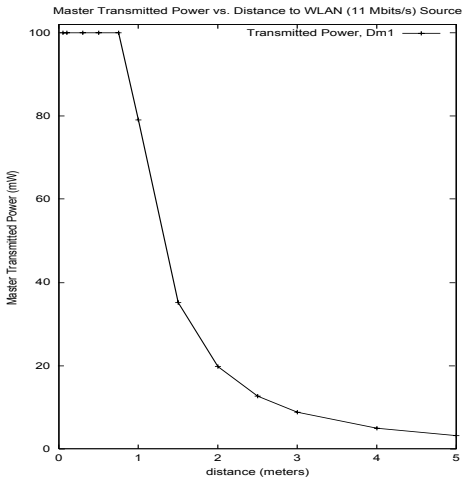


Fig. 3. Bluetooth Transmitted Power

As expected the transmitted power in Figure 3 varies between P_{max} and P_{min}. Figure 4 (a) and (b) give the packet loss and the access delay respectively with and without the power

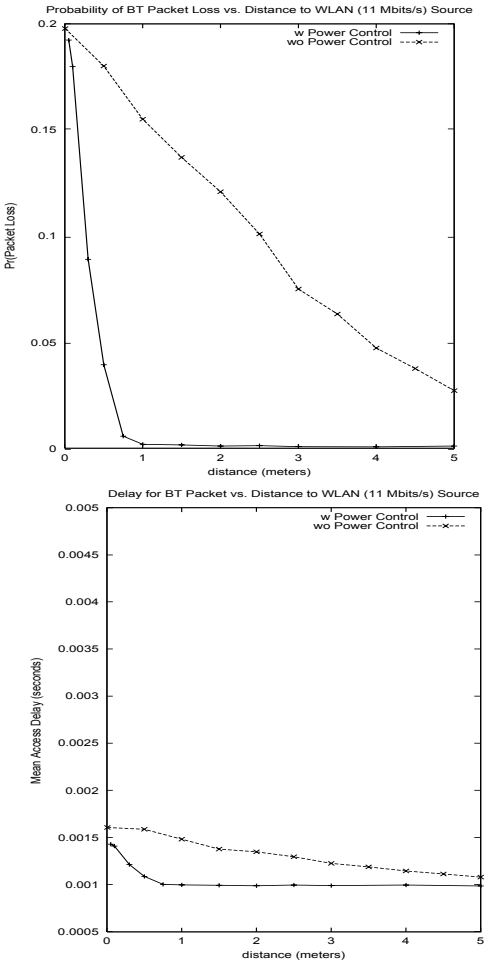


Fig. 4. $\frac{(a)}{(b)}$ Effect of Adaptive Power Control on Bluetooth Performance. (a) Probability of Packet Error vs. Distance. (b) Mean Access Delay vs. Distance.

control algorithm. Note that the WLAN transmitted power is x ed at 25 mW. For distances equal to 0.5 m from the interference source, increasing the transmitted power leads to lower packet losses, ≈ 4 % with power control instead of 18% without power control. A similar reasoning applies to the delays shown in Figure 4(b). However for distances less than 0.5m, the transmitted power is capped by P_{max} and the packet loss remains higher than $\approx 9\%$. A couple of observations are in order. We note that the power control algorithm can be effective in some scenarios; in the case studied here, lower packet losses and access delays are obtained for distances greater than 0.5m from the interference source. However, it should be made clear that this performance gain comes at the cost of increasing the interference level for other systems. As expected, increasing the Bluetooth transmitted power, has a negative impact on the interfering system; in Figure 5 we note a 17% packet loss at the WLAN AP device, even if it is about 15 meters away from the Bluetooth devices. As the Bluetooth transmitted power is weakened, the packet loss at the WLAN AP device drops to zero.

In a way, adjusting the power control can only be a partial solution. This may or may not constitute a problem for other systems depending on the con guration and the parameters used.

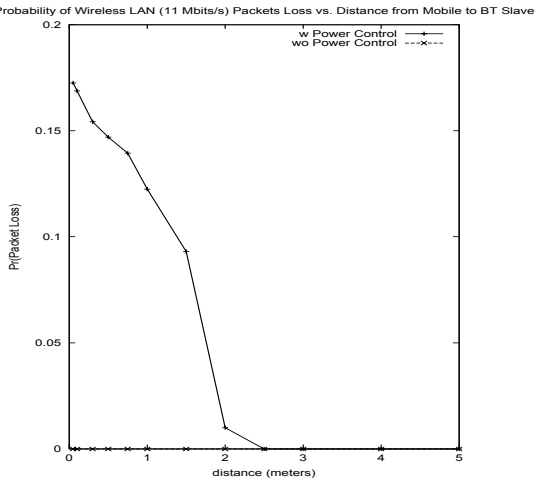


Fig. 5. Impact on the WLAN AP Device

III. MAC Scheduling

In this section, we investigate how scheduling techniques can be used to alleviate the impact of interference. We devise a mechanism for the Bluetooth MAC scheduler consisting of two components:

1. *Interference Estimation*

2. *Master Delay Policy*

In the *Interference Estimation* phase, the Bluetooth device detects the presence of an interfering device occupying a number of frequencies in the band. In this sequel, interfering devices are assumed to be WLAN DSSS systems. In order to detect the presence of interference, the Bluetooth device maintains a *Frequency Usage Table* where a bit error rate measurement, BER_f, is associated to each frequency as shown in Figure 6. Note that, a frame error rate or a packet loss measure can be used instead of the bit error rate (BER). Frequencies are classi ed according to a criteria that measures the level of interference in the channel and marked *used* or *unused* depending on whether their corresponding BER is above or below a threshold value, BER^T, respectively. This *Frequency Usage Table* is maintained at each receiver's side for both master and slave devices.

Use	Frequency Offset	BER_f
✓	0	10^{-3}
✗	1	10^{-1}
✗	2	10^{-2}
✗	3	10^{-1}
	...	
✓	76	10^{-4}
✓	77	10^{-3}
✓	78	10^{-3}

Fig. 6. Frequency Usage Table

The *Master Delay Policy* makes use of the measurements collected during the *Interference Estimation* phase in order to avoid

a packet transmission in a "bad" receiving channel, or a channel with a high level of interference. The basic idea is to "wait" for or choose an *unused* frequency for the receiver in the frequency hopping pattern. Thus the transmitter needs to consult the receiver's *Frequency Usage Table* before transmitting any packets. Alternatively, the receiver, can send status updates on its usage table to the transmitter. In Bluetooth, since the master device controls all transmissions in the piconet, the delay rule has to be implemented only in the master device. Furthermore, since following each master's transmission, there is a slave transmission, the master checks both the slave's receiving frequency and its own receiving frequency before chosing to transmit a packet in a given frequency hop as illustrated in Figure 7.

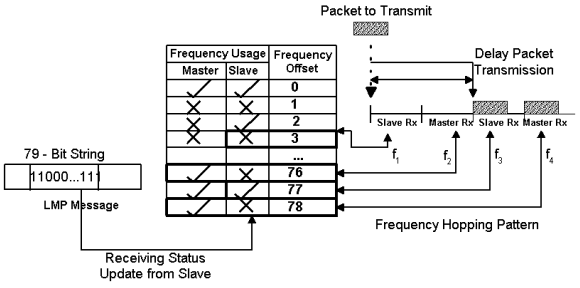


Fig. 7. Delay Scheduling Policy at Bluetooth Master

The main steps of the scheduling policy are summarized as follows.

1. Slave's End.
   (a) For every packet received, update BER_f which is an average value of the BER per frequency.
   (b) Every update interval, U, refresh the *Frequency Usage Table* by marking the frequencies, and
   (c) Send a status update message to the Master;
2. Master's End.
   (a) For every packet received, update BER_f which is an average value of the BER per frequency.
   (b) Every update interval, U, refresh the *Frequency Usage Table*, and
   (c) Before sending a packet, check slave's receiving frequency and master's following receiving frequency, delay transmission until both master and slave's receiving frequencies are available.

Implementation Considerations One of the advantages in using this scheduling policy is that it does not require any changes in the FCC rules. In fact, title 47, part 15 of the FCC rules on radio frequency devices [7], allows a frequency hopping system to recognize the presence of other users within the same spectrum band so that it adapts its hopsets to avoid hopping on occupied channels. However, coordination among hopping frequency systems in order to avoid simultaneous channel occupancy is not allowed.

Furthermore, scheduling in the Bluetooth speci cations is vendor implementation speci c. Therefore, one can easily implement a scheduling policy with the currently available Bluetooth chip set. Most importantly, the proposed scheduling algorithm does not require any changes to the Bluetooth frequency hopping pattern which is implemented in ASICs, and devices implementing scheduling can easily interoperate with other devices that do not.

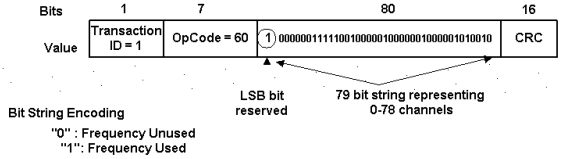


Fig. 8. LMP Interference Status PDU

As far as the status update message is concerned, we de ne an $LMP\_Interference\_Status$ PDU as shown in Figure 8. We use an Op_code value of 60 and set the $Transition\ ID$ to 1 in order to indicate that the message is sent from the slave to the master. The content eld uses 10 bytes to encode the slave's *Frequency Usage Table*. In fact, we reserve one bit for future use, and map the 79 channels in the *Frequency Usage Table* to a 79-bit string of 0's and 1's indicating the *used* and *unused* receiving frequencies respectively.

Numerical Results We simulate our proposed scheduling policy. We use the simulation environment, network topology and parameters described in section II. We use three types of Bluetooth packet encapsulations, namely, $DM1$, $DM3$, and $DM5$, that occupy 1, 3 and 5 slots respectively. The offered load for Bluetooth is set to 30% of the channel capacity which corresponds to a packet interarrival of 2.91 ms, 8.75 ms and 14.58 ms for $DM1$, $DM3$ and $DM5$ packets respectively. The transmitted power for Bluetooth and WLAN is x ed at 1mW and 25 mW respectively. Simulation parameters are summarized in Table III. Figure 9 (a) and (b) gives the packet loss and the mean access delay measured at the Bluetooth slave for varying distances of the interference source from the Bluetooth receiver. From Figure 9 (a) we observe that using the scheduling policy, leads to a packet loss of zero. We are basically able to avoid the channels occupied by the interfering system. When no scheduling policy is used the packet loss is $\sim 24\%$ for $DM5$, and $DM3$, and 19% for and $DM1$ packets respectively when the Bluetooth receiver is at a distance of 0.005 meters from the interference source. As the distance from the interference source is increased the packet loss drops to around 2.7% for $DM1$ packets. It is still around 6.7% for $DM3$ and $DM5$ packets.

For $DM1$, we observe an increase in delay from 1.6ms to 2.6ms when the scheduling policy is applied. On average the scheduling policy yields to a delay increase of 1ms (~ 1.6 Bluetooth slots). On the other hand, the scheduling policy reduces the delays by 0.8 ms and 2.6 ms for $DM3$ and $DM5$ respectively. Thus, delaying transmission to avoid bad channels pays off for packets occupying more than one slot. Note that, when bad channels are used, packets are dropped and have to be retransmitted which yields large delays. This effect does not apply to $DM1$ packets since they occupy only one slot.

In summary, we note that the scheduling policy is effective in reducing packet loss and delay (especially for multi-slot Bluetooth packets). Another advantage worth mentioning, are the additional savings in the transmitted power since packets are not transmitted when the channel is bad. Moreover, we note that by avoiding channels occupied by other devices, we elimi-

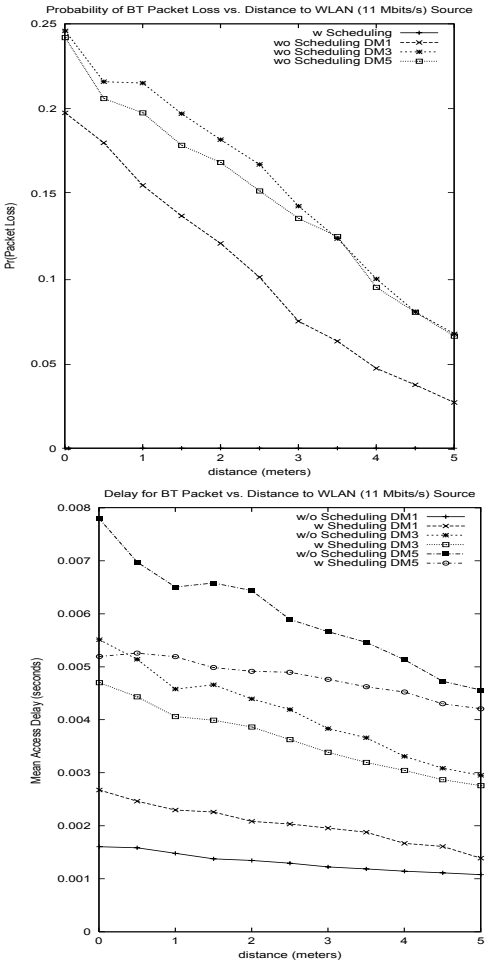


Fig. 9. $\frac{(a)}{(b)}$ Effect of MAC Scheduling on Bluetooth Performance. (a) Probability of Packet Error vs. Distance. (b) Mean Access Delay vs. Distance

nate interference on the other system sharing the same spectrum band. Figure 10 shows the packet loss for the WLAN Mobile device (receiving ACKs). We note that scheduling reduces the ACK packet loss to zero. Therefore scheduling can be considered as a neighbor friendly policy. Note that the packet loss at the WLAN AP located at (0,15) m is negligible in this case since the Bluetooth signal is too weak.

Finally, we note that scheduling policy proposed here works only with data traf c since voice packets need to be sent at x ed intervals. However, if the delay variance is constant and the delay can be limited to a slot (as was shown here), it may be worthwhile to use DM packets for voice using the same scheduling technique proposed here. This will constitute the basis of future work.

IV. Concluding Remarks

In this paper we explored two techniques for alleviating the impact of interference on the Bluetooth performance. While the power control approach may be useful and simple to use in some limited scenarios, it can only be a partial solution and thus can not be considered by itself. Our plan is to test the dynamics of the power control algorithm simultaneously on both the WLAN and the Bluetooth systems in order to gain additional insights

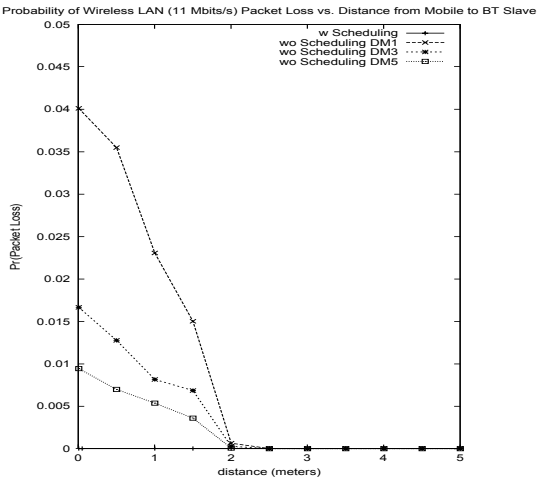


Fig. 10. Impact of MAC Scheduling on the WLAN Mobile Device

on its strengths and limitations in the context of interference.

Conversely, our simulation results indicate that the simple scheduling technique that we propose to delay the transmission of Bluetooth data packet once interference is detected can signi cantly lower the probability of packet loss for Bluetooth without much increase in the mean access delay.

The performance evaluation results obtained for the Bluetooth ACL link seems to be promising. We are currently looking at additional scenarios, and traf c conditions. We are also investigating the use of combined approaches such as packet encapsulation, scheduling, and ARQ o w control. Other future directions consist of exploring the interoperation of the coexistence techniques developed for Bluetooth and WLAN in dynamically changing environments in order to unravel their strengths and limitations.

References

[1] Bluetooth Special Interest Group, "Speci cations of the Bluetooth System, vol. 1, v.1.0B 'Core' and vol. 2 v1.0B 'Pro les', " December 1999.

[2] IEEE Std. 802-11, "IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Speci cation ," June 1997.

[3] G. Foschini, Z. Miljanic, "A Simple Distributed Autonomous Power Control Algorithm and its Convergence ," in *IEEE Journal on Vehicular Technology*, November 1993, vol. 42.

[4] J. Zander , "Distributed Cochannel Interference Control in Cellular Radio Systems," in *IEEE Journal on Vehicular Technology*, February 1992, vol. 41.

[5] Bambos, N., "Toward power-sensitive network architectures in wireless communications: concepts, issues, and design aspects," in *IEEE Personal Communications*, June 1998, vol. 5, pp. 50–59.

[6] M. Andersin, N. Mandayan, J. Zander, "A Subspace based Estimation of the Signal to Interference Ratio for TDMA Cellular Systems," in *IEEE VTC'96*, Atlanta, GA, April 1996.

[7] Federal Communications Commission, "Title 47, Code for Federal Regulations, Part 15," October 1998.

Interference Aware Bluetooth Packet Scheduling

N. Golmie, N. Chevrollier and I. ElBakkouri
National Institute of Standards and Technology
Gaithersburg, Maryland 20899

Abstract— **Bluetooth is a radio technology for Wireless Personal Area Networks operating in the 2.4 GHz ISM band. Since both Bluetooth and IEEE 802.11 devices use the same frequency band and may likely come together in a laptop or may be close together at a desktop, interference may lead to signi cant performance degradation. The main goal of this paper is to propose a scheduling algorithm aimed at reducing the impact of interference. This algorithm takes advantage of the fact that devices in the same piconet will not be subject to the same levels of interference on all channels of the band. The basic idea is to utilize the Bluetooth frequency hopping pattern and distribute channels to devices such that to maximize their throughput while ensuring fairness of access among users. Simulation results are given for selected scenarios and con gurations of interest.**

Keywords— **WPANs, Bluetooth, Interference, max-min fairness, scheduling.**

I. Introduction

An important requirement in the design of a scheduling mechanism for the Bluetooth technology is the support of heterogeneous traf c, a mix of voice and data applications such as email, ftp, remote login, and video with a wide range of delay, packet loss and throughput constraints.

Another key challenge in the design of a Bluetooth scheduling algorithm is probably the adaptiveness to a noisy environment. Today most radio technologies considered by Wireless Personal Area Network (WPAN) industry consortia and standard groups including the Bluetooth Special Interest Group [1], HomeRF [2], and the IEEE 802.15, employ the 2.4 GHz ISM frequency band. In addition both WPANs and Wireless Local Area Network (WLAN) devices implementing the IEEE 802.11 standard speci cations [3] will be sharing the same frequency band. Thus, WLAN devices operating in proximity to Bluetooth devices can signi cantly impact the performance of Bluetooth devices and vice versa as shown in [4][5][6].

Our goal in this paper is to propose a fair packet scheduling algorithm for Bluetooth that reduces the impact of interference. In Bluetooth, the master device controls both *downstream* (master-to-slave), and *upstream* (slave-to-master) traf c directions. The master can use odd numbered slots to send data *downstream* while slaves have to wait to be "polled" by the master in order to send data *upstream* in even numbered slots. Although in this paper, we do not make a speci c distinction between *upstream* and *downstream* traf c, we focus on a scheduling policy for polling slaves in order to allow them to access the channel. However our strategy or a similar policy can be used for either traf c directions. Furthermore, we assume that the source of interference to the Bluetooth system is an IEEE 802.11 system operating in a Direct Sequence Spread Spectrum (DSSS) mode. Note that our technique can be adapted to any other interference environment as well.

Recently, the issue of meeting different quality of service requirements in a wireless environment has been receiving more attention in the literature.

Fragouli, et. al. [7] proposed a strategy that combines class-based queuing [8] with channel-state based scheduling [9] that eliminates the Head of Line problem caused by FIFO queuing when certain devices suffer from a bad link. In [7], link sharing guidelines are provided to maximize channel utilization and limit the access of misbehaving sources.

Furthermore, a number of algorithms have been proposed on fair scheduling [10][11][12]. While there may be some differences in implementation and complexity, the basic idea in all these algorithms, is for sources experiencing a bad wireless link to relinquish the unutilized bandwidth to other sources that can take advantage of it. Compensation in bandwidth occurs when the channel conditions improve in order to achieve the so-called *Long Term Fairness* objective.

While the problem that we are trying to solve bears some resemblance with the problem addressed previously ([12] [7] [11][10]), we are more interested in an instantaneous measure of fairness rather than a *Long Term Fairness* objective. The reason is as follows. All previous work uses a two state Markov channel model for each link. The transition probabilities between the good and bad states are in the order of several seconds to account for periods of fading, multipath and various other wireless effects. The situation in our case is somewhat different due to the hopping nature of the Bluetooth device that uses a different frequency every 625 μs interval. Since different Bluetooth devices in a piconet will be subject to different interference levels due to parameters such as geometry and transmitted power, not all frequencies will be equally good to all devices. Therefore, our goal is to optimally assign frequencies such as to maximize channel utilization and guarantee fairness among the devices.

This paper is organized as follows. In section II we give some general insights on the Bluetooth protocol operation. In sections III and IV, we describe the scheduling mechanism and discuss its fairness properties respectively. In section V, we give simulation results and concluding remarks are offered in section VI.

II. Bluetooth Protocol Overview

In this section, we give a brief overview of the Bluetooth protocol [1]. Bluetooth is a short range (0 m - 10 m) wireless link technology aimed at replacing non-interoperable proprietary cables that connect phones, laptops, PDAs and other portable devices together. Bluetooth operates in the ISM frequency band starting at 2.402 GHz and ending at 2.483 GHz in the USA, and Europe. 79 RF channels of 1 MHz width are de ned. The raw data rate is de ned at 1 Mbits/s. A Time Division Multiplexing (TDM) technique divides the channel into 625 μs slots. Transmission occurs in packets that occupy an odd number of slots (up to 5). Each packet is transmitted on a different hop frequency with a maximum frequency hopping rate of 1600 hops/s.

Two or more units communicating on the same channel form

a piconet, where one unit operates as a master and the others (a maximum of seven active at the same time) act as slaves. A channel is de ned as a unique pseudo-random frequency hopping sequence derived from the master device's 48-bit address and its Bluetooth clock value. Slaves in the piconet synchronize their timing and frequency hopping to the master upon connection establishment. In the connection mode, the master controls the access to the channel using a polling scheme where master and slave transmissions alternate. The master uses even numbered slots while odd numbered slots are reserved for slave transmissions.

There are two types of link connections that can be established between a master and a slave: the Synchronous Connection-Oriented (SCO), and the Asynchronous Connection-Less (ACL) link. The SCO link is a symmetric point-to-point connection between a master and a slave de ned to carry 64 kbits/s of a voice stream.

In this paper, we focus on a scheduling strategy used for transmitting data on the ACL link that de nes an asymmetric point-to-point connection between a master and active slaves in the piconet. While the master can send data to a slave in the piconet on any even numbered slots, a slave has to be polled before it can transmit data. Therefore, the slave to master data rate is negotiated using Link Manager Protocol (LMP) messages at connection setup. The negotiated rate is usually de ned in terms of a poll interval, and a packet length. Additional Quality of Service (QOS) parameters can be exchanged in Link Layer Control Adaptation Protocol (L2CAP) messages and include parameters such as peak bandwidth, latency and delay variation.

Several packet formats are de ned for ACL, namely DM or DH packets that occupy either 1, 3, or 5 time slots. DM packets use Forward Error Correction (FEC) while DH packets do not have any FEC in the payload. An Automatic Repeat Request (ARQ) procedure is applied to ACL packets where packets are retransmitted in case of loss until a positive acknowledgement (ACK) is received at the source. The ACK is piggy-backed in the header of the returned packet where an ARQN bit is set to either 1 or 0 depending on whether the previous packet was successfully received or not. In addition, a sequence number (SEQN) bit is used in the packet header in order to provide a sequential ordering of data packets in a stream and lter out retransmissions at the destination.

In addition to ACL and SCO packets, the master and slave message exchange includes short POLL and NULL packets. POLL messages can be sent by the master and require an ACK while NULL messages can be sent by either the master or the slave and do not require an ACK.

III. Bluetooth Interference Aware Scheduling (BIAS)

In this section, we present the Bluetooth Interference Aware Scheduling (BIAS) algorithm. Our main objective is to alleviate the impact of interference while maintaining fairness and supporting different Quality of Service (QoS).

In this sequel, we assume that the traf c from slave S_i to the master is characterized by a data rate, r_i, equal to $\frac{l_i}{p_i}$ where l_i is the packet length in slots (1, 3 or 5 slots depending on the packet type), and p_i is the poll interval in (master/slave) slot pairs. In addition, we assume the following transmission rules for the master and slave.

Master - The master polls slave S_i every p_i in order to guarantee r_i in the upstream direction. A poll message can be either a data or NULL message. A data packet is sent to slave S_i if there is a packet in the queue for slave S_i. This packet contains the ACK of the previous packet received from slave S_i. In case there is no data to transmit and the master needs to ACK a previous slave transmission, it sends a NULL packet to slave S_i.

Slave S_i **-** Upon receipt of a packet from the master, the slave can transmit a data packet. This data packet contains the ACK information of the master to slave packet transmission. In case the slave does not have any data to send, it sends a NULL packet in order to ACK the previous packet reception from the master. No ACK is required for a NULL message from the master.

Our algorithm consists of several components, namely, a channel estimation procedure, a procedure that assigns weights to devices in order to determine a channel access priority, and a resource credit function that allocates bandwidth to each device according to its service requirements and the state of the channel.

The *estimate_channel()* procedure is used to detect the presence of interference in the frequency band. Thus, each Bluetooth receiver maintains a *Frequency Usage Table* where a bit error rate measurement, BER_f, is associated to each frequency as shown in Figure 1. Frequencies are classi ed according to a criteria that measure the level of interference in the channel and are marked *used* or *clear* depending on whether their corresponding BER is above or below a threshold value, BER^T, respectively. Note that, other criteria such as frame error rate, packet loss, or the received signal strength can be used in addition to the bit error measurement to detect a high level of interference in a speci c frequency band.

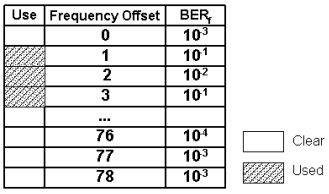


Fig. 1. Frequency Usage Table

Since the master device controls all transmissions in the piconet, the slaves need to send their *Frequency Usage Table* in the form of status update messages. The scheduler at the master can then make use of the measurements collected during the *Channel Estimation* phase in order to optimize the frequency allocation on each time slot and avoid a packet transmission in a receiving channel with a high level of interference. Figure 2 illustrates the frequency allocation that occurs at the master. In this case, frequency 78 is used to communicate with slave S_i, while frequencies 76, 1 and 0 are assigned to slaves S_i, S_{i+1}

and S_{i+1} in that order. Observe that the pattern of frequencies corresponds to the receiving frequencies. Thus, an M marks a receiving slot for the master device while an S is a receiving frequency for a slave device. Although, the master scheduler attempts to maximize channel utilization, it intentionally leaves certain slot pairs empty if either the master or the slave receiving frequency is *used*. Thus, in Figure 2, frequencies 16, 2, 7 and 77 are not used since frequencies 2 and 77 are not *clear* for the master.

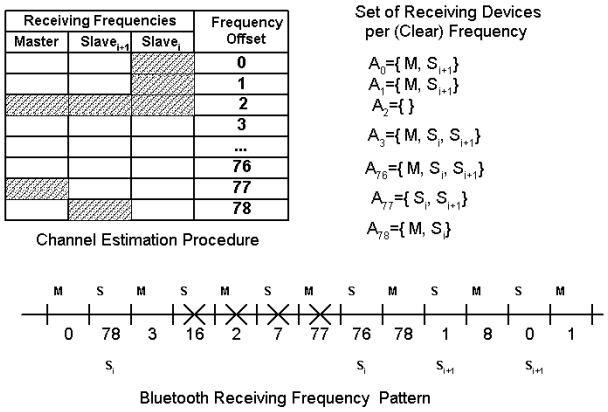


Fig. 2. Master Frequency Allocation Scheduling

The basic idea in the credit system is to control the bandwidth allocated to each device in order to ensure that no device gets more than its fair share of the available bandwidth. Thus, devices with a positive credit counter, c_i, are allowed to send data. There can be several ways to compute credits. Our method is based on the max-min fairness criteria [13]. Given r_i, we let u_i be the probability that a pair of slots (master/slave) are *clear*. Thus, u_i represents the available spectrum to slave i. Therefore, we write:

$$u_i = P(slave\ i\ has\ a\ clear\ receiving\ frequency) \times P(master\ has\ a\ clear\ receiving\ frequency) \quad (1)$$

where

$$P(device\ i\ has\ a\ clear\ receiving\ frequency) = \frac{Number\_of\_clear\_Channels_i}{Total\_Number\_of\_Channels} \quad (2)$$

We then de ne μ_i as

$$\mu_i = \min(u_i, r_i) \quad (3)$$

where μ_i is the minimum guaranteed rate for device i. Thus, in the case device i has a requested rate r_i, such that $r_i > u_i$, but is experiencing interference so it is not able to utilize more that u_i of the spectrum, and its rate is limited to u_i. We de ne a *constrained* device to be a device that is not able to use the entire frequency spectrum, such that $r_i > u_i$, while an *unconstrained* device is such that $r_i \leq u_i$. The next step is to reallocate the leftover bandwidth that is unused by the *constrained* devices and let g_i be the actual rate given to device i:

$$g_i = \begin{cases} \mu_i + \frac{r_i(B-\mu)}{\sum_{j \in Unconstrained} r_j} & \text{if } r_i < u_i \\ \mu_i & otherwise \end{cases} \quad (4)$$

where $\mu = \sum_i \mu_i$ and $B = 1 - \frac{Number\_of\_Used\_Channels_F}{Total\_Number\_of\_Channels}$. $Number\_of\_Used\_Channels_F$ is the number of frequencies that are marked *used* for all devices (in other words frequencies that can not be used by any device in the piconet). In essence, Equation 4 redistributes the leftover bandwidth to *unconstrained* devices proportionally to their service rate as in the Generalized Processor Scheduling (GPS) [14]. Thus, the *compute_credits()* function consists of computing the credits according to:

$$c_i = g_i \times N \quad (5)$$

where N is the number of slot pairs considered in the allocation.

The other component of the algorithm is to actually give the "right of way" or a priority of access to certain devices. We choose to give devices with fewer number of good channels a higher priority over other devices that have more channels available. Thus, in *compute_weights()* we set w_i as follows:

$$w_i = \min(\epsilon, P(slave\ i\ has\ a\ used\ receiving\ frequency)) \quad (6)$$

where we de ne

$$P(slave\ i\ has\ a\ used\ receiving\ frequency) = \frac{Number\_of\_used\_Channels_i}{Total\_Number\_of\_Channels} \quad (7)$$

and assume ϵ takes on values in $]0, \frac{1}{Total\_Number\_of\_Channels}]$ in the case all channels are *clear*. Finally, priority are assigned according to the "send factor", α_i, given by:

$$\alpha_i = w_i \cdot c_i \quad (8)$$

A. BIAS Pseudocode

The algorithm's pseudocode is as follows.

Every N slots
estimate_channel();
compute_weights();
compute_credits();
Every Even TS_f // Master Transmission Slot
if $TS_f + 1$ is clear // Master can receive in next slot
A_f = { set of slaves that can receive on frequency f }
$i = \max_{A_f}(\alpha_i)$ // Select device i with the largest send factor
if $\exists\ i$ s.t. $qsize_i > 0$ and $\alpha_i > 0$
c_i- -; //decrement credit counter
$\alpha_i = w_i \times c_i$; // update send factor
transmit packet for slave i

Table I summarizes the parameters used in the algorithm and their de nition.

TABLE I
DEFINITION OF PARAMETERS USED IN THE SCHEDULING ALGORITHM

Parameters	De nition
B	available spectrum
r_i	rate negotiated for device i
w_i	weight for device i
c_i	credit for device i
g_i	rate allocated for device i
α_i	send factor for device i
u_i	available frequency usage for device i
μ_i	minimum guaranteed rate for device i
ϵ	weight assigned to devices with $u_i = 1$

B. Numerical Example

Let's consider the *Frequency Usage Table* given in Figure 3 as an example. We consider 10 receiving frequencies, $f \in [0, 9]$. In order to keep the discussion simple, we show the frequency pattern for the *downstream* traf c and assume all 10 frequencies are *clear* for the master. We assume that there are 3 slaves in the piconet and each slave has a service rate equal to $r_1 = r_2 = r_3 = 1/3$ i.e. each slave gets polled every 6 slots. Since frequency 2 is marked *used* for all 3 slaves, noone can use it and $B = 9/10$. We compute u_i according to Equation 1. Slave, S_1, can use 2 out the 10 frequencies, therefore $u_1 = 2/10$. $u_2 = 7/10$ and $u_3 = 3/10$ for S_2 and S_3 respectively.

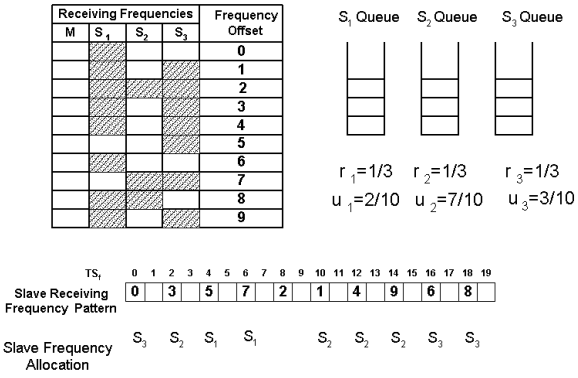


Fig. 3. Frequency Allocation Example

Similarly, $\mu_1 = 2/10$, $\mu_2 = 1/3$ and $\mu_3 = 3/10$. $\mu = \sum_{i=1}^{3} \mu_i = 25/30$. Since S_2 is *unconstrained*, it is a candidate device for receiving the leftover bandwidth. Therefore, $g_1 = 0.2$, $g_2 = \mu_2 + B - \mu = 0.4$, and $g_3 = 0.3$. The credits are $c_1 = 2$, $c_2 = 4$ and $c_3 = 3$ slots for $N = 10$. The weights are $w_1 = 8/10$, $w_2 = 3/10$ and $w_3 = 7/10$. At time $TS = 0$, $\alpha_1 = 2 * 8/10 = 8/5$ while $\alpha_2 = 4 * 3/10 = 6/5$ and $\alpha_3 = 3 * 7/10 = 27/10$. Therefore, S_3 is serviced at time $TS = 0$. A similar calculation determines the service order for S_1 on $TS = 4, 6$ and S_2 on $TS = 2, 10, 12, 14$, and S_3 on $TS = 16, 18$.

IV. BIAS PROPERTIES

In this section we summarize the features of the BIAS algorithm and highlight some of its fairness properties. Speci cally , Theorem 1 states that error-free connections are serviced according to their negotiated rate and gives an upper bound derivation for the initial delay. Theorem 2 says that *clear* channels are shared among error-free and error-prone devices according to their proportional rate.

The BIAS algorithm properties are summarized as follows.
P1. Service guarantees are provided to error-free connections including delay bounds and throughput. Although, error-prone connections are given a higher priority of access, interference-free devices are not affected by the interference conditions subsiding on some devices in the piconet.
P2. The scheduling policy is work conserving since no slots will be left idle if there is at least one device with a positive credit counter.
P3. Short term max-min fairness is guaranteed since the leftover bandwidth unused by the error prone sessions is redistributed to error-free sessions proportionally to their negotiated service rate.
P4. The sharing of *clear* channels is proportional to each session's negotiated rate regardless of whether they are error-free or error-prone.

Theorem 1
The number of slots allocated to an error-free session i over an interval of N slot pairs is equal to at least $r_i \times N$. The initial delay (in slot pairs), Δ, for an error free session, k, to send its rst packet over an interval of $1.25 \times N$ms is at most equal to:

$$\Delta = \sum_{i=1}^{k} \min(x_i, c_i) \tag{9}$$

where i represents the index of slave i,and c_i is the credit given to device i. x_i is de ned as:

$$x_i = \lceil \frac{c_i \cdot w_i - c_{i-1} \cdot w_{i-1}}{w_i} \rceil \tag{10}$$

where w_i is the weight of device i, and indices are assigned to devices such that $w_1 \cdot c_1 > w_2 \cdot c_2 > w_k \cdot c_k > ... > w_n \cdot c_n$, for all n devices in the piconet.

Proof
The minimum number of slots allocated to an error free connection follows directly from Equation 4. Since error free connections are not constrained by their spectrum usage, their allocation is greater or equal to their negotiated service rate, r_i.

In order to prove the initial access delay bound, we consider two devices i and j such that $w_i \cdot c_i > w_j \cdot c_j$. Since $\alpha_j < \alpha_i$ ($\alpha_i = w_i \cdot c_i$), device j will only be allowed to access the channel after device i has transmitted at least $x_i = \lceil \frac{c_i \cdot w_i - c_j \cdot w_j)}{w_i} \rceil$. Note that

$$w_i(c_i - x_i) \leq w_j \cdot c_i \tag{11}$$

and device j is allowed to transmit after device i has transmitted at least x_i packets. Also, observe that after the initial Δ slot pairs, both devices i and j will be serviced in a Round Robin (RR) fashion. This represents an upper bound since device i will keep its priority of access only if all Δ slot pairs are *clear*. In case, a *used* slot pair is encountered during the rst Δ slot

pairs and it is *clear* for device j, then j is allowed to transmit. □

Theorem 2
All devices sharing a set or subset of *clear* channels are serviced according to their allocated rate.

Proof
In case the devices sharing a set of *clear* frequencies have equal αs, they are serviced according to a RR policy. In the case devices have different αs, the device with the maximum α is serviced rst. Assuming equal credit counters, and without loss of generality, a high value for α indicates that the device has limited usage of the spectrum and is therefore self-constrained. That is, the device will not be able to use every slot in the set and thus will not deny service to other devices sharing the same subset with a smaller value of α. □

V. Simulation Results

In this section, we present simulation results to evaluate the performance of BIAS. The results obtained are compared with Round Robin (RR) scheduling. Our simulation environment is based on a detailed simulation environment consisting of the MAC, PHY and channel models for Bluetooth and IEEE 802.11 (WLAN) as described in [6]. We use the topology illustrated in Figure 4, and the simulation parameters presented in Table II.

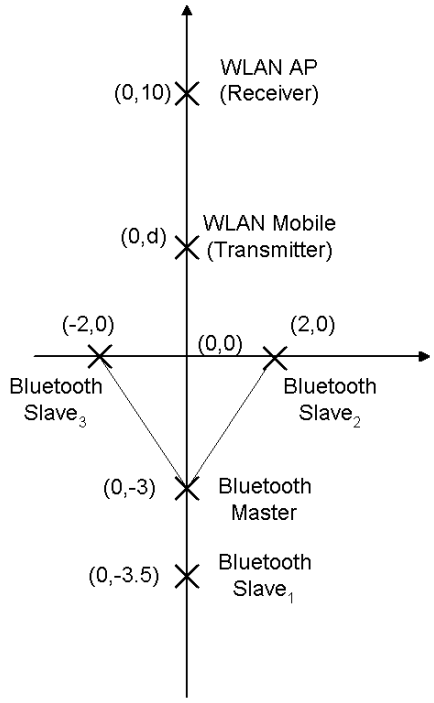


Fig. 4. Experiment Topology

We assume that WLAN is operating in the Direct Sequence Spread Spectrum (DSSS) mode. We vary the traf c distributions for WLAN and Bluetooth as follows. We assume that the WLAN Mobile device is transmitting data packets to the Access Point (AP) device which is responding with ACKs. The WLAN packet payload is set to 7776 bits transmitted at 11 Mbits/s, while the packet header is set to 224 bits transmitted at 1 Mbits/s. We assume that the WLAN packet interarrival rate is exponentially distributed with a mean of 1.86 ms corresponding to 50% of the offered load. For Bluetooth, we assume that the master device is transmitting DM1 packets with a mean arrival rate of λ where $\lambda = \frac{2*0.000625}{l} - 2 * 0.000625$ seconds, and $l = 25$ is the offered load in percent of the channel capacity. There are 3 slaves in the topology and they are sharing 25% of the total capacity. Thus, the offered load for each slave is set to 8.33%. The parameters used in the setup are summarized in Table II. Statistics are collected at the Bluetooth slave devices.

TABLE II
Simulation Parameters

Bluetooth Parameters	**Values**
ACL Baseband Packet Encapsulation	DM1 , 366 bits
Packet Interarrival Time for the master	2.91 ms
Transmitted Power	1 mW
Slave 1 Coordinates	(0, -3.5)
Slave 2 Coordinates	(2,0)
Slave 3 Coordinates	(-2,0)
Master Coordinates	(0,-3)
WLAN Parameters	**Values**
Packet Interarrival Time	1.86 ms
Offered Load	50 % of Channel Capacity
Transmitted Power	25 mW
Data Rate	11 Mbits/s
AP Coordinates	(0,10)
Mobile Coordinates	(0,d)
Packet Header	224 bits
Payload Size	7776 bits

The performance metrics that we use include the packet loss, the mean access delay and the fairness index. The packet loss is the probability that a packet is dropped at a device due to interference. The access delay measures the time it takes to transmit a packet from the time it is passed to the MAC layer until it is successfully received at the destination.The delay is measured at the L2CAP layer. We de ne the fairness index for device i, $F_i = \frac{Number\_of\_Packets\_Received}{Expected\_Number\_of\_Packets\_Received}$. The $Number\_of\_Packets\_Received$ is the number of packets sent minus the number of packets dropped.

Figure 5 gives the Bluetooth slave packet loss with respect to d, the y-coordinate of the WLAN transmitter. As the WLAN transmitter moves further away from the Bluetooth piconet (d increases), the packet loss measured at each of the slave devices decreases for RR scheduling and is kept at zero for BIAS. At $d = 0$, the WLAN transmitter is 2m away from slaves 2 and 3 and 3 meters away from slave 3, the packet loss is 1%, 17.3%, and 17.8% for slaves 1, 2 and 3 respectively for RR and 0% for BIAS. Even when $d = 8$m (i.e. the WLAN device is at 11.18m away from slaves 2 and 3, the packet loss is still in the order of 6.7% for slaves 2 and 3 with RR.

Figure 6 gives the mean access delay for servicing the Bluetooth slaves. With RR, the mean access delay is around 1.6ms for $d = 0$m for all 3 slaves. With BIAS, the delay increases to ~ 3.6 ms for slaves 2 and 3 and 5.2ms for slave 1. This increase in delay is expected since there is a trade-off between packet loss and delay.Thus, in order to bring the packet loss to zero, bad frequencies are skipped at the expense of increasing

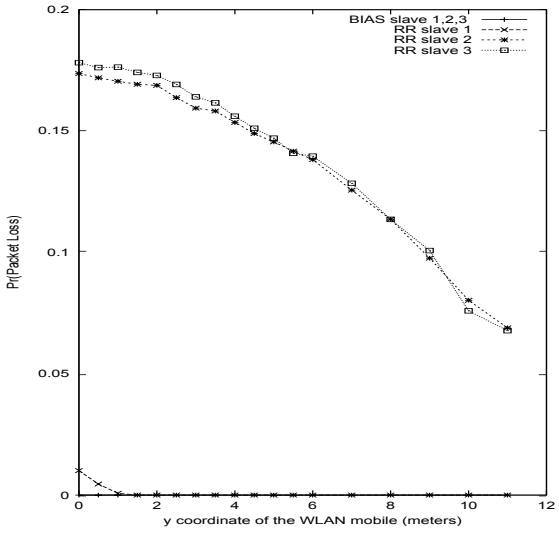


Fig. 5. Effect of Scheduling on Packet Loss.

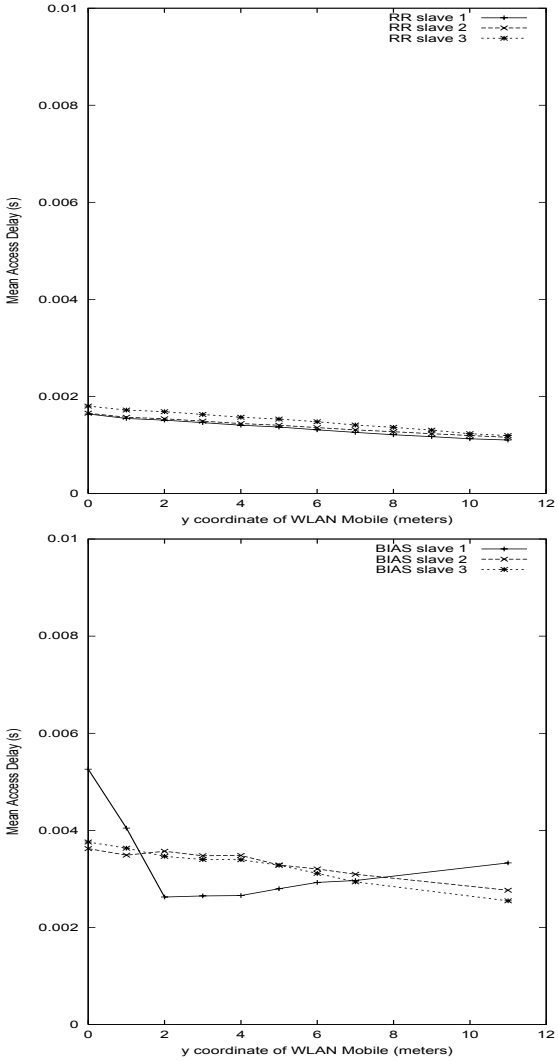


Fig. 6. $\boxed{\frac{(a)}{(b)}}$ Effect of Scheduling on the Mean Access Delay. (a) Round Robin Scheduling. (b) BIAS Scheduling

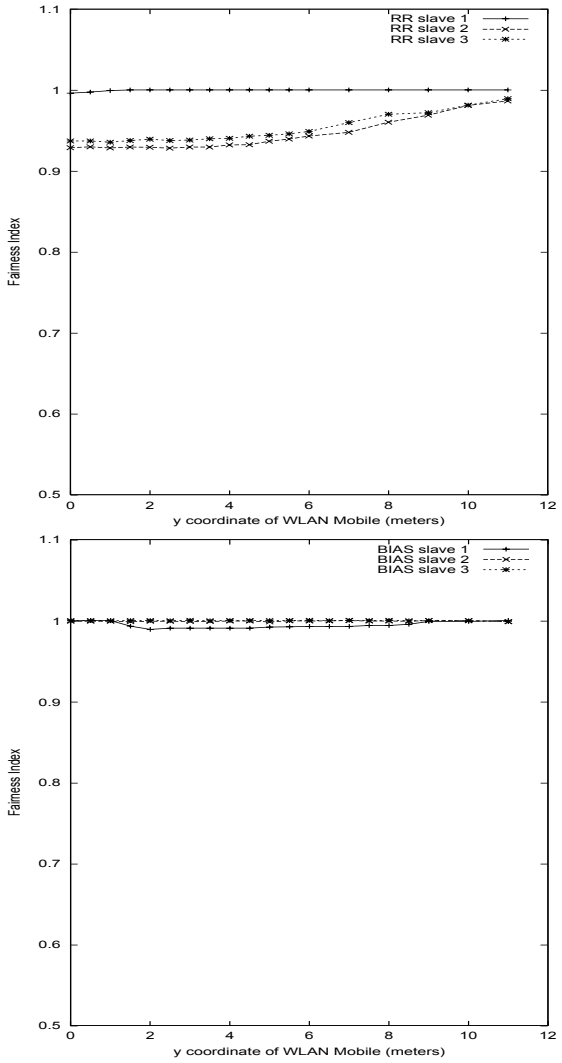


Fig. 7. $\boxed{\frac{(a)}{(b)}}$ Effect of Scheduling on Fairness. (a) Round Robin Scheduling. (b) BIAS Scheduling

the mean access delay. We veri ed that this result does not apply to multi-slot packets where the transmission time represents a larger fraction in the delay calculation. Therefore, reducing the packet loss when multi-slot packets are used reduces the mean access delay as well.

Figure 7 gives the fairness index with RR and BIAS. We note that all 3 slaves get the same number of packets with BIAS and their fairness index is 1 regardless of the position of the WLAN transmitter. For RR, the fairness index is 0.99 for slave 1, while it is 0.92 and 0.93 for slaves 2 and 3 respectively for $d = 0$m. Also, note that slaves 2 and 3 experience 17% of packet loss. Thus, more packets are being sent to slaves 2 and 3 with RR and the channel utilization is as not as ef cient as with BIAS. As the offered load increases, we expect this effect to be magni ed and lead to unfairness and degradation in servicing slave 1, which the error-free device in this case. Also, on the topic of channel utilization, we observe that with RR scheduling the number of NULL packets transmitted is 12% higher than with BIAS. This is mainly due to the packet loss that leads to

retransmissions and therefore the number of NULL packets to ACK data transmissions is higher.

VI. Concluding Remarks

In this paper we present, BIAS, a scheduling technique for alleviating the impact of interference on the Bluetooth performance. This technique attempts to redistribute the bandwidth unused by the interference-prone sessions to other error-free connections that can take advantage of it. Our goal is to guarantee fairness in scheduling while maximizing the channel utilization.

Our simulation results indicate that BIAS can signi cantly lower the probability of packet loss for sessions experiencing interference without much increase in the mean access delay for the worst case scenario. Furthermore, we demonstrate that BIAS can provide *Short Term Fairness* where error-free sessions are still serviced according to their negotiated rate regardless of the channel conditions of other devices.

Our current work is focused on extending BIAS and investigating the use of combined approaches such as packet encapsulation and o w control in order to support QoS in a Bluetooth environment.

References

[1] Bluetooth Special Interest Group, "Speci cations of the Bluetooth System, vol. 1, v.1.0B 'Core' and vol. 2 v1.0B 'Pro les', " December 1999.

[2] K. J. Negus, A. P. Stephens, and J. Lansford, "HomeRF: Wireless Networking for the Connected Home," in *IEEE Personal Communications*, February 2000, pp. 20–27.

[3] IEEE Std. 802-11, "IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Speci cation ," June 1997.

[4] N. Golmie and F. Mouveaux, "Interference in the 2.4 GHz ISM band: Impact on the Bluetooth access control performance," in *Proceedings of IEEE ICC'01*, 2001.

[5] A. Soltanian and R. E. Van Dyck, "Physical layer performance for coexistence of Bluetooth and IEEE 802.11b," in *in Virginia Tech. Symposium on Wireless Personal Communications*, June 2001.

[6] N. Golmie, R.E. Van Dyck and A. Soltanian, "Interference of Bluetooth and IEEE 802.11: Simulation Modeling and Performance Evaluation," in *Proceedings of the Fourth ACM International Workshop on Modeling, Analysis, and Simulation of Wireless and Mobile Systems, MSWIM'01*, Rome, Italy, July 2001.

[7] C. Fragouli, V. Sivaraman, and M. B. Srivastava, "Controlled Multimedia wireless link sharing via enhanced class-based queuing with channel-state-dependent packet scheduling," in *Proceedings of IEEE INFOCOM'98*, San Fransisco, CA, March 1998, pp. 572–580.

[8] S. Flyod and V. Jacobson, "Link Sharing and resource management models for packet networks," in *ACM/IEEE Transactions on Networking*, August 1995, vol. 3, pp. 365–386.

[9] P. Bhagwat, P. Bhattacharya, A. Krishna, and S. Tripathi, "Enhancing Throughput over Wireless LANs using Channel State Dependent Packet Scheduling," in *Proceedings of IEEE INFOCOM'96*, 1996, vol. 3, pp. 1133–1140.

[10] S. Lu, V. Bharghawan, and R. Srikant, "Fair Scheduling in wireless packet networks," in *Proceedings of ACM SIGCOMM'97*, September 1997, pp. 63–74.

[11] P. Ramanathan and P. Agrawal, "Adapting packet fair algorithms to wireless networks," in *Proceedings of ACM/IEEE MOBICOM'98*, October 1998, pp. 1–9.

[12] T.S.E. Ng, I. Stoica, and H. Zhang, "Packet fair queueing algorithms for wireless networks with location dependent errors," in *Proceedings of INFOCOM'98*, March 1998, pp. 1103–1111.

[13] D. Bertsekas and R. Gallager, *Data Networks, 2nd Ed.*, Prentice Hall, 1992.

[14] A. Parekh and R. Gallager, "A generalized processor sharing approach to o w control - the single node case," in *ACM/IEEE Transactions on Networking*, June 1993, vol. 1, pp. 344–357.

Techniques to Improve the Performance of TCP in a mixed Bluetooth and WLAN Environment

N. Golmie and O. Rebala
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
Email: nada.golmie@nist.gov

Abstract—**A major challenge for the WLAN technology stems from having to share the 2.4 GHz ISM band with other wireless devices such as Bluetooth radios. The main goal of this paper is to investigate the use of techniques to mitigate the effects of interference for Bluetooth and WLAN and discuss the resulting performance trade-offs. We compare the performance of the Bluetooth and WLAN systems and evaluate how each technique improves or degrades TCP performance. Simulation results for selected scenarios and configurations of interest are obtained and the performance of Bluetooth and WLAN is measured in terms of packet loss, TCP throughput and delay.**

Keywords— **WPANs, Bluetooth, Interference, MAC scheduling, TCP performance.**

I. Introduction

Since the Bluetooth and 802.11b technologies use the 2.4 GHz ISM band, devices operating in close proximity may suffer from mutual interference and significant performance degradation in terms of packet loss, lower throughputs and higher delays.

Various techniques and algorithms aimed at reducing the impact of interference have been considered [1]. These techniques range from collaborative schemes intended for Bluetooth and IEEE 802.11 protocols to be implemented in the same device [2] to fully independent solutions that rely on interference detection and estimation [3].

In this paper, we investigate the use of several techniques to mitigate interference for Bluetooth and WLAN and focus exclusively on schemes that do not require changes to either specifications. We consider rate scaling in conjunction with adaptive filtering for WLAN, and interference aware scheduling for Bluetooth. We compare the effects of using these techniques on performance for different scenarios and traffic types. Performance is measured in terms of packet loss, TCP delay and throughput.

The remainder of this paper is organized as follows. In section II, we describe the techniques used to mitigate interference. In section III, we give simulation results and concluding remarks are offered in section IV.

II. Techniques to Mitigate Interference

In this section, we present two techniques that can be used to mitigate the effect of interference. For WLAN, we consider data rate scaling, which is a common technique used in many implementations today to reduce the data rate from 11 down to 1 Mbit/s in a WLAN system. For Bluetooth, we consider a scheduling algorithm that avoids transmitting data on channels used by other wireless devices.

A. Bluetooth Interference Avoidance Scheduling

In this subsection, we give a brief overview of the Bluetooth Interference Aware Scheduling (BIAS) algorithm [4]. BIAS consists of three main components, namely a channel estimation procedure, a credit function that allocates bandwidth to each device according to its service requirements, and a priority scheduling function. Channel estimation can be based on either explicit or implicit methods. Explicit methods include BER calculation, packet loss, or frame error rate measurements performed on each receiver (master and slave device). The measurements are then collected by the master device at regular time intervals. Alternatively, implicit methods do not require the master and the slave to exchange information about the state of the channel. This information is derived by the master upon receipt of a negative ACK. We note that either channel estimation method allows the master device, which controls all data transmissions in the piconet, to avoid data transmission to a slave experiencing a "bad" frequency. Furthermore, since a slave transmission always follows a master transmission, using the same principle, the master avoids receiving data on a "bad" frequency, by avoiding a transmission on a frequency preceding a "bad" one in the hopping pattern.

This simple scheduling scheme needs only be implemented in the master device and translates into the following transmission rule. *The master transmits in a slot after it verifies that both the slave's receiving frequency and its own receiving frequency are "good". Otherwise, the master skips the current transmission slot and repeats the procedure over again in the next transmission opportunity.*

Additional considerations including bandwdith requirements and quality of service guarantees for each master/slave connection in the piconet can also be combined with the channel state information and mapped into transmission priorities given to each direction in the master/slave communication. Details on assigning transmission priorities are given in [5].

The algorithm's general steps are summarized below.

1: Every Even TS_f // Master transmits on frequency f
2: if $TS_f + l_{dn}$ is good // Master can receive in next slot
3: {
4: A^f_{data} = {set of slaves s.t. ((f "good") and ($qsize > 0$) }
5: if ($A^f_{data} \neq \emptyset$)
6: select slave i //according to a priority criteria
7: transmit data packet of size l_{dn} to slave i
8: }

where l_{dn} is the length of the packet from the master to the slave ($downstream$) and TS_f is the transmission slot using frequency f.

B. WLAN Rate Scaling

Rate scaling is used in most WLAN implementations in order to optimize the range performance since the 1 Mbit/s Barker code WLAN receiver performs better than the Complementary Code Keying (CCK) 11 Mbit/s [6] [7] [8]. The Barker code correlation effectively spreads noise or the interference signal while de-spreading the desired signal and leads to lower probability of bit error (BER) than CCK for the same signal-to-interference ratio (SIR).

While there is provision in the IEEE 802.11 standards [9] to implement a rate scaling algorithm, the details remain vendor implementation specific. In our study, we use a simple two-level threshold algorithm with some hysteresis margin in order to avoid unnecessary oscillations.

1: If $SIR_{measured} \geq SIR^{High}$ // the interference is low
2: PHY mode = 11 Mbit/s
3: If $SIR_{measured} < SIR^{Low}$ // the interference level is high
4: PHY mode = 1 Mbit/s

Basically, $SIR_{measured}$ is based on the Received Signal Strength Indicator (RSSI). The assumption is when the RSSI is low, the interference level is high (or the desired signal is weak), and therefore, the receiver reverts to the 1 Mbit/s mode. We set SIR^{High} and SIR^{Low} to 6 and 2 db respectively based on the BER performance of each receiver. Above 2 dB the BER for the 11 Mbit/s is below 10^{-4} [7].

In addition, we use an adpative filter in our 1 Mbit/s WLAN receiver that is able to estimate and cancel the Bluetooth interference. This technique is based on recursive least-squares lattice (RLSL) filters and generally more effective for the 1 Mbit/s WLAN receiver. It is adaptive in the sense that it does not require an a priori knowledge of the Bluetooth hopping patterns. Additional details on this method can be found in [10] where the authors discuss its effectiveness for both the 1 and 11 Mbit/s WLAN receivers.

III. Simulation Results

In this section, we present simulation results to evaluate the performance of the two techniques discussed in the previous section. We use a detailed simulation environment consisting of the MAC, PHY and channel models for Bluetooth and WLAN as described in [11]. We use the topology illustrated in Figure 1. The Bluetooth master and slave are placed one meter a part at (-0.5,0) and (0.5, 0) meters respectively. The WLAN station is located at (0,15) meters, while the WLAN server is located at (0,d) meters, where d varies along the y-axis between 0 and 10 meters.

We consider two application profiles, namely, FTP, and HTTP. We use the TCP/IP stack implemented in the OPNET library and configure the application profiles as shown in Table I. The parameters used in the setup are summarized in Table II. The simulations are run for 500 seconds of simulated time. We run 10 trials using a different random seed for each trial. In addition, to plotting the mean value, we verify that that the statistical variation around the mean values are very small (less than 1%).

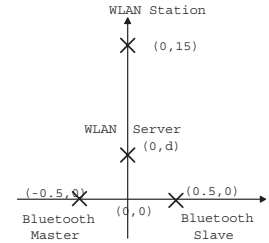


Fig. 1. Experiment Topology

The performance metrics include the packet loss, the average delay in seconds and the throughput in bytes/s. The packet loss is the percentage of packets dropped due to interference over the total number of packets received at the MAC layer. The average delay, measured at the TCP layer, indicates the time it takes to transmit a packet from the time it is passed to the TCP layer until it is successfully received at the destination. The throughput is the traffic received at the TCP layer and includes packet retransmissions.

TABLE I
Application Profile Parameters

Parameters	Distribution	Value
FTP		
Percentage of Put/Get		50%
Inter-Request Time (seconds)	Exponential	1
File Size (bytes)	Constant	2 M
HTTP		
Page Interarrival Time (seconds)	Exponential	10
Number of Objects per page	Constant	2
1st Object Size (bytes)	Constant	10000
2nd Object Size (bytes)	Uniform	(2000, 100000)

TABLE II
Simulation Parameters

Bluetooth Parameters	**Values**
ACL Baseband Packet Encapsulation	DH5
Transmitted Power	1 mW
Slave Coordinates	(-0.5, 0)
Master Coordinates	(0.5,0)
WLAN Parameters	**Values**
Transmitted Power	25 mW
Data Rate	11 Mbit/s if not rate scaling
Station Coordinates	(0,15)
Server Coordinates	(0,d)
PLCP Header	192 bits
Packet Header	224 bits

We run simulations for three different experiments where we vary the profiles used for the Bluetooth and WLAN applications as shown in Table III. In experiment 1, both WLAN and Bluetooth use the FTP profile, while in experiments 2 and 3, the WLAN (/Bluetooth) application uses FTP (/HTTP) and HTTP (/FTP) traffic respectively. Although a large amount of data was obtained at analyzed, due to space constraints, only a small subset of the results is shown here.

In the next two subsections, we discuss the performance of

TABLE III
EXPERIMENT SUMMARY

Scenario	WLAN	Bluetooth
1	FTP	FTP
2	FTP	HTTP
3	HTTP	FTP

TCP over WLAN and Bluetooth in terms of the techniques proposed. We compare the performance of WLAN and Bluetooth when rate scaling is used for WLAN and scheduling is used for Bluetooth. For each experiment, we run 4 simulations in order to identify the benefits of each algorithm and its interactions with other schemes. *None* refers to the case when no algorithm is used. *Rate Scaling* means that WLAN uses the rate scaling algorithm, while *Scheduling* means that Bluetooth uses BIAS. The case where WLAN uses rate scaling and Bluetooth uses BIAS simultaneously is refered to as *Rate Scaling + Scheduling*.

A. TCP over WLAN

Figure 2 (a) gives the packet loss with respect to the y-coordinate of the WLAN server, d, when both WLAN and Bluetooth use the FTP profile. When no algorithm is used, the packet loss can be up to 14% when the WLAN server is close to the Bluetooth piconet (d=0 meters). As the server moves away from the Bluetooth piconet, the packet loss drops to zero ($d \geq 5$ meters). When rate scaling is used, the packet loss drops to 5% when d=0 meters. This packet loss observed is due to the intermittent use of the 11 Mbit/s WLAN receiver before the 1 Mbit/s mode is used. While the adaptive filter used in the 1 Mbit/s receiver is able to reduce the packet loss to zero, the 11 Mbit/s receiver is less robust and yields a relatively high packet loss. Observe that the packet loss is zero when Bluetooth uses BIAS since the Bluetooth transmitter avoids using the same frequency used by WLAN.

Figure 2(b) illustrates the throughput of the WLAN server. When no algorithm is used, the throughput starts at 240 Kbyte/s when d=0 meters, and goes up to 350 Kbyte/s when $d \geq 5$ meters and the packet loss is zero. Observe that when BIAS is used, the throughput remains around 350 Kbyte/s since no packets are lost. Since rate scaling involves reducing the WLAN bit rate from 11 to 1 Mbit/s, this yields to reducing the throughput to 50 Kbyte/s. As expected, rate scaling can reduce the packet loss, at the cost of reducing the throughput.

Figure 3(a) and (b) give the WLAN packet loss and delay respectively for experiment 3. In this case, the WLAN uses the HTTP profile while the Bluetooth uses the FTP profile. The packet loss depicted in Figure 3(a)) is slightly less than when WLAN uses the FTP profile (Figure 2(a)), however it follows a similar trend. The packet loss with BIAS is around 1% when $d < 4$ meters.

An important metric for HTTP is the delay to access data, therefore in Figure 3(b), we plot the TCP delay. Note that it is 15 ms when the packet loss is 12% (Figure 3(a)) and drops down to 2.5 ms when the packet loss is zero. Observe that when rate scaling is used the delay remains flat at 5 ms. On the other hand, when Bluetooth uses BIAS, the delay starts at 5 ms and

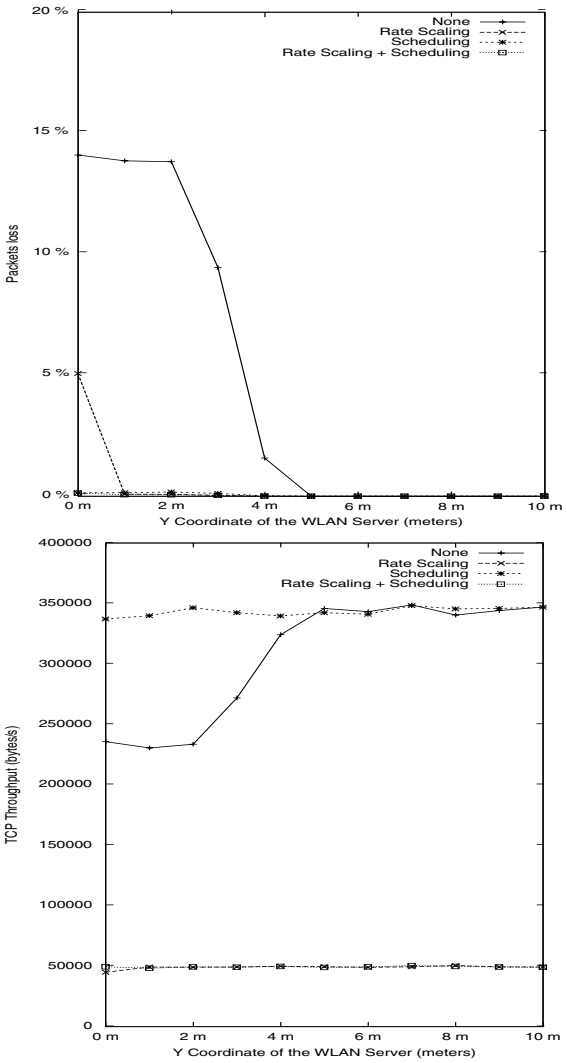


Fig. 2. $\boxed{\frac{(a)}{(b)}}$ Experiment 1. WLAN FTP Performance. (a) Probability of Packet Loss. (b) TCP Throughput

drops down to 2.5 ms.

Overall, we note that the use of Bluetooth scheduling improves the WLAN performance and brings it closer to the ideal case when no interference is present. The use of rate scaling produces interesting but expected trade-offs. While the WLAN packet loss is reduced, the delay is increased and the throughput is reduced.

B. TCP over Bluetooth

Figure 4(a) gives the packet loss for the Bluetoth master device as a function of the WLAN server y coordinate, d. When no algorithm is used, the packet loss is around 10% for d =0 meters. When $2 \leq d \leq 6$ meters, we observe a spike with a peak of 17% at d=4 meters. This is due to the closed loop interference between the WLAN and Bluetooth systems. To better understand the interactions, we look at Figure 2(a). Since less WLAN packets are lost (more WLAN packets are transmitted), this causes more interference on Bluetooth and thus more packet loss. This trend is valid until d=5 meters and the WLAN packet

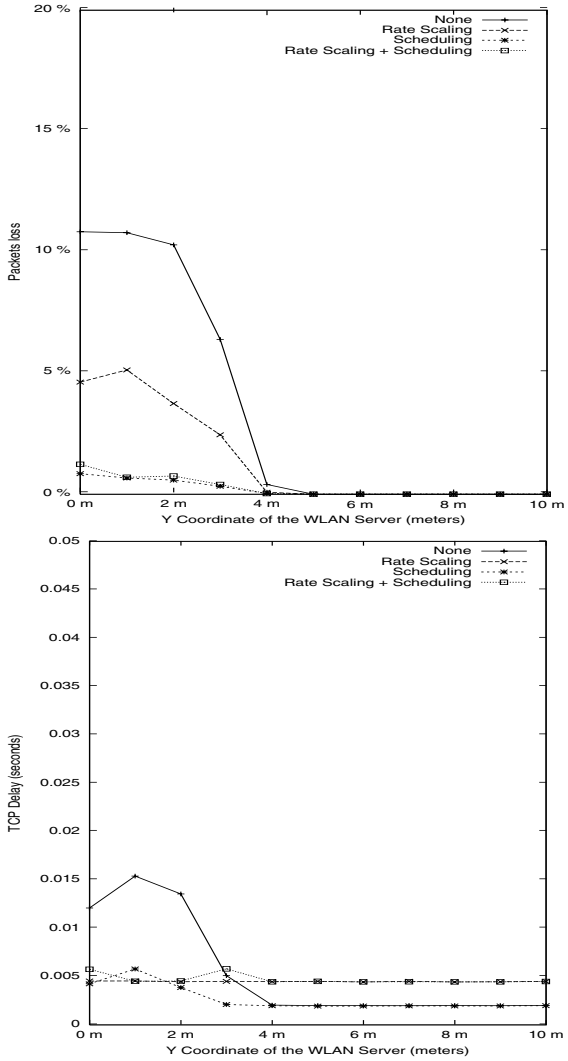


Fig. 3. $\boxed{\frac{(a)}{(b)}}$ Experiment 3. WLAN HTTP Performance. (a) Probability of Packet Loss. (b) TCP Delay

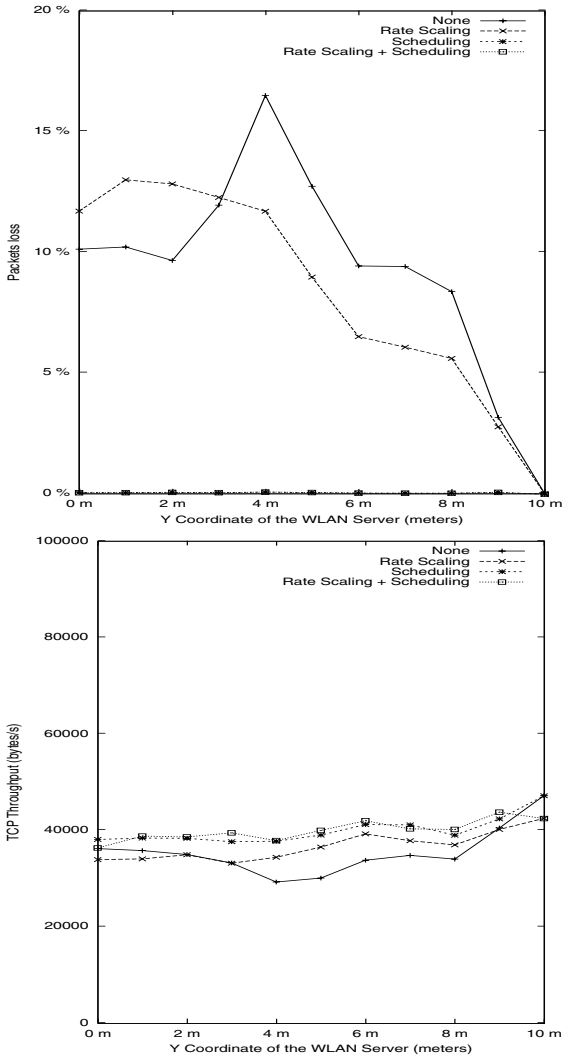


Fig. 4. $\boxed{\frac{(a)}{(b)}}$ Experiment 1. Bluetooth FTP Performance. (a) Probability of Packet Loss. (b) TCP Throughput

loss is zero. At that point, the Bluetooth packet loss start decreasing as the WLAN server moves further away. When rate scaling is used for the WLAN, we note a packet loss of 12% for Bluetooth at d=0 meters. The packet loss remains high until d=10 meters. This is due to the fact that rate scaling causes the WLAN to transmit packets at a lower rate, occupying more time in the air and causing more interference on Bluetooth. Note that when scheduling is used for Bluetooth, the packet loss is reduced to zero.

The TCP throughput depicted in Figure 4(b), closely follows the packet loss curves in Figure 4(a). When no algorithm is used, the throughput is 38 Kbyte/s when d=0 meters, 35 Kbyte/s when d=5 meters, and 45 Kbytes/s when d=10 meters, which clearly reflects a 12%, 17%, and 0% packet loss respectively. As expected, when rate scaling is used the throughput is about 10% lower than when scheduling is used reflecting the 10% packet loss observed in Figure 4(a).

The results for packet loss and delay when Bluetooth uses the HTTP profile (experiment 2), are illustrated in Figures 5(a) and (b) respectively. The packet loss when rate scaling is used is slightly higher (11%) than when no algorithm is used (8%). The packet loss is zero when scheduling is used.

The TCP delay in Figure 5(b) starts at 33 ms when rate scaling is used at d=0 meters. It is 7 ms and 12 ms when scheduling and no algorithm are used respectively. When no interference is present (d=10 meters), the delay is around 6 ms. Thus, the scheduling algorithm yields a slight increase in delay (around 1 ms) while reducing the packet loss to zero.

In summary, the main advantages of using scheduling in terms of the Bluetooth performance, are to reduce the packet loss to zero at almost no cost to either thoughput or delay. On the other hand the use of rate scaling for WLAN leads to higher packet losses for Bluetooth, including higher delays and lower throughput.

IV. Concluding Remarks

In this paper, we study the performance of TCP over Bluetooth and WLAN in a mutual interference environment consist-

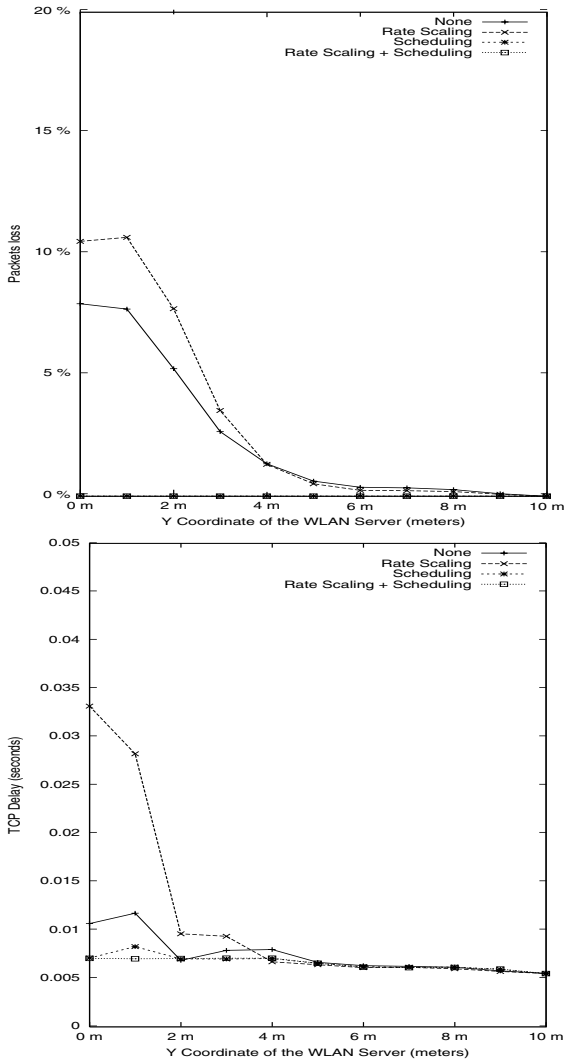


Fig. 5. $\boxed{\frac{(a)}{(b)}}$ Experiment 2. Bluetooth HTTP Performance. (a) Probability of Packet Loss. (b) TCP Delay

ing of two Bluetooth and two WLAN devices operating at the same time. We consider two application profiles, namely HTTP and FTP.

We investigate the use of two techniques to mitigate the effects of this mutual interference. Both techniques rely on detecting the presence of other wireless systems and adapting to the interference environment. For Bluetooth, we use a scheduling scheme that consists of avoiding to transmit a packet on a frequency used by the WLAN system. On the other hand, for WLAN we use rate scaling which consists of reverting to the more robust 1 Mbit/s mode. We also include in the 1 Mbit/s receiver used, an adaptive filter that can notch out the Bluetooth signal. Both techniques do not require any changes to either the Bluetooth or the IEEE 802.11 specifications.

Our simulation results indicate that the use of Bluetooth scheduling improves both the Bluetooth and WLAN systems' performance. The packet loss is reduced to zero, while the throughput is increased, and the delay decreased. On the other hand, the benefits of using rate scaling in the WLAN system are clearly less pronounced. While the packet loss is reduced for WLAN due to the the use of a more robust receiver and an adaptive filter, the performance of Bluetooth is degraded due to the increase of the WLAN packet transmission. As a result, the probability of a packet collision in time and frequency is much higher leading to higher packet loss and delays, and lower throughputs.

Finally, we note that these observations apply to either FTP or HTTP traffic. While the exact performance results depend on the parameters of the application profile used, the general trends hold in most cases studied.

ACKNOWLEDGEMENTS

The authors would like to thank Amir Soltanian for his help in the PHY layer simulation models and his assistance in making the 1 Mbit/s WLAN receiver with the adaptive filter available to use in the combined MAC and PHY simulation framework.

REFERENCES

[1] Carla F. Chiasserini, and Ramesh R. Rao, " Coexistence mechanisms for interference mitigation between IEEE 802.11 WLANs and bluetooth ," in *Proceedings of INFOCOM 2002*, 2002, pp. 590–598.

[2] J. Lansford, R. Nevo, E. Zehavi, "MEHTA: A method for coexistence between co-located 802.11b and Bluetooth systems," in *IEEE P802.11 Working Group Contribution, IEEE P802.15-00/360r0*, November 2000.

[3] B. Treister, A. Batra, K.C. Chen, O. Eliezer, "Adapative Frequency Hopping: A Non-Collaborative Coexistence Mechanism," in *IEEE P802.11 Working Group Contribution, IEEE P802.15-01/252r0*, Orlando, FL, May 2001.

[4] N. Golmie, N. Chevrollier, and I. Elbakkouri, "Interference Aware Bluetooth Packet Scheduling," in *Proceedings of GLOBECOM'01*, San Antonio, TX, November 2001.

[5] N. Golmie, "Bluetooth Dynamic Scheduling and Interference Mitigation," in *ACM Mobile Network, MONET*, 2002.

[6] B. Sklar, *Digital Communications: Fundamentals and Applications*, Prentice Hall, 1997.

[7] A. Soltanian and R. E. Van Dyck, "Physical layer performance for coexistence of Bluetooth and IEEE 802.11b," in *Virginia Tech Symposium on Wireless Personal Communications*, June 2001.

[8] N. Golmie, R.E. Van Dyck, A. Soltanian, A. Tonnerre, and O. Rebala, "Interference Evaluation of Bluetooth and IEEE 802.11b Systems," in *ACM Wireless Network, WINET*, 2002.

[9] IEEE Std. 802-11, "IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification ," June 1997.

[10] A. Soltanian, R. E. Van Dyck, and O. Rebala, "Rejection of Bluetooth Interference in 802.11 WLANs," in *Proceedings of IEEE VTC, Fall 2002*, September 2002.

[11] N. Golmie, R.E. Van Dyck and A. Soltanian, "Interference of Bluetooth and IEEE 802.11: Simulation Modeling and P erformance Evaluation," in *Proceedings of the Fourth ACM International Workshop on Modeling, Analysis, and Simulation of Wireless and Mobile Systems, MSWIM'01*, Rome, Italy, July 2001.

Bluetooth Dynamic Scheduling and Interference Mitigation

N. Golmie

National Institute of Standards and Technology

Gaithersburg, Maryland 20899

Email: nada.golmie@nist.gov

Abstract

Bluetooth is a cable replacement technology for Wireless Personal Area Networks. It is designed to support a wide variety of applications such as voice, streamed audio and video, web browsing, printing, and le sharing, each imposing a number of quality of service constraints including packet loss, latency, delay variation, and throughput. In addition to QOS support, another challenge for Bluetooth stems from having to share the 2.4 GHz ISM band with other wireless devices such as IEEE 802.11. The main goal of this paper is to investigate the use of a dynamic scheduling algorithm that guarantees QoS while reducing the impact of interference. We propose a mapping between some common QoS parameters such as latency and bit rate and the parameters used in the algorithm. We study the algorithm's performance and obtain simulation results for selected scenarios and con guratio ns of interest.

Keywords

WPANs, Bluetooth, Interference, MAC scheduling.

I. Introduction

Today most radio technologies considered by Wireless Personal Area Network (WPAN) industry consortia and standard groups including the Bluetooth Special Interest Group [1], HomeRF [2], and the IEEE 802.15, employ the 2.4 GHz ISM frequency band. This same frequency band is already in use by microwave ovens and the popular Wireless Local Area Network (WLAN) devices implementing the IEEE 802.11 standard speci cations [3].

However, instead of competing with WLANs for spectrum and applications, WPANs are intented to augment many of the usage scenarios and operate in conjunction with WLANs, i.e., come together in the same laptop, or operate in proximity in an of ce or conference room environment. For example, Bluetooth can be used to connect a headset, or PDA to a desktop computer, that in turn may be using WLAN to connect to an Access Point placed several meters away.

Thus, an issue of growing concern is the coexistence of WLAN and WPAN in the same environment. Several techniques and algorithms aimed at reducing the impact of interference have been considered. These techniques range from collaborative schemes

intended for Bluetooth and IEEE 802.11 protocols to be implemented in the same device to fully independent solutions that rely on interference detection and estimation. In particular:

- **Collaborative Mechanisms-**

  Mechanisms for collaborative schemes have been proposed to the IEEE 802.15 Coexistence Task Group and are based on a Time Division Multiple Access (TDMA) solution that alternates the transmission of Bluetooth and WLAN packets (assuming both protocols are implemented in the same device and use a common transmitter) [4]. A priority of access is given to Bluetooth for transmitting voice packets, while WLAN is given priority for transmitting data.

- **Non-Collaborative Mechanisms-**

  The non-collaborative mechanisms range from adaptive frequency hopping [5] to packet scheduling and traf c control [6]. They all use similar techniques for detecting the presence of other devices in the band such as measuring the bit or frame error rate, the signal strength or the signal to interference ratio (often implemented as the Received Signal Indicator Strength (RSSI)). Frequency hopping devices may be able to detect that some frequencies are used by other devices and thus modify their frequency hopping pattern. They can also choose not to transmit on ”bad” frequencies. The rst technique is known as adaptive frequency hopping, while the second technique is known as MAC scheduling. The main advantage of scheduling is that it does not require changes to the Bluetooth speci cations.

In this paper we present a Bluetooth Interference Aware Scheduling (BIAS) algorithm to deal with coexistence. This algorithm takes advantage of the fact that devices in the same piconet will not be subject to the same levels of interference on all channels of the band. The basic idea is to utilize the Bluetooth frequency hopping pattern and distribute channels to devices such that to maximize their throughput while ensuring fairness of access among users.

In this paper, we propose several extensions to a preliminary discussion of the the algorithm [7] in order to address (1) priority scheduling, (2) dynamic changes in the environment, and (3) asymmetric scenarios where packet lengths and data rates are chosen differently in the upstream (slave to master transmission) and downstream (master to slave transmission) directions. In addition, we describe how to map commonly used QOS parameters, namely bit rate, and jitter and the parameters used in BIAS. Simulation results for scenarios and con gurations of interest are presented and performance is measured in terms of packet loss and mean access delay.

The remainder of this paper is organized as follows. In section II we give some general insights on the Bluetooth interference environment. In sections III, we describe the scheduling algorithm and discuss the mapping of the QOS parameters. In section IV, we present simulation results and offer concluding remarks in section V.

II. Interference Environment

Since Bluetooth operates in the 2.4 GHz band along with other wireless technologies such as 802.11, high and low rate WPAN (802.15.3 and 4), the resulting mutual interference leads to signi cant performance degradation.

In this paper, we assume that interference is caused by an 802.11 spread spectrum network operating in proximity of the Bluetooth piconet. This represents the worst case interference for Bluetooth. Golmie *et al.* [8][9] use a detailed MAC and PHY simulation framework to evaluate the impact of interference for a pair of WLAN devices and a pair of Bluetooth devices. The results indicate that Bluetooth performance may be severely impacted by interference with packet loss of 8% and 18% for voice and data traf c respectively. In [9], the authors investigate the effect of several factors, such as transmitted power, offered load, packet size, hop rate, and error correction on performance. First, they note that power control may have limited bene ts in an interference environment. Increasing the Bluetooth transmission power even ten times is not suf cient to reduce the Bluetooth packet loss. Second, using a shorter packet size leads to less packet loss for Bluetooth at the cost of causing more interference on WLAN. Overall, the results exhibit a strong dependence on the type and characteritics of the traf c distribution used.

Additional analytical [10] [11] and experimentation [12] [13] results con rm these ndings.

III. Bluetooth Scheduling Algorithm

In this section, we present a Bluetooth Interference Aware Scheduling (BIAS) algorithm that consists of several components, namely, (i) dynamic channel estimation, (ii) credit computation, and (iii) access priority. A preliminary discussion of BIAS appeared in [7].

In this sequel, we assume that traf c from slave S_i to the master (upstream) is characterized by a data rate, γ_{up}^i, equal to $\frac{N_{peak}^i \times l_{up}^i}{p^i}$ where N_{peak}^i is the number of packets sent back-to-back within a poll interval, p^i, and l_{up}^i is the packet length (1, 3, or 5 slots depending on the packet type). Similarly, the data rate in the downstream (from the master to slave S_i) is characterized by γ_{dn}^i equal to $\frac{N_{peak}^i \times l_{dn}^i}{p^i}$. Note that N_{peak}^i and p^i are the same in the upstream and downstream, since every packet in the upstream corresponds to one in the downstream. In addition, we assume the following transmission rules for the master and slave.

Master - The master polls S_i every p^i slots in order to guarantee γ_{up}^i in the upstream direction. A poll message can be either a data or POLL packet. A data packet is sent if there is a packet in the queue destined for S_i. This packet contains the ACK of the previous packet received from S_i. In case there is no data to transmit and the master needs to ACK a previous slave transmission, it sends a NULL packet.

Slave S_i **-** Upon receipt of a packet from the master, the slave can transmit a data packet. This data packet contains the ACK

information of the master to slave packet transmission. In case the slave does not have any data to send, it sends a NULL packet in order to ACK the previous packet reception from the master. No ACK is required for a NULL message from the master.

In a nutshell, we propose a method that allows the master device, which controls all data transmissions in the piconet, to avoid data transmission to a slave experiencing a ”bad” frequency. Furthermore, since a slave transmission always follows a master transmission, using the same principle, the master avoids receiving data on a ”bad” frequency, by avoiding a transmission on a frequency preceding a ”bad” one in the hopping pattern.

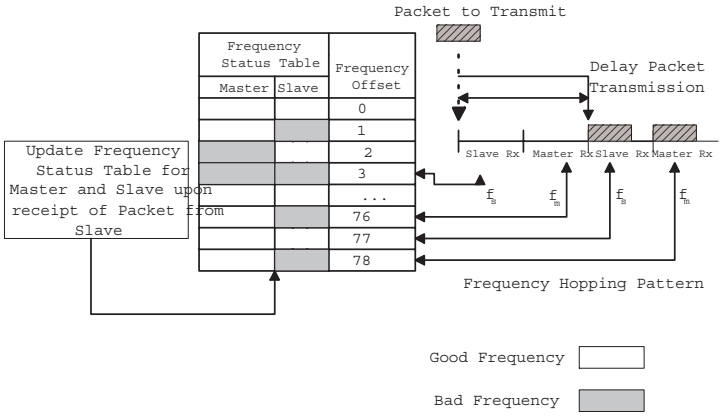


Fig. 1. Interference Aware Scheduling

This simple scheduling scheme illustrated in Figure 1 needs only be implemented in the master device and translates into the following transmission rule. *The master transmits in a slot after it veri es that both the slave's receiving frequency, f_s, and its own receiving frequency, f_m, are ”good”. Otherwise, the master skips the current transmission slot and repeats the procedure over again in the next transmission opportunity.*

Figure 2 describes the master's transmission o w diagram. In addition, to checking the slave's and the master's receiving frequencies pair, (f_s,f_m), the algorithm incorporates bandwidth requirements, and quality of service guarantees for each master/slave connection in the piconet. This bandwidth allocation is combined with the channel state information and mapped into transmission priorities given to each direction in the master/slave communication. It is shown in the ”choose slave” routine in the o w diagram. Note that the master invokes the ”choose” routine after serving the retransmission and ACK queues for packets sent by the master requiring retransmissions and packets received by the master requiring acknowlegments respectively.

In the remainder of this section, we discuss (a) a dynamic channel estimation procedure, (b) a credit allocation function, and (c) a service priority routine that schedules packet transmissions to devices according to their service requirements and the state of the channel.

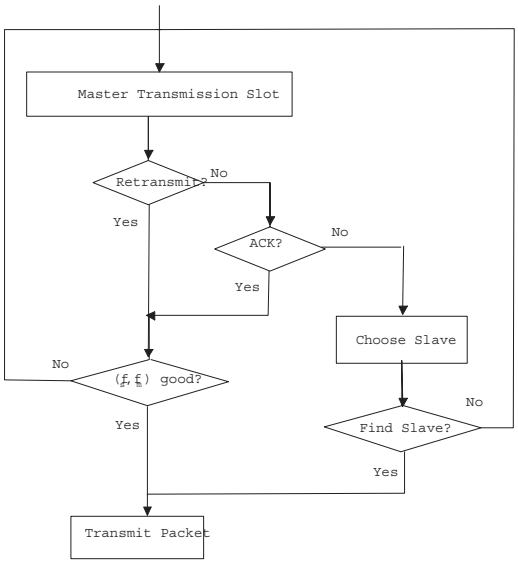


Fig. 2. Master Packet Transmission Flow Diagram

A. Dynamic Channel Estimation

Estimation is mainly based on measurements conducted on each frequency or channel in order to determine the presence of interference. Several methods are available ranging from BER, RSSI, packet loss rate, and negative ACKs. In this discussion, the estimation is based on negative ACKs, which belongs to the class of implicit methods that do not require messages to be exchanged between the master and the slave devices. First, we de ne two phases in the channel estimate procedure. During the *Estimation Window* packets are sent on all frequencies regardless of their classi cation. Note that in case no data traf c is available for transmission, POLL/NULL packets could be exchanged between the master and the slave in order to probe the channel and collect measurements. The *Estimation Window* takes place every estimation interval, EI, and is followed by an *Online* phase where the master uses only ”good” frequencies to selectively send data and POLL packets to slaves in the piconet.

Next, we give a lower bound on the *Estimation Window* and describe how to adjust EI based on the environment’s dynamics.

Estimation Window -

Since the slave does no need to send information to the master about the state of its channel, as soon as a frequency is determined to be ”bad” it can be classi ed as such right away and skipped at the next hop. A simple rule such as the number of times each frequency is visited before a frequency is classi ed can be used in order to derive the status of a frequency. Thus, the boundary between the Estimation Window and the online phase is blurred as illustrated in Figure 3. While the channel estimation procedure is still performed ever EI, the size of the Estimation Window does not need to be predetermined.

Estimation Interval - How often to update the channel estimation depends on the application and the dynamics of the scenario

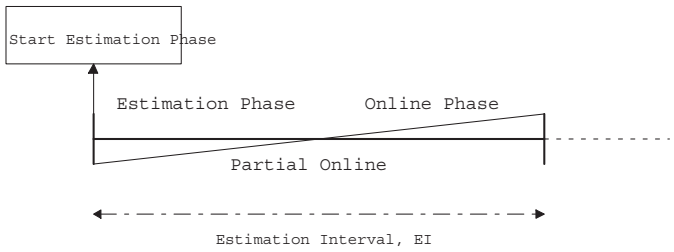


Fig. 3. Implicit Estimation

used. We propose an adaptive procedure to adjust EI, which the interval between two consecutive estimation windows.

First, we let δ, be the percentage of frequencies that change classi cation status (from "good" to "bad" or vice versa) during the previous estimation phase. More formally, let S(f,t) be the status of frequency f at time t.

$$S(f,t) = 1; \quad if\ f\ is\ "good"$$
$$S(f,t) = 0; \quad otherwise \tag{1}$$

Using the exclusive 'OR' ($\otimes$) operation between S(f,t) and S(f, t+1) represents the change of status of frequency f from time t to $t+1$. A change of status leads to a logic "1" while a no change yields a logic "0". Summing over all frequencies and dividing by the number of frequencies available, which is 79 in this case, leads to the following

$$\delta = \frac{1}{79}\sum_{f}^{79}(S(f,t) \otimes S(f,t+1)) \tag{2}$$

We can then de ne, a procedure for adapting EI. Initially, EI is set to EI_{min}. Then, EI is updated every interval, k, according to the rationale that if a change were to happen it is likely to happen again in the near future and therefore EI is set to EI_{min}. Otherwise, the window is doubled.

$$EI_{k+1} = max\ (2 * EI_k, EI_{max}); \quad if\ \delta \leq 0.1$$
$$EI_{k+1} = EI_{min} \quad otherwise \tag{3}$$

B. Credit Allocation

The credit system controls the bandwidth allocated to each device in order to ensure that no device gets more than its fair share of the available bandwidth. Thus, devices with a positive credit counter, c_i, are allowed to send data. Since the rate in the upstream can be different from the rate in the downstream, we de ne c^i_{up} and c^i_{dn} for both the upstream and downstream credits. Credits can be computed according to the upstream and downstream rates negotiated as follows:

$$c^i_{up} = \gamma^i_{up} \times N \tag{4}$$

$$c_{dn}^{i} = \gamma_{dn}^{i} \times N$$

where N is the number of slots considered in the allocation and $\gamma_{up/down}^{i} = l_{up/down}^{i} * N_{peak}^{i}/p^{i}$. Credits are decremented by the number of slots used in each data packet transmission. The transmission of POLL and NULL packets does not affect the credit count based on the rationale that credits are not required for the transmission of POLL and NULL messages. An interesting question is how to compute γ or derive it from application QOS parameters such as delay, peak bandwidth, and jitter. Let d (seconds), r (bits/s), σ (seconds) represent delay, peak bandwidth, and jitter respectively. r is part of the L2CAP QOS parameters and for some applications is negotiated between the master and the slave at connection setup. r is equal to $(N_{peak} \times E_l * 8)/(p \times 625 \times 10^{-6})$ and $\gamma = (r \times l \times 625 \times 10^{-6})/(E_l \times 8)$. Note that E_l is the number of information bytes contained in a packet of length l. Table I gives E_l corresponding to the various DH formats.

TABLE I

PACKET ENCAPSULATION RATE FOR DH PACKETS

Packet Type	l	E_l **(Bytes)**
DH1	1	27
DH3	3	183
DH5	5	339

The choice of l depends on the L2CAP packet size, k. When $k \leq E_5$, $N_{peak} = 1$ and l is such that:

$$l = \begin{cases} 1 & if \quad 0 < k \leq 27 \\ 3 & if \quad 27 < k \leq 183 \\ 5 & if \quad 183 < k \leq 339 \end{cases} \tag{5}$$

However, when $k > E_5$, higher layer packets (L2CAP) are segmented into N_{peak} packets. The aim is to nd N_{peak} equal to

$$N_{peak} = \lceil \frac{k}{E_l} \rceil \tag{6}$$

such as to minimize $N_{peak} \times l$, or the total number of slots needed. Furthermore, since master and slave transmission alternate, the end-to-end delay of a packet accounts for the segmentation and the transmission of packets in both directions. Therefore, the choice of l_{up} and l_{dn} are loosely constrained by the delay requirements as follows:

$$N_{peak} \times (l_{up} + l_{dn}) \leq \frac{d}{625 \times 10^{-6}} \tag{7}$$

where 625×10^{-6} is the length of a slot in seconds. Finally, the choice of p is determined by σ as follows.

$$2 \leq p \leq \frac{\sigma}{625 \times 10^{-6}} \tag{8}$$

where 2 is the minimum value for the poll interval since every other slot is dedicated to a master (or slave) transmission.

In case r, d, and σ cannot be determined from the application QOS, γ can be set to $1 - \sum \gamma^i$, the leftover bandwidth after having calculated γ for all other applications with known service rates ($\sum \gamma$).

C. Service Priority

The third component of the algorithm is to give an access priority to devices based on their channel conditions and their allocated credits.

We let u_i be the probability that a pair of master/slave transmission slots are "good". Thus, u_i represents the available spectrum to slave S_i, and we write:

$$\begin{aligned} u_i \quad = \quad & max((1 - 1/79), P(slave\ i\ has\ a\ good\ receiving\ frequency) \\ & \times P(master\ has\ a\ good\ receiving\ frequency)) \end{aligned} \tag{9}$$

where

$$\begin{aligned} & P(device\ i\ has\ a\ good\ receiving\ frequency) = \\ & Number\ of\ good\ Channels_i / Total\ Number\ of\ Channels \end{aligned} \tag{10}$$

We use a two-tier system with high and low priorities, denoted by A, and B respectively. Priority A is used to support delay constrained applications such as voice, MP3, and video. On the other hand, priority B, is used to support best effort connections such as ftp, http, print, email. The scheduling routine services priority A devices rst, and priority B devices second. Also, among same tier connections, we choose to give devices with fewer number of good channels the right of way over other devices that have more channels available. The priority access is determined according to a weight factor, w, that is the product of the credits and the probability of experiencing a bad frequency. w^i_{up} and w^i_{dn} are computed as follows:

$$\begin{aligned} w^i_{up} &= c^i_{up} \times (1 - u_i) \\ w^i_{dn} &= c^i_{dn} \times (1 - u_i) \end{aligned} \tag{11}$$

The master schedules a data transmission for slave i such as to maximize the product of the weights in the up and downstreams.

$$i = max^f_S(w^i_{up} \times w^i_{dn}) \tag{12}$$

To transmit a POLL packet, the master looks only at the weight function in the upstream:

$$i = max^f_S(w^i_{up}) \tag{13}$$

The selection of a slave is restricted over the set of slaves S that can receive on the master's current transmission frequency, f. Thus, any slave that experiences a "bad" channel on the current transmission frequency is not considered. Four sets of slaves are formed,

A^f_{data}, A^f_{poll}, B^f_{data}, and B^f_{poll}. A_{data} and A_{poll} represent the set of high priority connections requiring data and POLL packet transmissions respectively. Similarly, B_{data} and B_{poll} represent low priority connections. First, the algorithm tries to schedule a packet to high priority slaves in group A, then a POLL packet, before it moves to group B. The credit counters and weights are updated accordingly after every master's transmission. Table II summarizes the parameters used in the algorithm and their de nition. The algorithm's pseudocode is given in the appendix.

TABLE II

DEFINITION OF PARAMETERS USED IN THE SCHEDULING ALGORITHM

Parameters	**De nition**
$\gamma^i_{up,dn}$	rate allocated for device i in the upstream and downstream
$w^i_{up,dn}$	weight for device i
$c^i_{up,dn}$	credit for device i
N	Number of slots considered in the allocation
u^i	available frequency usage for device i

IV. PERFORMANCE EVALUATION

In this section, we present simulation results to evaluate the performance of BIAS. The experiments illustrate the algorithm's responsiveness to changes in the environment and the support of QOS. The results obtained are compared with Round Robin (RR) scheduling. Our simulation environment is based on a detailed MAC, PHY and channel models for Bluetooth and IEEE 802.11 (WLAN) as described in [8]. The parameters used in the setup vary according to the experiment. The common simulation parameters are summarized in Table III. The simulations are run for 300 seconds of simulated time unless speci ed otherwise. We run 10 trials using a different random seed for each trial. In addition, to plotting the mean value, we verify that that the statistical variation around the mean values are very small (less than 1%).

The performance metrics include the **packet loss**, the **mean access delay**, and the channel **estimation transient time**. The packet loss is the percentage of packets dropped due to interference over the total number of packets received. The access delay measures the time it takes to transmit a packet from the time it is passed to the MAC layer until it is successfully received at the destination. The delay is measured at the L2CAP layer. The estimation transient time measures the time it takes a Bluetooth device to detect the presence of a "bad" frequency, i.e. from the time a packet loss occurs until the frequency is classi ed "bad". This average is provided on a per frequency basis.

TABLE III

COMMON SIMULATION PARAMETERS

Bluetooth Parameters	**Values**
ACL Baseband Packet Encapsulation	DH5
Transmitted Power	1 mW
WLAN Parameters	**Values**
Packet Interarrival Time	2.172 ms
Offered Load	60 % of Channel Capacity
Transmitted Power	25 mW
Data Rate	11 Mbit/s
PLCP Header	192 bits
Packet Header	224 bits
Payload Size	12000 bits

A. Experiment 1: Base Case

This experiment includes Bluetooth performance results for the reference scenario when no interference is present. It represents a base case since the effects of BIAS are quanti ed and compared against the reference scenario. It also covers different levels of interference caused by WLAN systems operating in close proximity. Thus, we examine Bluetooth's performance when 1, 2, and 3 WLAN interfering systems are operational and compare that to the ideal performance when no interference is present. Note that, the maximum number of non-overlapping channels for WLAN systems is 3, i.e. there could be up to 3 WLAN networks operating simultaneously using different non-overlapping channels. In each case, results are obtained with BIAS and RR scheduling. The bene ts of using BIAS are discussed in terms of packet loss and access delay.

Topology - We use the topology illustrated in Figure 4 that consists of 3 WLAN systems (source-sink pairs), and one Bluetooth piconet with one master and one slave device. In a rst step, we record the results of Bluetooth when no WLAN system is present. Then, we add one WLAN system at a time starting with WLAN (Source/Sink) 1, followed by WLAN (Source/Sink) 2, and 3.

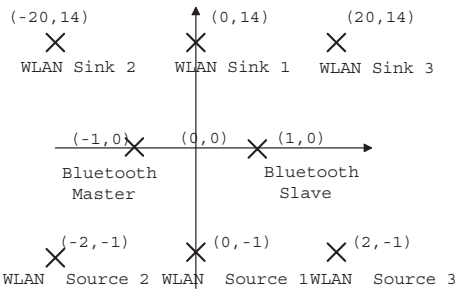


Fig. 4. Topology for Experiments 1 and 2

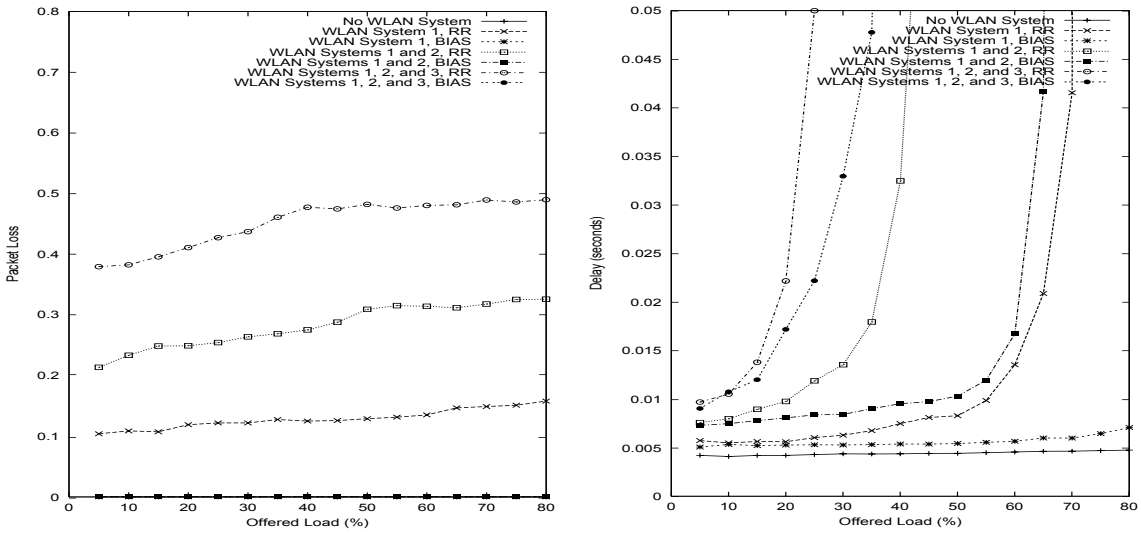


Fig. 5. (a) (b) Experiment 1. Variable Number of WLAN Interfering Systems. (a) Probability of Packet Loss. (b) Mean Access Delay

Traf c - For Bluetooth, a generic source that generates $DH5$ packets is considered. The packet interarrival mean time in seconds, t_B, is exponentially distributed and is computed according to

$$t_B = 2 \times l \times 0.000625 \times (\frac{1}{\lambda} - 1) \tag{14}$$

where l is the packet length in slots and λ is the offered load. We assume that WLAN is operating in the Direct Sequence Spread Spectrum (DSSS) mode. The WLAN source is transmitting data packets to the sink which is responding with ACKs. The WLAN packet payload is set to 12000 bits transmitted at 11 Mbit/s, while the PLCP header of 192 bits is transmitted at 1 Mbit/s. The packet interarrival time in seconds, t_W, is exponentially distributed and its mean is computed according to

$$t_W = (\frac{192}{1000000} + \frac{12224}{11000000})/\lambda \tag{15}$$

Results - Figure 5 gives the packet loss (a) and the mean access delay (b) measured at the slave for a variable Bluetooth offered load (5-80%). Observe that when no WLAN system is present, the packet loss is zero and the access delay remains at at around 4 ms. This represents a reference measure for the Bluetooth performance when there is no interference. Each WLAN system addition an increase of 15% in packet loss as shown in Figure 5(a). The packet loss is around 15%, 30% and 45% when one, two, and three WLAN systems are present respectively. Repeating the same experiments using BIAS, brings the packet loss down to zero for any number of WLAN systems. The delay trends captured in Figure 5(b) are consistent with the packet loss results. Using BIAS yields lower delays than when RR is used. When one WLAN system is present, the delay curve with BIAS is at at 5 ms (a 1 ms increase compared to the reference case when no interference is present). When 2 WLAN systems are present, the delay curve takes off at 35% with RR, while the curve remains at until 60% with BIAS. When 3 WLAN systems are present, the delay curve takes off sharply at 15% with RR, while the knee of the curve remains lower with BIAS (shifted to the right).

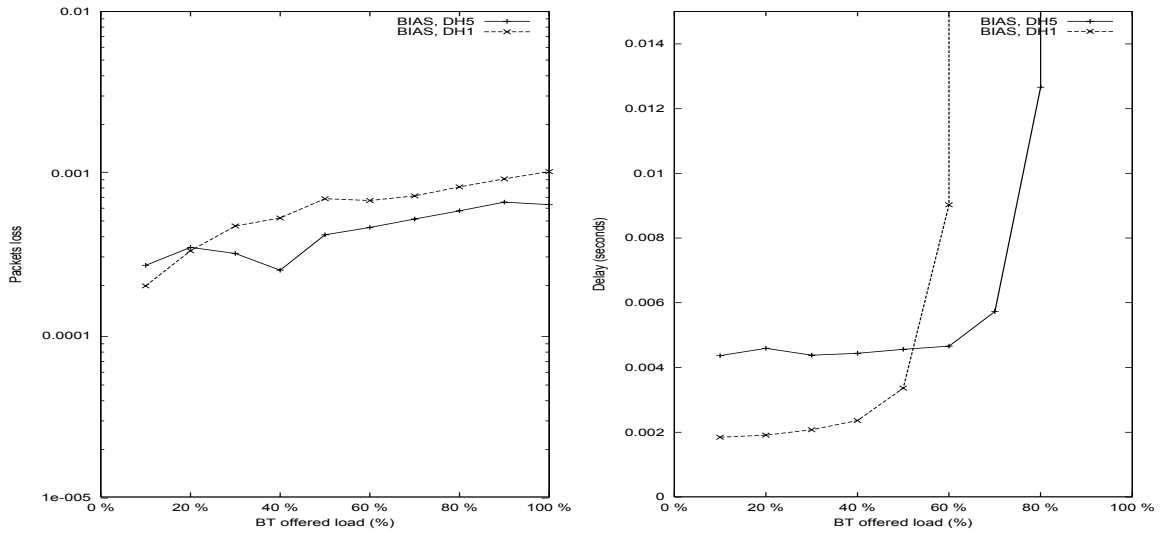


Fig. 6. (a) (b) Experiment 2. Variable Bluetooth Offered Load. (a) Probability of Packet Loss. (b) Mean Access Delay

B. Experiment 2: Dynamic Behavior

In this experiment, we focus on BIAS's responsiveness to transient effects and sudden changes in the environment. We measure the channel estimation transient time per frequency and over the entire spectrum. We design an experiment where the WLAN traf c is turned on and off several times during each simulation run (about 20 times).

Topology - We use the topology of Figure 4 with one WLAN system (Source/Sink 1) and the Bluetooth master/slave pair.

Traf c - The offered load for Bluetooth is varied between 5 and 100%, while for WLAN the offered load is set to 60%. For Bluetooth, both DH1 (1 slot) and DH5 (5 slots) packets are used in order to compare the difference in transient times. The interarrival time is computed according to Equations 14 and 15. In addition, the time the WLAN connection is on, T_{ON}, is exponentially distributed with a mean equal to 30 seconds, while the time the WLAN connection is off, T_{OFF}, is also exponentially distributed with mean equal to 60 seconds. Each simulation is run for 1800 seconds. In addition, we set $E_{min} = 5$ seconds, $E_{max} = 900$ seconds, and $\alpha = 0.9$.

Results - Figure 6(a) and (b) give the packet loss and access delay respectively measured at the Bluetooth slave device. The packet loss is negligible (less than 0.1%) for both DH1 and DH5 packets. The delay for DH1 packets is lower than the delay for DH5 packets for offered loads under 40% (it is around 2 ms for DH1 packets, and 4 ms for DH5 packets). The knee of the curve for DH5 packets is located around 70% of the offered load while it is at 50% for DH1 packets. Figure 7 gives the time it takes to estimate a "bad" frequency using DH1 and DH5 packets. The use of DH5 packets leads to a slower hopping rate and therefore increases the transient times, up to 7.5 ms while it is around 900 μs for DH1 packets.

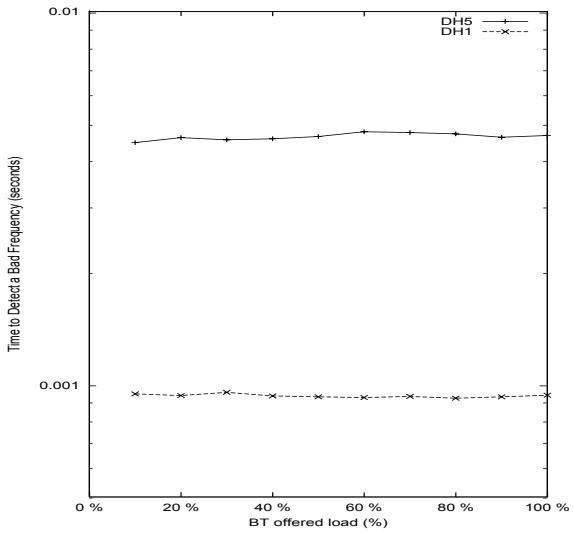


Fig. 7. Experiment 2. Variable Bluetooth Offered Load. Time to Estimate a ”Bad” Channel.

C. Experiment 3: QOS Support

This experiment highlights the support of QOS in an environment where devices experience different levels of interference and connections have a range of service requirements.

Topology - We use the topology illustrated in Figure 8. Slaves 1 and 2 experience the same level of interference, while slave 3 does not experience any interference. The y-coordinate of the WLAN FTP server is varied along the y-axis in order to vary the level of interference on the Bluetooth piconet.

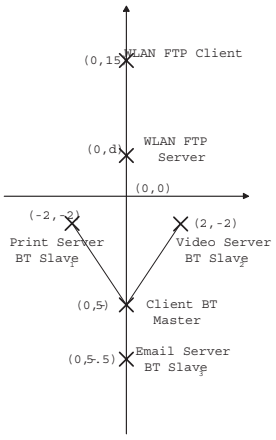


Fig. 8. Topology for Experiment 3

Traf c - For Bluetooth, we consider three application pro les, namely, Print, Video, and Email. We use print, video, and email traf c between slaves 1, 2, 3 and the master respectively. Note that the master is the client process in all three connections. The pro le parameters are given in Table IV. The WLAN uses the FTP pro le described in Table V.

TABLE IV

BLUETOOTH APPLICATION PROFILE PARAMETERS

Parameters	Distribution	Value
Email		
Send Interarrival Time (seconds)	Exponential	120
Send Group	Constant	3
Receive Interarrival Time (secondS)	Exponential	60
Receive Group	Constant	3
Email Size (bytes)	Exponential	1024
Print		
Print Requests Interarrival Time (seconds)	Exponential	30
File Size	Normal	(30000,9000000)
Video		
Frame Rate	Constant	1 Frame/s
Frame Size (bytes)	Constant	17280 (128 x 12 pixels)

TABLE V

WLAN APPLICATION PROFILE PARAMETERS

Parameters	Distribution	Value
FTP		
File Interarrival Time (seconds)	Exponential	5
File Size (bytes)	Exponential	5000000
Percentage of Get		100%

Since the video application uses 475 out of 1600 slots, we set $\gamma_{up} = 0.3$ and $\gamma_{dn} = 0.05$. The two other applications, share the leftover bandwidth ($1 - 0.35 = 0.65$). Since in a realistic environment it is often dif cult to predict the exact traf c distribution in the upstream and downstream, 0.65 is divided equally between the upstream and downstream, and each direction gets 0.17.

Results - Figure 9 depicts the results when the WLAN y-coordinate is varied between 0 and 11 meters. In Figure 9 (a), the packet loss with BIAS is below 0.5% for all three slaves. With RR, slave 1 (Print) and slave 2 (Video) vary between 15% and 2% of packet loss between 0 and 11 meters respectively. Slave 3 (Email) has a low packet loss with both BIAS and RR since it is far from the WLAN server.

The access delay for slave 2 (Video) in Figure 9(b) stays around 1.5 ms with BIAS, while it is up to ten times higher with RR (15 ms). For Print, delays with BIAS are half the delays with RR. The delays for Email are also reduced by half with BIAS (7.5 ms as opposed to 15 ms).

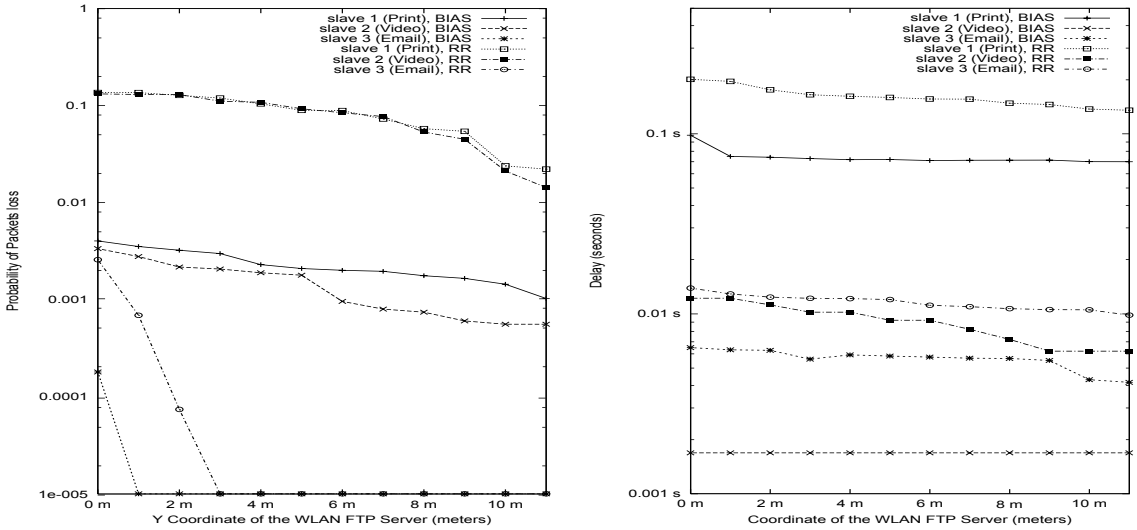


Fig. 9. (a) (b) Experiment 3. Variable Distance (a) Probability of Packet Loss. (b) Access Delay

D. Experiment 4: WLAN and Multi-Bluetooth Piconets Interference

When two or more Bluetooth piconets are proximally located, one expects few collisions when the packets happen to be transmitted on the same frequency. However, the probability of such collisions is low as discussed in [14] since each piconet has a unique frequency sequence. Given that these packet collisions are random in nature and are already mitigated by frequency hopping, we do not expect signi cant performance improvements when BIAS is used since the packet loss is already very low. Furthermore, the fact that frequencies are eliminated due to other Bluetooth piconet interference may even cause delay increases. We illustrate this particular issue using the following scenario.

Topology - We use the topology illustrated in Figure 10 representing a conference hall environment. It consists of one WLAN AP located at (0,15) meters, and one WLAN mobile at (0,0) meters. The WLAN mobile is the server device, while the AP is the client. The distance between the WLAN AP and mobile is $d_W = 15$ meters. There are ten Bluetooth piconets randomly placed, covering a disk. The center of the disk is located at (0,0) and its radius is $r = 10$ meters. We de ne d_B as the distance between a Bluetooth master and slave pair. $d_B = 1$ meter for half of the master and slave pairs, while $d_B = 2$ meters for the other half of the master and slave pairs.

Traf c - We use the application pro les available in the OPNET library and con gure the parameters according to Table VI. Four piconets use the Voice application, while four other piconets use the FTP pro le. Two piconets use the HTTP pro le. Half of the Bluetooth piconets have the slave and the master device set one meter apart, while the other half have the slave and the master device set two meters apart. The WLAN is either using the FTP pro le de ned in Table V, or the HTTP pro le de ned in Table VI.

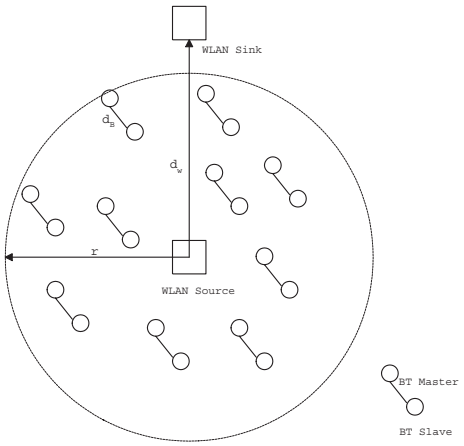


Fig. 10. Topology for Experiment 4

TABLE VI

PROFILE PARAMETERS

Parameters	Distribution	Value
Voice		
Encoder		G.711
Silence Length (seconds)	Exponential	0.65
Talk Spurt (seconds)	Exponential	0.352
Bluetooth FTP		
Percentage of Put/Get		100%
Inter-Request Time (seconds)	Exponential	5
File Size (bytes)	Exponential	250000
HTTP		
Page Interarrival Time (seconds)	Exponential	30
Number of Objects per page	Constant	2
Object 1 Size (bytes)	Constant	1000
Object 2 Size (bytes)	Uniform	(2000,100000)

Results - The results for Bluetooth are shown in Table VII and VIII for the packet loss and the access delay respectively. The results are separated by application category (FTP, HTTP, Voice), and d_B, for each of the WLAN pro les.

First, we observe that the packet loss is slightly lower with BIAS for all Bluetooth and WLAN traf c types. Second, the change is more signi cant when WLAN is set to the FTP pro le, since this latter application transmits more packets and thus causes more interference on the Bluetooth piconets. In addition, we note that the decrease in packet loss is most noticeable for $d_B = 2$ m. The packet loss for the Bluetooth FTP application goes from 17% to 8% (for RR and BIAS respectively), while for the Bluetooth voice application it goes from 16% to 5%.

Observe that when the WLAN HTTP application is used, the Bluetooth packet loss remains the same with BIAS and RR. In this

TABLE VII

BLUETOOTH PACKET LOSS PROBABILITY FOR EXPERIMENT 4

BT Traf c		WLAN Traf c			
		FTP		HTTP	
		BIAS	RR	BIAS	RR
FTP	$d_B = 1$ m	0.0309	0.0410	0.0161	0.029
	$d_B = 2$ m	0.0841	0.1705	0.0381	0.045
HTTP	$d_B = 1$ m	0.0009	0.0057	0.0024	0.0010
	$d_B = 2$ m	0.035	0.0652	0.0291	0.0309
Voice	$d_B = 1$ m	0.0023	0.0339	0.0003	0.0009
	$d_B = 2$ m	0.0581	0.1627	0.046	0.0373

TABLE VIII

BLUETOOTH MAC DELAY (SECONDS) FOR EXPERIMENT 4

BT Traf c		WLAN Traf c			
		FTP		HTTP	
		BIAS	RR	BIAS	RR
FTP	$d_B = 1$ m	0.2905	0.3749	0.1595	0.1480
	$d_B = 2$ m	0.5078	1.3942	0.2668	0.1970
HTTP	$d_B = 1$ m	0.0890	0.084	0.0829	0.0800
	$d_B = 2$ m	0.1087	0.1148	0.1155	0.0770
Voice	$d_B = 1$ m	0.0018	0.0014	0.0014	0.0013
	$d_B = 2$ m	0.0070	0.0034	0.0050	0.0015

case, the packet loss is mainly due to other piconet interference, which BIAS was not designed to mitigate.

The delays given in Table VIII are consistent with the packet loss results. We note a signi cant delay decrease (by up to fty percent) when the WLAN is set to the FTP pro le especially for the the Blueototh FTP and voice applications. The delay remains unchanged when the WLAN uses the HTTP application.

Table IX and X give the packet loss and the access delay respectively for the WLAN FTP and HTTP pro les. Observe a signi cant reduction in packet loss with BIAS for both WLAN applications, where the packet loss drops from 60% and 68% to 1% and 4% for the FTP and HTTP application respectively.

As expected, the access delay shown in Table X is improved by at least an order of magnitude.

In summary, the use of BIAS in a multi-Bluetooth piconet environment leads to performance improvements for Bluetooth when the WLAN interference is signi cant, and does not affect performance when most of the interference is due to other Bluetooth piconets. Note that BIAS always improves the performance of the WLAN since Bluetooth packets are no longer transmitted on the

TABLE IX

WLAN PROBABLITY OF PACKET LOSS FOR EXPERIMENT 4

WLAN Traf c			
FTP		HTTP	
BIAS	RR	BIAS	RR
0.0125	0.6834	0.0450	0.6174

TABLE X

WLAN MAC DELAY (SECONDS) FOR EXPERIMENT 4

WLAN Traf c			
FTP		HTTP	
BIAS	RR	BIAS	RR
0.0011	0.0154	0.000795	0.0127

frequencies used by the WLAN.

V. CONCLUDING REMARKS

In this paper we propose a scheduling technique, BIAS, aimed at eliminating interference on WLAN and alleviating the impact of interference on the Bluetooth performance. This work addresses the need to adjust to changes in the environment, support asymmetric traf c in the upstream and downstream, in addition to the use of different scheduling priorities.

The performance results obtained are summarized as follows. First, BIAS eliminates packet loss even in the worst interference case when more than 3/4 of the spectrum are occupied by other devices. Delay is slightly increased over the reference scenario (when no interference is present). This increase varies between 1 to 5 ms on average. Furthermore, BIAS is able to rapidly adjusts to changes in the channel. The channel estimation estimation time is around 7 ms and 900 μs for DH5 and DH1 packets respectively. Finally, BIAS supports QOS and maintains a low access delay for delay-sensitive traf c such as video applications.

Our future work is aimed at further investigating the performance of BIAS for additional scenarios of interest where connections with different QOS are set up and torn down over time. Mainly, we focus on studying the results' dependence on the application pro le used and the tuning of the algorithm's QOS parameters.

ACKNOWLEDGEMENTS

The author would like to thank O. Rebala and A. Tonnerre for their help in developing the simulation models and compiling the results.

REFERENCES

[1] Bluetooth Special Interest Group, "Speci cations of the Bluetooth System, vol. 1, v.1.0B 'Core' and vol. 2 v1.0B 'Pro les', " December 1999.

[2] K. J. Negus, A. P. Stephens, and J. Lansford, "HomeRF: Wireless Networking for the Connected Home," in *IEEE Personal Communications*, February 2000, pp. 20–27.

[3] IEEE Std. 802-11, "IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Speci cation ," June 1997.

[4] J. Lansford, R. Nevo, E. Zehavi, "MEHTA: A method for coexistence between co-located 802.11b and Bluetooth systems," in *IEEE P802.11 Working Group Contribution, IEEE P802.15-00/360r0*, November 2000.

[5] B. Treister, A. Batra, K.C. Chen, O. Eliezer, "Adapative Frequency Hopping: A Non-Collaborative Coexistence Mechanism," in *IEEE P802.11 Working Group Contribution, IEEE P802.15-01/252r0*, Orlando, FL, May 2001.

[6] N. Golmie, and N. Chevrollier, "Techniques to Improve Bluetooth Performance in Interference Environment," in *Proceedings of MILCOM'01*, McLean, Virginia, October 2001.

[7] N. Golmie, N. Chevrollier, and I. Elbakkouri, "Interference Aware Bluetooth Packet Scheduling," in *Proceedings of GLOBECOM'01*, San Antonio, TX, November 2001.

[8] N. Golmie, R.E. Van Dyck and A. Soltanian, "Interference of Bluetooth and IEEE 802.11: Simulation Modeling and Performance Evaluation," in *Proceedings of the Fourth ACM International Workshop on Modeling, Analysis, and Simulation of Wireless and Mobile Systems, MSWIM'01*, Rome, Italy, July 2001.

[9] N. Golmie, R.E. Van Dyck, A. Soltanian, A. Tonnerre, and O. Rebala, "Interference Evaluation of Bluetooth and IEEE 802.11b Systems," in *to appear in ACM Wireless Network, WINET*, 2002.

[10] S. Shellhammer, "Packet Error Rate of an IEEE 802.11 WLAN in the Presence of Bluetooth," in *IEEE P802.15 Working Group Contribution, IEEE P802.15-00/133r0*, Seattle, Washington, May 2000.

[11] N. Golmie and F. Mouveaux, "Interference in the 2.4 GHz ISM band: Impact on the Bluetooth access control performance," in *Proceedings of IEEE ICC'01*, 2001.

[12] I. Howitt, V. Mitter, J. Gutierrez, "Empirical Study for IEEE 802.11 and Bluetooth Interoperability," in *in IEEE Vehicular Technology Conference (VTC), Spring 2001*, May 2001.

[13] D. Fumolari, "Link Performance of an Embedded Bluetooth Personal Area Network," in *Proceedings of IEEE ICC'01*, Helsinki, Finland, June 2001.

[14] A. El-Hoiydi, "Interference Between Bluetooth Networks - Upper Bound on the Packet Error Rate," in *IEEE Communications Letters*, June 2001, vol. 5, pp. 245–247.

Appendix: BIAS Pseudocode

1: Every N Slots

2: estimate_channel();

3: compute_credits();

4: Every Even TS_f // Master Transmission Slot

5: if $TS_f + l_{dn}$ is clear // Master can receive in next slot

6: {

7: A^f_{data} = {set of high priority slaves s.t. ((f "good") and ($qsize > 0$) and ($c_{dn} > 0$) }

8: A^f_{poll} = {set of high priority slaves s.t. ((f "good") and ($c_{up} > 0$)) }

9: B^f_{data} = {set of low priority slaves s.t. ((f "good") and ($qsize > 0$)) }

10: B^f_{poll} = {set of low priority slaves s.t. ((f "good") and ($c_{up} \times c_{dn} > 0$)) }

11: // **Service high priority slaves rst**

12: if ($A^f_{data} \neq \emptyset$) // transmit data packets

13: {

14: $i = \max_{A^f_{data}} (w^i_{up} \times w^i_{dn})$ // Select device i with the largest weight

15: transmit data packet of size l_{dn} to slave i

16: $c^i_{dn,up} = c^i_{dn,up} - l^i_{dn,up}$; //decrement credit counter

17: $w^i_{dn,up} = (1 - u_i) \times c^i_{dn,up}$; // update weights

18: }

19: else if ($A^f_{poll} \neq \emptyset$) // transmit polls

20: {

21: $i = \max_{A^f_{poll}} (w^i_{up})$ // Select device i with the largest weight

22: transmit poll to slave i

23: $c^i_{up} = c^i_{up} - l^i_{up}$; //decrement credit counter

24: $w^i_{up} = (1 - u_i) \times c^i_{up}$; // update weights

25: }

26: // **Then service low priority slaves**

27: else if ($B^f_{data} \neq \emptyset$)

28: {

29: $i = \max_{B^f_{data}} (w^i_{up} \times w^i_{dn})$ // Select device i with the largest weight

30: transmit data packet of size l_{dn} to slave i

31: if ($c^i_{dn} > 0$) $c^i_{dn} = c^i_{dn} - l^i_{dn}$; //decrement credit counter

32: else $c^i_{up} = c^i_{up} - l^i_{dn}$; //decrement credit counter

33: $w^i_{dn,up} = (1 - u_i) \times c^i_{dn,up}$; // update weights

34: }

35: else if ($B^f_{poll} \neq \emptyset$) // transmit polls

36: {

37: $i = \max_{B^f_{poll}} (w^i_{up})$ // Select device i with the largest weight

38: transmit poll to slave i

39: if ($c_{up} > 0$) $c^i_{up} = c^i_{up} - l^i_{up}$; //decrement credit counter

40: else $c^i_{dn} = c^i_{dn} - l^i_{up}$; //decrement credit counter

41: $w^i_{dn,up} = (1 - u_i) \times c^i_{dn,up}$; // update weights

42: }

43: }

Bluetooth Adaptive Techniques to Mitigate Interference

N. Golmie and O. Rebala
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
Email: nada.golmie@nist.gov

Abstract—**In this paper, we investigate the use of adaptive techniques to mitigate interference for Bluetooth systems in the presence of WLAN direct sequence spread spectrum devices. We consider two different techniques that attempt to avoid time and frequency collisions of WLAN and Bluetooth transmissions. We conduct a comparative analysis of their performance for several dynamic scenarios where the WLAN interference varies over time due to either change in user activity or number of non-overlapping WLAN systems. We discuss the trade-offs involved in terms of delay, packet loss performance, and synchronization.**

I. Introduction

Recently, there has been a growing number of industry led activities focused on the coexistence of wireless devices in the 2.4 GHz band. Both, the IEEE 802.15.2 Coexistence Task Group and the Bluetooth Special Interest Group (SIG) are looking at similar techniques for alleviating the impact of interference. The proposals considered by the groups range from collaborative schemes intended for Bluetooth and IEEE 802.11 protocols to be implemented in the same device to fully independent solutions that rely on interference detection and estimation. Except for a Time Division Multiple Access (TDMA) technique aimed at time sharing the Bluetooth and 802.11 signals [1], most mechanisms considered do not require any direct communication between the protocols. These so-called non-collaborative mechanisms range from adaptive frequency hopping [2] to packet scheduling and traf c control [3]. The techniques used for detecting the presence of other devices in the band are based on measuring the bit or frame error rate, and the signal to interference ratio. For example, each device can maintain a packet error rate measurement per frequency used. Frequency hopping devices can then know which frequencies are occupied by other users of the band and thus modify their frequency hopping pattern. They can even choose not to transmit on a certain frequency if that frequency is occupied. The rst technique is known as adaptive frequency hopping, while the second technique is known as MAC scheduling or Bluetooth Interference Aware Scheduling (BIAS).

In this paper, we investigate the use of BIAS and an AFH technique. Although both techniques rely on similar methods for estimating the interference environment before a packet transmission, they differ signicantly in terms of complexity and performance. Our goal is to bring to light some of the trade-offs associated with different interference scenarios, applications, and parameters.

The remainder of this paper is organized as follows. In section II, we brie y describe BIAS and an AFH technique. In section III, we discuss the performance results obtained. In section IV, we offer concluding remarks.

II. Adaptive Interference Mitigation Techniques

Central to most interference mitigation techniques is the ability to detect the presence of other systems operating in the band. One method to estimate interference consists of measuring the percentage of packets dropped, Pr(Ploss), per frequency per receiver. Thus, given Pr(Ploss) and let's say a threshold value of 0.5, frequencies at each receiver are classi ed "good" or "bad" depending on whether their packet loss rate is less than or greater than 0.5 respectively. Also, updating Pr(Ploss) may vary according to the application, the environment, and the level of accuracy and interference tracking desired. In our simulations, we use a minimum update interval of 2 and a maximum of 100 seconds. Details on the dynamic procedure used to vary the update interval between this minimum and maximum values are found in [4].

Since in a Bluetooth piconet, the master device controls all packet transmission, the measurements collected by the slave devices are sent to the master (or implied in acknowledgement packets) that decides to (1) either avoid data transmission to a slave experiencing a "bad" frequency in the case of BIAS, and/or (2) modify the frequency hopping pattern in the case of AFH. While in the former case the decision remains local to the master, in the latter case, the master needs to communicate the changes in the hopping sequence to all slaves in the piconet in order to maintain synchronization. Thus, AFH requires the exchange of Layer Management Protocol (LMP) messages in order to advertise the new hopping sequence. Observe that this constitutes one of the major differences between BIAS and AFH.

A. Interference Aware Scheduling

The basic idea of the so-called Bluetooth Interference Aware Scheduling (BIAS) is that a data transmission to a slave experiencing a "bad" frequency is postponed until a "good" frequency is found in the hopping pattern. Furthermore, since a slave transmission always follows a master transmission, using the same principle, the master avoids receiving data on a "bad" frequency, by avoiding a transmission on a frequency preceding a "bad" one in the hopping pattern. Further details on BIAS are found in [5].

B. Adaptive Frequency Hopping

From the many AFH algorithms possible, we propose an AFH algorithm that modi es the original Bluetooth frequency

hopping scheme as follows.

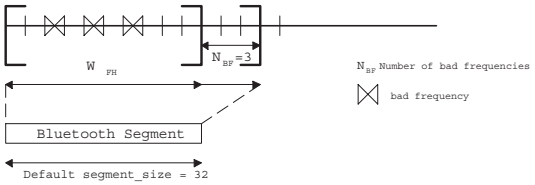


Fig. 1. Resizing the Frequency Hopping Window, W_{FH}

Given a segment of 32 "good" and "bad" frequencies that we call the Frequency Hopping Window (W_{FH}), the algorithm visits each "good" frequency exactly once. Each "bad" frequency in the segment is replaced with a "good" frequency selected from outside the original segment of 32 as shown in Figure 1. Thus, W_{FH} is increased by the number of "bad" frequencies, N_{BF}, in the original segment. Thus, the difference between AFH and the original Bluetooth hopping sequence algorithm is in the selection of only "good" frequencies in order to ll up the segment size.

Finally, AFH does not preclude additional scheduling techniques to control the transmission (and possibly the retransmission) of packets on the medium.

III. Performance Evaluation

In this section we present simulation results to evaluate the performance of the proposed AFH algorithm. We ran several experiments using different applications, traf c types, and network topologies. Given the lack of space, only a representative set of the data obtained is discussed in this paper. While the advantages of AFH may be obvious in terms of mitigating the interference between Bluetooth and WLAN in a stationary environment, we focus mainly on its dynamic behavior and its ability to adjust to time-varying and different interference levels. Although, more dif cult to solve, the scenarios selected are perhaps more realistic. A summary of the experiments is given in Table I.

TABLE I
Experiment Summary

Experiment	Traf c	Topology
1	ON/OFF Exponential	1
2	FTP/HTTP	1
3	ON/OFF Exponential	2 w/ 1 Bluetooth Piconet
4	ON/OFF Exponential	2 w/ 3 Bluetooth Piconets

A. Experiment 1: Variable Offered Load

We use Topology 1 illustrated in Figure 2 with one WLAN system (Access Point/Station) and a Bluetooth master/slave pair. The WLAN access point (AP) is located at (0,15) meters, and the WLAN station is x ed at (0,1) meters. The Bluetooth slave device is x ed at (0,0) meters and the master is x ed at (1,0) meters.

In Experiment 1, we use an on-off traf c generation model with parameters to characterize burstiness and user activity as seen at layer 2, such as packet interarrival, and packet size. For Bluetooth the packet size is x ed to either 1, or 5 slots using the

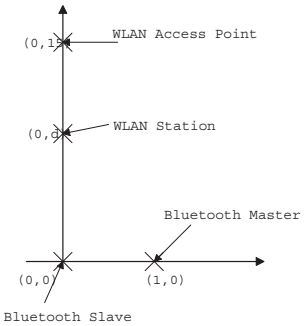


Fig. 2. Topology 1 - Two WLAN devices and one Bluetooth piconet

DH format and the mean interarrival time is computed according to

$$t_B = 2 \times n_s \times T_s/\lambda, \quad (1)$$

where λ is the offered load, n_s is the number of slots occupied by a packet. For $DH5$, $n_s = 5$. T_s is the slot size equal to 625 μs.

For WLAN, we x the packet payload to $12,000$ bits assuming an IP packet of 1500 bytes and compute the mean packet interarrival time according to

$$t_W = (\frac{192}{1,000,000} + \frac{12,224}{data\_rate})/\lambda, \quad (2)$$

where λ is the offered load, 224 is the MAC layer header, 192 is the PLCP header (always sent at 1 Mbit/s). The payload is transmitted at the data_rate of 11 Mbit/s. The offered load for Bluetooth is varied between 10 and 100%, while the WLAN offered load is set to 60%. The time the WLAN connection is ON, T_{ON}, is exponentially distributed with a mean equal to 10 seconds, while the time the WLAN connection is OFF, T_{OFF}, is also exponentially distributed with mean equal to 20 seconds. Each simulation is run for 900 seconds. Averages and con - dence intervals are obtained over 10 simulation runs.

Figure 3(a) gives the packet loss results at the Bluetooth slave. Observe that with AFH the packet loss is close to 8% and 4% for DH5 and DH1 packets respectively. These results are close to the ones obtained when no interference mitigation technique is used and depend on the frequency of the synchronization messages exchanged between the Bluetooth master and the slave. Thus there is a trade-off between the communication overhead and the response to changes in the interference environment. A fast responding system will incurr a lower packet loss at the cost of a higher communication overhead. In this experiment, synchronization messages are exchanged every 2 seconds, which is also the value used for E_{min}. On the other hand, using BIAS without AFH reduces the packet loss to less than 2% and 0.9% for DH5 and DH1 packets respectively at 10% offered load. As the offered load is increased, the packet loss even drops further to negligible levels. Since no explicit message exchange is required for BIAS, the response time to changes in the interference environment happen within a packet round trip time. Figure 3(b) illustrates the access delay results. For DH1 packets, AFH yields lower delays than BIAS. Delaying the transmission of a short packet in the case where the probability of packet collision is less than 22/79 leads to higher access delays. The access delay curve for AFH takes off at around

70% load. For DH5 packets, the delays obtained with AFH are comparable to the results obtained with BIAS loads less than 50%. However, as the offered load is increased the access delays obtained with BIAS are lower mainly due to fast response times and low packet loss.

B. Experiment 2: FTP and HTTP Pro le

In this experiment, we use Topology 1 given in Figure 2. We consider two application pro les, namely, FTP and HTTP. We use the TCP/IP stack implemented in the OPNET library and con gure the application parameters provided. For the FTP pro le, the parameters are the percentage of put/get, the inter-request time, and the le size. The percentage of put/get represents the number of times the put command is executed in an FTP connection over the total number of put and get commands, i.e., a fty percent indicates that half of the FTP commands executed are put, and the other half are get. The inter-request time is the interval between two FTP commands, and the le size represents the size of the le requested in bytes. The HTTP pro le is described by parameters characterizing a web page such as the page interarrival time, the number of objects in each page and their size in bytes. Two pro le sets are de ned for each of the Bluetooth and the WLAN in Table II.

TABLE II
PROFILE PARAMETERS

Parameters	Distribution	Value
Bluetooth FTP		
Percentage of Put/Get		100%
Inter-Request Time (seconds)	Exponential	5
File Size (bytes)	Constant	2M
WLAN FTP		
Percentage of Put/Get		100%
Inter-Request Time (seconds)	Exponential	1
File Size (bytes)	Constant	2M
HTTP		
Page Interarrival Time (seconds)	Exponential	5
Number of Objects per page	Constant	2
Object 1 Size (bytes)	Constant	10K
Object 2 Size (bytes)	Uniform	(200K,600K)

We ran two simulations where in each case both the WLAN and Bluetooth devices are using either the HTTP or the FTP pro le. Table III gives the packet loss using FTP and HTTP traf c. Observe that the packet loss with BIAS alone is an order of magnitude lower than with AFH due to the dynamic nature of the traf c. The packet loss with AFH is comparable to the packet loss obtained when no algorithm is used.

On the other hand, the FTP access delay with BIAS is slightly higher than with AFH and None in that order. This increase in delay can be attributed to the policy of delaying the transmission of packets on the so-called "bad" frequencies, although the probability that a packet collision occurs is less than 22/79 (in fact it is proportional to the offered load and the number of channels occupied by WLAN divided by the number of frequencies used by Bluetooth). There is no noticeable difference in delay for the HTTP application given that it does not generate as much traf c as the FTP application.

TABLE III
EXPERIMENT 2: BLUETOOTH PROBABILITY OF PACKET LOSS

	BIAS	AFH	None
FTP	0.005	0.037	0.059
HTTP	0.005	0.019	0.021

TABLE IV
EXPERIMENT 2: BLUETOOTH ACCESS DELAY (SECONDS)

	BIAS	AFH	None
FTP	0.0040	0.0033	0.0029
HTTP	0.0026	0.0023	0.0023

C. Experiment 3: Multi-WLAN Interference

In this experiment, our goal is to study the performance of AFH in a multi-WLAN environment, where the Bluetooth hopping sequence is close to the minimum number of hops allowed, which in our case is set to 15. We use Topology 2 illustrated in Figure 4, consisting of 3 WLAN systems (source-sink pairs). In this experiment (Experiment 3) we use one Bluetooth piconet, while in Experiment 4, we use all three Bluetooth piconets. We use the same traf c parameters described in Experiment 1. Figure 5 gives the packet loss and access delay measured at the Bluetooth slave. In this case, there are three WLAN systems occupying about 9-11 frequencies each. That leaves about 18-20 frequencies in the band to be used by Bluetooth. With BIAS, the Bluetooth piconets only transmits on "good" frequencies, and therefore has to skip approximately 1 in every 4 transmission opportunity. With AFH, the frequency hopping sequence is modi ed in order to include only "good" frequencies. Therefore, we expect signi cant throughput and delay improvements with AFH. Observe in Figure 5(a) the packet loss with AFH (DH5) is around 2%. It is negligible with AFH for DH1 packets, and BIAS for both DH1 and DH5 packets. As expected, the access delay with AFH, shown in Figure 5(b), is several orders of magnitude lower than with BIAS. The delay for DH1 packets stays around 1 ms until an offered load of 70%. The delay for DH5 packets starts at 10 ms with a steep slope between 10 and 60%. At 60% the delay reaches 100 ms. On the other hand, the delay with BIAS for DH1 packets is 15 ms at 10% load and quickly reaches 10 seconds at 20% load. For DH5 packets, the delay between 10 and 40% load is comparable to the delay obtained with AFH. However, the delay curve takes off sharply at 40%. The delay is around 10 seconds between 40 and 100% offered load.

D. Experiment 4: Multi-WLANs and Bluetooth Piconets

In this experiment we use Topology 2 illustrated in Figure 4, with all three Bluetooth piconets (two additional Bluetooth piconets to what was used in Experiment 3) to highlight the performance of AFH in an extreme case of interference where multi-Bluetooth and WLAN devices are operating in the same environment. Our goal is to verify that the performance improvements observed with AFH in the previous experiment still apply in this case.

Figure 6 gives the packet loss and access delay measured at the Bluetooth slaves and averaged over all three slaves. As ex-

pected in this case, the presence of two more Bluetooth piconets makes the interference detection more challenging. As a result the packet loss observed in Figure 6 is higher than in Figure 5. The packet loss for AFH (8%) is about an order of magnitude higher than with BIAS (around 1% and 0.8% for DH5 and DH1 packets respectively). The difference in packet loss between the two schemes is due using a reduced hopping set with AFH while increasing the number of Bluetooth piconets. Similarly, the access delay in Figure 6(b) is higher than the delay in Figure 5(b) for both schemes. The delay with AFH for DH1 packets is around 2 ms. On the other hand the delay for DH5 packets is between 20 and 100 ms for offered loads between 10 and 70%. The delay with BIAS for both types of packets is between 10 and 20 seconds.

IV. Concluding Remarks

In this paper, we study using adaptive frequency hopping for Bluetooth devices when operating in close proximity to WLAN systems. We present the details of an AFH algorithm and compared its performance to BIAS, a delay transmission method aimed at interference mitigation. A summary of our ndings is as follows. In the case of WLAN interference, when the interference levels vary over time and the channel estimation is performed often, the main advantages of AFH in terms of lowering the access delay, applies only to short packets (DH1). BIAS is more effective for longer packets (DH5) and leads to lower packet loss and comparable access delays. In the case of multi-WLANs (three non-overlapping systems), AFH keeps the delay low and doubles the throughput obtained with BIAS. This result holds even in the presence of two additional Bluetooth piconets. Thus, signi cantly reducing the frequency hopping sequence due to WLAN interference, causes only a slight increase in the probability of packet collisions among the different Bluetooth piconets and does not affect performance.

References

[1] J. Lansford, A. Stephens, and R. Nevo, "Wi-Fi (802.11b) and Bluetooth: Enabling Coexistence," in *IEEE Network Magazine*, Sept/Oct. 2001, vol. 15, pp. 20–27.

[2] B. Treister, A. Batra, K.C. Chen, O. Eliezer, "Adapative Frequency Hopping: A Non-Collaborative Coexistence Mechanism," in *IEEE P802.11 Working Group Contribution, IEEE P802.15-01/252r0*, Orlando, FL, May 2001.

[3] Carla F. Chiasserini, and Ramesh R. Rao, " Coexistence mechanisms for interference mitigation between IEEE 802.11 WLANs and bluetooth ," in *Proceedings of INFOCOM 2002*, 2002, pp. 590–598.

[4] N. Golmie, O. Rebala, and N. Chevrollier, "Bluetooth Adaptive Frequency Hopping and Scheduling," in *Proceedings of MILCOM'03*, Boston, MA, October 2003.

[5] N. Golmie, "Bluetooth Dynamic Scheduling and Interference Mitigation," in *to appear in ACM Mobile Network, MONET*, 2003.

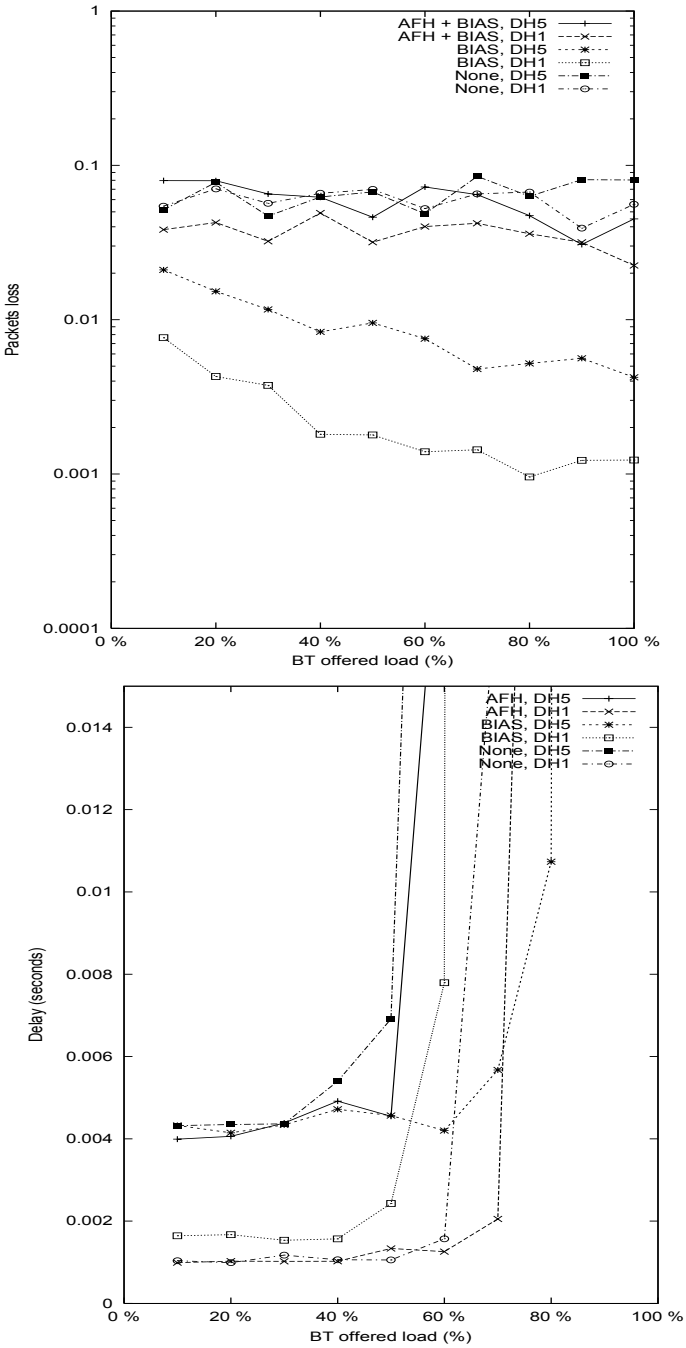


Fig. 3. $\frac{(a)}{(b)}$ Experiment 1. (a) Probability of Packet Loss. (b) Mean Access Delay (seconds)

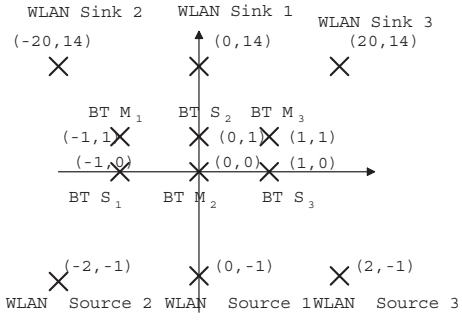


Fig. 4. Topology 2 - Multi-WLANs and Bluetooth piconets interference

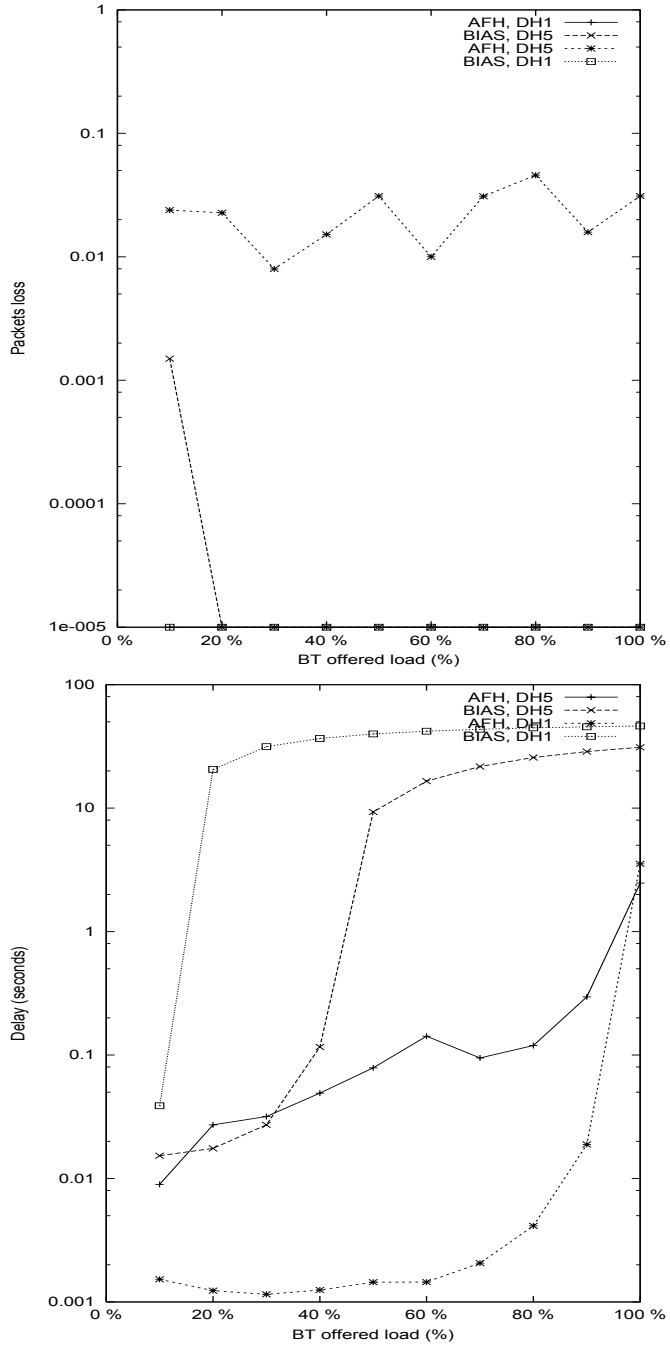


Fig. 5. $\boxed{\frac{(a)}{(b)}}$ Multi-WLANs Interference. (a) Probability of Packet Loss. (b) Mean Access Delay (seconds)

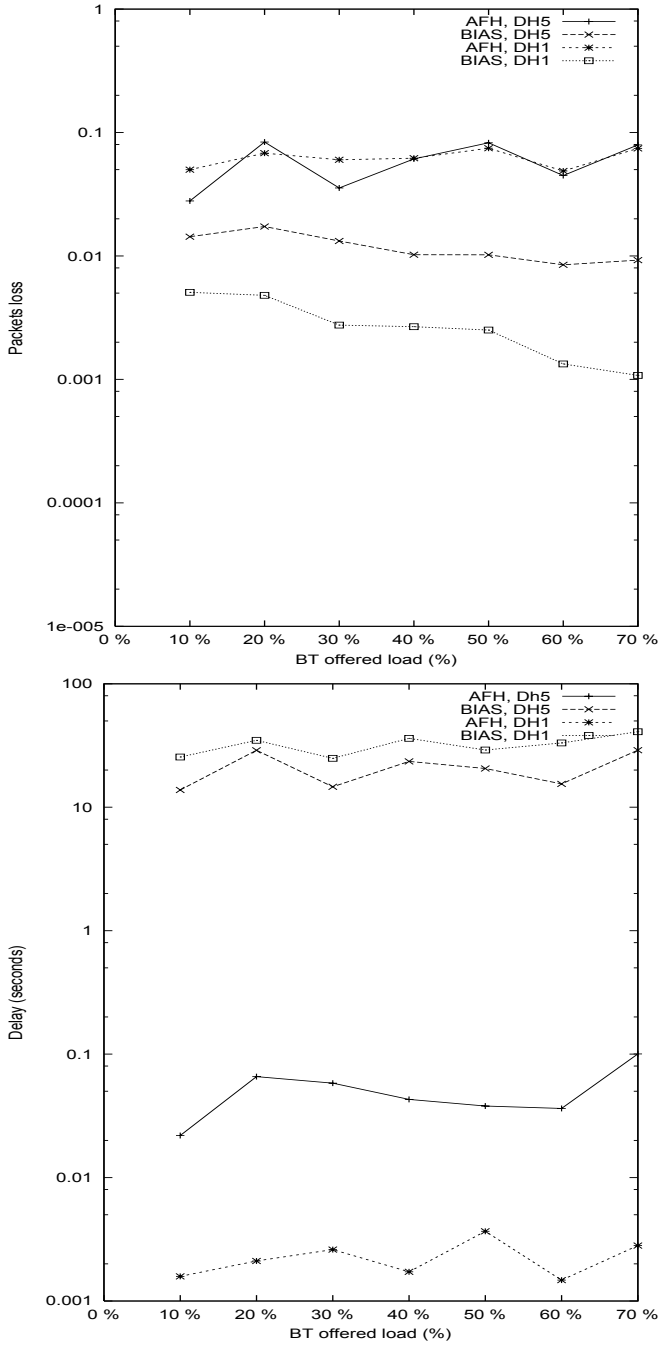


Fig. 6. $\boxed{\frac{(a)}{(b)}}$ Multi-WLANs and Multi-Bluetooth Piconets Interference. (a) Probability of Packet Loss. (b) Mean Access Delay (seconds)

Bluetooth Adaptive Frequency Hopping and Scheduling

N. Golmie, O. Rebala, N. Chevrollier
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
Email: nada.golmie@nist.gov

Abstract— **In this paper, we investigate the use of Adaptive Frequency Hopping (AFH) techniques aimed at modifying the Bluetooth frequency hopping sequence in the presence of WLAN direct sequence spread spectrum devices. We examine the conditions such as the applications, topologies, and scenarios under which AFH techniques improve performance that is measured in terms of packet loss, TCP delay, and channel ef ciency . We also compare the results obtained with AFH to others obtained using a scheduling technique that consist in delaying the transmission of a Bluetooth packet until the medium is "idle". Our results show that an obvious performance improvement with AFH is in terms of delay and throughput. AFH brings the delay down to the same level than when no interference is present. On the other hand, AFH is rather slow in responding to changes in the environment and the packet loss is more signi cant than with the scheduling. This is probably due to the limitations imposed by the communication overhead. The main dif culty for AFH is having to dynamically communicate the changes to all slaves in the piconet in order to keep the synchronization.**

I. Introduction

The deployment of different wireless devices for mobile, home, and enterprise networks, all operating in the 2.4 GHz unlicensed band, is met with growing concerns about signal interference and performance degradation. To address these challenges, a number of industry led activities have focused on the coexistence of these devices in the same environment. For example, the IEEE 802.15.2 Coexistence Task Group and the Bluetooth Special Interest Group (SIG) are looking at techniques for alleviating the impact of interference between IEEE 802.11b and Bluetooth devices.

A solution that has gained acceptance in both groups is based on modifying the frequency hopping sequence of Bluetooth in order to make it avoid direct sequence spread spectrum devices such as IEEE 802.11b. This so-called Adaptive Frequency Hopping (AFH) has gained momentum especially after the Federal Communications Commission, a US government agency in charge of telecommunication regulations, has relaxed the minimum frequency hop requirement to 15 (down from 75). AFH is expected to be included in the new release of the Bluetooth speci cations, Version 1.2.

Other proposals considered by the groups range from collaborative schemes intended for Bluetooth and IEEE 802.11 protocols to be implemented in the same device to fully independent solutions that rely on interference detection and estimation. Except for a Time Division Multiple Access (TDMA) technique aimed at time sharing the Bluetooth and 802.11 signals [1], most mechanisms considered do not require any direct communication between the protocols. For example, Bluetooth Interference Aware Scheduling (BIAS) is a MAC scheduling technique [2] that is aimed at delaying packet transmission if the medium is used by other devices. Another technique known as OverLap Avoidance (OLA) [3] uses different Bluetooth encapsulations to avoid a frequency collision between Bluetooth and 802.11.

Our goals in this paper are to investigate the use of AFH techniques aimed at modifying the Bluetooth frequency hoping sequence in the presence of WLAN direct sequence spread spectrum devices. Mainly, under what conditions – i.e., interference levels, topologies, scenarios, and applications – is it practical to use either AFH or BIAS? Which mechanisms is more effective for a given application? How fast can either technique adjust to changes in the environment? We conduct numerous simulation experiments to evaluate and quantify the operation range of AFH and BIAS. To answer the question of application sensitivity, we consider four applications, namely, voice, video, HTTP, and FTP. We set the application pro les available in the OPNET library including the details of the entire TCP/IP stack.

In section II, we describe an AFH algorithm implementation. In section III, we describe BIAS. Section IV discusses channel estimation techniques and their use with interference mitigation schemes. In section V, we consider several experiments to evaluate the performance of AFH and how it compares to BIAS. In section VI, we offer concluding remarks.

II. Bluetooth Adaptive Frequency Hopping

We devise an AFH algorithm that modi es the original Bluetooth frequency hopping scheme as follows.

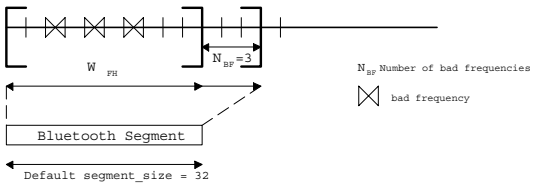


Fig. 1. Resizing the Frequency Hopping Window, W_{FH}

Given a sorted list of odd and even frequencies, and a segment of 32 frequencies, $W_{FH} = 32$, including "good" and "bad" frequencies, the algorithm visits each "good" frequency exactly once. While the segment size is the same as the one used in the current Bluetooth speci cations [4], in order to lter out the so-called "bad" frequencies, the window, W_{FH}, over which frequencies are selected is increased by the number of "bad" frequencies, N_{BF}, in W_{FH}. Thus, the main difference between the scheme we propose and the current Bluetooth speci cations

is the resizing of the interval over which frequencies are randomly selected for each segment as illustrated in Figure 1.

Note that in order for a frequency to be classi ed "bad", it has to be "bad" for at least one device in the piconet. Thus, N_{BF} represents the union of the sets of "bad" frequencies collected from of all devices.

```
1: W_FH = segment_size;          // Initialize the hopping algorithm window size
2: W_FH += N_BF;                 // Increase by the number of "bad" frequencies
3: If (W_FH > 79)
4:       W_FH = 79;              // limit to the list size
5:       N_BF = min(N_BF, 79 - H_min)
6:              //use at least H_min different frequencies
```

After, each "good" frequency is visited once, a new segment is set including 16 frequencies of the previous segment and 16 new frequencies in the sorted list.

When W_{FH} is greater than 79, the number of "good" frequencies may be less than 32 and therefore there are not enough "good" frequencies to ll in the segment. In that case, we allow each "good" frequency to be visited more than once, with the condition to use at least H_{min} different frequencies. In other words, we impose the minimum hop set to be at least equal to H_{min} different frequencies. In our simulations, we set $H_{min} = 15$.

In summary, the difference between AFH and the original Bluetooth hopping sequence algorithm is the dynamic resizing of W_{FH} based on the frequency classi cation status. The other requirement for AFH is the exchange of LMP messages between the master and the slaves in the piconet in order to advertise the new hopping sequence.

Finally, it is worth pointing out that the details presented here give an example of how "bad" frequencies can be eliminated from the Bluetooth hopping sequence. Other variants are also possible. For example, the IEEE 802.15.2 Task Group on coexistence considers a more general algorithm that allows one to choose which "bad" frequencies to keep and which to eliminate. However, for all practical scenarios considered, most AFH algorithms will give comparable performance. In fact, this is easily veri ed by implementing the AFH in [5], denoted by *AFH-IEEE*, and comparing the results obtained to the algorithm proposed in this paper.

III. Bluetooth Interference Aware Scheduling

The Bluetooth Interference Aware Scheduling (BIAS) algorithm [6] is a delay policy implemented at the master device that postpones the transmission of a packet until a slot associated with a "good" frequency becomes available. The master device, which controls all data transmissions in the piconet, uses information about the state of the channel in order to avoid data transmission to a slave experiencing a "bad" frequency. Furthermore, since a slave transmission always follows a master transmission, using the same principle, the master avoids receiving data on a "bad" frequency, by avoiding a transmission on a frequency preceding a "bad" one in the hopping pattern.

This simple scheduling scheme needs only be implemented in the master device and translates into the following transmission rule. *The master transmits in a slot after it veri es that both the slave's receiving frequency and its own receiving frequency are "good". Otherwise, the master skips the current transmission slot and repeats the procedure over again in the next transmission opportunity.*

Additional considerations including bandwidth requirements and quality of service guarantees for each master/slave connection in the piconet can also be combined with the channel state information and mapped into transmission priorities given to each direction in the master/slave communication. Details on assigning transmission priorities are given in [6].

The algorithm's general steps are summarized below.

```
1: Every Even TS_f                  // Master transmits on frequency f
2:       if TS_f + l_dn is good     // Master can receive in next slot
3:          {
4:          A^f_data = {set of slaves s.t. (( f "good") and ( qsize > 0) }
5:          if (A^f_data ≠ ∅)
6:                select slave i    //according to a priority criteria
7:                transmit data packet of size l_dn to slave i
8:          }
```

where l_{dn} is the length of the packet from the master to the slave and TS_f is the transmission slot using frequency f.

IV. Channel Estimation

Channel estimation methods include BER calculation, packet loss, or frame error rate measurements performed by each receiver (master and slave device). Since in a Bluetooth piconet, the master device controls all packet transmission, the measurements collected by the slave devices are sent to the master that decides to (1) either avoid data transmission to a slave experiencing a "bad" frequency, and/or (2) modify the frequency hopping pattern. While in the former case the decision remains local to the master, in the latter case, the master needs to communicate the change to all slaves in the piconet in order to maintain synchronization. Also, the former method falls into the scheduling policy category, while the latter is in the AFH category.

Channel estimation is based on measurements conducted on each frequency in order to determine the presence of interference. Although our discussion exclusively focuses on packet loss, other measurements can be used. In a nutshell, channel estimation works as follows. Each Bluetooth receiver maintains a *Frequency Status Table* (FST) where a percentage of packets dropped due to errors, Pr(PLoss), is associated to each frequency offset, f, as shown in Figure 2. Frequencies are classi ed "good" or "bad" depending on whether their packet loss rate is below or above a threshold value respectively. In Figure 2 the threshold value is equal to 0.5. Each slave has its own FST maintained locally. However, the master has in addition to its FST, a copy of each slave's FST.

At regular time intervals each slave updates its FST copy kept at the master using a status update message that can be de ned in the Layer Management Protocol (LMP). Alternatively, the master can derive information about each slave's FST by keeping track of the ACK bit sent in the slave's response packet.

First, we de ne two phases in the channel estimate procedure. During the *Estimation Window*, EW, packets are sent on all frequencies regardless of their classi cation. EW is followed

Status	Frequency Offset	Pr[PLoss]
good	0	10^{-3}
bad	1	0.75
bad	2	1
bad	3	0.89
	...	
good	76	10^{-4}
good	77	10^{-3}
good	78	10^{-3}

Fig. 2. Frequency Status Table

by an interval, EI, in which slaves have updated their FST at the master (refer to Figure 3). The master uses the channel information collected during EW in order rearrange the frequency hopping pattern in case of AFH and/or selectively avoid to transmit packets on so-called "bad" frequencies. In order to avoid having to manually compute or pick an arbitrary value for EW, we use a technique to dynamically adjusts the window based on the number of times, N_f, each frequency in the band should be visited. Further details on channel estimation parameter tuning are available in [6]. In our simulations, we use $N_f = 1$, $EI_{min} = 2$s, $EI_{max} = 100$s.

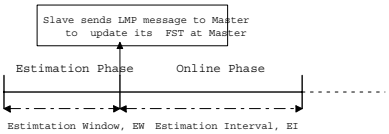


Fig. 3. Explicit Estimation

Note that during both phases, Pr(PLoss) is measured and continuously updated. Although the local FSTs can be updated every time a packet is received, the copy of the slave FST kept at the master is updated either at the end of each EW using an LMP de ned message, or every time a packet acknowledgement (ACK) is received by the master. It is important to point out that for scheduling purposes, the master can make use of the ACK feedback information as soon as it becomes available. On the other hand, AFH requires a master to slave message exchange in order to keep the piconet synchronized. In our study, we assume that updates are based on ACK feedback for BIAS and LMP messages for AFH sent at the end of each EW.

V. Performance Evaluation

In this section we present simulation results to evaluate the performance of AFH in a realistic environment. We ran several experiments using different applications, and network topologies. We consider four application pro les, namely, FTP, HTTP, voice, and video. We use the TCP(UDP)/IP stack implemented in the OPNET library and con gure the application parameters provided. For the FTP pro le, the parameters are the percentage of put/get, the inter-request time, and the le size. The percentage of put/get represents the number of times the put command is executed in an FTP connection over the total number of put and get commands, i.e., a fty percent indicates that half of the FTP commands executed are put, and the other half are get. The inter-request time is the interval between two FTP commands, and the le size represents the size of the le requested in bytes. The HTTP pro le is described by parameters characterizing a web page such as the page interarrival time, the number of objects in each page and their size in bytes. For the voice application, we use the encoding de ned in the G.723.1 speci cations. The video application uses a 1 Frame/s rate and a frame size of 17280 bytes. The application pro le parameters are summarized in Table I.

TABLE I
Application Profile Parameters

Parameters	Distribution	Value
Bluetooth FTP		
Percentage of Put/Get		100%
Inter-Request Time (seconds)	Exponential	5
File Size (bytes)	Constant	2M
Bluetooth HTTP		
Page Interarrival Time (seconds)	Exponential	5
Number of Objects per page	Constant	2
Object 1 Size (bytes)	Constant	10K
Object 2 Size (bytes)	Uniform	(20K,600K)
Bluetooth Voice		
Encoder		G.723.1
Bluetooth Video		
Frame Rate	Constant	1 Frame/s
Frame Size (bytes)	Constant	17280 (128 x 120 pixels)
WLAN FTP		
Percentage of Put/Get		0%
Inter-Request Time (seconds)	Exponential	1
File Size (bytes)	Constant	2M
Connection Duration (seconds)	Exponential	20
Interval between Connections (seconds)	Exponential	20

For each network topology considered, we run a set of 16 simulations covering each application and algorithm combination. *None* refers to the case when no algorithm is present, while *BIAS* and *AFH* refer to using BIAS and AFH respectively. Note that *AFH-IEEE* refers to the AFH algorithm included in the draft IEEE Recommended Practice on Coexistence [5]. Performance is measured in terms of the packet loss, the delay measured at the TCP layer (in seconds), and the channel ef cienc y. The channel ef cien y measures the normalized number of data packets received minus the number of packets lost and packets ignored in the case of duplicate transmissions. Averages are obtained and reported for each simulation set consisting of 10 simulation runs. Each simulation is run for 900 seconds. The packet loss and channel ef cienc y are measured at the application client (master device), while the TCP access delay is measured at the application server (slave device).

A. Experiment 1: WLAN Interference

We use Topology 1 illustrated in Figure 4 with one WLAN system (Access Point/Station) and a Bluetooth master/slave pair. The WLAN access point (AP) is located at (0,15) meters, and the WLAN station is x ed at (0,1) meters. The Bluetooth slave device is x ed at (0,0) meters and the master is x ed at (1,0) meters.

In this case, the WLAN station is "uploading" les to WLAN server using the FTP *put* command. A summary of the application pro le is described in Table I.

Table II gives the performance of the Bluetooth FTP application. First, observe that the results with AFH and AFH-IEEE are comparable and therefore in our discussion we will not distinguish between the two algorithms unless speci ed otherwise.

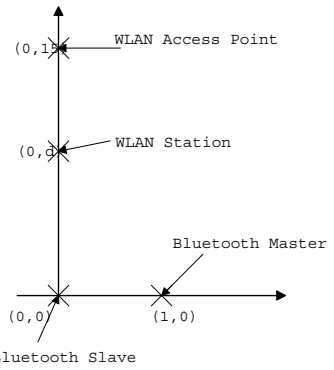


Fig. 4. Topology 1 - Two WLAN devices and one Bluetooth piconet

TABLE II
EXPERIMENT 1: BLUETOOTH FTP PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.1633	0.0009	0.0748	0.0721
TCP Delay (seconds)	0.0201	0.0178	0.0167	0.0184
Channel Ef cienc y	0.6921	0.9981	0.9306	0.9336

When no interference mitigation algorithm is present, which represents a base case, the packet loss is around 16%. The effects of BIAS are summarized in comparison to the base case as follows. First, we observe a decrease in packet loss to negligible levels, a decrease of 3 ms in the delay (from 20.1 to 17.8 ms), and an increase of 30% in the ef cienc y. Similarly the effects of AFH are characterized by a lower packet loss (to 7%), lower delay (16.7$\sim$ 18.4ms), and higher ef cienc y ($\sim$93%). The delays observed with BIAS and AFH are almost comparable, while the difference in ef cienc y is more striking. Although more packets are sent with AFH, they are more likely due to duplicate transmissions.

The observations noted for FTP are also consistent with the HTTP results given in Table III. Similarly, BIAS reduces the packet loss to zero, the access delay by 6 ms (to 11 ms), and increases the ef cienc y by 30% (to 99%). On the other hand, AFH gives a packet loss of 5%, reduces the delay by 3 ms (to $\sim$ 10 ms) and increases ef cienc y by 20% (to $\sim$ 95%).

The results of the video application are shown in Table IV. Here again, BIAS reduces the packet loss to a negligible level,

TABLE III
EXPERIMENT 1: BLUETOOTH HTTP PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.1487	0.0012	0.0585	0.0445
TCP Delay (seconds)	0.0171	0.0112	0.0109	0.0107
Channel Ef cienc y	0.6943	0.9976	0.9453	0.9557

TABLE IV
EXPERIMENT 1: BLUETOOTH VIDEO PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.1310	0.0043	0.0455	0.0269
Channel Ef cienc y	0.6974	0.9914	0.9503	0.9611

TABLE V
EXPERIMENT 1: BLUETOOTH VOICE PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.1359	0.0091	0.0400	0.0212
Channel Ef cienc y	0.6901	0.9840	0.9631	0.9722

and increases the ef cienc y to 99%. On the other hand, AFH causes a decrease in packet loss to 4.5% and 2.6% for AFH and AFH-IEEE respectively (down from 13%).

Table V shows the results of the voice application. We observe a packet loss of 4 and 2% with AFH and AFH-IEEE respectively compared to 0.9% with BIAS. The channel ef cienc y is 98%, 96%, and 97% for BIAS, AFH, and AFH-IEEE respectively.

For AFH, the time it takes to estimate the channel and communicate the changes is usually longer than for BIAS leading to a higher packet loss and a lower channel ef cienc y. This signi es that a number of packets transmitted are due to duplicate transmissions that end up getting discarded at the destination and therefore do not lead to a higher goodput. This observation captures the essence of the performance trade-offs between AFH and BIAS. AFH increases the total number of packets sent at the cost of higher packet loss, and lower ef cienc y. This may be acceptable for some bandwidth hungry applications such as FTP and HTTP, but perhaps less desirable for real-time applications such as voice and video. In summary, there are de nite trade-offs for using AFH versus BIAS depending on the application considered.

B. Experiment 2: Multi-WLAN Interference

In this experiment, our goal is to study the performance of AFH in a multi-WLAN environment, where the Bluetooth hopping sequence is further reduced. We use Topology 2 illustrated in Figure 5, consisting of 2 WLAN systems (source-sink pairs) operating on non-overlapping frequencies (each WLAN system operates on a different center channel). We use the same traf c parameters described in Table I.

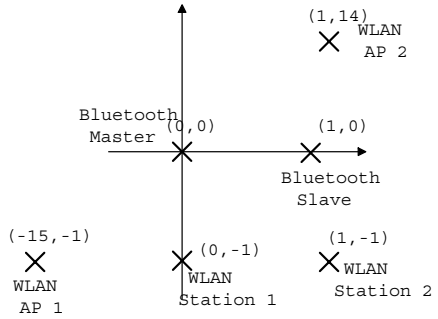


Fig. 5. Topology 2 - Multi-WLANs and Bluetooth piconets interference

Since there are two WLAN systems occupying about 16 frequencies each, that leaves about 47 frequencies in the band to be used by Bluetooth. With BIAS, the Bluetooth piconet only transmits on "good" frequencies, and therefore has to skip approximately 1 in every 3 transmission opportunities. With AFH,

TABLE VI
EXPERIMENT 2: BLUETOOTH FTP PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.3431	0.0183	0.1524	0.1542
TCP Delay (seconds)	0.0322	0.0213	0.0218	0.0242
Channel Ef cienc y	0.4500	0.9684	0.8486	0.8552

TABLE VII
EXPERIMENT 2: BLUETOOTH HTTP PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.2535	0.0169	0.1350	0.1172
TCP Delay (seconds)	0.0181	0.0191	0.0160	0.0152
Channel Ef cienc y	0.4725	0.9705	0.8668	0.8849

the frequency hopping sequence is modi ed in order to include only "good" frequencies. Therefore, one expects signi cant throughput and delay improvements with AFH. Our goals in this experiment are to verify that our previous conclusions about AFH and BIAS still hold even in the case of severe interference.

Table VI gives the performance results for the Bluetooth FTP application. The packet loss when no algorithm is present is around 34% for Bluetooth. Note that it is more than double the packet loss obtained in Experiment 1. The packet loss is 1.8%, 15.24%, 15.42% with BIAS, AFH, and AFH-IEEE respectively. Delays with AFH and BIAS are comparable (21 ms). On the other hand, the channel ef cienc y is only 84% and 85% with AFH, while it is around 96% with BIAS.

Table VII gives the results for the Bluetooth HTTP application. The results are consistent with the FTP results for the most part. There are additional delay improvements with AFH.

Tables VIII and IX give the results for the video and voice applications respectively. The general trends observed in Experiment 1 are still valid. In general, BIAS leads to lower packet loss and higher or equal channel ef cienc y than AFH.

VI. CONCLUDING REMARKS

In this paper, we study using adaptive frequency hopping for Bluetooth devices when operating in close proximity to WLAN systems. We present the details of an AFH algorithm and compare its performance to BIAS, a delay transmission method aimed at interference mitigation.

A summary of our ndings is as follows. For the applications considered, BIAS leads to a lower packet loss and an equal or higher channel ef cienc y than AFH. Basically, when the channel estimation has to be performed often, the synchronization overhead associated with AFH leads to an additional packet loss. In fact, our results indicate that this packet loss is often accompagnied with additional duplicate packet transmissions, which in turn lead to a lower channel ef cienc y. Thus, the number of additional packets transmitted with AFH is often offset by an additional number of packets lost or ignored. In other words, the adaptive part of AFH is constrained by the channel estimation and how often to synchronize the devices in the piconet. That in turn determines the response time and the performance.

Having said that, AFH may be more suitable for slow-changing environments where the same sequence could be used for a long period of time. On the other hand, in environments where the interference levels vary more rapidly, BIAS would be the interference mitigation solution of choice.

An area of future investigations would be combining BIAS and AFH within the same scenario, where BIAS would be used to respond quickly to a change in the environment, before an AFH policy is put in place if the interference persists for a long period of time.

TABLE VIII
EXPERIMENT 2: BLUETOOTH VIDEO PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.2725	0.0230	0.1070	0.0750
Channel Ef cienc y	0.2079	0.9803	0.8485	0.8878

TABLE IX
EXPERIMENT 2: BLUETOOTH VOICE PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.2126	0.0433	0.0940	0.0564
Channel Ef cienc y	0.4543	0.9269	0.9088	0.9300

REFERENCES

[1] J. Lansford, A. Stephens, and R. Nevo, "Wi-Fi (802.11b) and Bluetooth: Enabling Coexistence," in *IEEE Network Magazine*, Sept/Oct. 2001, vol. 15, pp. 20–27.

[2] N. Golmie, "Interference Aware Bluetooth Scheduling Techniques," in *IEEE P802.11 Working Group Contribution, IEEE P802.15-01/143r0*, Hilton Head, NC, March 2001.

[3] Carla F. Chiasserini, and Ramesh R. Rao, " Coexistence mechanisms for interference mitigation between IEEE 802.11 WLANs and bluetooth ," in *Proceedings of INFOCOM 2002*, 2002, pp. 590–598.

[4] Bluetooth Special Interest Group, "Speci cations of the Bluetooth System, vol. 1, v.1.0B 'Core' and vol. 2 v1.0B 'Pro les', " December 1999.

[5] IEEE Std. 802-15 Task Group on Coexistence, "Draft Recommended Practice for Information Technology, Part 15.2: Coexistence of Wireless Personal Area Networks with Other Wireless Devices Operating in the Unlicensed Frequency Bands," March 2003.

[6] N. Golmie, "Bluetooth Dynamic Scheduling and Interference Mitigation," in *ACM Mobile Network, MONET*, 2002.

Bluetooth and WLAN Coexistence: Challenges and Solutions

N. Golmie, N. Chevrollier, and O. Rebala
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
Email: nada.golmie@nist.gov

Abstract

In this article we discuss solutions to the interference problem caused by the proximity and simultaneous operation of Bluetooth and WLAN networks. We consider different techniques that attempt to avoid time and frequency collisions of WLAN and Bluetooth transmissions. We conduct a comparative analysis of their respective performance and discuss the trends and trade-offs they bring for different applications and interference levels. Performance is measured in terms of packet loss, TCP goodput, delay, and delay jitter.

I. Introduction

The Bluetooth technology [1] is considered a Wireless Personal Area Network (WPAN) system, intended for cable replacement and short distance ad hoc connectivity. WPAN is distinguished from other types of wireless networks in both size and scope. Communications in WPAN are normally con ned to a person or object and extend up to 10 meters in all directions. This is in contrast to Wireless Local Area Networks (WLANs) employing the IEEE 802.11 speci cations [2] that typically cover a moderately sized geographic area such as a single building or campus. In this sequel, we will use WLAN and IEEE 802.11 interchangeably. WLANs operate in the 100 meter range and are intended to augment rather than replace traditional wired LANs. They are often used to provide the nal few feet of connectivity between the main network and the user.

However, instead of competing with WLANs for applications, WPANs are intended to augment many of the usage scenarios and operate in conjunction with WLANs, i.e., come together in the same laptop, or operate in proximity in an of ce or conference room environment. For example, Bluetooth can be used to connect a headset, or PDA to a desktop computer, that in turn may be using WLAN to connect to an Access Point placed several meters away.

Bluetooth and several cordless phone manufacturers plan to operate in the 2.4 GHz Industry Scienti c and Medical (ISM) unlicensed band since it is suitable for low cost radio solutions such as the ones proposed for WPANs. In addition, IEEE 802.11 [2] has standards for WLANs operating in this band as well. However, the major down side of the unlicensed ISM band is that frequencies must be shared and potential interference tolerated as de ned in the the Federal Communications Commission Title 47 of the Code for Federal Regulations Part 15 [3]. While the spread spectrum and power rules are fairly effective in dealing with multiple users in the band provided the radios are physically separated, the same is not true for close proximity radios such as IEEE 802.11 and Bluetooth that may likely come together in a laptop or a desktop. An issue of growing interest is the coexistence of these devices in the same environment.

Recently, there has been a growing number of industry led activities focused on the coexistence of wireless devices in the 2.4 GHz band. Both, the IEEE 802.15.2 Coexistence Task Group [4] and the Bluetooth Special Interest Group (SIG) are looking at similar techniques for alleviating the impact of interference. The proposals considered by the groups are intended for Bluetooth and IEEE 802.11 direct sequence spread spectrum protocols. They range from collaborative schemes to be implemented in the same device to fully independent solutions that rely on interference detection and estimation. Except for a Time Division Multiple Access (TDMA) technique aimed at time sharing the Bluetooth and 802.11 signals [5], most mechanisms considered do not require any direct communication between the protocols. These so-called non-collaborative mechanisms are intended mainly for Bluetooth since it is easier for a frequency hopping system to avoid frequencies occupied by a spread spectrum system such as WLAN. The techniques considered range from adaptive frequency hopping [6] to packet scheduling and traf c control [7]. The techniques used for detecting the presence of WLAN devices in the band are based on measuring the bit or frame error rate, the signal strength or the signal to interference ratio (often implemented as the Received Signal Strength Indicator (RSSI)). For example, each device can maintain a packet error rate measurement per frequency visited. Frequency hopping devices can then know which frequencies are occupied by other users of the band and modify their frequency hopping pattern. They can even choose not to transmit on a certain frequency if that frequency is occupied. The rs t technique is known as adaptive frequency hopping, while the second technique is known as Medium Access Control (MAC) scheduling. Other scheduling techniques known as packet encapsulation rules or OverLap Avoidance (OLA) [8], use the variety of Bluetooth packet lengths to avoid the overlap in frequency between 802.11 and Bluetooth. In other words, the Bluetooth scheduler knows to use the packet length of proper duration (1, 3 or 5 slots) in order to skip the so-called ”bad” frequency. This was shown to provide goodput improvements for both 802.11 and Bluetooth data traf c.

In this article, we investigate two solutions to the interference problem, namely, (1) an Adaptive Frequency Hopping (AFH) mechanism aimed at modifying the Bluetooth frequency hopping sequence in the presence of WLAN direct sequence spread spectrum devices [9], (2) a Bluetooth Interference Aware Scheduling (BIAS) strategy that postpones the transmission of packets on so-called “bad” frequencies [7]. Each of these two techniques considered imposes a number of implementation implications.

For example, the implication with AFH is that the chipset has to be modi ed in order to support a new Bluetooth hopping sequence that does not contain any frequencies used by WLAN. On the other hand, the backoff strategy applies to the Bluetooth master device rmw are that is responsible for transmitting packets on the medium.

The remainder of this article is organized as follows. Section II discusses interference detection methods used to determine the presence of WLAN interference. In section III and IV, we describe the backoff and AFH procedures respectively. In section V, we consider realistic scenarios to discuss performance trends and trade-offs. In section VI, we offer some concluding remarks.

II. Bluetooth Interference Estimation

Central to most interference mitigation techniques is the ability to detect the presence of other systems operating in the band, or in other words, estimate interference. Techniques that do not require interference estimation belong to the collaborative category where both the Bluetooth and WLAN protocols are implemented on the same device in order for each protocol to be aware of the traf c and packet transmissions in both the WLAN and the Bluetooth networks.

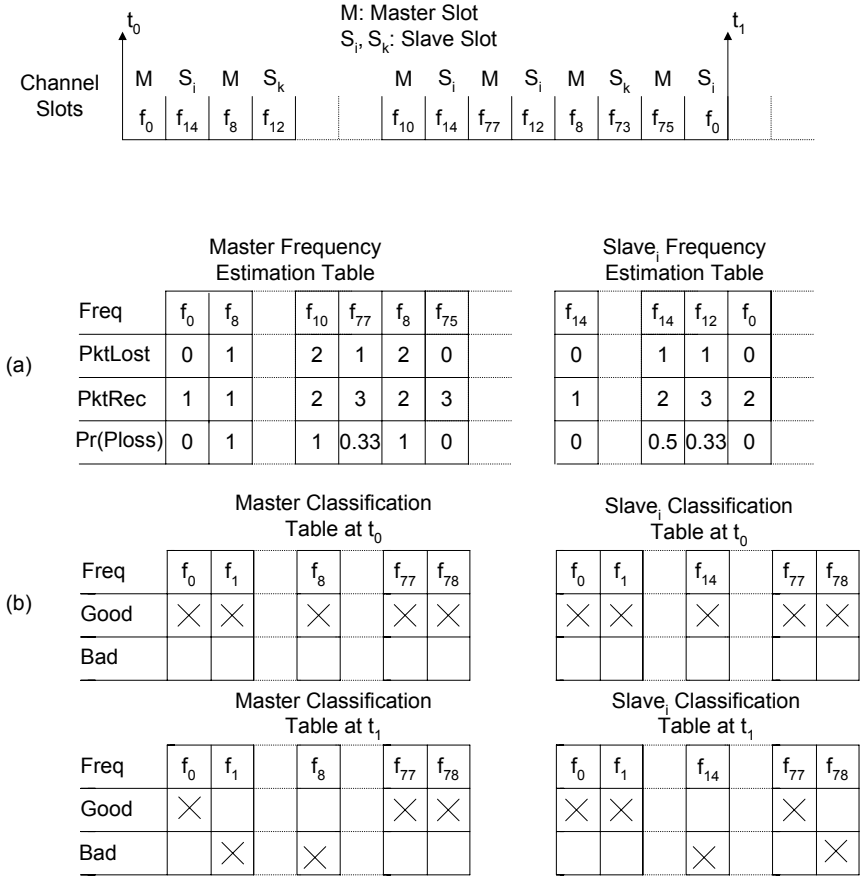


Fig. 1. Interference Estimation and Frequency Classi cation

Interference estimation methods include Signal to Interference Ratios (SIR), Bit Error Rate (BER) calculation, packet loss, or frame error rate measurements performed by a device receiver. We use packet loss measurements in our performance evaluation although other measurements can be used as well without affecting the outcome of the experiments studied. In addition, we limit our discussion to interference estimation for Bluetooth since that pertains to the solutions presented here.

In a nutshell, here is how a Bluetooth receiver detects the presence of a WLAN spread spectrum system. Measurements are collected by each receiver in the piconet since interference depends on the device location and transmitted power. These measurements consist of a percentage of packets dropped due to errors, Pr(Ploss), that is associated with each frequency in the hopset, f, as shown in Figure 1(a). Given Pr(Ploss) and a packet loss threshold, frequencies are classi ed "good" or "bad" depending on whether their packet loss rate is less than or greater than the threshold value respectively. In Figure 1(b), we use a packet loss threshold equals to 0.5.

Since in a Bluetooth piconet, the master device controls all packet transmissions, the measurements collected by the slaves are mostly useful if available at the master. There are at least two ways of sharing these measurements among the devices of the piconet. One approach would be for the master and slaves to periodically exchange their measurements via management messages.

Another method would be for the master to derive information about each slave's measurements by looking at the ACK bit sent in the slave's response packets. Observe that in this latter approach, the master can make use of the ACK feedback information as soon as it becomes available, and thus speed up the estimation time by few tens to hundreds of milliseconds depending on the traf c load and packet sizes considered. Scanning the entire frequency band using ACK feedback may take between 0.5 to 1.5 seconds depending on the application and the traf c load considered.

A nal point of observation is concerned with the classi cation update interval. Since the master uses the packet loss information collected in order to rearrange the frequency hopping pattern in case of AFH and/or selectively avoid packet transmissions on so-called "bad" frequencies, one needs to ask how often should frequencies be classi ed? If the classi cation update period is relatively short, the classi cation re ects more accurately the state of the channel at a higher communication overhead cost in case the measurements are distributed via management messages. Also, frequent classi cations may lead to a higher packet loss. On the other hand, a long classi cation period may not be able to keep up with rapid changes in the interference environment, when traf c is bursty and users are mobile. A number of techniques can be used in order to make the update interval track changes in the channel dynamics. In our evaluation, we x ed the update interval to 4 seconds in order to highlight the effects of synchronization messages.

III. Bluetooth Interference Aware Scheduling

Since the interference mitigation approach that we discuss is concerned primarily with packet scheduling and transmission in Bluetooth, we will rst give a brief overview of how packets are transmitted in Bluetooth, and we will then show how to modify the packet scheduler in order to mitigate interference.

The Bluetooth transmission channel is divided into 625 μs slots. Transmission occurs in packets that occupy an odd number of slots (1, 3, or 5). Each packet is transmitted on a different hop frequency with a maximum frequency hopping rate of 1600 hops/s in case packets occupy a single slot, and a minimum hopping rate of 320 hops/s in case packets occupy 5 slots. Note that every slot has a frequency associated with it; however transmission of a packet occupying multiple slots always uses the frequency associated with the rst slot.

A slave packet always follows a master packet transmission as illustrated in Figure 2(a), which depicts the master's view of the slotted channel. A slave needs to respond to a master's packet that is speci cally addressed to it. In case it does not have any data to send, it sends a NULL packet. Moreover, each packet contains the ACK information of the previous packet received.

Since the master is in charge of all transmissions in the piconet and chooses which slave to transmit to, it is easy to envision a scheduling policy at the master that considers the frequency classi cation information before sending packets on the medium. The so-called Bluetooth Interference Aware Scheduling (BIAS) [7] is a backoff policy that postpones the transmission of a packet until a slot associated with a "good" frequency becomes available. Here is how it works. The master continuously classi es each frequency as either "bad" or "good" based on a prede ned criterion, for example a packet loss threshold as mentioned in section II. Given a master/slave slot pair and their associated frequencies as illustrated in Figure 2(a), the master transmits in a slot after it veri es that both the slave's receiving frequency and its own receiving frequency are "good". Thus, the master avoids receiving data on a "bad" frequency, by avoiding a transmission on a frequency preceding a "bad" one in the hopping pattern. If either frequency in the pair is "bad", the master skips the current transmission slot and repeats the procedure over again in the next transmission opportunity.

Finally, Figure 2(a) shows an example of transmission priority that can be built into the master scheduler. In this case, the master schedules retransmissions rst, then data packet, and nally acknowledgment packets. Note that in all three cases the pair of frequencies associated with the master and slave slots need to be "good". Additional considerations including bandwidth requirements and quality of service guarantees for each master/slave connection in the piconet can also be combined with the channel state information and mapped into transmission priorities given to each direction in the master/slave communication. Details on assigning transmission priorities are given in [7].

IV. Bluetooth Adaptive Frequency Hopping

The key idea in BIAS is to wait for a slot associated with a "good" frequency in order to transmit a packet. The question that comes up is, can the frequency and slot association be modi ed in order to eliminate the so-called "bad" frequencies? In other words, can "bad" frequencies be replaced with "good" ones so that transmissions need not be postponed? That's the main idea in adaptive frequency hopping.

First, we describe the Bluetooth frequency hopping sequence de ned in the Bluetooth speci cations [1] , then we present an AFH algorithm that modi es it in order to mitigate interference.

Frequency hopping in Bluetooth is achieved as follows. Frequencies are sorted into a list of even and odd frequencies in the 2.402-2.480 GHz range. A segment consisting of the rst 32 frequencies in the sorted list is chosen. After all 32 frequencies in that window are visited once in a random order, a new window is set including 16 frequencies of the previous window and 16 new frequencies in the sorted list. From the many AFH algorithms possible, here is an implementation that eliminates "bad" frequencies in the sequence.

Given a segment of 32 "good" and "bad" frequencies, the algorithm visits each "good" frequency exactly once. Each "bad" frequency in the segment is replaced with a "good" frequency selected from outside the original segment of 32 as shown in

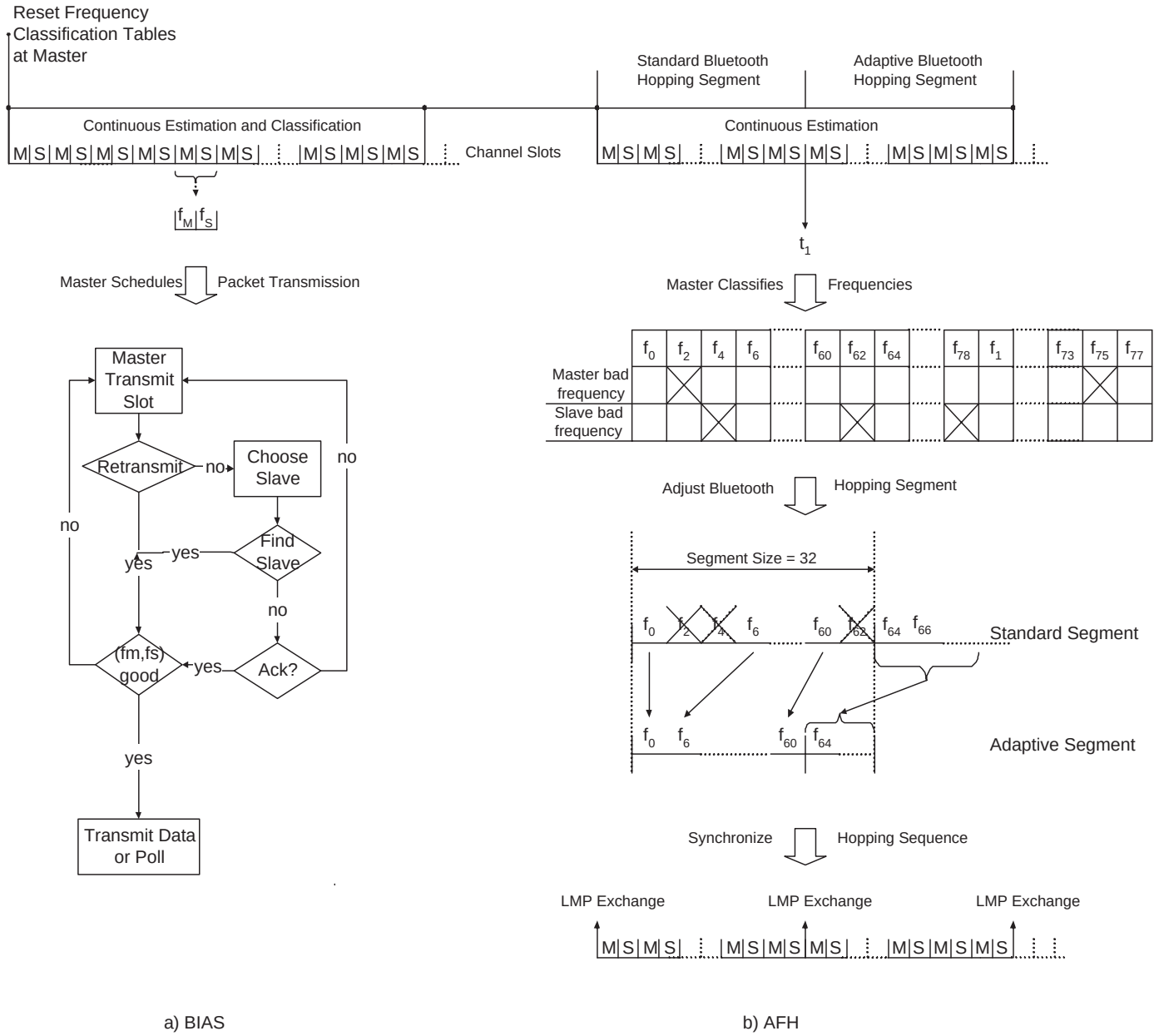


Fig. 2. Bluetooth Scheduling and Adaptive Hopping Techniques

Figure 2(b). Thus, the difference between AFH and the original Bluetooth hopping sequence algorithm is in the selection of only "good" frequencies in order to ll up the segment size. Some additional constraints can be imposed on the maximum number of "bad" frequencies to eliminate if a minimum number of different frequencies is to be kept in the sequence. In their most recent ruling the FCC recommends using at least 15 different frequencies.

Changing the frequency patterns requires changes in the Bluetooth hardware implementations. Another requirement is the advertisement of the new hopping pattern among devices in the piconet in order to keep synchronization. This is typically done using Link Management Protocol (LMP) messages exchanged between the master and the slaves in the piconet in order to advertise the new hopping sequence. This last requirement imposes some limitations on how often a new hopping pattern should be advertised and used. Improving performance such as lowering the packet loss, the access delay, and increasing the throughput should outweigh the communication overhead associated with synchronization. As suggested in section II, the synchronization update interval could be dynamically adjusted so that it tracks changes in the channel. In our simulations the LMP messages were sent twice in a 4 seconds update interval. The rst LMP message was sent when the frequency tables were reset, while the second message was sent about 1.5 seconds later to signify the use of a new hopping pattern.

Finally, AFH does not preclude additional scheduling techniques to control the transmission (and possibly the retransmission) of packets on the medium.

V. Performance Evaluation Results

We present simulation results to evaluate the performance of Bluetooth and WLAN and discuss some of the trade-offs associated with the backoff and the frequency hopping schemes presented earlier. Our simulation environment is based on detailed MAC, PHY and channel models for Bluetooth and IEEE 802.11 (WLAN) as described in [10]. The channel model consists of a geometry-based propagation model for the signals, as well as a noise model based on Additive white Gaussian noise (AWGN). For the indoor channel, we apply a propagation model consisting of two parts: (1) line-of-sight propagation (free-space) for the rst 8 meters, and (2) a propagation exponent of 3.3 for distances over 8 meters [11]. The transmitters, channel, and receivers are implemented at complex baseband. We develop models for the Bluetooth and the IEEE 802.11 access protocols using the OPNET network simulator and con gure the applications available in the simulator library.

In general, we nd that performance results vary according to the network con guration, usage scenario and application considered [10]. In this paper, we vary the application and the interference level considered, as these two factors are most likely to dominate the performance results.

For Bluetooth, we consider two applications, FTP and voice. FTP is a bandwidth hungry application that stresses the throughput requirement, while voice has strict delay and jitter requirements. Together, these two applications constitute a representative set of the application space used in a Bluetooth piconet. For WLAN, we use FTP to upload a large le (for instance a movie) to a server. For the FTP pro le, the parameters are the inter-request time and the le size. The inter-request time is the interval between two FTP commands, and the le size represents the size of the le requested in bytes. For Bluetooth we vary the le sizes from 200 bytes to 500 Kbytes (every 5 seconds), while for WLAN we use a single le of 960 Mbytes. The voice application used in Bluetooth is based on the G.723.1 encoder (with silence). The simulation and pro le parameters are given in Table I.

TABLE I
Simulation Parameters

Simulation Parameters		**Values**
Propagation delay		5 μs/km
Length of simulation run		1600 seconds
Bluetooth Parameters		
ACL Baseband Packet Encapsulation		DH5
Transmitted Power		1 mW
WLAN Parameters		
Transmitted Power		25 mW
Packet Header		224 bits
Packet Payload		12,000 bits
Application Pro le Parameters	**Distribution**	**Values**
Bluetooth FTP		
Inter-Request Time (seconds)	Exponential	5
File Size (Kbytes)	varies in	[0.2,500]
WLAN FTP		
File Size (Mbytes)	Constant	960
Bluetooth Voice		
Encoder		G.723.1
Silence Length (seconds)	Exponential	0.65
Talk Spurt (seconds)	Exponential	0.352

We use the four-device con guration shown in Figure 3 that is common to some of ce or home environments. It consists of a laptop computer connected to the Internet via WLAN, while a desktop located at a distance d from it, is also connected to either a PDA or a wireless headset over a Bluetooth link. By varying d, the level of interference on each of the Bluetooth and WLAN receivers is effected. For example, as d is increased, the level of interference is decreased. Other usage scenarios can also be obtained by putting both WLAN and Bluetooth receivers on the same device, for example the laptop computer in this case. Although some variations in the performance results are to be expected, the differences in the results remain minor.

Now, we discuss the details of two experiments involving a voice and an Ftp application for Bluetooth and an Ftp application for WLAN. For each experiment we set d=1 and 3 meters. In addition, in experiment 1, we vary the le size of the Bluetooth FTP application. Each data point collected is averaged over 15 simulation trials using a different random seed for each trial. In addition to the mean value, we verify that statistical variation around the mean values are small and fall within a 95% con dence interval.

A. Effects on Bluetooth Data Traf c

In this experiment, we consider the effects of BIAS and the AFH schemes on the performance of a Bluetooth FTP connection when it is operating in close proximity to a WLAN FTP connection. While the WLAN connection is used to upload a 960 Mbytes le to a server, a Bluetooth FTP connection is used to download les (email, attachment documents) from a PDA to a desktop computer. This latter operation produces similar traf c characteristics than that of a ”HOT SYNC” even if the le sharing protocol used in that case is speci c to the PDA manufacturer.

Figure 4(a) gives the packet loss results at the Bluetooth receiver located on the desktop computer. *None* refers to the case when no algorithm is used, while *AFH* and *Scheduling* refer to the use of AFH and BIAS respectively. Also, the distance between

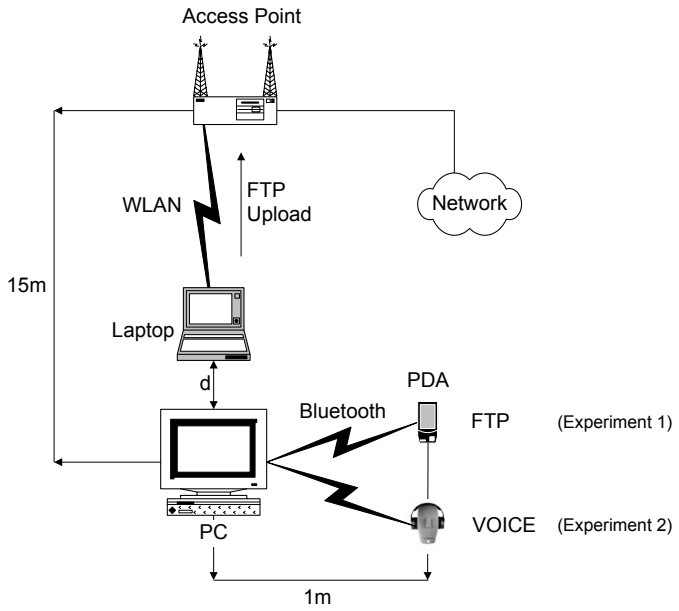


Fig. 3. Topology 1 - Two WLAN devices and one Bluetooth piconet

the Bluetooth desktop and the WLAN laptop is either *1 m* or *3 m* as indicated after the dash. First, observe that the curves are grouped into 3 distinct pairs according to the scheme used. Also, the packet loss corresponding to 1 m is always higher than the one corresponding to 3 m. This is expected since the packet loss is higher when the WLAN node is closer to the Bluetooth device. When no scheme is used the packet loss starts at 12% and 4% for 1 and 3 m respectively. The packet loss for AFH starts at 2% and increases to 6% as the offered load is increased to 800 Kbit/s. There is less than 1% difference between the packet loss for 1 and 3 m. The packet loss for BIAS is negligible and is at least two orders of magnitude lower than the ones observed for *None*.

Note that the relatively higher packet loss observed with AFH depends on the frequency of the synchronization messages exchanged between the Bluetooth master and the slave. There is a trade-off between the communication overhead and the response to changes in the interference environment. A fast responding system will incur a lower packet loss at the cost of a higher communication overhead. In this experiment, synchronization messages are exchanged on average every few seconds (1.5 and 2.5). Since no explicit message exchange is required for the scheduling algorithm, the response time to changes in the interference environment happen within a packet round trip time.

Figures 4(c) and (d) illustrate the TCP goodput and delay results respectively. Observe that the goodput is directly proportional to the offered load until about 480 Kbit/s for all 6 curves. We have computed that about 660 Kbit/s is the maximum application goodput available considering the choice of the simulation parameters. This includes a 10% overhead for the packet headers of all layers between the application and the Bluetooth baseband link and assuming a maximum TCP packet payload of 1460 bytes. Thus, 480 Kbit/s corresponds to 72% of the Bluetooth medium capacity. As the offered load is increased beyond 500 Kbit/s, the difference between the various schemes becomes more signi cant. The maximum goodput obtained is 600 and 550 Kbit/s with AFH and BIAS respectively. When no algorithm is used the maximum goodput is 480 Kbit/s.

The TCP le transfer delay shown in Figure 4(d) is consistent with the goodput results. The le transfer delay remains below 4 seconds until 500 Kbit/s for AFH and BIAS. It is 2 seconds higher when no algorithm is used. All delay curves take off sharply when the offered load increased above 500 Kbit/s.

In summary, AFH improves the maximum Bluetooth goodput by 25%, while BIAS brings only a 14% improvement. It is important to point out that in this experiment the interference level remains the same for several minutes since the WLAN connection is transmitting during the entire simulation time. Therefore, the throughput advantage brought by AFH can be further increased as the communication overhead is kept low and the channel update interval is increased to several hundred seconds. Had the WLAN traf c been more bursty, additional packet loss could have been incurred with AFH, and the throughput advantage may not have been as signi cant. On the other hand, BIAS produces a lower packet loss due to its ability to avoid frequencies that have become "bad" within a packet round trip time.

B. Effects on the Bluetooth Voice Application

While in the previous experiment, the objective was to maximize the throughput of an FTP connection, in this experiment the goal is the minimize the delay and most importantly the delay jitter for a Bluetooth voice connection. We use the same parameters used in Experiment 1 and replace the Bluetooth FTP connection with a voice connection as shown in Figure 3. Table II gives the Bluetooth performance results collected on the desktop for d =1 m. The packet loss is 11%, 2.9% and 0.6% with None, AFH

TABLE II
EXPERIMENT 2: BLUETOOTH VOICE PERFORMANCE

	BIAS	AFH	None
d=1 meter			
Probability of Packet Loss	0.0064	0.0294	0.1101
Delay (seconds)	0.0832	0.0014	0.0018
Delay Jitter (seconds)	0.0770	0.0769	0.0767
Goodput (Kbit/s)	2.9096	2.9124	2.9197
d=3 meter			
Probability of Packet Loss	0.0064	0.0155	0.0320
Delay (seconds)	0.0836	0.0015	0.0017
Delay Jitter (seconds)	0.0770	0.0764	0.0768
Goodput (Kbit/s)	2.9109	2.9332	2.9189

and BIAS respectively. Note that the delay jitter is around 76 ms with all three schemes. On the other hand, the delay measured with BIAS is 83 ms, while it is 14 and 18 ms with AFH and None respectively. This result points out the main disadvantage of BIAS in terms of increasing the access delay while lowering the packet loss. However since the delay jitter obtained with BIAS is comparable to what is obtained with AFH and None, then BIAS is still a viable option for voice applications.

The results for d=3 meters are consistent with the discussion presented earlier. In this case the packet loss is lower than with d=1 m since the Bluetooth receiver and the WLAN transmitter are further apart.

C. Effects on the WLAN Performance

Although the interference mitigation schemes presented mostly impact the performance of Bluetooth, it is equally important to consider any effects on the WLAN performance. Before we discuss the effects of the algorithms implemented for Bluetooth on the WLAN, it is important to keep in mind that in the simulation setup used, the WLAN node that is close to the Bluetooth piconet is mainly functioning as a transmitter of data packets and not a receiver. Thus, the impact of the Bluetooth interference is not as signi cant since the WLAN node only receives short ACK packets. Figure 4(b) shows the WLAN packet loss observed on the WLAN receiver located on the laptop computer. When no interference mitigation algorithm is implemented for Bluetooth, the packet loss is 17% and 10% at a distance of 1 and 3 meters respectively. The packet loss when AFH is implemented drops to 7% and 5% at d=1 and 3 m respectively. The packet loss is less than 1% with BIAS. Note that, we expect the packet loss to be more signi cant with None and AFH (up to 30% and 15% respectively) when the WLAN node is receiving long packets.

In summary, BIAS not only gives the lowest packet loss results for Bluetooth, but is also a neighbor friendly strategy for WLAN. Since “bad” frequencies can be avoided quickly that reduces the packet loss for both Bluetooth and WLAN.

VI. CONCLUDING REMARKS

In this paper, we study the use of interference mitigation techniques for Bluetooth when operating in close proximity to WLAN systems. We consider a backoff strategy (BIAS) for Bluetooth that avoids the transmission of packets in the WLAN spectrum. We also look at adapting the Bluetooth frequency hopping pattern (AFH) in order to avoid the WLAN spectrum. The former method does not require any changes to the Bluetooth speci cations. On the other hand, changing the frequency hopping pattern requires changes to the Bluetooth speci cations. The two techniques considered capture the range of solutions considered for the interference problem in the 2.4 GHz band.

Furthermore, while BIAS can be viewed as an intermediate or a temporary x to the problem, AFH is expected to be part of the next generation Bluetooth speci cations and perhaps chipsets if interoperability issues with legacy devices do not hinder its deployment and rapid market acceptance. However, taking a step back from speculative market analysis and technology hypes, our goals in this paper are to examine some of the strategies available for users and vendors and discuss the performance implications and trade-offs they bring.

A summary of our ndings is as follows. First, an obvious trade-off lies in terms of communication overhead, and performance improvement. Although partially explored in this study by imposing a synchronization interval, dynamic scenarios where the WLAN interference is intermittent may be dif cult to track using AFH. This is probably due to limitations imposed by the communication overhead. The main dif culty is having to dynamically communicate the changes to all slaves in the piconet in order to keep the synchronization. Nevertheless, the use of AFH in environments where the level of interference does not change often, brings additional performance improvements. More speci cally , AFH maximizes the throughput for bandwidth hungry applications such as FTP, most le sharing, synchronization applications where the packet loss requirement is not as stringent. On the other hand, the bene ts of AFH may not be as obvious for delay jitter and packet loss constrained applications such as voice and video, where packets are never retransmitted and the packet interarrival time is required to be relatively constant. For those applications, BIAS seems to give better performance results, mainly negligible packet loss and low delay jitters.

Finally, our results strongly suggest that no single technique could optimize performance for all scenarios and applications. Perhaps, combining BIAS and AFH could lead to widening the solution space and applying an appropriate technique for each scenario and application considered.

REFERENCES

[1] Bluetooth Special Interest Group, "Speci cations of the Bluetooth System, vol. 1, v.1.0B 'Core' and vol. 2 v1.0B 'Pro les', " December 1999.

[2] IEEE Std. 802-11, "IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Speci cation ," June 1997.

[3] Federal Communications Commission, "Title 47, Code for Federal Regulations, Part 15," October 1998.

[4] IEEE 802.15.2-2003, "IEEE Recommended Practice for Information Technology -Part 15.2: Coexistence of Wireless Personal Area Networks with Other Wireless Devices Operating in the Unlicensed Frequency Bands," 2003.

[5] J. Lansford, A. Stephens, and R. Nevo, "Wi-Fi (802.11b) and Bluetooth: Enabling Coexistence," in *IEEE Network Magazine*, Sept/Oct. 2001, vol. 15, pp. 20–27.

[6] B. Treister, A. Batra, K.C. Chen, O. Eliezer, "Adapative Frequency Hopping: A Non-Collaborative Coexistence Mechanism," in *IEEE P802.11 Working Group Contribution, IEEE P802.15-01/252r0*, Orlando, FL, May 2001.

[7] N. Golmie, "Bluetooth Dynamic Scheduling and Interference Mitigation," in *to appear in ACM Mobile Network, MONET*, 2004.

[8] Carla F. Chiasserini, and Ramesh R. Rao, " Coexistence mechanisms for interference mitigation between IEEE 802.11 WLANs and bluetooth ," in *Proceedings of INFOCOM 2002*, 2002, pp. 590–598.

[9] N. Golmie, "Bluetooth Adaptive Frequency Hopping and Scheduling," in *Proceedings of MILCOM'03*, Boston, MA, October 2003.

[10] N. Golmie, R.E. Van Dyck, A. Soltanian, A. Tonnerre, and O. Rebala, "Interference Evaluation of Bluetooth and IEEE 802.11b Systems," in *ACM Wireless Network, WINET, Vol. 9, pp. 200-211, May 2003*.

[11] A. Kamerman, "Coexistence between Bluetooth and IEEE 802.11 CCK: Solutions to avoid mutual interference," in *IEEE P802.11 Working Group Contribution, IEEE P802.11-00/162r0*, July 2000.

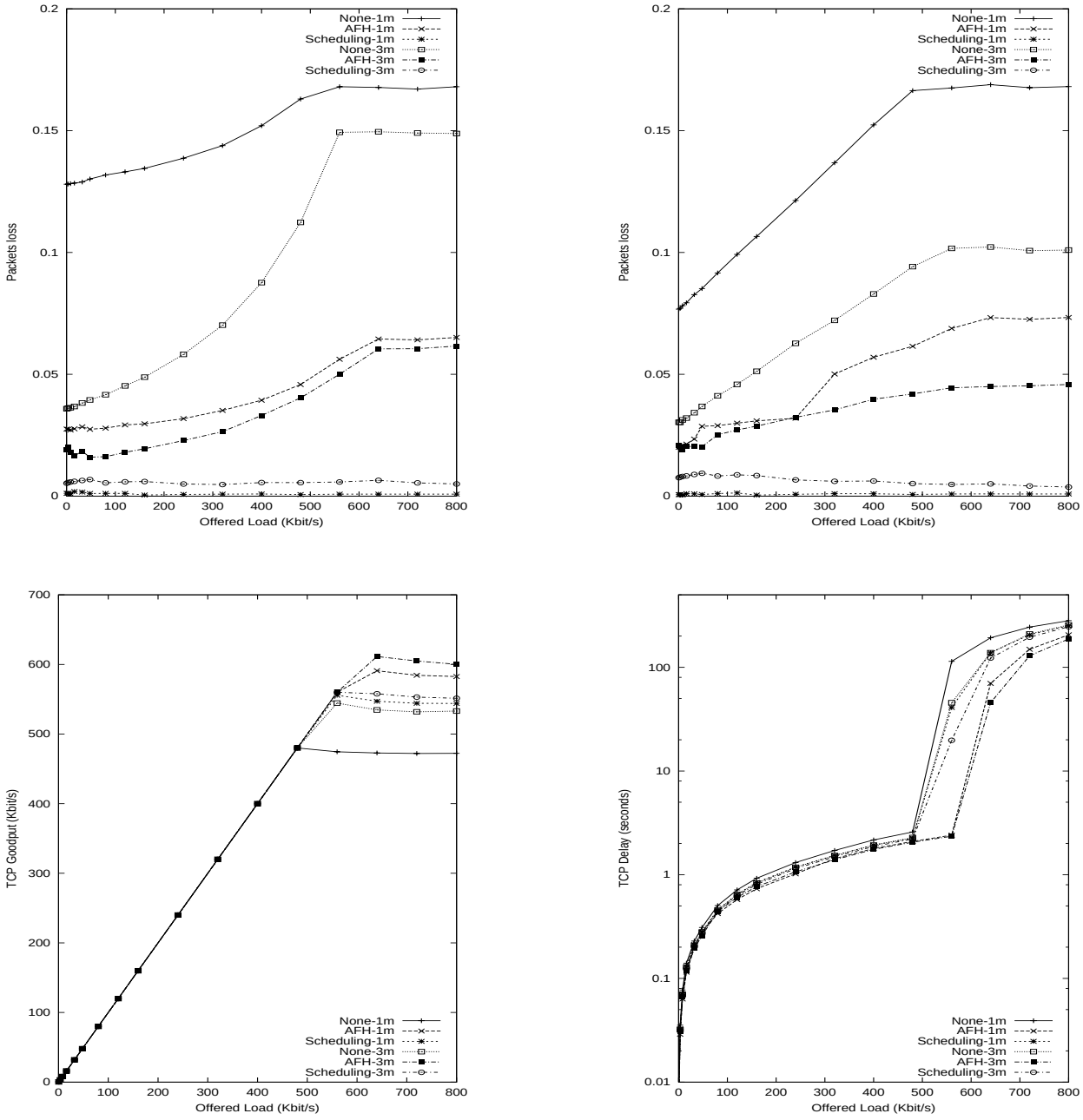


Fig. 4. $\frac{(a)\ (b)}{(c)\ (d)}$ Experiment 1. (a) Bluetooth Probability of Packet Loss (b) WLAN Probability of Packet Loss (c) Bluetooth Goodput (Kbit/s). (d) Bluetooth TCP Delay (seconds)

SUMMARY OF CONTRIBUTIONS TO WLAN-WPAN TECHNOLOGY

As part of the ITL research program in networking for pervasive computing, NIST researchers published an expansive, coherent, and focused evaluation of interference between WLAN and WPAN technology. The published results were expansive in the sense that they covered physical, access control, transport, and application layers, coherent in the sense that they considered a selected set of topologies likely to reveal useful findings, and focused in the sense that they investigated specifications for the most commonly deployed WLAN (IEEE 802.11b DSSS) and WPAN (Bluetooth) technologies. NIST researchers provided convincing evidence that interference could be quite a problem for proximal WLAN and WPAN devices. NIST researchers helped to form and then lead an IEEE 802.15 task group to consider possible coexistence strategies for WLAN and WPAN technologies. The interference results published by NIST researchers provided insights into a range of techniques for interference mitigation. NIST researchers investigated both physical layer approaches (such as receive filtering, adaptive power control, and rate scaling) and access-control approaches (such as adaptive frequency hopping and interference-aware scheduling) in isolation and combination. The findings from NIST researchers were conveyed to the IEEE 802.15 task group that considers wireless coexistence strategies, and variants of approaches proposed by NIST researchers were adopted in selected standards. As specific industry segments, such as healthcare providers, consider adoption of wireless WLAN and WPAN technologies, the evaluation approaches pioneered by NIST researchers could be applied to estimate likely interference problems and to investigate the properties of various coexistence strategies.

FIRST-GENERATION SERVICE DISCOVERY SYSTEMS

Software systems are evolving to a so-called network-centric (or net-centric) form, where distributed components are composed together dynamically and then cooperate to provide information processing in support of application requirements. This mode of operation is projected to occur throughout a pervasive computing environment, as software components acting on mobile computing devices communicate over a wireless network to rendezvous and configure into component collections in support of user needs. Once configured, component collections must monitor themselves, detect component failures, and then discover and configure replacement components. Various groups in the information-technology industry have conceived of software infrastructures to provide these functions (of discovery, configuration, and monitoring) in the form of libraries that can be used generally by distributed components. For example, engineers and researchers at Sun Microsystems conceived Jini™ Networking Technology to provide service discovery support for Java components. In addition, a team of engineers meeting in the Internet Engineering Task Force (IETF) designed and specified the Service Location Protocol (SLP) to provide service discovery functionality for Internet applications. Further, a group of designers and engineers from Microsoft and Intel devised a set of protocols and description techniques, later standardized under the auspices of the Universal Plug-and-Play (UPnP) Forum, to extend plug-and-play technology to encompass local-area networks. Several industry groups also develop service discovery technology that is more narrowly construed. For example, the Home Audio Video interoperability (HAVi) protocol provides service discovery for home entertainment and multimedia applications connected over IEEE 1394 (Firewire) networks. In addition, the Bluetooth Consortium specified a service discovery protocol that operates over top Bluetooth (or IEEE 802.15) wireless networks. Further, the Salutation Consortium defined a vertically integrated service discovery system to support office automation and related devices, such as copiers and fax machines. The fact that numerous competing designs have appeared indicates a substantial industry interest in using dynamic service discovery as a means to deploy and evolve component-based technology for pervasive computing.

Two key questions arise when considering the potential for service discovery technologies to revolutionize our ability to deploy and configure components and services in pervasive computing applications. First, what behavioral and performance characteristics should users expect from the current generation of designs for service discovery systems? Second, what techniques might be used to improve the performance of the current generation of service discovery systems? These questions are addressed in the following set of thirteen papers that document findings by researchers in the Information Technology Laboratory at NIST. The papers divide naturally into two sets. Seven papers (Paper #15 through #21) characterize the behavior and performance of designs for the first-generation of service discovery protocols. The majority of the papers focus on robustness of the designs while supporting several applications (e.g., information dissemination and real-time control) when subjected to various failures, such as message loss, communication failure, node failure, and power failure. One paper (#20) aggregates all failure-related results, providing a single characterization of failure response for the three major designs for service discovery systems. Another paper (#21) provides a complete model-based analysis and comparison of the major designs along several dimensions: functionality, structure, scalability, and service guarantees. A set of six papers (Paper #22 through #27) investigates various self-adaptive algorithms that could be implemented to improve the scalability, responsiveness, and fairness of service discovery systems. These algorithms may become key assets in pervasive computing environments, where the number of communication components can vary over a wide range within a short time.

BEHAVIORAL AND PERFORMANCE CHARACTERIZATION OF DISCOVERY SYSTEMS

The designs for various service discovery systems share a similar logical structure. All service discovery systems encompass at least three component types: a service user (SU) and a service provider (SP) and supporting service manager (SM). Component instances communicate via messages exchanged over a network. Each SU attempts to discover available SPs. Each SM advertises the availability of a set of one or more associated SPs. The primary objective of a service discovery system is to enable a SU with specific requirements for some service to rendezvous with any available SP that satisfies those requirements. The secondary objective of a service discovery system is to enable all components to monitor the availability and characteristics of other available components. This secondary objective allows, for example, a SU to determine when a new SP satisfying some requirements arrives, or to learn when a previously discovered SP is no longer available or has altered characteristics.

Designs for service discovery systems generally encompass one of three architectures. In a two-party architecture a SU can discover an available SP directly from a SM, which acts as a proxy for the SP. In a three-party architecture a SU must first discover another component, the service cache manager, or SCM, and then query that component for any available SPs. This implies that SMs will also discover SCMs on behalf of associated SPs and then deposit and maintain (on the SCMs) descriptions of the SPs. In an adaptive architecture the SUs and SMs operate in a three-party mode unless or until no SCMs can be discovered, after which the SUs and SMs switch to a two-party mode of operation. In general, when a system operates without interfering failures, any of the architectures should offer reasonable robustness. On the other hand, will the various architectures offer similar robustness when subjected to failures that could arise in a distributed system?

Numerous types of failure can interfere with the operation of a distributed system. For example, message losses could cause processes to exchange only partial information, and might also lead to situations where some communicating processes are unsure about the state of the information received by corresponding processes. More pernicious results could arise when a partial communication failure, of a transmitter or receiver, affects all communicating processes on a node. On the other hand, individual nodes could fail, taking down all processes on the node, or individual processes may fail or become subverted on particular nodes. Even more routine failures, such as power loss and restoration, present challenges for distributed systems.

Another factor may well complicate the failure response of particular service discovery systems. Service discovery systems consist of general middleware functions implemented to support application-specific logic. This implies that some failures will be resolved, if possible, by the service discovery functions, while other failures will be referred to the application software for resolution. For this reason, any fair assessment of the robustness of designs for service discovery systems must compare the designs not only under identical failure scenarios but also under identical application-specific processing, and related assumptions.

For all service discovery architectures modeled for the work reported in this publication, the NIST researchers provide identical failure models and application-specific processing. Further, where specifications permit, the researchers strive to configure the various service discovery protocols with parameter values that yield similar behavior. In addition, the models incorporate identical assumptions about characteristics of the underlying protocols – either the transmission-control protocol (TCP) or the unicast and multicast versions of the user datagram protocol (UDP) – used to exchange messages among service discovery processes. The goal of this modeling approach is to eliminate behavioral and performance differences that could be attributed to differences in failure models, application models, protocol configuration, and communication mechanisms. Under this regime, any evident performance differences should be due to differences in system architecture and protocol design.

All the models underlying the results reported in this publication represent multiple, independent nodes that execute an application supported by service discovery middleware and also by communication protocols. The service discovery middleware is modeled as a collection of independent processes, each

supporting some service discovery function. Application-specific logic is modeled as a process separate from, but interacting with, the service discovery processes. Processes within each modeled node communicate using an appropriate protocol (either TCP or unicast or multicast UDP, depending on the particular process and system design) over a communication channel with similar transmission and propagation delays.

Along with developing simulation models for various service discovery systems, applications, and supporting communication protocols, the NIST researchers had to devise metrics to compare behavior and performance. This required some degree of innovation because the literature did not previously contain any comparisons of the behavior and performance of service discovery systems. For this reason, each of the following papers defines the specific performance metrics used to compare system behavior.

In Paper #15, "Analyzing Properties and Behavior of Service Discovery Protocols Using an Architecture-Based Approach", Dabrowski and Mills define a partial set of consistency conditions that they believe a service discovery system should strive to achieve, and then they show some scenarios under which the design for one service discovery system fails to provide the specified consistency. The paper is motivated by the fact that no extant designs for service discovery systems provide a specification of consistency goals. By proposing some consistency goals and then showing how a model of a service discovery system can be evaluated against those goals, Dabrowski and Mills suggest a concrete approach to improve the specification, design, and testability of service discovery systems (and distributed systems in general). In this case, the authors uncover some specification ambiguities and omissions that could lead to feature interference and race conditions. The researchers also show, using a power-outage-and-restart scenario, how the same model used to assess logical properties can be employed to investigate performance characteristics, at least for relatively small topologies. Because the results reported in this paper were obtained using a model constructed with an architecture-description language (ADL), the authors close with a critique of the use of such models.

In Paper #16, "Understanding Consistency Maintenance in Service Discovery Architectures during Communication Failure", Dabrowski, Mills, and Elder study the ability of selected designs for service-discovery protocols to maintain consistency in a distributed system during catastrophic communication failure (e.g., jamming). The researchers use an architectural-description language, called Rapide, to model two different architectures (two-party and three-party) and two different consistency-maintenance mechanisms (polling and notification). The paper investigates performance differences among combinations of architecture and consistency-maintenance mechanism as communication-failure rate increases. The paper reports system performance along three dimensions: (1) update responsiveness (How much latency is required to propagate changes?), (2) update effectiveness (What is the probability that a node receives a change?), and (3) update efficiency (How many messages must be sent to propagate a change throughout the topology?). The paper reveals lower than expected update effectiveness for the notification mechanism over failure rates from 5% to 35%. This performance deficiency arises when temporary failures block dissemination of notifications, which rely upon retransmission within TCP, leading to a remote exception. The service discovery systems investigated do not persist in attempts to deliver notifications, but instead seem to assume that the communications failure will be detected and recovered by other discovery processes. This assumption proves unwarranted in the 5%-35% range of failure rates.

In Paper #17, "Understanding Consistency Maintenance in Service Discovery Architectures in Response to Message Loss", Dabrowski, Mills, and Elder study the ability of selected designs for service-discovery protocols to maintain consistency in a distributed system during severe message loss. This paper uses the same models and experiment design employed in Paper #16, except that they replace communication-failure rate with message-loss rate. Again, the researchers model two different architectures (two-party and three-party) and two different consistency-maintenance mechanisms (polling and notification). The paper characterizes performance (update responsiveness, effectiveness, and efficiency) differences among combinations of architecture and consistency-maintenance mechanism as message-loss rate increases. All the systems studied prove remarkably robust, providing substantial ($\geq$85%) update effectiveness even as the message-loss rate reaches 85%. The paper also finds that update

responsiveness is better at lower (≤25%) message-loss rates when using notification, but polling proves better at higher (>25% and ≤ 85%) message-loss rates. Notification also yields lower update effectiveness in the range of 25%-85% message-loss rate. Beyond 85% message-loss rate the performance of all architectures and mechanisms diminishes substantially.

In Paper #18, "Understanding Self-healing in Service-Discovery Systems", Dabrowski and Mills dissect the effectiveness of two failure-recovery mechanisms often embedded in distributed applications supported by service discovery systems. The researchers quantify the proportion of update effectiveness achieved through soft-state (heartbeat) mechanisms included in service discovery systems against the proportion of update effectiveness that can be attributed to application-level persistence (retries). In effect, this paper further examines the results reported in Paper #16 for the notification mechanism during communication failure. The aim is to better understand when to rely on soft state and when to rely on application-level persistence. The results suggest that soft state and application-level persistence provide complementary recovery mechanisms when deployed in a two-party architecture. At lower (≤30%) failure rates, application persistence provides the most value, while soft state contributes most at higher failure rates. The results also reveal that soft state and application persistence appear redundant when deployed in a three-party architecture.

In Paper #19, "Performance of Service-Discovery Architectures in Response to Node Failures", Dabrowski, Mills, and Rukhin investigate the ability of selected designs for service-discovery protocols to detect and recover from failure of remote services when used to support real-time distributed control applications. The researchers model two architectures (two-party and three-party) underlying most commercial service-discovery systems, and use simulation to quantify functional effectiveness (proportion of time an application is functional) and efficiency achieved by each of the architectures as the rate of failure increases among remote services. The results suggest that a two-party architecture yields better robustness than a three-party architecture. The paper also decomposes non-functional periods into failure-detection latency and restoration latency, which reveals that for the two-party architecture 80% of non-functional periods are due to failure-detection latency. For the three-party architecture failure-detection and restoration latency each compose about 50% of non-functional periods. This occurs because the three-party architecture depends upon the presence of SCMs; thus, is unable to recover a lost service until at least one SCM is operational.

In Paper #20, "Failure Response in First-Generation Service Discovery Systems", Dabrowski, Mills, and Quirolgico compile and present a comprehensive collection of simulation results that characterize the performance of multiple architectures (two-party, three-party, and adaptive) operating under a range of failure scenarios (node failure, communications failure, message loss, and power failure and restart) in selected applications (real-time distributed control, information dissemination, and configuration recovery). On the one hand, this paper collects results previously reported individually in Papers #15, #16, #17, and #19. On the other hand, this paper extends those results in several ways. First, the paper includes an adaptive architecture for which no results have been reported previously. The paper also reports for the first time the performance of the two-party architecture in the face of power failure and restart. Second, the paper increases the number of experiment repetitions significantly to produce performance graphs that exhibit much less noise than the graphs published in previous papers. The main aim of this paper is to provide an archival set of results characterizing failure response for state-of-the-art designs for first-generation service discovery systems.

In Paper #21, "A Model-based Analysis of First-Generation Service Discovery Systems", Dabrowski, Mills, and Quirolgico compare and contrast state-of-the-art designs for first-generation service discovery systems. The approach, unique within existing literature, first constructs a generic object-oriented meta-model and model (documented in the Unified Modeling Language, or UML) for the domain of service discovery systems. The generic model is based on an analysis of representative specifications for first-generation service discovery systems. The authors also identify a set of open issues in existing designs. The authors demonstrate how their generic model can be used to represent specific service discovery systems, including three – Universal Plug-and-Play (UPnP), Jini, and the Service Location Protocol (SLP) –analyzed in creating the model, but also including two service discovery

systems – the Web Services Dynamic Discovery and the Globus Monitoring and Discovery Service (MDS) – not analyzed in creating the model. Beyond an analysis of the structure and behavior of first-generation service discovery systems, the authors consider two other issues. First, the authors identify three classes of performance concerns that might arise in first-generation service discovery systems, and they suggest a range of solutions that implementers could adopt to solve each issue. Second, the authors propose a set of service guarantees that they believe service discovery systems should aim to achieve, and they provide a formal specification of those guarantees. The authors also make available a UML description for the model described in the paper.

Analyzing Properties and Behavior of Service Discovery Protocols Using an Architecture-Based Approach

Christopher Dabrowski and Kevin Mills
National Institute of Standards and Technology
Gaithersburg, MD USA 20899
{cdabrowski, kmills} @nist.gov

Abstract

Current trends suggest that future software systems may appear as collections of distributed components that combine and recombine dynamically in response to changing conditions. Such dynamic environments will require new analysis approaches and tools for software design. In this paper, we investigate an architecture-based approach to evaluate and compare designs for service discovery protocols operating under network and node failures. We elaborate our approach, using Jini as a specific example, and show how Jini can be analyzed using Rapide, an Architecture Description Language (ADL). Our analyses take two forms: property analysis and event analysis. We use property analysis to investigate robustness to dynamic change, while we use event analysis to discern underlying causes of observed behavior and performance. We evaluate how well Rapide supported our modeling and analyses. We also recommend improvements in ADLs to help test and analyze designs for distributed systems.

1. Introduction

Numerous trends suggest that future software will operate in an environment much more uncertain than today's typical client-server paradigm. Increased deployment of wireless communications, implying greater user mobility, coupled with proliferation of personal digital assistants and other information appliances, foretell a future where software components can never be quite sure about the network connectivity available, about the other software services and components nearby, or about the state of the network neighborhood a few minutes in the future. In the most extreme situations, as found for example in military applications [1], software components composing a distributed system may find that cooperating components disappear due to physical or cyber attacks or due to jamming of communication channels or movement of computing platforms beyond communications range. Even in less demanding circumstances, increased use of computer chips, network communications, and software to implement a growing range of consumer appliances portends the need for simple, self-contained units that, when powered on, can discover their technical surroundings and then automatically configure themselves into a larger system that might already be deployed. Further, as the consumer rearranges components in such a system, then the system must automatically adapt its configuration as necessary. Such environments demand new analysis approaches and tools for software design, implementation, and testing.

Our work considers how one might rigorously assess the robustness of distributed software systems in response to dynamic change, such as process, node, and link failures of both a temporary and permanent nature. More particularly we seek techniques to test the behavior and resilience of dynamic distributed systems, and to compare and contrast various approaches to design such systems. As a challenging application we investigate service discovery protocols, which provide mechanisms for rendezvous and robustness in the face of uncertainty. Such mechanisms enable dynamic elements in a network: 1) to discover each other, 2) to express opportunities for collaboration, and 3) to compose themselves into larger collections that cooperate to meet an application need. In this paper, we limit our analysis to Jini(tm)[1] Networking Technology, one of at least six service discovery protocols [2]-[7] designed to date. Future papers will consider additional discovery protocols.

We wish to address software robustness as early as possible in the engineering lifecycle because the earlier a design error can be uncovered, the lower the cost to repair. For this reason, we use an Architectural Description Language (ADL) [12]-[19] to transform natural-language specifications into architectural models that provide rigorous representation of system structure and behavior. Such architectural models, coupled with appropriate automated analysis tools, permit designers to uncover and correct errors and omissions, and to clarify ambiguities that would otherwise lead to incorrect behavior, or to performance problems, after a specification has been implemented and the resulting

[1] Certain commercial products or company names are identified in this report to describe our study adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the products or names identified are necessarily the best available for the purpose.

software deployed. Architectural models also provide significant advantages over less formal approaches when comparing and contrasting alternate designs for dynamic distributed systems, such as service discovery protocols.

Other authors compare various service discovery protocols [8]-[11], [22], [24]. While instructive, these comparisons exhibit significant limitations. For example, existing comparisons are largely functional in nature and informal in presentation. Such comparisons cannot capture nor express a deep understanding of the behavioral properties of the protocols, nor can these comparisons uncover areas of ambiguity, inconsistency, and incompleteness within the specifications. Further, existing comparisons use concepts and terminology taken from individual specifications. Since each specification adopts a unique language for describing its design, it becomes difficult to compare the designs directly. In future work, we aim to contribute a more rigorous comparison of three discovery protocols: Jini [4], UPnP [3], and SLP [6].

The current study serves two purposes: 1) validate our approach against the specification for Jini and 2) evaluate the suitability of ADLs to model and analyze dynamic distributed systems. To perform this study, we examined several ADLs [12]-[19], selecting Rapide [12], an ADL developed at Stanford University. Rapide specializes in modeling architectures for real-time, distributed systems and therefore represents behavior in a form suitable to investigate discovery protocols. Rapide also comes with an accompanying suite of analysis tools that can execute a specification and can record and visualize system behavior.

This paper reports our initial results with respect to modeling and analyzing the Jini specification. The paper is organized as five sections. First, we describe our approach to model and analyze discovery protocols. We provide a general architecture intended to encompass all the protocols we studied. Using Jini as an example, we illustrate how this architecture can be used to model a specific protocol, and then how the model can be converted to an executable specification, described using Rapide. In the second section, Analysis Approaches, we discuss the application of ADL tools to analyze logical properties of our models, and in the process to uncover specification deficiencies, and to assess the degree to which the model satisfies selected consistency conditions. Further, we show how behavior traces from our model can be analyzed to produce quantitative metrics. In the third section, we report and discuss the results obtained from our initial analysis of Jini. We examine how well our Jini model satisfies selected consistency conditions, and we characterize the behavior and performance of Jini with respect to particular scenarios. In the fourth section, we assess our experiences using an ADL and related tools to model and analyze Jini. We report our positive findings, along with recommendations for improvements. In the fifth section, we provide our conclusions and outline future work.

2. Modeling with an Architecture-based Approach

Most extant discovery protocols are specified statically, using natural language, and supplemented with reference software that provides one presumably legitimate implementation of the specification. The static specification expresses the appropriate behavior of system components in reaction to particular events and conditions. The reference implementation contains incidental complexity needed to fit the protocol into a software framework that includes various supporting components. Typically, static specifications cannot be used effectively to understand the dynamic behavior of distributed systems. Such specifications do not express collective behavior very well and often do not define consistency conditions against which dynamic behavior can be evaluated. Further, natural-language specifications usually lack completeness, and suffer from ambiguities and inconsistencies. On the other hand, reference software includes complexity irrelevant to the fundamental requirements of the specification. Further, reference software typically will implement one particular design choice in cases where a specification may allow various alternatives.

To overcome these shortcomings, we adopted an approach that entails the following general steps: 1) construct an architectural model of each discovery protocol, 2) identify and specify relevant consistency conditions that each model should satisfy, 3) define appropriate metrics for comparing the behavior of each model, 4) construct interesting scenarios to exercise the models and to probe for violations of consistency conditions, and 5) compare the results from executing similar scenarios against each model. Below, we elaborate our approach, using Jini as a specific example, and show how Jini can be modeled using Rapide, an Architecture Description Language (ADL). We also discuss the Rapide run-time, which converts our Jini model to an executable specification. First, we introduce discovery protocols, and define some consistent terminology that we can use to build comparable architectural models.

2.1. Discovery Protocols in Essence

Discovery protocols enable software components to find each other on a network, and to determine if discovered components match their requirements. Further, discovery protocols include techniques to detect changes in component availability, and to maintain,

within some time bounds, a consistent view of components in a network. Many diverse industry activities explore different approaches to meet such requirements; leading to a variety of proposed designs for service discovery protocols [2]-[7]. Some industry groups approach the problem from a vertically integrated perspective, coupled with a narrow application focus. Other industry groups propose more widely applicable solutions. For example, a team of researchers and engineers at Sun designed a general service discovery mechanism atop Java(tm), which provides a base of portable software technology. The proliferation of service discovery protocols motivates deeper analyses of their designs. Beyond this, given the level of debate within the industry, a comparative analysis can help to assess the relative merits of particular protocols.

To help us compare protocols, we developed a general UML (Unified Modeling Language) model, expressed with a consistent terminology (see Table 1) that provided a basis for the Rapide architectural model. The main components in our general model include: 1) service manager (SM), 2) service user (SU), and 3) service cache manager (SCM), where the service cache manager is an optional element not supported by all discovery protocols.

Table 1. Mapping Concepts Among Various Discovery Protocols.

Generic Model	Jini	UPnP	SLP
Service User	Client	Control Point	User Agent
Service Manager	Service or Device Proxy	Root Device	Service Agent
Service Provider	Service	Device or Service	Service
Service Description	Service Item	Device/Service Description	Service Registration
Identity	Service ID	Universal Unique ID	Service URL
Type	Service Type	Device/Service Type	Service Type
Attributes	Attribute Set	Device/Service Schema	Service Attributes
User Interface	Service Applet	Presentation URL	Template URL
Program Interface	Service Proxy	Control/Event URL	Template URL
Service Cache Manger	Lookup Service	not applicable	Directory Service Agent (optional)

These components participate in the discovery, registration, and consistency maintenance processes that comprise dynamic discovery protocols. A service manager maintains a database (Service Repository) of records (Service Descriptions, or SDs), where each record describes the essential characteristics of a particular service or device (Service Provider, or SP). Each SD contains the identity, type, and attributes that characterize a SP. Each SD also provides up to two interfaces (an application-programming interface and a graphic-user interface) to access a service. Table 1 shows how these general concepts map to specific concepts for Jini, UPnP, and SLP. Since the paper uses Jini as an example, we provide a brief synopsis.

2.2. Jini in Brief

Upon startup, a Jini component (SU, SM, or SCM) engages in a discovery process to locate other, relevant Jini components within the network neighborhood. To oversimplify things: 1) SMs attempt to discover relevant SCMs with which to register a SD for each SP managed and 2) SUs attempt to discover relevant SCMs to query for SDs that lead to desired SPs. In other words, SUs and SPs rendezvous through SDs registered by SMs with particular SCMs, where the SCMs are found through a discovery process.

2.2.1. Jini Discovery. Jini encompasses two discovery modes, *multicast and directed*, supported by three discovery processes, which we call aggressive, lazy, and directed. Both aggressive and lazy discovery involve multicast communication among Jini components participating in two multicast groups. Upon initiation, a Jini component enters aggressive discovery, where it transmits probes at a fixed interval for a specified period, or until it has discovered a sufficient number of SCMs. Upon cessation of aggressive discovery, a component enters lazy discovery, where it listens for announcements sent at intervals by SCMs. This implies that during lazy discovery a SCM both listens for announcements by other SCMs and sends its own announcements at the required intervals. Figure 1 gives a simplified illustration of the two Jini discovery modes, and the three supporting processes.

During aggressive discovery, probes sent by Jini components identify interest in one or more administrative scopes, which Jini calls groups; probes also contain a list of SCMs already discovered by the Jini component. Each SCM must reply to a probe only when the list of groups contained within the probe intersects with the SCM's own list of groups in which it is a member, and also provided that the probe does not indicate that the SCM has already been discovered. Once a relevant SCM is discovered, the discovering component requests an application-programming interface (API) that enables the component to interact with the SCM.

Lazy discovery operates similarly. Announcements sent by SCMs identity group membership. A Jini component requests an API from an announcing SCM when the following conditions hold: 1) the group membership of the SCM intersects with the groups of interest to the component, 2) the component has not already discovered the SCM, and 3) the component has not already discovered enough SCMs. Receipt of an API from the SCM ends the discovery process between the component and the SCM.

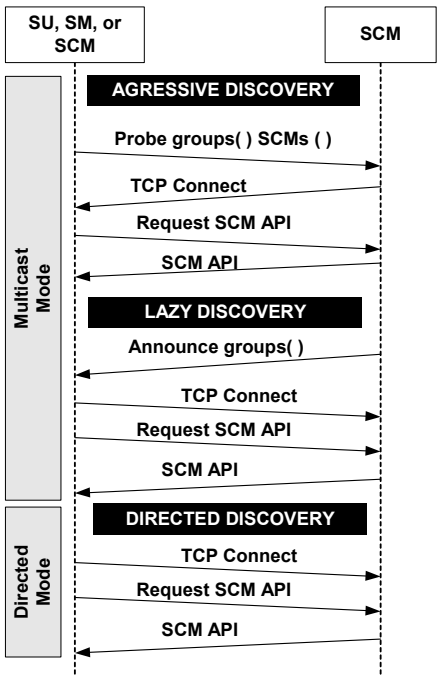


Fig. 1. Jini has two discovery modes (multicast and directed) that encompass three discovery processes. In multicast mode, aggressive discovery is initiated on node startup, and then lazy discovery begins after aggressive discovery completes. In directed mode, directed discovery is used to look for specified SCMs.

Directed discovery operates differently from multicast discovery. Each Jini component may be given a specific list of SCMs to discover. For each SCM on the list, a Jini component establishes a connection and requests an API. Should the SCM prove unavailable, the component can continue to retry connecting. As explained later, ambiguities regarding interaction between directed and multicast discovery lead to several problems for the Jini specification.

Once a Jini component obtains an API from a SCM, the component can use the API to access services provided by the SCM. To allow the component and the SCM to reside on different network nodes, the API must use a communication protocol, such as Java Remote Method Invocation (RMI)[2], which enables the component to access SCM services as if they resided within the same Java Virtual Machine (JVM). In general, SCM services can be classified as registration and consistency maintenance, which Jini refers to as leasing.

2.2.2. Jini Registration. A SM holds a SD for one or more SPs. The SM must register each of these SDs with each SCM discovered. As part of the registration request, the SM asks that the registration remain valid for some duration. If the SCM agrees to add the SD to its set of registered services, then the SCM grants a lease time (not more than requested) and returns a service item and lease to the SM. Once a SD is registered with a SCM, SUs can discover the existence of the related SP by querying the SCM, or by receiving notifications from the SCM. Before receiving notifications, a SU must register notification requests with a SCM. A SU can register a request that a SCM notify the SU whenever the SCM adds, deletes, or changes a SD of interest. As with service registrations, notification requests will be maintained by a SCM only for an agreed time (the lease period).

2.2.3. Jini Consistency Maintenance. In a distributed system, new services and devices can be deployed, obsolete services and devices can be removed, and nodes, processes, and links can fail. These facts imply that replicated state, distributed throughout a system, can become inconsistent. To time bound such inconsistencies, Jini requires each SCM to periodically purge SD registrations and notification requests. For this reason, a SCM assigns a lease to each registration and notification request. The lease indicates when the SCM plans to purge the item. To prevent its removal by the SCM, the registering component must renew the lease prior to the purge time. In this way, if the registering component fails (or the network path fails), then the SCM can, within a bounded delay, remove reference to the item, and, when appropriate, can notify other interested components. Once the failure is resolved, the discovery and registration processes can be restarted for the failed component, and the previous state might be recovered eventually.

Interactions with SCMs provide another means for Jini to maintain consistent state. Each component may register some items with a SCM. In addition, leases for these registered items must be renewed periodically. Whenever a component attempts to invoke a SCM method across a network the possibility exists for a remote exception. Remote exceptions indicate that the corresponding SCM (process or node) might have failed, or that the network link between the component and the SCM might have failed. A component is free to retry a method invocation, and to give up after some period of time.

2.3. Complexity and Uncertainty

The foregoing discussion of Jini, while oversimplified, highlights the inherent complexity and uncertainty associated with discovery protocols. Complexity arises from several sources. The protocol involves multiple parties communicating across a network, which introduces asynchrony, and which can also introduce variable delays. Multiparty interactions can be quite difficult to specify and understand. Further, the protocol defines various operating modes that could potentially interfere with one another, and each protocol entity maintains independently operating behavioral threads,

[2] Jini does not require the use of any particular technique for remote procedure calls. In this paper, we use RMI for illustrative purposes.

which implement features that can interact in unanticipated ways.

Uncertainty also arises because nodes, processes, and links can appear and disappear without warning. Discovery protocols must include behavior to cope with such changes. The coping behavior itself can exhibit unexpected interactions with the already complex behavior defined to implement multiparty communication. Together, this complexity and uncertainty discourage protocol designers from attempting to specify the properties of a particular discovery protocol. Yet, we desire to compare and contrast the protocols based on such properties. This conundrum led us to the idea of constructing an architectural model for each discovery protocol, and using the models to investigate various properties.

2.4. An Architectural Model for Jini

Broadly speaking, an architectural model comprises a set of components, and the connections among them, along with the relationships and interactions among the components. In our application, an architectural model expresses structure (as components, connections, and relations), interfaces (as messages received by components), behavior (as actions taken in response to messages received, including generation of new messages), and consistency conditions (as Boolean relations among state variables maintained across different components).

Figure 2 depicts the top level of our Jini architecture that was realized in Rapide. This architecture consists of three component types (SU, SM, and SCM) together with three connection types: Aggressive Discovery Multicast Group (ADMG), Lazy Discovery Multicast Group (LDMG), and Remote Method Invocation Unicast Link (RMIUL). Only one instance each can exist for the LDMG and ADMG but the SU, SM, SCM, and RMIUL can be instantiated as multiple instances. Each SU, SM, and SCM resides on a network node and participates in service discovery, registration, and consistency maintenance. To perform these functions, each type of Jini component is decomposed into subcomponents (not described in this paper due to lack of space). Jini components use the ADMG to distribute probes to any SCMs listening. SCMs use the LDMG to distribute announcements to any Jini component listening. When asked to engage in directed discovery, a Jini component uses one RMIUL to contact each SCM on its directed-discovery list. To invoke methods on a specific SCM, a Jini component must use an appropriate RMIUL.

We implement SMs, SCMs, and SUs, as Rapide interfaces. We define connections, also implemented as Rapide interfaces, to link Jini components that exchange events. We use Rapide services to constrain the event types allowed on each connection. We model two classes of connection: 1) fan-out multicast links (ADMG and LDMG) for discovery and 2) unicast links (RMIUL) for directed discovery and for remote-method invocation.

Modeling connections as Rapide interfaces allows the links to encapsulate logic: 1) to control link state (up or down) and 2) to send appropriate remote exceptions in response to events sent over a failed link. The remote exception logic proves significant because some events require remote exceptions to be sent in one direction, while other events require bi-directional remote exceptions. Since nodes may come up or go down at any time, our model also includes specific events to start and stop nodes. As we discuss later in Section 5, these requirements have implications for how ADLs should model connections.

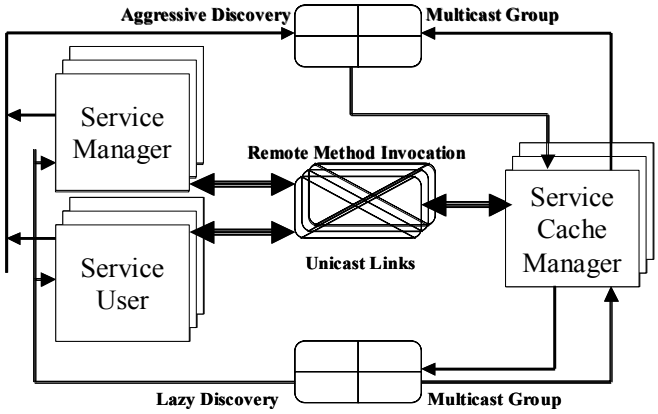


Fig. 2. Our top-level architecture models a distributed Jini by using two multicast groups and a set of unicast links to connect Jini components into a topology.

3. Analysis Approach

Our specification analyses take two forms: property analysis and event analysis. Both depend upon Rapide's ability to execute a specification and to generate events. We use property analysis to investigate robustness to dynamic change, including network failure. Property analysis also provides insight into processes defined in a protocol specification, and helps to identify ambiguity, inconsistency, incompleteness, and other flaws. Event analysis examines *Rapide* POSETs (partially ordered sets of events exchanged among components) to discern underlying causes of observed behavior and performance, and especially to assess the protocol's capacity to recover from network disruption. We also use event analysis to understand circumstances surrounding specific protocol design issues, such as race conditions. Property and event analysis can be used together to evaluate a protocol's resilience in the face of network failure. We also suspect that POSETs can provide a basis for complexity metrics,

another dimension along which we expect to compare discovery protocols. Our current work has not developed such complexity metrics. Below, we describe our use of *Rapide* to analyze properties and behavior of Jini.

3.1. Property Analysis

To implement property analysis we define consistency conditions and then use the Rapide constraint language to express the negation of each consistency condition. If a negation is satisfied, then Rapide has detected an inconsistency. We stimulate periodic events, called *consistency probes*, which retrieve values from the internal state variables of appropriate components. At each probe interval Rapide checks for the presence of an inconsistency. In general, discovery protocols attempt to guarantee time-bounded inconsistency. Our analysis strives to verify such guarantees. We also seek to identify unbounded inconsistencies, which persist indefinitely. Unbounded inconsistencies suggest areas of a specification, or protocol design, which merit further attention. Below we give some examples of consistency conditions. In Section 4, we discuss circumstances in our Jini model where these consistency conditions do not hold.

We posited the quality of service that users might expect from discovery protocols. Then we defined these ideas as consistency conditions that specify relationships a protocol should strive to maintain among state variables across interacting components. In this paper, we define selected consistency conditions[3] that should hold in the absence of failures or other dynamic changes that could permit the conditions to be violated for a transient period. Several consistency conditions concern the SCM and the SM. Analogous conditions could also be defined for the SCM and the SU. For example, a SM can only register a service description with a SCM it has discovered. This can be expressed as the following consistency condition:

```
For All (SM, SD, SCM):                                   (CC1)
        (SM, SD) IsElementOf SCM registered-services
         implies SCM IsElementOf SM discovered-SCMs
```

In our model, we express the negation of this consistency condition as a Rapide constraint. Consistency probes return the contents of each SM's list of discovered SCMs and of each SCM's list of registered services. Rapide checks various combinations of values for specific pairs of SMs and SCMs at each probe time. When the negation is true, an inconsistency exists.

A second example consistency condition states that if a SM has discovered a SCM and the SM has a SD for a service that it is managing, then the SM should have registered the SD with the SCM. Here, a service is managed if the SM is required to advertise its availability. This may be expressed as:

```
For All (SM, SD, SCM):                                   (CC2)
        SCM IsElementOf SM discovered-SCMs &
        (SD) IsElementOf SM managed-services
        implies (SM, SD) IsElementOf SCM registered-services
```

This consistency condition amounts to an inverse view of CC1. This inverse view can catch specification issues that CC1 would miss.

A third example consistency condition states that if a SM has discovered a SCM through multicast discovery and has registered its services on that SCM, then there should be an intersection between the list of groups the SM is to join and at least one group in which the SCM holds membership. This can be expressed as:

```
For All (SM, SD, SCM):                                   (CC3)
        SCM IsElementOf SM discovered-SCMs &
        (SM, SD) IsElementOf SCM registered-services &
        NOT (SCM IsElementOf SM persistent-list)
        implies Intersection
                (SM GroupsToJoin, SCM GroupsMemberOf)
```

Reference to the absence of membership of the SCM in the SM persistent list eliminates SCMs that the SM found through directed discovery.

3.2. Event Analysis

We use event analysis to understand underlying causes for the observed behavior and performance of discovery protocols. The general idea is to define a set of usage scenarios that can be executed against the models of several discovery protocols. Table 2 provides an excerpt from a scenario we defined, and provides a sense of the stimuli that can be simulated. While executing scenarios, the Rapide run-time produces POSETs that provide a basis for analyses. POSETs help us to understand relationships among events, which trace back to specific behavior in components, and to possible issues within a specification. The POSETs may also be used to compute simple metrics, such as number of events generated or time taken by the model to transition between two configurations of interest. To support such computation, we insert performance probes at key points in the Rapide model. Such probes can compute the desired measurements, or can place markers in the POSET for off-line computation. While event analyses can be applied individually to specific protocols, greater value may accrue in comparative analysis. Following we give examples of some event analyses of interest.

[3] Consistency conditions we define here do not necessarily reflect the intent of Jini's designers.

Table 2. Sample Scenario Commands with Parameters and Intended Execution Times.

Time	Command	Parameters
5	NodeFail	SM4
5	LinkFail	SCM1 SM4
10	GroupJoin	SM4 GROUP1
10	FindService	SU8 5 1 2 S XYZ ALL
50	AddService	SM4 SCM3 T ATT API GUI 20 30

3.2.1. Identifying and Understanding Race Conditions. Due to asynchronous processing and associated delays in communications among components, distributed systems often exhibit race conditions, where system behavior can vary depending upon the order in which events arrive at cooperating components. Though such problems cannot always be eliminated, it remains important to identify the existence of specific race conditions so that application programmers can adopt appropriate safeguards. We can use Rapide to find race conditions by asserting and testing consistency conditions. For example, consider the following:

For All (SM, SD, SCM, SU, NR): (CC4)
(SU, NR) IsElementOf SCM requested-notifications &
(SM, SD) IsElementOf SCM registered-services &
Matches((SM, SD), (SU,NR))
implies (SM, SD) IsElementOf SU matched-services

This consistency condition indicates that if a SU has requested notification when a certain service (SM, SD) registered at a SCM matches specified criteria, then the SU should become aware of the matching service. While the Jini specification does not guarantee CC4, we would be interested to identify situations where the condition does not hold. In such cases, we can analyze the POSET to determine specific causes. In this way, we might uncover race conditions that require an application programmer to take particular care when using Jini's matching mechanisms.

3.2.2. Measuring and Understanding Protocol Performance. When comparing various discovery protocols, we can use Rapide to define and compute performance metrics, and then use POSET analysis to investigate the underlying behaviors. Of course, comparative performance must be considered in light of selected scenarios of interest. For example, consider a scenario where a major power failure occurs after the discovery phase has completed, services and notification requests are registered, and SUs have received SDs for services that meet their requirements. During the failure, most Jini entities lose some internal state: all nodes lose discovered SCMs; SUs lose SDs for services previously discovered; but SCMs and SMs must retain specified persistent information. Upon power restoration, the Jini components restart and recover. To assess recovery performance we define two metrics, restoration latency and restoration overhead, which measure the efficiency of recovery in terms of total time and number of messages generated before all SUs rediscover their original set of SDs. Restoration latency covers node start-up delays, transmission times, processor background workload, and times for processing transaction data. Restoration overhead includes all events exchanged by Jini components from power up through complete restoration of the desired state.

4. Selected Analysis of the Jini Service Discovery Protocol

In this section we discuss some results obtained running scenarios against our Jini architectural model. We were able to verify the robustness of Jini's design in a range of failure scenarios that are not presented here due to lack of space. However, we found the Jini specification unclear regarding interactions between multicast and directed discovery. In particular, we could not discern whether discovered SCMs should be kept on a single list or whether SCMs found by directed discovery should be kept on a separate list from SCMs found by multicast discovery. We included both interpretations in our Jini model, and we ran related scenarios to evaluate CC1 and CC2. We also noticed that the Jini reference implementation permits administrators to alter the operation of a running SCM. We were interested to consider if such changes could adversely affect a Jini network, so we ran related scenarios to evaluate CC3. Further, we discovered an apparent race condition that is difficult to discern from reading the Jini specification, so we ran scenarios to evaluate CC4. We also executed selected scenarios to understand some performance characteristics of Jini systems. Here, we discuss restart from power failure. While the work described here suggests some incompleteness and ambiguity (already shared with Sun) in the Jini specification, our purpose is to illustrate an architecture-based approach to model, analyze, and compare service discovery protocols. Regardless of any ambiguity and incompleteness discussed here, overall we found Jini to operate as specified.

4.1. Interfering Interactions between Directed and Multicast Discovery

The Jini specification permits a Jini component to engage simultaneously in two modes of discovery:

directed and multicast. However, the specification is unclear with respect to issues that arise regarding interactions between these two modes. This means that an implementer must make some decisions, which can lead to various difficulties. We identified decisions that cause local interference between independent processes on the same Jini node. We also found decisions that cause independent processes on the same Jini node to interfere with the node's remote state on discovered SCMs. We discuss these situations below.

4.1.1. Local Interference. For the following discussion, assume that the implementer decides to maintain a single list of SCMs discovered by a SM. Figure 3 illustrates (using a simplified description) what occurs during a scenario where SM4 uses multicast discovery to find SCMs in a Jini group (GROUP2).

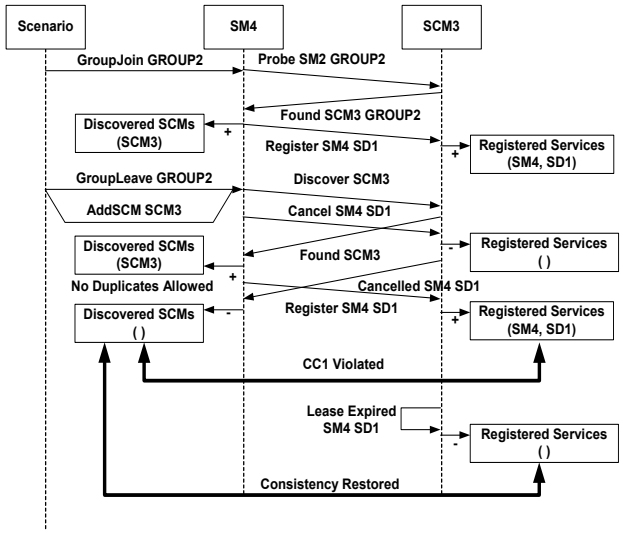


Fig. 3. Example of local interference between directed and multicast discovery modes.

In this case SM4 discovers SCM3, also a member of GROUP2. Shortly after, SM4 is told to discover SCM3 (AddSCM) through directed discovery, and at the same time SM4 is told to drop membership in GROUP2. The resulting behavior leads to a time-bounded violation of CC1, which states that a SD should not be registered on a SCM if the SCM is not on the discovered list of the SM managing the SD. The specific behavior follows.

Through multicast discovery SM4 finds SCM3 and adds it to the list of discovered SCMs. Subsequently, SM4 is asked simultaneously to leave GROUP2 and to discover SCM3. The group leave causes SM4 to first cancel leases for SDs held on SCM3 and then to remove SCM3 from its list of discovered SCMs. Between these two events, SM4 uses directed discovery to find SCM3 and then attempts to add SCM3 to its list of discovered SCMs. Since our model assumes that probes will be built from the list of discovered SCMs, we decided not to insert duplicate SCMs in that list.[4] This rule is enforced by the list maintenance function. Therefore, in Figure 3, the second discovery of SCM3 is not added to the list of discovered SCMs because it's already there. Soon thereafter, SM4 completes lease cancellation for SDs on SCM3 and then removes SCM3 from its list of discovered SCMs. In the meantime, the directed discovery process in SM4 registers SDs with SCM3. At that point, CC1 is violated, and remains so until the leases for the SM4 SDs expire on SCM3.

4.1.2. Remote Interference. Suppose that an implementer decides to maintain SCMs discovered by multicast and directed discovery on separate lists? In this case, local interference disappears, only to be replaced by a form of remote interference, where two discovery processes within the same node independently manipulate the state of SDs on SCMs. Figure 4 illustrates behavior from a scenario that uncovers this problem through violation of CC2, which states that services managed by a SM must be registered on all discovered SCMs. In the scenario, SM4 uses directed discovery to find SCM1. Later SM4 is instructed to join GROUP1, which includes SCM1. This causes a duplicate service registration, which leads SCM1 to abrogate the existing lease for (SM4, SD1). Subsequently, SM4 is told to leave GROUP1. In the end, this causes SM4 to cancel leases for its SDs held on SCM1, resulting in a situation where SCM1 is on the list of SCMs discovered directly by SM4 but where the SDs from SM4, which were originally registered through the directed discovery action, are not now registered on SCM1. Assuming that SM4 maintains a single registration process, this violation of CC2 is unbounded in time.

4.2. Insensitivity to Changes in Group Membership by SCMs

The Jini reference implementation includes an interface that permits an administrator to alter parameters associated with a running SCM. We mirrored this behavior within our Jini model, and then exercised the option to change group membership of a running SCM. Figure 5 illustrates the relevant subset of a related scenario. First, SM4 is instructed to join GROUP1, which leads to the multicast discovery of SCM1 (a member of GROUP1).

Subsequently, an administrator removes (AdminDelete Group) SCM1 from membership in GROUP1. Once this occurs, CC3 is violated because: (1) SM4 has found SCM1 with multicast discovery, (2) SDs managed by SM4 are registered with SCM1, and yet (3) SM4 and

[4] Allowing duplicates on a single list leads to a number of other problems, which are beyond the scope of the discussion here.

SCM1 have no common group membership. The violation of CC3 continues in a time-unbounded form so long as SM4 renews leases on SCM1.

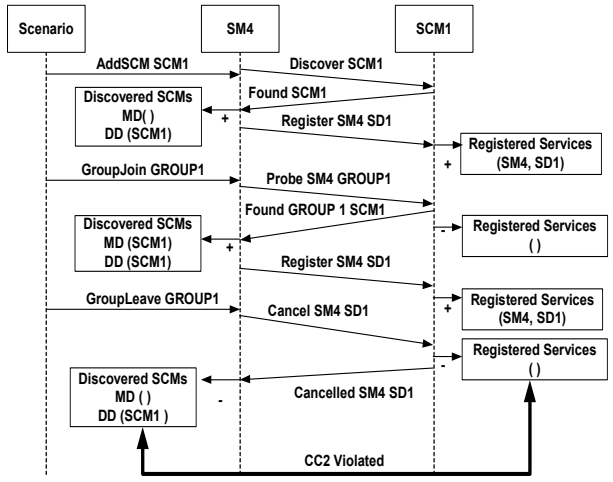


Fig. 4. Example of remote interference between directed and multicast discovery modes

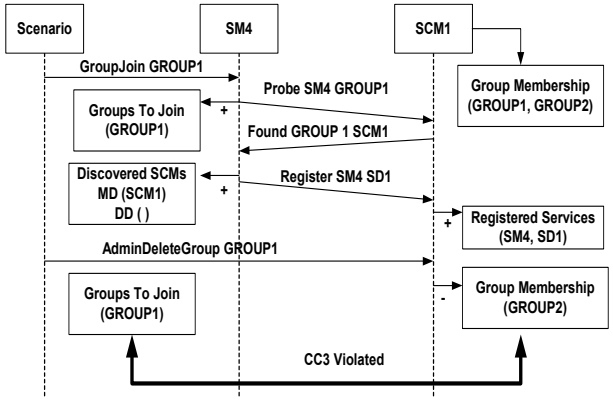


Fig. 5. Example of insensitivity to group membership changes by the SCM.

These results suggest that the Jini specification may be incomplete with regard to this issue. While an administrator can remove group membership from a running SCM, the Jini protocol specifies no behavior in reaction to this new information. As a SCM continues to issue announcements, which contain its current group membership, other Jini components are told to ignore announcements from SCMs that do not belong to groups of interest. As shown in the discussion above, this can lead to a situation where SMs (as well as SUs) may continue to maintain registration with SCMs no longer relevant. This might or might not be the intent of Jini's designers; however, the issue should be addressed in the specification.

4.3. Race Conditions

All distributed systems exhibit the possibility for race conditions. Our architectural model permits us to investigate how such conditions can arise. Figure 6 presents a portion of a scenario illustrating a race between service registration by SMs and registration of notification requests by SUs.

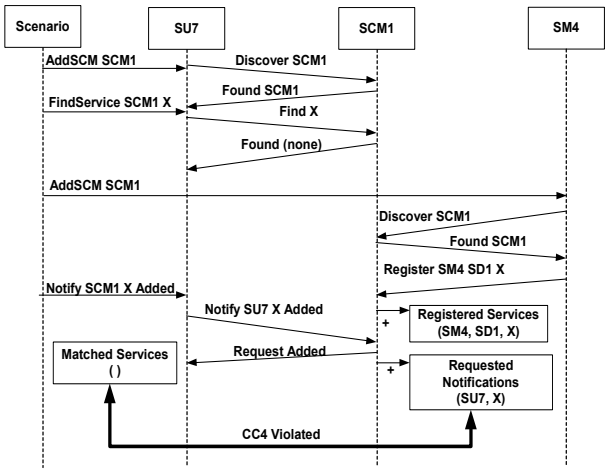


Fig. 6. Example race condition between service registration by an SM and notification request registration by an SU.

In this case, SU7 discovers SCM1 and then queries it for a matching service. At the time of the query, SCM1 does not contain a SD for a matching service and so replies without matches. In this particular scenario, SU7 delays for 10s its request to be notified by SCM1 when a SD for a matching service is added to the SCM cache. In the interim, SM4 discovers SCM1 and registers a SD for a service matching the needs of SU7. Unfortunately, the only matching service was registered during the interval between the query and the request for notification by SU7. In Jini's definition of matching semantics, SU7 can continue to renew leases for its request for notification and SM4 can continue to renew leases for its SD and the two will never learn of each other. This situation results in a time-unbounded violation of CC4, which states that if a SCM holds a notification request from a SU, which matches a SD also held by the SCM, then the SU should know about the matching SD.

While this violation of CC4 can be attributed to the 10s delay before SU7 sends a notification request, a number of other situations can lead to similar results. For example, network congestion can delay the reply to the original query by SU7 or can delay the request by SU7 for SCM1 to register its notification. Alternatively, competing processing within the node supporting SU7 could delay the generation of its notification requests. To account for this, SUs might issue a second query for a

matching service after the notification request is registered with a SCM. In this way, the SU can detect any matching SDs registered by the SCM after the first query but before the notification request.

4.4 Restart Performance

To demonstrate the ability of our architectural model to provide insight into performance-related behavior, we describe the results of an experiment to investigate the restart of a Jini network following recovery from a major power failure. The experiment topology consists of nine nodes (three of each type: SU, SM, and SCM). We partition the nodes into threes, where each partition consists of one SU attempting to rendezvous with one SM through a SCM. Once all SUs have found their assigned SMs, we simulate a major power failure, which causes all nodes to crash for 40s. We then restore the power and wait for all SUs to rendezvous with their assigned SMs. Table 3 gives the values for relevant experiment parameters. Upon restart, each Jini node chooses a random delay before beginning discovery; we used delays uniformly distributed between two and 15s. We also had each SU and SM request leases of 30s for notification requests and service registrations, and we had each node renew the leases for a period of 100s. For each link, we introduced variable transmission delays; for each node, we introduced variable processing-load delays. We also introduced processing delays for manipulating items in the discovery databases and the SCM registration databases. Since the Jini specification did not address the persistence of notification requests upon SCM failure, we assumed that this information was purged on failure.

Table 3. Parameter values used in the power-failure restart performance experiment. Some values reflect settings of Jini protocol parameters, while others reflect assumptions regarding transmission and processing delays.

Parameter Class	Parameter	Value
Jini Protocol Parameters	Node Restart Delay	2s – 15s uniform
	Probe Interval (and period)	5s (7 times)
	Announce Interval	120s
	Per Lease Time	30s
	Total Leasing Duration	100s
	Notification Requests	Purge on SCM Failure
Delays	Transmission Delay	1us – 10us uniform
	Processing Load Delay	10us – 100 us uniform
	Per Item Processing	10 us (discovery DBs) 100 us (SCM cache)

We ran the experiment 30 times, measuring the restoration latency and overhead. In this experiment, before the original state could be recovered, all nodes had to restart. For that reason, the maximum node restart delay dominates the restoration delay. For example, for our experiment runs, the average maximum node restart delay was 12.56s (2.09s variance), and the average restoration latency was 14.76s (3.31s variance). The restoration overhead in each run depends upon the restoration latency, because periodic message exchanges associated with Jini discovery and leasing continue through the restoration. In this experiment, the restoration latencies were relatively close, as were the number of messages exchanged, differing only in the number of probes sent during aggressive discovery and in the subsequent number of discoveries. In our runs, the number of messages exchanged to achieve restoration ranged approximately between 70 and 90. These results demonstrate that the same architectural model can be used to investigate both performance and logical properties of a distributed system.

4.5 Summary of Findings

Using our architectural model and usage scenarios we were able to verify the robustness of Jini mechanisms in a range of failure scenarios. Further, as supported by the analyses above, we were able to uncover areas of incompleteness and ambiguity in the natural-language specification for Jini. While a static, natural-language specification, such as Jini's, contains a reasonable description of the behavior of each component in response to specific events, such specifications largely miss collective behavior arising when various components interact together in a distributed system, and especially when pieces of the system change state during the interactions. In addition, our dynamic, executable model of the Jini specification permitted us to explore the behavior and performance of Jini systems in various realistic scenarios. A static specification cannot hope to provide similar insights.

5. Assessment of the Architecture-based Approach

As part of our work, we assessed how well the Rapide ADL and analysis tools supported our modeling and analyses of Jini, with specific attention to analysis of dynamic behavior. We found that the Rapide ADL provided valid abstractions to represent and analyze the structure and behavior of Jini under conditions of dynamic change. Using Rapide interfaces we easily represented the major service discovery components, and subcomponents (not discussed in this paper). The components proved easy to connect into architectures that model a network of Jini entities. Our analyses relied upon Rapide's ability to represent dynamic behavior through

events, rules, and constraints, and then to analyze the resulting POSETs. The ability to represent the behavior of individual components and to analyze the collective behavior resulting from interactions was key, without which this analysis could not have been performed. We did identify some suggestions for improving specific capabilities that apply generally to all ADLs. Before discussing these suggestions, we describe general merits of using an architectural model.

5.1. Merits of using an Architectural Model

Our Rapide model provided benefits for analysis. Some of these benefits apply to all ADLs. First, the architectural model proved more precise, concise, and informative than the natural-language specification. For example, the architectural model provided executable behavior so that we could discover interactions not apparent from the paper specification. As a consequence, we were able to identify and address areas of ambiguity, inconsistency, and incompleteness. While the Jini specification was supported by a reference implementation, the architectural model proved easier to understand and analyze, and permitted us to focus on the essential complexity inherent in the specification. The reference implementation entailed incidental complexity that interfered with our ability to gain a clear understanding of the behavior of the specification. Second, a single architectural model can be analyzed for behavioral, performance, and logical properties. Using a single model limits the errors and inconsistencies that can creep in when multiple models must be used to represent the same specification. Third, using an architectural model enabled us to readily consider alternative implementation options, where they were allowed by the specification, and to identify specification ambiguities. When addressing ambiguities, the architectural model enabled us to investigate the ramifications of various alternate resolutions.

5.2 Areas for Improvement

Below, we identify and discuss some suggestions for improving ADLs in several areas: domain-specificity, simplification through views, representation of structure and behavior, and support for analysis. While we discuss these suggestions in the context of Rapide, we believe they apply more generally to use of ADLs for modeling architectures for dynamic systems.

5.2.1. Need for customizable domain-specific syntax and semantics. Constructing an architectural model typically entails a partnership between a domain expert and a system architect. The partnership proceeds more smoothly when the architecture reflects the terminology of the domain, allowing the domain expert to review the specification with less help from the architect. For this reason, ADLs should support renaming common ADL constructs such as interfaces, components, connectors or modules to use terms familiar in the domain. This would allow the expert to more easily read the specification without having to learn the ADL in detail. The same benefit may accrue from allowing customization of language syntax to be more familiar to domain practitioners, especially with respect to system behavior.

5.2.2. Improvement to representation of structure. Rapide, and other ADLs, connect components to subcomponents and pass events in a strictly hierarchical manner. One purpose in doing this is to constrain communications among subcomponents of different hierarchies in order to limit the introduction of errors when replacing subcomponents. This requires inter-component events to propagate through multiple levels in two hierarchies, leading to several inefficiencies. First, if the same events must be duplicated as a result, an unnecessarily large set of events will be created for analysis. Second, the architecture entails an increased number of connections, resulting in a larger specification, which is more difficult to maintain and modify. This inhibits revision and evolution of system designs, especially important when modeling dynamic systems, and also discourages investigation of alternative design approaches. Third, the strict hierarchy arrangement does not agree with real-world designs in which subcomponents of different systems often communicate directly. To address these problems, we recommend investigating alternative ways to specify connectivity between top-level components and subcomponents in an architectural model, while preserving correct communications. We plan to address this area further in future work.

Beyond the question of number, connections take on importance for modeling reasons. Specifically, we believe that connections should be represented as first-class entities [17], [20], [21], [23]. Many domains, including networking, have numerous, well-known connection classes. Such domain-specific knowledge can be encoded as taxonomies of connection types, provided that connections can be represented as first-class entities within the ADL. For example, we found the need to specify classes of multicast groups and RMI connections in order to represent systems that dynamically "plug-and-play" with network components, and to simulate transient failures, transmission delays, and other network characteristics. Using connection types allowed us to more easily specify restrictions on events that pass among components, and to define constraints on inter-component behavior, while associating them directly with appropriate places in an architectural model. Making connections first class permits still further semantic distinction

between components and connections, thus facilitating clear and explicit description of architecture. First class connections also encourage designers to define constraints for specific connection types and type hierarchies, so that formal reasoning about connector behavior can be localized. We suspect this may be of particular importance for architectures of dynamic systems, where connections provide a focal point for analysis.

5.2.3. Improvement to representation of behavior. As an adjunct to sending and receiving events, a Rapide component encapsulates a set of state variables. To test consistency conditions during execution, we needed to capture and analyze state variables maintained by multiple components. This required us to adopt several cumbersome solutions. We believe modeling of architectures for dynamic systems is greatly facilitated by permitting definition of component state from a subset of internal state variables. Component states should be selectively exported and recorded along with events. Linking events to changes in state [13] then allows recording of dependencies for analysis.

Assuming appropriate state variables are exported, further investigation is needed to determine how best to define, implement, and evaluate consistency conditions that involve the state of two or more components *and* that account for time. ADL constraint representation should include rich semantics for this purpose. Further, analyses of architectures for dynamic systems benefit greatly when ADL run-time environments include support for automated evaluation of inter-component consistency conditions (as some already do), and especially constraint languages and related constraint-analysis engines that account for time.

6. Conclusions and Future Work

Our current work illustrates the viability of an architecture-based approach to investigate and evaluate logical and behavioral properties of discovery protocols under conditions of dynamic change. Our results show that executable architecture models prove essential to understand the collective behavior of distributed systems. In this paper, we demonstrated how such models help to uncover ambiguity, incompleteness, and other issues in static, natural-language specifications. Our demonstration contributes to improving the specification for Jini. We also argue that a single executable architecture model can be used to investigate system performance as well as logical properties. Beyond this, we offered some recommendations, based on our experience, to improve the suitability of ADLs to model and analyze distributed systems.

In the next phase of our project, we intend to demonstrate that using architectural models provides a sound basis on which to compare and contrast the technical merits of various discovery protocols. The results from our analyses should provide industry with better understanding of the design and behavior of discovery protocols. We will define a generic set of usage scenarios to measure interesting events common among all protocols. These scenarios will exploit a common vocabulary and set of protocol features derived from our UML model. Similarly, we will identify a set of consistency conditions, design issues, and performance metrics that provide a suitable basis for comparison among discovery protocols. We suspect relevant consistency conditions and metrics will involve only SMs and SUs, because not all protocols require SCMs.

The next phase of the project will also provide a vehicle for continued appraisal of our architecture-based approach to investigate distributed system designs under dynamic conditions. We intend to sharpen and refine our current assessment. We also hope to make more specific recommendations on ADL features to better support domain-specific models, to represent connections, and to analyze internal state of components. Modeling additional discovery protocols also provides an opportunity to examine reuse of architectural components as we attempt to adapt common functions in architectures for different protocols. This work should reveal insights regarding ADL features that facilitate reuse.

Finally, we suspect, but cannot yet conclude, that the nature of dynamism in the service-discovery domain differs from other real-time domains. The next phase of the project, together with the results of concurrent research in dynamic change within the defense software research community, should illuminate this issue as well. Since automatic component discovery and collaborative composition will be essential capabilities of future defense systems, early insights gained into this issue will likely prove important.

7. Acknowledgments

The work described in this paper benefits from financial support provided by the National Institute of Standards and Technology (NIST), the Advanced Research Development Agency (ARDA), and the Defense Advanced Research Projects Agency (DARPA). In particular, we acknowledge the support of Greg Puffenbarger from ARDA and John Salasin, DARPA's program manager for Dynamic Assembly for System Adaptability, Dependability, and Assurance (DASADA). We also gratefully acknowledge the insights that Jim Waldo, Jini Architect, provided us during several hours of

discussion about our approach and preliminary results, and in written comments on an earlier draft of this paper.

8. References

[1] G. Bieber and J. Carpenter, "Openwings A Service-Oriented Component Architecture for Self-Forming, Self-Healing, Network-Centric Systems," on the www.openwings.org web site.

[2] Salutation Architecture Specification, Version 2.0c, Salutation Consortium, June 1, 1999.

[3] Universal Plug and Play Device Architecture, Version 1.0, Microsoft, June 8, 2000.

[4] Ken Arnold et al, The Jini Specification, V1.0 Addison-Wesley 1999. Latest version is 1.1 available from Sun.

[5] Specification of the Home Audio/Video Interoperability (HAVi) Archiecture, V1.1, HAVi, Inc., May 15, 2001.

[6] Service Location Protocol Version 2, Internet Engineering Task Force (IETF), RFC 2608, June 1999.

[7] Specification of the Bluetooth System, Core, Volume 1, Version 1.1, the Bluetooth SIG, Inc., February 22, 2001.

[8] B. Miller and R. Pascoe, Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer, Version 1.0, Bluetooth SIG White paper, July 1, 1999.

[9] C. Bettstetter and C. Renner, "A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol", *Proceedings of the Sixth EUNICE Open European Summer School: Innovative Internet Applications*, EUNICE 2000, Twente, Netherlands, September, 13-15, 2000.

[10] G. Richard, "Service Advertisement and Discovery: Enabling Universal Device Cooperation," *IEEE Internet Computing*, September-October 2000, pp. 18-26.

[11] B. Pascoe, "Salutation Architectures and the newly defined service discovery protocols from Microsoft and Sun: How does the Salutation Architecture stack up," Salutation Consortium whitepaper, June 6, 1999.

[12] Luckham, D. "Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Ordering of Events," http://anna.stanford.edu/rapide, August 1996.

[13] Allen, R. "A Formal Approach to Software Architecture", Ph.D. Thesis, Carnegie Mellon University, CMU Technical Report CMU-CS-97-144, May 1997.

[14] Garlan, D, Monroe, R., and Wile, D., "Acme: An Architecture Description Interchange Language", Proceedings of CASCON '97, Nov. 1997.

[15] Melton, R. "The Aesop System: A Tutorial," Carnegie Mellon University, Pittsburgh, Pennsylvania, 1998.

[16] Moriconi, M & Riemenschneider, R. "Introduction to SADL 1.0: A Language for Specifying Software Architecture Hierarchies," TR SRI-CSL-97-01, March 1997.

[17] Medvidovic, N., P.Oreizy, J. Robbins, and R. Taylor. "Using Object-Oriented Typing to Support Architectural Design in the C2 Style", *Proceedings of SIGSOFT'96: The Fourth Symposium on the Foundations of Software Engineering (FSE4)*, San Francisco, CA, October 16-18, 1996.

[18] Shaw, M. R. DeLine, D.V. Klein, T.L. Ross, D.M. Young, and G. Zelesnik, "Abstractions for Software Architecture and Tools to Support Them," *IEEE Trans. Software Eng.*, vol. 21, no. 4, pp. 314-335, Apr. 1995.

[19] Vestal, S. MetaH User's Manual, Version 1.27, Honeywell Technology Center, Minneapolis, MN 55418, 1997.

[20] Shaw, M., R. DeLine, and G. Zelesnik, "Abstractions and Implementations for Architectural Connections," *Proc. Third Int'l Conf. Configurable Distributed Systems*, May 1996.

[21] Allen, R. and D. Garlan. "Formalizing Architectural Connection", *Proceedings of the Sixteenth International Conference on Software Engineering*, Sorrento, Italy, Ma 1994, pp. 71-80.

[22] J. Rekesh, UPnP, Jini and Salutation - A look at some popular coordination framework for future network devices, Technical Report, California Software Lab, 1999. Available online from http://www.cswl.com/whiteppr/tech/upnp.html.

[23] Garlan, D. "Higher Order Connectors", Workshop on Compositional Software Architectures, Monterey, CA, January, 1998.

[24] R. Pascoe, "Building Networks on the Fly", *IEEE Spectrum*, March 2001, pp. 61-65.

Understanding Consistency Maintenance in Service Discovery Architectures during Communication Failure

Christopher Dabrowski
NIST
NN Room 560
Gaithersburg, Maryland USA 20899
1-301-975-3249
cdabrowski@nist.gov

Kevin Mills
NIST
NN Room 445
Gaithersburg, Maryland USA 20899
1-301-975-3618
kmills@nist.gov

Jesse Elder
NIST
NN Room 579
Gaithersburg, Maryland USA 20899
1-301-975-4411
jelder@nist.gov

ABSTRACT

Current trends suggest future software systems will comprise collections of components that combine and recombine dynamically in reaction to changing conditions. Service-discovery protocols, which enable software components to locate available software services and to adapt to changing system topology, provide one foundation for such dynamic behavior. Emerging discovery protocols specify alternative architectures and behaviors, which motivate a rigorous investigation of the properties underlying their designs. Here, we assess the ability of selected designs for service-discovery protocols to maintain consistency in a distributed system during catastrophic communication failure. We use an architecture description language, called Rapide, to model two different architectures (two-party and three-party) and two different consistency-maintenance mechanisms (polling and notification). We use our models to investigate performance differences among combinations of architecture and consistency-maintenance mechanism as interface-failure rate increases. We measure system performance along three dimensions: (1) update responsiveness (How much latency is required to propagate changes?), (2) update effectiveness (What is the probability that a node receives a change?), and (3) update efficiency (How many messages must be sent to propagate a change throughout the topology?). We use Rapide to understand how failure-recovery strategies contribute to differences in performance. We also recommend improvements to architecture description languages.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications – *methodologies and tools.*
D.2.5 [**Software Engineering**]: Testing and debugging – *symbolic execution and tracing.*
D.2.8 [**Software Engineering**]: Metrics – *performance measures.*

1. INTRODUCTION

Growing deployment of wireless communications, implying greater user mobility, coupled with proliferation of personal digital assistants and other information appliances, foretell a future where software components can never be quite sure about the network connectivity available, about the other software services and components nearby, or about the state of the network neighborhood a few minutes in the future. In extreme situations, as found for example in military applications [1], software components composing a distributed system may find that cooperating components disappear due to physical or cyber attacks or due to jamming of communication channels or movement of nodes beyond communications range. Such environments demand new analysis approaches and tools to design and test software.

In this paper, we use architectural models to assess the ability of selected designs for service-discovery protocols to maintain consistency in a distributed system during catastrophic communication failure. Using an architecture description language (ADL), we model two different architectures (two-party and three-party) and two different consistency-maintenance mechanisms (polling and notification). To provide our models with realistic behaviors, we incorporate consistency-maintenance mechanisms adapted from two specifications: Jini™ Networking Technology[1] [2] and Universal Plug-and-Play (UPnP) [3]. We use our models to investigate performance differences among combinations of architecture and consistency-maintenance mechanism as interface-failure rate increases. We measure system performance along three dimensions: (1) update responsiveness (How much latency is required to propagate changes?), (2) update effectiveness (What is the probability that a node receives a change?), and (3) update efficiency (How many messages must be sent to propagate a change throughout the topology?).

Our modeling and analysis approach builds on earlier work [4] where we derived benefits by creating dynamic models from specifications for service-discovery protocols. Dynamic models

[1] Certain commercial products or company names are identified in this paper to describe our study adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor to imply that the products or names identified are necessarily the best available for the purpose.

enable us to understand collective behavior among distributed components, and to detect ambiguities, inconsistencies and omissions in specifications. In this paper, we apply the same method: (1) construct an architectural model of each discovery protocol, (2) identify and specify relevant consistency conditions that each model should satisfy, (3) define appropriate metrics for comparing the behavior of each model, (4) construct relevant scenarios to exercise the models and to probe for violations of consistency conditions, and (5) compare results from executing similar scenarios against each model. To implement the method, we rely on Rapide [5], an ADL developed at Stanford University. Rapide represents behavior in a form suitable to investigate distributed systems, and comes with an accompanying suite of analysis tools that can execute a specification and can record and visualize system behavior. In this paper, we use Rapide to understand how failure-recovery strategies contribute to differences in performance. Based on our experiences, we also recommend improvements to architecture description languages.

The remainder of the paper is organized in six sections. We begin, in Section 2, by introducing service-discovery protocols and architectures, including a description of procedures to maintain consistency in replicated information. Section 2 also discusses various failures that can interfere with consistency maintenance. In Section 3, we outline some techniques, included in our models, to recover from failures. Section 4 defines an experiment, and related metrics, to compare the performance and overhead exhibited by selected pairings of architecture and consistency-maintenance mechanism while attempting to propagate changes during interface failures. In Section 5, we present results from the experiment, and we discuss causes underlying some of the results. In Section 6, we outline future work to evaluate service-discovery architectures and protocols during message loss and node failure. We conclude in Section 7.

2. SERVICE DISCOVERY SYSTEMS

Service-discovery protocols enable software components in a network to discover each other, and to determine if discovered components meet specific requirements. Further, discovery protocols include *consistency-maintenance mechanisms*, which can be used by applications to detect changes in component availability and status, and to maintain, within some time bounds, a consistent view of components in a network. Many diverse industry activities explore different approaches to meet such requirements, leading to a variety of proposed designs for service-discovery protocols [2, 3, 6-14]. Some industry groups approach the problem from a vertically integrated perspective, coupled with a narrow application focus. Other industry groups propose more widely applicable solutions. For example, a team of researchers and engineers at Sun Microsystems designed Jini Networking Technology [2], a general service-discovery mechanism atop Java™, which provides a base of portable software technology. As another example, a group of engineers at Microsoft and Intel conceived Universal Plug-and-Play [3] in an attempt to extend plug-and-play, an automatic intra-computer device-discovery and configuration protocol, to distributed systems. The proliferation of service discovery protocols motivates deeper analyses of their designs.

To help us compare designs, we developed a general structural model, documented using the UML (Unified Modeling Language). Our general model provides a basis for comparative analysis of various discovery systems by representing the major architectural components with a consistent and neutral terminology (see first column in Table 1). The main components in our general model include: (1) service user (SU), (2) service manager (SM), and (3) service cache manager (SCM), where the SCM is an optional element not supported by all discovery protocols. These components participate in the discovery, information-propagation, and consistency-maintenance processes that comprise discovery protocols. A SM maintains a database of service descriptions, (SDs), each SD encoding the essential characteristics of a particular service or device (Service Provider, or SP). Each SD contains the identity, type, and attributes that characterize a SP. Each SD also includes up to two software interfaces (an application-programming interface and a graphic-user interface) to access a service. A SU seeks SDs maintained by SMs that satisfy specific requirements. Where employed, the SCM operates as an intermediary, matching advertised SDs of SMs to requirements provided by SUs. Table 1 shows how these general concepts map to specific concepts from Jini, UPnP, and the Service Location Protocol (SLP) [8]. The behaviors by which SUs discover and maintain consistency in desired SDs depend partly upon the service-discovery architecture employed.

Table 1. Mapping concepts among service-discovery systems.

Generic Model	Jini	UPnP	SLP
Service User (SU)	Client	Control Point	User Agent
Service Manager (SM)	Service or Device Proxy	Root Device	Service Agent
Service Provider (SP)	Service	Device or Service	Service
Service Description (SD)	Service Item	Device/Service Description	Service Registration
Identity	Service ID	Universal Unique ID	Service URL
Type	Service Type	Device/Service Type	Service Type
Attributes	Attribute Set	Device/Service Schema	Service Attributes
User Interface	Service Applet	Presentation URL	Template URL
Program Interface	Service Proxy	Control/Event URL	Template URL
Service Cache Manager (SCM)	Lookup Service	not applicable	Directory Service Agent (optional)

2.1 Alternative Architectures

Broadly speaking, system architecture comprises a set of components, and the connections among them, along with the relationships and interactions among the components. In our application, we represent the architecture of a discovery system using an architectural model, which expresses structure (as components, connections, and relations), interfaces (as messages received by components), behavior (as actions taken in response to messages received, including generation of new messages), and consistency conditions (as Boolean relations among state variables maintained across different components). Our initial analysis of six distinct discovery systems revealed that most designs use one of two underlying architectures: two-party and three-party.

2.1.1 Two-Party Architectures

A two-party architecture consists of two major components: SMs and SUs. In this study, we use a two-party architecture arranged in a simple topology consisting of one SM and five SUs, as depicted in Figure 1. To animate the architecture, we chose behaviors for discovery, information propagation, and consistency maintenance, as described in the specification for UPnP. Upon startup, each SU and SM engages in a discovery process to locate other relevant components within the network neighborhood. In a lazy-discovery process, each SM periodically announces the existence of its SDs

over the UPnP multicast group, used to send messages from a source to a group of receivers. Upon receiving these announcements, SUs with matching requirements use a HTTP/TCP (HyperText Transfer Protocol/transmission-control protocol) unicast link (for message exchanges between two specific parties) to request, directly from the SM, copies of the SDs associated with relevant SPs. The SU stores SD copies in a local cache. Alternatively, the SU may engage in an aggressive-discovery process, where the SU transmits SD requirements, as *Msearch* queries, on the UPnP multicast group. Any SM holding a SD with matching requirements may use a HTTP/UDP (user-datagram protocol) unicast link to respond (after a jitter delay) directly to the SU. Whenever a UPnP SM responds to an *Msearch* query (or announces itself), it does so with a train of $(3 + 2d + k)$ messages, where d is the number of distinct devices and k is the number of unique service types managed by the SM. For each appropriate response, the SU uses a HTTP/TCP unicast link to request a copy of the relevant SDs, caching them locally.

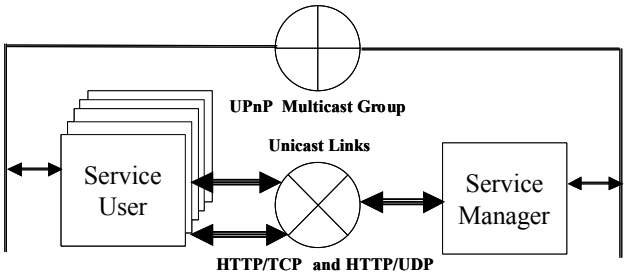


Figure 1. Two-party service-discovery architecture deployed in a six-node topology: five service users (SUs) and one service manager (SM).

To maintain a SD in its local cache, a SU expects to receive periodic announcements from the relevant SM. In UPnP, the SM announces the existence of SDs at a specified interval, known as a Time-to-Live, or TTL. Each announcement specifies the TTL value. If the SU does not receive an announcement from the SM within the TTL (or a periodic SU *Msearch* does not succeed within that time), the SU may discard the discovered SD. We selected the minimum TTL of 1800 s, as recommended by the UPnP specification. (See Tables 2 and 4 for a summary of relevant parameter values used in this paper.)

2.1.2 Three-Party Architectures

A three-party architecture consists of SMs, SUs, and SCMs, where the number of SCMs represents a key variable. In this study, we model a three-party architecture with one SM and five SUs, as shown in Figure 2. We anticipate that under failure conditions, increasing the number of SCMs will increase the chance of successful rendezvous among components, leading to better propagation of information updates from SMs to SUs. To investigate this, we vary the number of SCMs in our three-party architectural model. To animate our three-party model, we choose behaviors described in the Jini specification.

In Jini, the discovery process focuses upon discovery by SMs and SUs of any intermediary SCMs that exist in the network neighborhood. Elsewhere [4], we describe these procedures in detail. Here, we simply summarize. Upon initiation, a Jini component enters aggressive discovery, where it transmits probes on the aggressive-discovery multicast group at a fixed interval (5 s recommended) for a specified period (seven times recommended), or until it has discovered a sufficient number of SCMs. Upon cessation of aggressive discovery, a component enters lazy discovery, where it listens on the lazy-discovery multicast group for announcements sent at intervals (120 s recommended) by SCMs. Our three-party model implements both the aggressive and lazy forms of Jini multicast discovery.

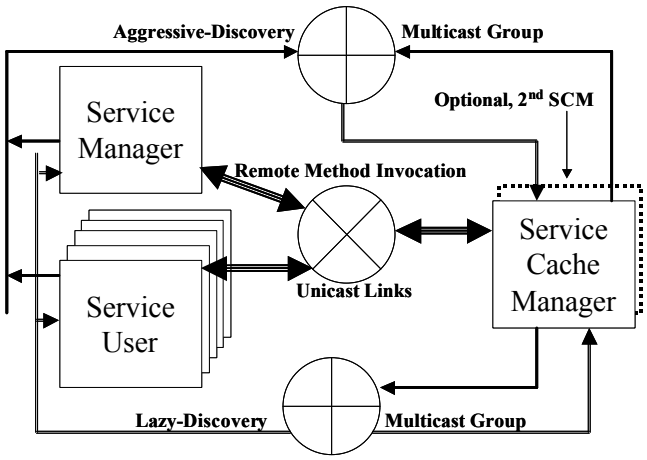


Figure 2. Three-party service-discovery architecture deployed in a seven- or eight-node topology: five service users (SUs), a service manager (SM), and a service cache manager (SCM), with an optional 2nd SCM.

Once discovery occurs, a SM deposits a copy of the SD for each of its services on the discovered SCM. The SCM caches this deposited state, but only for a specified length of time, or TTL. To maintain a SD on the SCM beyond the TTL, a SM must refresh the SD. In this way, if the SM fails, then the SCM can purge any SDs deposited by the SM. To make behavior as consistent as possible across our models for both the two-party and three-party architectures, we selected 1800 s as TTL for a SD to be cached by a SCM. Using these techniques, SUs and SPs rendezvous through SDs registered by SMs with particular SCMs, where the SCMs are found through a discovery process. The SCMs match SDs provided by SMs to SU requirements, and forward matches to SUs, which then access the appropriate SPs.

2.2 Consistency Maintenance Mechanisms

After initial discovery and information propagation (through SDs), service-discovery protocols provide consistency-maintenance mechanisms that applications can use to ensure that changes to critical information propagate throughout the system. Critical information may consist of service availability and capacity, or updates to descriptive information about service capabilities, which may be necessary for a SU to effectively use a discovered service. In our study, we consider two basic consistency-maintenance mechanisms, polling and notification, along with accompanying mechanisms to propagate new information.

2.2.1 Polling

In polling, a SU periodically sends queries to obtain up-to-date information about a SD that was previously discovered, retrieved, and cached locally. In a two-party architecture, the SU issues the

query directly to the SM from which the SD was obtained. In this study, we use the UPnP *HTTP Get* request mechanism to poll the SM to retrieve a SD associated with a specific URL (uniform-resource locator). In response, the SM provides a SD containing a list of all supported services, including their relevant attributes.

Polling in a three-party architecture consists of two independent processes. In one process, a SM sends a *ChangeService* request to propagate an updated SD to each SCM where the SD was originally cached. In the second process, each SU polls relevant SCMs by periodically issuing a *FindService* request, effectively a query with a set of desired SD requirements. The SCM replies with a *MatchFound* that contains the relevant information for any matching SDs. In our study, we adopt a 180-second interval for polling in both architectures.

2.2.2 Notification

In notification, immediately after an update occurs, a SM sends events that announce a SD has changed. To receive events about a SD of interest, a SU must first register for this purpose. In the two-party architecture, the SU registers directly with a SM. We model this procedure using the UPnP event-subscription mechanism, where the SU sends a *Subscribe* request, and the SM responds by either accepting the subscription, or denying the request. The subscription, if accepted, is retained for a TTL, which may be refreshed with subsequent *Subscribe* requests from the SU. In our experiment, we chose 1800 s as TTL for event subscriptions in both architectures.

In a three-party architecture, a SU registers with a SCM to receive events using a procedure analogous to that used by a SM to propagate a SD. As with SD propagation, the SCM grants event registrations for a TTL, which may be refreshed. When a SD update occurs, the SM first issues a *ChangeService* request to all SCMs to which it originally propagated the SD. The SCM then issues a *MatchFound* to propagate the event to all SUs that have registered to receive events about the SD.

2.3 The Nature and Import of Failures

The foregoing discussion, while oversimplified, highlights the complexity inherent in discovery protocols. Additional complexity arises from uncertainty, as nodes, processes, and links can appear and disappear without warning. Discovery protocols must include behavior to cope with such changes. In this section, we address the nature of various failures that can arise, and we consider the implication of such failures on the behavior of discovery protocols, and on the application software that depends upon them.

2.3.1 Classifying Failures

In our research, we focus particularly on failures that can exist within a hostile environment, such as encountered during military or emergency-response operations. We can classify such failures in two general categories: (1) communication failures and (2) process failures. Communication failures can arise due to enemy jamming, or other interference, due to congestion, due to physical severing of cables, due to improperly configured or sabotaged routing tables, or due to multi-path fading as nodes move across a terrain. We can subdivide communication failures into three classes: interface failures, message loss, and path failures. This paper considers only interface failure. A communication interface in a node may fail fully (both transmit and receive) or partially (either transmit or receive). All outbound messages from an interface will be lost when the transmitter fails, while all inbound messages will be lost when the receiver fails. Message loss, a less severe failure, implies that individual messages may be lost, either sporadically or in bursts. Path loss appears as a blocked communication route between two nodes, or areas, in the network. A path can be blocked in one or both directions.

Process failures can be caused by enemy bombardments or cyber attacks, by programming errors, or by hardware failures. We can subdivide process failures into node and thread failures. During a catastrophic failure, processing in a node ceases, and the node must reinitialize before processing resumes. Some information maintained by the node may persist across the failure, while other information may be lost. The nature and condition of persistent information could prove crucial to a node's behavior after processing resumes. Of course, the node might never reappear. Thread failures, while less catastrophic, can be more troublesome than node failures. A node might rely on certain long-running threads to react to events from other nodes. Failure of selected threads can interfere with the operation of the node, as well as other nodes in a distributed system. In some cases, a node can appear to be present, while being effectively inoperable.

2.3.2 Failure Recovery in Service Discovery Systems

In service-discovery systems, failure-recovery responsibilities are divided among three parties: (1) lower-layer protocols, (2) discovery protocols, and (3) applications. Discovery protocols and applications use the services of three classes of lower-layer protocols: (1) unreliable unicast protocols, (2) unreliable multicast protocols, and (3) reliable unicast protocols. Unreliable protocols, whether unicast or multicast, neither recover nor signal lost messages; thus, neither source nor destination will learn of a loss. Further, multicast protocols exchange messages along a tree of receivers. For this reason, a multicast message might be received by some nodes, but not by others. A failure near a multicast source prevents messages from being received by any node in the multicast tree, while a failure near a receiver prevents messages from being received by only a single node in the multicast tree. Of course, failures at intermediate points in the multicast tree could result in messages being lost to subsets of receivers. Since unreliable protocols provide no guarantees, recovery must be provided by mechanisms at a higher layer.

Reliable unicast protocols include mechanisms that attempt to ensure delivery of messages by detecting and retransmitting lost messages. Of course, the reliability schemes may eventually give up if too many retransmissions are needed (which might indicate node, interface, or path failure). In such cases, the reliable unicast protocol will signal to a higher layer that a message could not be delivered. Some ambiguity does exist, however, when using reliable unicast protocols to send request-response message pairs, as is the case for discovery systems. After submitting a request through a reliable unicast protocol, a requesting process might wait for a corresponding response from a remote process. For example, Jini can use Remote Method Invocation (RMI) over TCP to invoke a method on a remote object, and to receive a response. Similarly, UPnP uses TCP to submit HTTP requests and receive HTTP responses. In such cases, the RMI layer or the TCP layer can signal a remote exception (REX). The requesting process cannot determine whether a REX was caused by failure to transmit the request or by failure to receive a response from the remote process. The responding process has more information, as

it does not receive a REX when an inbound request fails, but does receive a REX when its outbound response fails. In essence, while reliable unicast protocols attempt to deliver messages in the face of various communication failures, ultimately the reliability mechanisms might prove insufficient, causing a higher-layer process to be notified of the failure. In such cases, the higher-layer process is free to determine an appropriate recovery strategy.

3. MODELING RECOVERY STRATEGIES

Our architectural models incorporate three classes of failure-recovery strategies: (1) recovery by lower-layer protocols, (2) recovery by discovery protocols, and (3) recovery by application software. For each class, we outline the strategies (see Table 2) included in our models.

Table 2. Summary of recovery responsibilities and strategies as implemented within our models for two- and three-party architectures.

Responsible Party	Recovery Mechanism	Two-Party Architecture (UPnP)	Three-Party Architecture (Jini)
Lower-Layer Protocols	UDP	No recovery	No recovery
	TCP	Issue REX in 30-75 s	Issue REX in 30-75 s
Discovery Protocols	Lazy Discovery	SM: announces with n ($3+2d+k$) messages every 1800 s	SCM: announces every 120 s
	Aggressive Discovery	SU: issues *Msearch* every 120 s (after purging SD)	SU and SM: issue seven probes (at 5 s intervals) only during startup
Application Software	Ignore REX	SU: *HTTP Get* Poll SM: Notification	SU: *FindService* Poll SCM: Notification
	Retry after REX	SU: *HTTP Get* after discovery retry in 180 s (retries ≤ 3) *Subscribe* requests retry in 120s	SM: depositing or refreshing SD copy on SCM retry in 120s SU: registering and refreshing notification requests with SCM retry in 120 s
	Discard Knowledge	SU: purge SD after failure to receive SM announcement within 1800 s	SU and SM: purge SCM after 540 s of continuous REX

3.1 Recovery by Lower-Layer Protocols

Our models operate over two types of channels: unreliable, simulating the UDP in both multicast and unicast forms, and reliable, simulating the TCP. In UDP simulation, we discard messages lost due to transmission errors, and we discard messages lost due to path and interface failures. During path failure, messages can be discarded in one or both directions. During interface failure, we discard all messages sent from a node with a failed transmitter, and we discard all messages inbound for a node with a failed receiver. Neither sender nor receiver learns the fate of lost messages.

In the TCP simulation, our model proves more complex. For messages lost to transmission errors, we schedule a retransmission (roughly within a round-trip time, or RTT). We increase the RTT by about 25% with each successive retransmission. If successive retransmissions exceed a threshold (three in the current study), then we discard the message and issue a REX. For messages lost to interface or path failure, we model TCP connection establishment procedures by discarding the message and waiting for a period, uniformly distributed between an upper and lower bound (30-75 s in the current study), then we signal a REX. When discarding a request, we signal a REX to the requester, but when discarding a response, we signal a REX to both parties.

3.2 Recovery by Discovery Protocols

Discovery protocols include built-in robustness measures to deal with the possibilities of UDP message loss and node failure. Discovery protocols specify periodic transmission of key messages. For example, Jini requires a node to engage in aggressive discovery on startup, and then to enter lazy discovery, where all SCMs periodically announce their presence. In a similar lazy discovery, UPnP requires SMs to periodically announce their presence. While not specifying aggressive discovery, UPnP permits SUs to issue *Msearch* queries at any time. To compensate for the different announcement intervals recommended for Jini and UPnP, we chose to have UPnP SUs issue *Msearch* queries every 120 s, but only after a SU purges a SD from its local cache. Once a SU regains its desired SD, the related *Msearch* queries cease. Whenever a UPnP SM announces itself or responds to an *Msearch* query, it sends n copies of each message, where n is a retransmission factor (two in the current study) recommended by the UPnP specification to compensate for possible UDP message loss. In both Jini and UPnP, each announcement includes a TTL. Receiving nodes can cache the information in the announcement until the TTL expires; then the information must be purged from the cache. In this way, each node in the system eliminates residual information about failed or unreachable nodes. Our models incorporate these failure-recovery behaviors.

3.3 Recovery by Application Software

When discovery nodes communicate over a reliable channel, a REX may occur. Response to a REX is left to the application. In our models, depending on the situation, we implement three different strategies: (1) ignore the REX, (2) retry the operation for some period, and (3) discard knowledge. The retry strategy attempts to recover from transient failures. The discard strategy, which occurs following repeated failure of the retry strategy, relies upon discovery mechanisms to recover from more persistent failures.

3.3.1 Ignore the Remote Exception

In many cases, we simply ignore a REX. In general, our models ignore a REX received when attempting to respond to a request. A SU can ignore a REX received in response to a poll, *FindService* or *HTTP Get*, because the poll recurs at an interval. The SCM (three-party model) or the SM (two-party model) also ignores a REX received while attempting to issue a notification. This behavior, which is described in both the Jini and UPnP specifications, depends upon reliable lower-layer protocols to provide robustness for notifications. Notifications include sequence numbers that allow a receiving node to determine whether or not previous notifications were missed.

3.3.2 Retry the Operation

In our models, we retry selected operations in the face of a REX. The UPnP specification separates the operation of discovering a resource from obtaining a description of the resource (Jini combines these operations). Without a description, the resource cannot be used. For this reason, in our two-party model, a SU must issue a *HTTP Get* to obtain a description. If no description arrives within 180 s, then our model retries the *HTTP Get*. If unsuccessful after three attempts, the SU ceases the retries, but sets a flag reminding itself to reissue a *HTTP Get* when the resource is next announced. Our three-party model, based on Jini,

also contains a retry strategy, but associated with attempts to register or change a SD with a SCM. In these cases, the SM retries a *ChangeService* or *ServiceRegistration* 120 s after receiving a REX. Similarly, when a SU receives a REX (from either a SM or SCM) in response to a request to register for notification, the SU retries the registration in 120 s. These retries occur until some time bounds, after which the SM discards knowledge of the SCM.

3.3.3 Discard Knowledge

Both our two-party and three-party models include the possibility that an application can discard knowledge of previously discovered nodes. In UPnP, after failure to receive announcements from the SM within a TTL, a SU discards a SM and any related SDs. We implement this behavior in our two-party model. In Jini, the specification states that a discovering entity *may* discard a SCM with which it cannot communicate. In our three-party model, a SM or SU deletes a SCM if it receives only REXs when attempting to communicate with the SCM over a 540-s interval. After discarding knowledge of a SM (UPnP) or SCM (Jini), all operations involving the node cease until it is rediscovered, either through lazy discovery (Jini or UPnP announcements) or aggressive discovery (UPnP *Msearch* queries).

4. EXPERIMENT DESIGN AND METRICS

In this paper, we investigate the following question: How do alternative service-discovery architectures, topologies, and consistency-maintenance mechanisms perform under deadline during interface failure? To address this question, we deploy a two-party and three-party architecture (recall Figures 1 and 2), each in a topology that includes one SM and five SUs. In the three-party case, we use two topologies, one with one SCM and another with two SCMs. To establish initial conditions, we exercise each topology until discovery completes, and the initial information (a SD) propagates to all SUs. To begin the experiment, we introduce a change in the SD at the SM, and we establish a deadline, *D*, before which the change must propagate to all SUs. We measure the number of messages exchanged and the latency required to propagate the new information, or until *D*, under two different consistency-maintenance mechanisms: polling and notification. We repeat this experiment while varying the percentage of interface-failure time for each node up to 75% (in increments of 5%). We provide further details below.

Table 3. Experiment combinations.

Architectural Variant	Protocol Basis	Consistency-Maintenance Mechanism
Two-Party	UPnP	Polling
Two-Party	UPnP	Notification (with notification registration on SM)
Three-Party (Single SCM)	Jini	Polling (with service registration on SCM)
Three-Party (Single SCM)	Jini	Notification (with service registration and notification registration on SCM)
Three-Party (Dual SCM)	Jini	Polling (with service registration on SCM)
Three-Party (Dual SCM)	Jini	Notification (with service registration and notification registration on SCM)

4.1 Experiment Combinations

To compare change propagation in two- and three-party architectures, we use our models to combine the architectures with different consistency-maintenance mechanisms. Table 3 depicts the six combinations. Each experiment runs one combination from time zero until *D*, while introducing failures at each node (see 4.3). Each experiment aims to restore consistency among the changed SD held by the SM and the cached copies of the SD held by all of the SUs.

4.2 Tracking Consistency

To track consistency in our experiment, we employ property analysis [4], using a single consistency condition: service attributes for a SD discovered by a SU should have the same values as the attributes of the SD being maintained by the SM that manages the SD. More formally,

FOR All (SM, SU, SD)
(SM, SD [Attributes1]) **isElementOf** SM managed-services **AND**
(SM, SD [Attributes2]) **isElementOf** SU discovered-services
implies Attributes1 **equals** Attributes2

The condition is incorporated directly into our models and checked using Rapide procedural code. We establish an initial system state in which this condition holds, and then introduce a change in (SM, SD [Attributes1]), which negates the condition for all SUs. Then, we monitor updates to (SM, SD) tuples in the set of discovered-services maintained by individual SU's to determine if the condition becomes true. Note that if a SU discards its (SM, SD) tuple, the tuple must be recovered before the condition can be satisfied. These consistency checks form the basis for our measurements.

4.3 Generating Interface Failures

We set aside an interval, up to time *Q*, to complete initial discovery and information propagation. In our experiments, Q = 100 s and D = 5400 s. We choose a time, randomly distributed on the uniform interval Q to $D/2$, to introduce a change into the SD on the SM. We also choose times, randomly distributed on the uniform interval Q to $[D - (D \times F)]$, for each node to suffer an interface failure, where F is the interface-failure rate, which defines the duration of failures as follows. Once activated, each failure remains in effect for a duration of $D \times F$, after which the failure is remedied. We choose interface failures to be of equal and increasing length to give a suitable basis for comparative analysis. When activating each interface failure, we choose with equal likelihood that the transmitter, receiver, or both fail. Table 4 summarizes most of the relevant parameters and values for our experiments.

4.4 A Sample Run

Figure 3 shows partial results from a sample run for the three-party architecture, with two SCMs, using notification as the consistency-maintenance mechanism. In this run, F was 0.05, and so each failure occurred between 100 and 5130 s $[D - (D \times F)]$, and lasted for 270 s $(D \times F)$. Figure 3 shows the time when each interface failed, and recovered. The performance section of the figure lists two times for each node: loss of consistency and restoration of consistency, or D where inconsistency remains. The figure also lists two message counts for each node: messages sent to restore consistency and total messages sent. For each SM and SCM, the first count includes messages sent while any SU remains inconsistent. In this sample run, SUs 1, 2, 4, and 5 and both SCMs became consistent quickly, within 0.00109 s, which represents the time necessary to propagate the change from the SM to at least one SCM, match the changed SD registration to all the SU notification requests registered on the SCM, and forward the matches. However SU 3, whose receiver failed at an inopportune time, never heard the notification and continued in an

inconsistent state for the remainder of the run. This illustrates how lack of robustness in the notification mechanism can lead to prolonged inconsistent states.

Table 4. Values for relevant parameters.

	Parameter	Value
Behavior in both two- and three-party architectures	Polling interval	180 s
	Registration TTL	1800 s
	Time to retry after REX (if applicable)	120 s
UPnP-specific behavior for two-party architecture	Announce interval	1800 s
	Msearch query interval	120 s
	SU purges SD	At TTL expiration
Jini-specific behavior for three-party architecture	Probe interval	5 s (7 times)
	Announce interval	120 s
	SM or SU purges SCM	After 540 s with only REX
Interface failure parameters	Failure incidence	Once per run for each node
	Failure scope	Transmitter, receiver, or both with equal likelihood
	Failure duration	5% increments of 5400 s from 0 to 75%
Transmission and processing delays	UDP transmission delay	10 us constant
	TCP transmission delay	10-100 us uniform
	Per-item processing delay	100 us for cache items 10 us for other items

```
Rate - 5
Run number - 21

SM 1  OUT Interface       down 365, up 635

SCM 1 OUT Interface       down 2417, up 2687
SCM 2 IN & OUT Interface  down 519, up 789

SU 1  IN Interface        down 2238, up 2508
SU 2  IN Interface        down 3256, up 3526
SU 3  IN Interface        down 207,  up 477
SU 4  OUT Interface       down 2876, up 3146
SU 5  IN Interface        down 4478, up 4748

Performance:

  SM  1 346.00000 346.00000 6 17
  SCM 1 346.00000 346.00016 61 102
  SCM 2 346.00000 346.00015 61 105
  SU  1 346.00000 346.00109 0 11
  SU  2 346.00000 346.00109 0 11
  SU  3 346.00000 5400.00000 4 11
  SU  4 346.00000 346.00109 0 11
  SU  5 346.00000 346.00114 0 11
```

Figure 3. Console output from a sample run: three-party, two SCMs, notification, F = 5%, Q =100 s, and D = 5400 s.

4.5 Metrics

We use the data collected from experiment runs to compute three metrics: update responsiveness, update effectiveness, and update efficiency. We define these below.

4.5.1 Update Responsiveness

Assuming information is created at a particular time and must be propagated by a deadline, then the difference between the deadline and the creation time represents available time in which to propagate the information. Update Responsiveness, *R,* measures the proportion of the *available time remaining* after the information is propagated. More formally, let D be a deadline by which we wish to propagate information to each SU-node n in a service discovery topology. Let t_C be the creation time of the information that we wish to propagate, where $t_C < D$. Let $t_{U(n)}$ be the time that the information is propagated to SU n, where $n = 1$ to N, and N is the total number of SUs in a topology. Define change-propagation latency (L) for SU n as: $L_n = (t_{U(n)} - t_C)/(\max(D, t_{U(n)}) - t_C)$. This is effectively the proportion of available time used to propagate the change to SU n. The numerator represents the time at which the SU achieved consistency after the update occurred. The denominator represents the time available to propagate the change. The term $\max(D, t_{U(n)})$ accounts for cases where $t_{U(n)} > D$. Define R for SU n as: $R_n = 1 - L_n$. R_n is the proportion of *available time remaining* after propagating a change to SU n.

4.5.2 Update Effectiveness

Update Effectiveness, U, measures the probability that a change will propagate successfully for a given SU, i.e., $t_{U(n)} < D$. More formally, assuming definitions from 4.5.1 hold, let X be the number of runs (30 here) during which a particular topology is observed under identical conditions. Recalling that N is the total number of SUs in a topology, define the number of SUs observed under identical conditions as: $O = X \cdot N$. Define U, the probability that $t_{U(n)} < D$, as: $U = 1 - P(F)$, where $P(F) = (\Sigma i \Sigma j$ (one if $R_{i,j}$ equals 0 and zero otherwise))$/O$ and where $i = 1...X$ and $j = 1...N$.

4.5.3 Update Efficiency

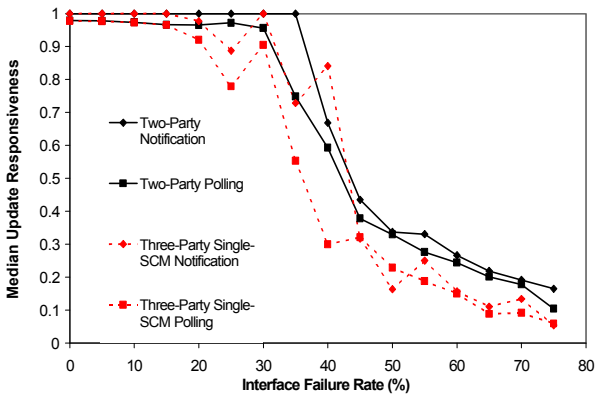
Given a specific service-discovery topology, examination of the available architectures (two-party and three-party) and consistency-maintenance mechanisms (polling and notification) reveals a minimum number of messages, M, that must be sent to propagate a change to all SUs. In our topology, M ($M = 7$) occurs when using notification to propagate information in a three-party architecture with one SCM. Update Efficiency, E, can be defined as the ratio of M to the actual number of messages observed. More formally, let S be the number of messages sent while attempting to propagate a change from a SM to SUs in a given run. Define average E as: $E_{avg} = (\Sigma k(M/S_k))/X$, where $k = 1..X$.

5. RESULTS AND DISCUSSION

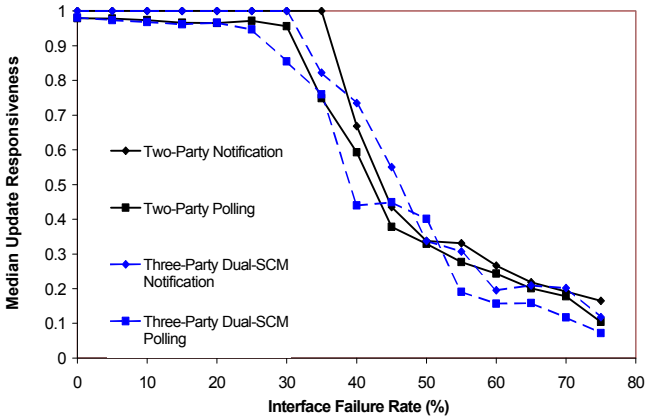
In this section, after showing results from our experiments, we consider the relative performance of our models. We propose reasons for performance differences, subject to further analysis and verification by on-going research. We also use Rapide to examine selected saw-tooth behaviors, and we outline suggestions for improving ADLs (based on our experiences with Rapide).

5.1 Results

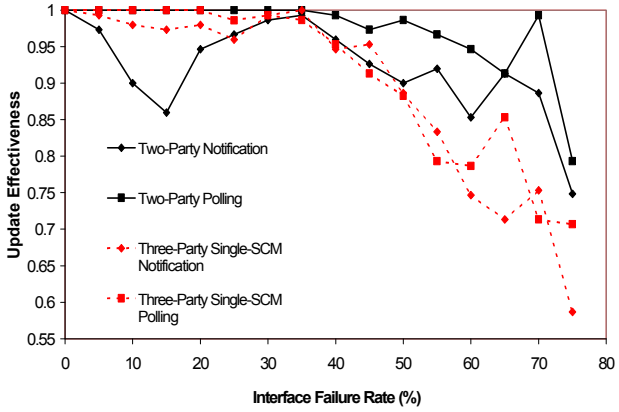
In a series of six graphs, which have identical abscissas (interface-failure rate, increasing from 0% to 75% in increments of 5%) and ordinates (one of the three metrics ranging between 0 and 1), we plot selected measurements generated from our models. Each graph compares four of the configurations in Table 3 against one of the metrics: update responsiveness (median), effectiveness, or efficiency (average). We choose the median as a measure of update responsiveness because the measured data tend to clump in distinct concentrations. Averages proved less representative of the data. Figure 4(a) compares responsiveness from our two-party model against that from our single-SCM, three-party model, for both polling and notification. Figure 4(b) provides a similar comparison, but substitutes the results from our dual-SCM, three-party model in place of results from our one-SCM, three-party model. Figures 4(c) and 4(d) compare update effectiveness using



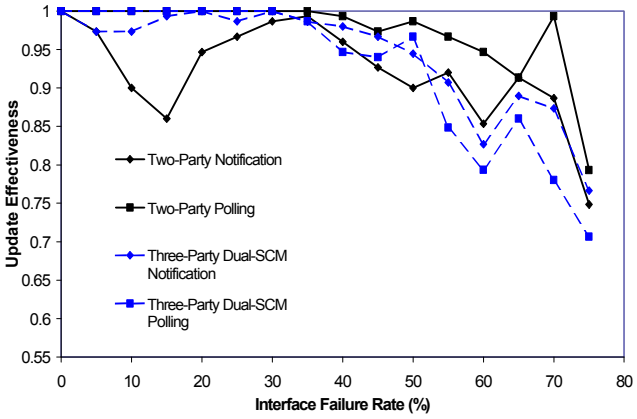
(a) Median Update Responsiveness of Two-Party vs. Three-Party (Single-SCM)



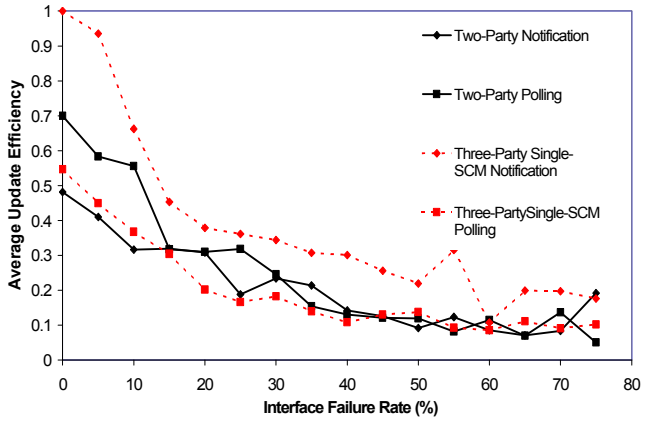
(b) Median Update Responsiveness of Two-Party vs. Three-Party (Dual-SCM)



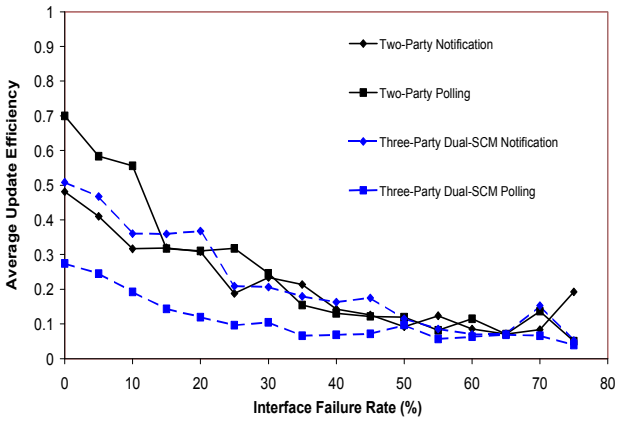
(c) Update Effectiveness of Two-Party vs. Three-Party (Single-SCM)



(d) Update Effectiveness of Two-Party vs. Three-Party (Dual-SCM)



(e) Update Efficiency of Two-Party vs. Three-Party (Single-SCM)



(f) Update Efficiency of Two-Party vs. Three-Party (Dual-SCM)

Figure. 4. Graphs comparing combinations of architecture, topology, and consistency-maintenance mechanism.

the same combinations. Figures 4(e) and 4(f) use the same combinations, but compare update efficiency. The graphs reporting measures of responsiveness and effectiveness depict a system undergoing a phase-transition from peak performance (where changes propagate quickly) to non-performance (where changes fail to propagate). Regarding efficiency, the graphs show a system that begins at its best efficiency (without interfering failures) and then asymptotically approaches zero efficiency as the failure rate increases toward 100%. The graphs (particularly those showing update effectiveness) also depict several eccentricities, in the form of saw-tooth behaviors. Using the analysis and visualization tools provided by Rapide, we were able to investigate the causes underlying these eccentricities (see 5.3). Because the graphs can be difficult to interpret, we compute summary statistics (see Table 5) for each of our six combinations. Each summary statistic reflects the mean of a particular metric, when averaged across all interface-failure rates, for a specified configuration. To indicate the uncertainty associated with our measurements, we also give (see Table 6) the upper and lower bounds (computed using an appropriate standard error formula for each metric) associated with selected interface-failure rates (5%, 40%, and 75%) for each of our curves.

Table 5. Summary statistics (mean across all interface-failure rates) computed for each curve given in the graphs shown in Figures 4(a) through 4(f).

	Mean (across all interface-failure rates)		
	Median Responsiveness	Effectiveness	Average Efficiency
Two-Party Notification	0.663	0.921	0.212
Two-Party Polling	0.615	0.973	0.251
Three-Party Notification (Single SCM)	0.601	0.894	0.389
Three-Party Polling (Single SCM)	0.530	0.911	0.201
Three-Party Notification (Dual SCM)	0.655	0.942	0.221
Three-Party Polling (Dual SCM)	0.587	0.927	0.110

5.2 Understanding Relative Performance

Below, we discuss the results for each of our three metrics. The reader should note that engineering trade-offs exist among these metrics: responsiveness, effectiveness, and efficiency.

5.2.1 Responsiveness

Results in Figs. 4(a) and 4(b) and the first column of Table 5 show that the various combinations of architecture and behavior exhibit similar responsiveness, where the mean median ranges between 0.663 and 0.530. Table 6, which reports uncertainty in the results, confirms a rough similarity in responsiveness. Similarity arises because interface failures interfere with both polling and notification, requiring nodes to rely on recovery mechanisms in the underlying discovery protocols to restore consistency. Absent failures, notification proves more responsive because change notices are issued to interested parties immediately after a change occurs, while polling incurs some lag time. The presence of interface failures complicates the situation. First, if a required interface is not operating when a notification is issued, then an update will be lost. Second, when polls fail for an extended period (likely during high interface-failure rates), then polling ceases and updates can be missed. Under both (polling and notification) mechanisms, restoring consistency depends upon the recovery mechanisms in the discovery protocol.

Table 6. Depicts upper and lower bounds of the 95% C.I., computed using appropriate statistical techniques, for each metric and all experiment combinations at selected interface-failure rates.

	Responsiveness			Effectiveness			Efficiency		
	5%	40%	75%	5%	40%.	75%	5%	40%	75%
Two-Party Notification	1.000 1.000	0.561 0.783	0.111 0.162	0.970 0.977	0.954 0.966	0.709 0.787	0.354 0.467	0.065 0.220	0.031 0.354
Two-Party Polling	0.975 0.980	0.501 0.849	0.076 0.138	1.000 1.000	0.993 0.993	0.760 0.826	0.501 0.666	0.031 0.230	0.042 0.059
Three-Party Notification (Single SCM)	1.000 1.000	0.605 1.000	0.042 0.095	0.993 0.993	0.939 0.955	0.521 0.652	0.827 1.000	0.099 0.504	0.033 0.320
Three-Party Polling (Single SCM)	0.974 0.980	0.244 0.412	0.043 0.083	1.000 1.000	0.946 0.960	0.660 0.753	0.387 0.512	0.043 0.173	0.040 0.164
Three-Party Notification (Dual SCM)	1.000 1.000	0.562 1.000	0.099 0.143	0.970 0.977	0.977 0.983	0.730 0.803	0.335 0.599	0.035 0.290	0.009 0.096
Three-Party Polling (Dual SCM)	0.974 0.986	0.391 0.543	0.056 0.096	1.000 1.000	0.939 0.955	0.660 0.753	0.218 0.273	0.033 0.103	0.019 0.059

The recovery mechanisms, as implemented in our models, exhibit similar responsiveness: rediscovery of lost nodes will occur within 120 s after restoration of a failed interface. In the three-party case, periodic (120 s) announcements by each SCM (lazy-discovery procedures) ensure rediscovery. Similarly, in the two-party model, the periodic (120 s) *Msearch* queries by each SU (aggressive-discovery procedures) also ensure rediscovery. In this way, restoration of a failed interface leads to rediscovery of lost nodes, and to restoration of consistency in cached copies of SDs. As the interface-failure rate increases beyond 30%, the rediscovery machinery tends to dominate the responsiveness results (see 5.4 for further discussion of recovery mechanisms).

5.2.2 Effectiveness

Results in Figs. 4(c) and 4(d) and the second column of Table 5 show that certain combinations lead to better update effectiveness, and Table 6 suggests that these differences could be significant. Differences in effectiveness may be partly attributed to architecture and topology. For example, each SD copy must propagate over either one link (two-party case) or two links (three-party case). For this reason, the three-party architecture (single SCM) can prove more vulnerable to interface failures (two links must be operational). This suggests that a two-party architecture will be more effective under severe interface failures, and our results support this. On the other hand, the three-party architecture allows replication of SCMs, which provides a greater number of paths through which information can propagate. This suggests (and our results agree) that the three-party architecture with the dual SCM should provide superior effectiveness over the single-SCM, three-party architecture. Our results also indicate that the dual-SCM three-party architecture yields effectiveness close to that of the two-party architecture. Adding SCMs will likely improve the effectiveness of the three-party architecture by increasing path redundancy in the topology.

Differences in effectiveness may also be attributed in part to consistency-maintenance mechanism. In general, polling should lead to better effectiveness than notification. Our results support this for the two-party architecture and for the three-party architecture with a single SCM. Polling has built-in robustness from issuing periodic requests. On the contrary, in both two- and three-party architectures, each notification is issued only once

with no further action by the sender in response to a REX (recall Table 2). In two-party notification, effectiveness suffers from situations where the notice is lost but the SM is not lost (because announcements occur only every 1800 s and thus an interface failure can be restored before the next announcement). In these situations, rediscovery does not occur and the change will not be propagated (see 5.3).

5.2.3 Efficiency

For a given combination of architecture and topology, we expect that notification would be more efficient than polling. We also expect that the two-party architecture would be more efficient than the three-party architecture, and that the single-SCM topology would be more efficient than the dual-SCM topology. In general, our results support these expectations, but with a few twists. The three-party, single-SCM architecture with notification proves more efficient than the two-party architectures because in Jini the SD arrives with the notification, while in UPnP notifications indicate only that a change has occurred, requiring a SU to exchange a request-response message pair to obtain the updated SD.

In notification, efficiency also decreases as the failure rate increases because SUs need to recover from REXs associated with refreshing remote registrations. Each SU must periodically refresh notification requests deposited on the SM (two-party case) or SCM (three-party case). Interface failures lead to REXs during refresh attempts. A REX invokes retry procedures: every 120 s until 540 s of continuous REX (three-party case) or every 120 s until a SM is purged (two-party case).

5.3 Investigating Saw-Tooth Phenomena

A number of the curves shown in Figures 4(a)-(f), exhibit saw-tooth phenomena, most pronounced for update effectiveness, particularly for the two-party architecture with notification. Our uncertainty calculations suggest that at failure rates above 40% these spikes may be attributed to random variations, which might be reduced by increasing the number of runs at each failure rate (currently 30) and the corresponding number of data points (currently 5 SUs x 30 runs = 150). On the other hand, spikes at lower failure rates appear more likely due to causal behavior in our models. For example, the two-party architecture with notification exhibits a significant dip at 15% interface-failure rate.

Using visualization and analysis tools included with Rapide, we examined the partially ordered sets of events (POSETs) that display the complete causal behavior of our model. The POSETs revealed that at the 15% interface-failure rate a large number of notifications were lost when either the SM transmitter was inoperable (causing notifications to all SUs to be lost) or when SU receivers were inoperable (causing lost notifications to individual SUs). Recovery from notification loss depends upon a SU discarding a SM, and then rediscovering the SM, and retrieving related SDs. A SU discards a SM when it fails to receive an announcement from the SM within the specified time. Unfortunately, in many cases, a failed interface that caused a notification loss was repaired prior to the next SM announcement (announcements come every 1800 s). In such cases, the SU does not purge the SM, and therefore there is no rediscovery. Without rediscovery, there is no mechanism to restore consistency; thus, lost notifications lead to inconsistencies that persist to the deadline (and beyond).

Why does this behavior not appear with notification in the three-party architecture? The three-party architecture requires a SM to first propagate a change to a SCM. The SCM then propagates the change on to SUs that requested notification. While notification from SCM to SU is unprotected, on failure a SM retries change propagation to a SCM. An inoperable SCM transmitter leads not only to failure to propagate notifications to SUs, but also to failure to confirm the change propagated by the SM. Absent confirmation, the SM retries the change for up to 540 s, during which time the SCM transmitter might be restored. Each repeated change that propagates to the SCM also causes notifications to be sent to the relevant SUs. Thus for SCM transmitter failures, we conclude that robustness in change propagation from SM to SCM compensates for lack of robustness in notifications from SCM to SU. No equivalent serendipity occurs in the two-party architecture. These cases suggest relationships between the timing and scope of failures and the role of recovery mechanisms in the different architectures.

5.4 Role of Recovery Mechanisms

Under hostile conditions, such as those in our experiments, recovery mechanisms play a key role in consistency maintenance. For example, a detailed analysis of results from our two-party architectural model show that at 30% failure rate and below, interface failures tend to be restored more frequently within the REX retry period associated with *HTTP Get* requests; thus, application recovery contributes substantially to update effectiveness. Above 30% failure rate, application recovery tends to exhaust its allotted time, leading a SU to discard knowledge of the SM. In such cases, update effectiveness depends primarily on robustness mechanisms built into the discovery protocol. We plan additional analysis to establish the contribution to update effectiveness of various recovery strategies in both two- and three-party architectures.

5.5 Recommendations for Improving ADLs

While the Rapide ADL provided useful abstractions to represent and analyze the structure and behavior of service-discovery protocols under failure, we recommend some improvements that apply generally to ADLs. First, this study reinforces our previous recommendations [4] that component states should be selectively exportable to allow data extraction and recording for analysis. Such an export mechanism would also assist in implementing techniques to evaluate consistency conditions that involve state variables from two or more components *and* that consider time, two important considerations when analyzing component interactions. We note that some ADLs include constraint-analysis engines that consider time [e.g., 15]. Second, ADLs, and especially their tools, must provide representations of behavior that can be evaluated efficiently. For example, to bound POSET size in this study, we were forced to substitute procedure calls in place of Rapide constraint evaluation. Third, we would find it convenient if ADL tools supported the same statistical techniques available from commercial simulation systems. For example, ADL tools might include mechanisms to track and summarize statistics about selected state variables. ADLs might also include machinery to apply statistical tests to selected variables across experiment runs in order to automate halting decisions. We expect to develop additional recommendations as our work proceeds.

6. FUTURE WORK

We envision future work along three general directions. First, we intend to complete our characterization of performance for various combinations of architecture, topology, and behavior during failures. We will model the effects of message loss, which appear likely to differ significantly from those described in this study, and we will assess the ramification of node failure on discovery and recovery mechanisms in various architectures and topologies. Second, we plan to model and evaluate selected changes to improve the performance of discovery architectures and protocols in response to failure. Here, our goal is to increase the fault-tolerance of such systems. We intend to implement and evaluate our most promising suggested changes in publicly available service-discovery software. Third, we will expand our generic structural model of service-discovery architectures to include message exchanges and verifiable consistency conditions.

Along a different dimension, we hope to improve methodologies available to design and engineer distributed software systems. At present, many publicly available specifications come with one or more reference implementations. We hope to demonstrate that architectural models lead to better understanding of the properties of distributed systems. In addition, we aim to improve ADLs, and associated tools, by providing recommendations based on our experience. We are also considering developing our own modeling and analysis tools especially designed for understanding collective behavior in multi-party distributed systems.

7. CONCLUSIONS

Emerging service-discovery protocols provide the foundation for software components to discover each other, to organize themselves into a system, and to adapt to changes in system topology. While likely suitable for small-scale commercial applications, questions remain regarding the performance of such protocols at large scale, and during periods of high volatility and duress, such as might exist in military and emergency-response applications. In this paper, we used architectural models to characterize the performance of selected combinations of system topology and consistency-maintenance mechanism during catastrophic communication failure. Further, we used behavioral analysis to investigate causes underlying observed performance. Our initial investigations show significant differences in update effectiveness can be obtained by varying aspects of the design (architecture, topology, consistency-maintenance mechanism, and recovery strategies). Our results also suggest relationships among interface-failure rate, failure timing, and recovery strategies.

8. ACKNOWLEDGMENTS

The work described benefits from financial support provided by the National Institute of Standards and Technology (NIST), the Defense Advanced Research Projects Agency (DARPA), and the Advanced Research Development Agency (ARDA). In particular, we acknowledge the support of Susan Zevin from NIST, Doug Maughan and John Salasin from DARPA, and Greg Puffenbarger from ARDA. We also thank Stefan Leigh of NIST and the anonymous WOSP reviewers for insightful comments that helped us to improve the manuscript.

9. REFERENCES

[1] G. Bieber and J. Carpenter, "Openwings A Service-Oriented Component Architecture for Self-Forming, Self-Healing, Network-Centric Systems," on the web site: http://www.openwings.org.

[2] Ken Arnold et al, The Jini Specification, V1.0 Addison-Wesley 1999. Latest version is 1.1 available from Sun.

[3] Universal Plug and Play Device Architecture, Version 1.0, Microsoft, June 8, 2000.

[4] Dabrowski, C. and Mills, K., "Analyzing Properties and Behavior of Service Discovery Protocols Using an Architecture-Based Approach", *Proceedings of Working Conference on Complex and Dynamic Systems Architecture*, Brisbane, Australia, December 2001.

[5] Luckham, D. "Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Ordering of Events," http://anna.stanford.edu/rapide, August 1996.

[6] Salutation Architecture Specification, Version 2.0c, Salutation Consortium, June 1, 1999.

[7] Specification of the Home Audio/Video Interoperability (HAVi) Archiecture, V1.1, HAVi, Inc., May 15, 2001.

[8] Service Location Protocol Version 2, Internet Engineering Task Force (IETF), RFC 2608, June 1999.

[9] Specification of the Bluetooth System, Core, Volume 1, Version 1.1, the Bluetooth SIG, Inc., February 22, 2001.

[10] B. Miller and R. Pascoe, Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer, Version 1.0, Bluetooth SIG White paper, July 1, 1999.

[11] C. Bettstetter and C. Renner, "A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol", *Proceedings of the Sixth EUNICE Open European Summer School: Innovative Internet Applications*, EUNICE 2000, Twente, Netherlands, September, 13-15, 2000.

[12] G. Richard, "Service Advertisement and Discovery: Enabling Universal Device Cooperation," *IEEE Internet Computing*, September-October 2000, pp. 18-26.

[13] B. Pascoe, "Salutation Architectures and the newly defined service discovery protocols from Microsoft and Sun: How does the Salutation Architecture stack up," Salutation Consortium whitepaper, June 6, 1999.

[14] J. Rekesh, UPnP, Jini and Salutation - A look at some popular coordination framework for future network devices, Technical Report, California Software Lab, 1999. Available online from http://www.cswl.com/whiteppr/tech/upnp.html.

[15] Allen, R. A Formal Approach to Software Architecture, Ph.D. Thesis, Carnegie Mellon University, CMU Technical Report CMU-CS-97-144, May 1997.

Understanding Consistency Maintenance in Service Discovery Architectures in Response to Message Loss

Christopher Dabrowski, Kevin Mills, Jesse Elder
National Institute of Standards and Technology
Gaithersburg, Maryland USA
{cdabrowski, kmills, jelder}@nist.gov

Abstract

Current trends suggest future software systems will comprise collections of components that combine and recombine dynamically in reaction to changing conditions. Service-discovery protocols, which enable software components to locate available software services and to adapt to changing system topology, provide one foundation for such dynamic behavior. Emerging discovery protocols specify alternative architectures and behaviors, which motivate a rigorous investigation of the properties underlying their designs. Here, we assess the ability of selected designs for service-discovery protocols to maintain consistency in a distributed system during severe message loss. We use an architecture description language, called Rapide, to model two different architectures (two-party and three-party) and two different consistency-maintenance mechanisms (polling and notification). We use our models to investigate performance differences among combinations of architecture and consistency-maintenance mechanism as message-loss rate increases. We measure system performance along three dimensions: (1) update responsiveness (How much latency is required to propagate changes?), (2) update effectiveness (What is the probability that a node receives a change?), and (3) update efficiency (How many messages must be sent to propagate a change throughout the topology?).

1. Introduction

Successful deployment of active middleware services, which can detect and adapt to changes in topologies of distributed components, will depend upon a foundation layer of service-discovery software that can monitor the state of nearby software services and components and that can detect changes in network connectivity. Already, military organizations are investigating the applicability of commercial service-discovery systems to meet such requirements in hostile and volatile environments [1]. In military and civil emergency response situations, software components in a distributed system may find that cooperating components disappear due to physical or cyber attacks, to jamming of communication channels or to movement of nodes. Such environments demand new analysis approaches and tools to design and test software that will be used to provide active middleware services.

In this paper, we use architectural models to assess the ability of selected designs for service-discovery protocols to maintain consistency in a distributed system during severe message loss. (A companion paper investigates robustness in the face of interference due to node interface failure [2].) Using an architecture description language (ADL), we model two different architectures (two-party and three-party) and two different consistency-maintenance mechanisms (polling and notification). To provide our models with realistic behaviors, we incorporate consistency-maintenance mechanisms adapted from two specifications: Jini™ Networking Technology[1] [3] and Universal Plug-and-Play (UPnP) [4]. We use our models to investigate performance differences among combinations of architecture and consistency-maintenance mechanism as message-loss rate increases. We measure system performance along three dimensions: (1) update responsiveness (How much latency is required to propagate changes?), (2) update effectiveness (What is the probability that a node receives a change?), and (3) update efficiency (How many messages must be sent to propagate a change throughout the topology?).

Our modeling and analysis approach builds on earlier work [5] where we derived benefits by creating dynamic models from specifications for service-discovery protocols. Dynamic models enable us to understand collective behavior among distributed components, and to detect ambiguities, inconsistencies and omissions in specifications. In this paper, we apply the same method: (1) construct an architectural model of each discovery protocol, (2) identify and specify relevant consistency conditions that each model should satisfy, (3) define appropriate metrics for comparing the behavior of each model, (4) construct relevant scenarios to exercise the

[1] Certain commercial products or company names are identified in this paper to describe our study adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor to imply that the products or names identified are necessarily the best available for the purpose.

models and to probe for violations of consistency conditions, and (5) compare results from executing similar scenarios against each model. To implement the method, we rely on Rapide [6], an ADL developed at Stanford University. Rapide represents behavior in a form suitable to investigate distributed systems, and comes with an accompanying suite of analysis tools that can execute a specification and can record and visualize system behavior. In this paper, we use Rapide to understand how failure-recovery strategies contribute to differences in performance.

The remainder of the paper is organized in six sections. We begin, in Section 2, by introducing service-discovery protocols and architectures, including a description of procedures to maintain consistency in replicated information. In Section 3, we outline some techniques, included in our models, to recover from failures. Section 4 defines an experiment, and related metrics, to compare the performance and overhead exhibited by selected pairings of architecture and consistency-maintenance mechanism while attempting to propagate changes during message loss. In Section 5, we present results from the experiment, and we discuss causes underlying some of the results. We conclude in Section 6.

2. Service discovery systems

Service-discovery protocols enable software components in a network to discover each other, and to determine if discovered components meet specific requirements. Further, discovery protocols include *consistency-maintenance mechanisms*, which can be used by applications to detect changes in component availability and status, and to maintain, within some time bounds, a consistent view of components in a network. Many diverse industry activities explore different approaches to meet such requirements, leading to a variety of proposed designs for service- discovery protocols [3, 4, 7-10]. Some industry groups approach the problem from a vertically integrated perspective, coupled with a narrow application focus. Other industry groups propose more widely applicable solutions. For example, a team of researchers and engineers at Sun Microsystems designed Jini Networking Technology [3], a general service-discovery mechanism atop Java™, which provides a base of portable software technology. As another example, a group of engineers at Microsoft and Intel conceived Universal Plug-and-Play [4] in an attempt to extend plug-and-play, an automatic intra-computer device-discovery and configuration protocol, to distributed systems. The proliferation of service-discovery protocols motivates deeper analyses of their designs.

To help us compare designs, we developed a general structural model, documented using the UML (Unified Modeling Language). Our general model provides a basis for comparative analysis of various discovery systems by representing the major architectural components with a consistent and neutral terminology (see first column in Table 1). The main components in our general model include: (1) service user (SU), (2) service manager (SM), and (3) service cache manager (SCM). The SCM is an optional element not supported by all discovery protocols. These components participate in the discovery, information-propagation, and consistency-maintenance processes that comprise discovery protocols. A SM maintains a database of service descriptions, (SDs), each SD encoding the essential characteristics of a particular service or device (Service Provider, or SP). Each SD contains the identity, type, and attributes that characterize a SP. Each SD also includes up to two software interfaces (an application-programming interface and a graphical-user interface) to access a service. A SU seeks SDs maintained by SMs that satisfy specific requirements. Where employed, the SCM operates as an intermediary, matching advertised SDs of SMs to requirements provided by SUs. Table 1 shows how these general concepts map to specific concepts from Jini, UPnP, and the Service Location Protocol (SLP) [9]. The behaviors by which SUs discover and maintain consistency in desired SDs depend partly upon the service-discovery architecture employed.

Generic Model	Jini	UPnP	SLP
Service User (SU)	Client	Control Point	User Agent
Service Manager (SM)	Service or Device Proxy	Root Device	Service Agent
Service Provider (SP)	Service	Device or Service	Service
Service Description (SD)	Service Item	Device/Service Description	Service Registration
Identity	Service ID	Universal Unique ID	Service URL
Type	Service Type	Device/Service Type	Service Type
Attributes	Attribute Set	Device/Service Schema	Service Attributes
User Interface	Service Applet	Presentation URL	Template URL
Program Interface	Service Proxy	Control/Event URL	Template URL
Service Cache Manager (SCM)	Lookup Service	not applicable	Directory Service Agent (optional)

Table 1. Mapping concepts among service-discovery systems.

2.1 Alternative architectures

Broadly speaking, system architecture comprises a set of components, and the connections among them, along with the relationships and interactions among the components. In our application, we represent the architecture of a discovery system using an architectural model, which expresses structure (as components, connections, and relations), interfaces (as messages received by components), behavior (as actions taken in response to messages received, including generation of new messages), and consistency conditions (as Boolean relations among state variables maintained across different components). Our initial analysis of six distinct discovery systems revealed that most designs use one of two underlying architectures: two-party or three-party.

2.1.1 Two-party architectures. A two-party architecture consists of two major components: SMs and SUs. In this study, we use a two-party architecture arranged in a simple topology consisting of one SM and five SUs, as depicted in Figure 1. To animate the architecture, we chose behaviors for discovery, information propagation, and consistency maintenance, as described in the specification for UPnP. Upon startup, each SU and SM engages in a discovery process to locate other relevant components within the network neighborhood. In a lazy-discovery process, each SM periodically announces the existence of its SDs over the UPnP multicast group, used to send messages from a source to a group of receivers. Upon receiving these announcements, SUs with matching requirements use a HTTP/TCP (HyperText Transfer Protocol/transmission-control protocol) unicast link (for message exchanges between two specific parties) to request, directly from the SM, copies of the SDs associated with relevant SPs. The SU stores SD copies in a local cache. Alternatively, the SU may engage in an aggressive-discovery process, where the SU transmits SD requirements, as *Msearch* queries, on the UPnP multicast group. Any SM holding a SD with matching requirements may use a HTTP/UDP (user-datagram protocol) unicast link to respond (after a jitter delay) directly to the SU. Whenever a UPnP SM responds to an *Msearch* query (or announces itself using the lazy discovery process), it does so with a train of $(3 + 2d + k)$ messages, where d is the number of distinct devices and k is the number of unique service types managed by the SM. For each appropriate response, the SU uses a HTTP/TCP unicast link to request a copy of the relevant SDs, caching them locally.

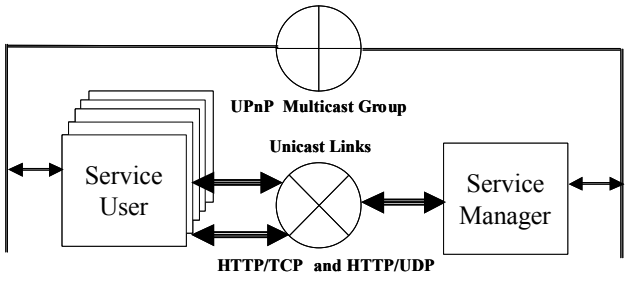


Figure 1. Two-party service-discovery architecture deployed in a six-node topology: five service users (SUs) and one service manager (SM).

To maintain a SD in its local cache, a SU expects to receive periodic announcements from the relevant SM. In UPnP, the SM announces the existence of SDs at a specified interval, known as a Time-to-Live, or TTL. Each announcement specifies the TTL value. If the SU does not receive an announcement from the SM within the TTL (or a periodic SU *Msearch* does not succeed within that time), the SU may discard the discovered SD. We selected the minimum TTL of 1800 s, as recommended by the UPnP specification. (See Tables 2 and 4 for a summary of relevant parameter values used in this paper.)

2.1.2 Three-party architectures. A three-party architecture consists of SMs, SUs, and SCMs, where the number of SCMs represents a key variable. In this study, we model a three-party architecture with one SM and five SUs, as shown in Figure 2. We anticipate that under failure conditions, increasing the number of SCMs will increase the chance of successful rendezvous among components, leading to better propagation of information updates from SMs to SUs. To investigate this, we vary the number of SCMs in our three-party architectural model. To animate our three-party model, we chose behaviors described in the Jini specification.

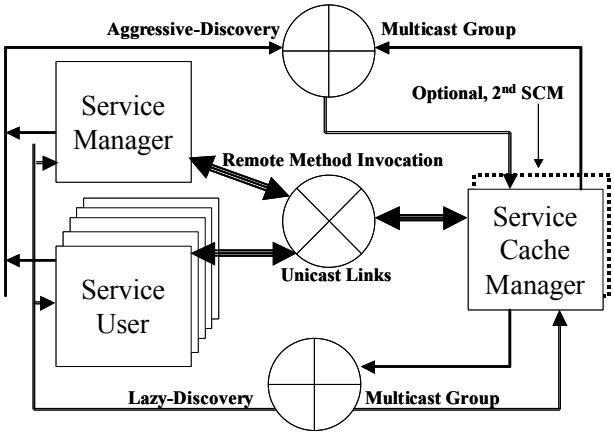


Figure 2. Three-party service-discovery architecture deployed in a seven- or eight-node topology: five service users (SUs), a service manager (SM), and a service cache manager (SCM), with an optional 2nd SCM.

In Jini, the discovery process focuses upon discovery by SMs and SUs of any intermediary SCMs that exist in the network neighborhood. Elsewhere [5], we describe these procedures in detail. Here, we simply summarize. Upon initiation, a Jini component enters aggressive discovery, where it transmits probes on the aggressive-discovery multicast group at a fixed interval (5 s recommended) for a specified period (seven times recommended), or until it has discovered a sufficient number of SCMs. Upon cessation of aggressive discovery, a component enters lazy discovery, where it listens on the lazy-discovery multicast group for announcements sent at intervals (120 s recommended) by SCMs. Our three-party model implements both the aggressive and lazy forms of Jini multicast discovery. Once discovery occurs, a SM deposits a copy of the SD for each of its services on the discovered SCM. The SCM caches this deposited state, but only for a specified length of time, or TTL. To

maintain a SD on the SCM beyond the TTL, a SM must refresh the SD. In this way, if the SM fails, then the SCM can purge any SDs deposited by the SM. To make behavior as consistent as possible across our models for both the two-party and three-party architectures, we selected 1800 s as TTL for a SD to be cached by a SCM. Using these techniques, SUs and SPs rendezvous through SDs registered by SMs with particular SCMs, where the SCMs are found through a discovery process. The SCMs match SDs provided by SMs to SU requirements, and forward matches to SUs, which then access the appropriate SPs.

2.2 Consistency maintenance mechanisms

After initial discovery and information propagation (through SDs), service-discovery protocols provide consistency-maintenance mechanisms that applications can use to ensure that changes to critical information propagate throughout the system. Critical information may consist of service availability and capacity, or updates to descriptions of service capabilities, which may be necessary for a SU to effectively use a discovered service. In our study, we consider two basic consistency-maintenance mechanisms, polling and notification, along with accompanying mechanisms to propagate updates.

2.2.1 Polling. In polling, a SU periodically sends queries to obtain up-to-date information about a SD that was previously discovered, retrieved, and cached locally. In a two-party architecture, the SU issues the query directly to the SM from which the SD was obtained. In this study, we use the UPnP *HTTP Get* request mechanism to poll the SM to retrieve a SD associated with a specific URL (uniform resource locator). In response, the SM provides a SD containing a list of all supported services, including their relevant attributes.

Polling in a three-party architecture consists of two independent processes. In one process, a SM sends a *ChangeService* request to propagate an updated SD to each SCM where the SD was originally cached. In the second process, each SU polls relevant SCMs by periodically issuing a *FindService* request, effectively a query with a set of desired SD requirements. The SCM replies with a *MatchFound* that contains the relevant information for any matching SDs. In our study, we adopt a 180-s interval for polling in both architectures.

2.2.2 Notification. In notification, immediately after an update occurs, a SM sends events that announce a SD has changed. To receive events about a SD of interest, a SU must first register for this purpose. In the two-party architecture, the SU registers directly with a SM. We model this procedure using the UPnP event-subscription mechanism, where the SU sends a *Subscribe* request, and the SM responds by either accepting the subscription, or denying the request. The subscription, if accepted, is retained for a TTL, which may be refreshed with subsequent *Subscribe* requests from the SU. In our experiment, we chose 1800 s as TTL for event subscriptions in both architectures.

In a three-party architecture, a SU registers with a SCM to receive events using a procedure analogous to that used by a SM to propagate a SD. As with SD propagation, the SCM grants event registrations for a TTL, which may be refreshed. When a SD update occurs, the SM first issues a *ChangeService* request to all SCMs to which it originally propagated the SD. The SCM then issues a *MatchFound* to propagate the event to all SUs that have registered to receive events about the SD.

3. Modeling recovery strategies

Elsewhere [2], we discuss the classes of network failures occurring in hostile environments and describe failure-recovery mechanisms of lower-layer protocols in more detail. Here we address recovery in response to message loss at a more general level. Our architectural models incorporate three classes of failure-recovery strategies: (1) recovery by lower-layer protocols, (2) recovery by discovery protocols, and (3) recovery by application software. For each class, we outline the strategies (see Table 2) included in our models.

3.1 Recovery by lower layers

Our models operate over two types of channels: unreliable, simulating the UDP in both multicast and unicast forms, and reliable, simulating the TCP. UDP provides no guarantee of message delivery; therefore our simulated unreliable channels discard messages lost due to transmission errors. Neither sender nor receiver learns the fate of lost messages.

Responsible Party	Recovery Mechanism	Two-Party Architecture (UPnP)	Three-Party Architecture (Jini)
Lower-Layer Protocols	UDP	No recovery	No recovery
	TCP	Issue REX in 78 s if connection establishment fails	Issue REX 78 s if connection establishment fails
Discovery Protocols	Lazy Discovery	SM: announces with n ($3+2d+k$) messages every 1800 s	SCM: announces every 120 s
	Aggressive Discovery	SU: issues *Msearch* every 120 s (after purging SD)	SU and SM: issue seven probes (at 5 s intervals) only during startup
Application Software	Ignore REX	SU: *HTTP Get* Poll SM: Notification	SU: *FindService* Poll SCM: Notification
	Retry after REX	SU: *HTTP Get* after discovery retry in 180 s (retries ≤ 3) *Subscribe* requests retry in 120s	SM: depositing or refreshing SD copy on SCM retry in 120s SU: registering and refreshing notification requests with SCM retry in 120 s
	Discard Knowledge	SU: purge SD after failure to receive SM announcement within 1800 s	SU and SM: purge SCM after 540 s of continuous REX

Table 2. Summary of recovery responsibilities and strategies as implemented within our models for two- and three-party architectures.

Reliable unicast protocols attempt to ensure delivery of messages by detecting and retransmitting lost messages. Accordingly in the TCP simulation, our model is more complex, including both connection establishment and data transfer. During connection establishment, we allow up to four attempts to initiate a connection. An attempt fails if either the connection request or accept is lost. If no accept arrives, then the request is resent in 6 s for the first retry, but we wait 24 s for each subsequent retry. If all attempts fail, then we signal a REX to the requester. During data transfer, messages lost to transmission errors are scheduled for retransmission (roughly within a round-trip time, or RTT). We increase the retransmission timeout by 25% with each successive retransmission. We place no bound on the number of retransmissions during data transfer.

3.2 Recovery by discovery protocols

Discovery protocols include built-in robustness measures to deal with the possibilities of UDP message loss and node failure. Discovery protocols specify periodic transmission of key messages. For example, Jini requires a node to engage in aggressive discovery on startup, and then to enter lazy discovery, where all SCMs periodically announce their presence. In a similar lazy discovery, UPnP requires SMs to periodically announce their presence. While not specifying aggressive discovery, UPnP permits SUs to issue *Msearch* queries at any time. To compensate for the different announcement intervals recommended for Jini and UPnP, we chose to have UPnP SUs issue *Msearch* queries every 120 s, but only after a SU purges a SD from its local cache. Once a SU regains its desired SD, the related *Msearch* queries cease. Whenever a UPnP SM announces itself or responds to an *Msearch* query, it sends *n* copies of each message, where *n* is a retransmission factor (two in the current study) recommended by the UPnP specification to compensate for possible UDP message loss. In both Jini and UPnP, each lazy announcement recurs periodically. Receiving nodes can cache information from the announcements; the cached information may be purged if communication fails. In this way, each node in the system eliminates residual information about failed or unreachable nodes. Our models incorporate these failure-recovery behaviors.

3.3 Recovery by application software

When discovery nodes communicate over a reliable channel, a REX may occur. Response to a REX is left to the application. In our models, depending on the situation, we implement three different strategies: (1) ignore the REX, (2) retry the operation for some period, and (3) discard knowledge. The retry strategy attempts to recover from transient failures. The discard strategy, which occurs following repeated failure of the retry strategy, relies upon discovery mechanisms to recover from more persistent failures.

3.3.1 Ignore REX. In general, our models ignore a REX received when attempting to respond to a request. A SU can ignore a REX received in response to a poll, *FindService* or *HTTP Get*, because the poll recurs at an interval. The SCM (three-party model) or the SM (two-party model) also ignores a REX received while attempting to issue a notification. This behavior, which is described in both the Jini and UPnP specifications, depends upon reliable lower-layer protocols to provide robustness for notifications. Notifications include sequence numbers that allow a receiving node to determine if previous notifications were missed.

3.3.2 Retry the operation. In our models, we retry selected operations in the face of a REX. The UPnP specification separates the operation of discovering a resource from obtaining a description of the resource (Jini combines these operations). Without a description, the resource cannot be used. For this reason, in our two-party model, a SU must issue a *HTTP Get* to obtain a description. If no description arrives within 180 s, then our model retries the *HTTP Get*. If unsuccessful after three attempts, the SU ceases the retries, but sets a flag reminding itself to reissue a *HTTP Get* when the resource is next announced. Our three-party model, based on Jini, also contains a retry strategy, but associated with attempts to register or change a SD with a SCM. In these cases, the SM retries a *ChangeService* or *ServiceRegistration* 120 s after receiving a REX. Similarly, when a SU receives a REX (from either a SM or SCM) in response to a request to register for notification, the SU retries the registration in 120 s. All retries occur until some time bounds, after which knowledge of the discovery is discarded.

3.3.3 Discard knowledge. Both our two-party and three-party models include the possibility that an application can discard knowledge of previously discovered nodes. In UPnP, after failure to receive announcements from the SM within a TTL, a SU discards a SM and any related SDs. We implement this behavior in our two-party model. In Jini, the specification states that a discovering entity *may* discard a SCM with which it cannot communicate. In our three-party model, a SM or SU deletes a SCM if it receives only REXs when attempting to communicate with the SCM over a 540-s interval. After discarding knowledge of a SM (UPnP) or SCM (Jini), all operations involving the node cease until it is rediscovered, either through lazy discovery (Jini or UPnP announcements) or aggressive discovery (UPnP *Msearch* queries).

4. Experiment design and metrics

In this paper, we investigate the following question: How do alternative service-discovery architectures, topologies, and consistency-maintenance mechanisms perform under deadline during message loss? To address this question, we deploy a two-party and three-party architecture (recall Figures 1 and 2), each in a topology that includes one SM and five SUs. In the three-party case, we use two topologies, one with one SCM and another with two SCMs. To compare change propagation in two- and three-party architectures, we then combine the architectures with different consistency-maintenance mechanisms. Table 3 depicts the six combinations. To establish initial conditions, we exercise each topology until discovery completes, and the initial information (a SD) propagates to all SUs. To begin the experiment, we introduce a change in the SD at the SM, and we establish a deadline, *D*, before which the change must propagate to all SUs. We measure the number of messages exchanged and the latency required to propagate the new information, or until *D*, under two different consistency-maintenance mechanisms: polling and notification. We repeat this experiment while varying the message-loss rate up to 95% (in increments of 5%). We provide further details below.

Architectural Variant	Protocol Basis	Consistency-Maintenance Mechanism
Two-Party	UPnP	Polling
Two-Party	UPnP	Notification (with notification registration on SM)
Three-Party (Single SCM)	Jini	Polling (with service registration on SCM)
Three-Party (Single SCM)	Jini	Notification (with service registration and notification registration on SCM)
Three-Party (Dual SCM)	Jini	Polling (with service registration on SCM)
Three-Party (Dual SCM)	Jini	Notification (with service registration and notification registration on SCM)

Table 3. Experiment combinations.

4.1. Tracking consistency

To track consistency in our experiment, we employ property analysis [5], using a single consistency condition: service attributes for a SD discovered by a SU should have the same values as the attributes of the SD being maintained by the SM that manages the SD, expressed as:

FOR All (SM, SU, SD)
(SM, SD [Attributes1]) isElementOf SM managed-services &
(SM, SD [Attributes2]) isElementOf SU discovered-services
implies Attributes1 equals Attributes2

The condition is incorporated directly into our models and checked using Rapide procedural code. We establish an initial system state in which this condition holds, and then introduce a change in (SM, SD [Attributes1]), which negates the condition for all SUs. Then, we monitor updates to (SM, SD) tuples in the set of discovered-services maintained by individual SU's to determine if the condition becomes true. Note that if a SU discards its (SM, SD) tuple, the tuple must be recovered before the condition can be satisfied. These consistency checks form the basis for our measurements.

4.2. Generating message loss

We set aside an interval, up to time *Q*, to complete initial discovery and information propagation. In our experiments, Q = 100 s and D = 5400 s. We define F as the message-lost rate, which represents the independent variable in our experiment, ranging from 0.00 to 0.95 in increments of 0.05. For each attempt to transmit a data message, whether on a reliable or unreliable channel, or to retransmit a data message on a reliable channel, or to send or retry a connection request or accept message on a reliable channel, we select a uniform random number, V, from the unit interval 0 to 1. If $V < F$, we discard the message, which in the case of messages sent on the reliable channel will stimulate a retransmission after the appropriate timeout period (recall 3.1). Table 4 summarizes most of the relevant parameters and values for our experiments.

	Parameter	Value
Behavior in both two- and three-party architectures	Polling interval	180 s
	Registration TTL	1800 s
	Time to retry after REX (if applicable)	120 s
UPnP-specific behavior for two-party architecture	Announce interval	1800 s
	Msearch query interval	120 s
	SU purges SD	At TTL expiration
Jini-specific behavior for three-party architecture	Probe interval	5 s (7 times)
	Announce interval	120 s
	SM or SU purges SCM	After 540 s with only REX
Message loss parameters and protocol response	Loss Probability (*F*)	Each transmission attempt fails with *P*(*F*)
	Unreliable protocol response	Message discarded. No retransmission.
	Reliable protocol response	Connection Establishment - 4 retransmission attempts with delays of 6 s, 24 s, 24 s and 24 s; then REX if unsuccessful. Data Transfer – retransmit until success, increasing time-out by 25% on each retry (first time-out is round-trip time).
	Transmission delay without message loss	0.14 – 0.42 s uniform (range is multiplied by 1+*F*)
	Per-item processing delay within node	100 us for cache items 10 us for other items

Table 4. Values for relevant parameters.

4.3. Metrics

We use the data collected from experiment runs to compute three metrics: update responsiveness, update effectiveness, and update efficiency.

4.3.1 Update Responsiveness. Assuming information is created at a particular time and must be propagated by a deadline, then the difference between the deadline and the creation time represents available time in which to propagate the information. Update Responsiveness, *R,* measures the proportion of the *available time remaining* after the information is propagated. More formally, let *D* be a deadline by which we wish to propagate information to each SU-node *n* in a service-discovery topology. Let t_C be the creation time of the information that we wish to propagate, where $t_C < D$. Let $t_{U(n)}$ be the time that the information is propagated to SU *n*, where $n = 1$ to *N*, and *N* is the total number of SUs in a topology. Define change-propagation latency (*L*) for SU *n* as: $L_n = (t_{U(n)} - t_C)/(\max(D, t_{U(n)}) - t_C)$. This is effectively the proportion of available time used to propagate the change to SU *n*. The numerator represents the time at which the SU achieved consistency after the update occurred. The denominator represents the time available to propagate the change. The term $\max(D, t_{U(n)})$ accounts for cases where $t_{U(n)} > D$. Define *R* for SU *n* as: $R_n = 1 - L_n$. R_n is the proportion of *available time remaining* after propagating a change to SU *n*.

4.3.2 Update Effectiveness. Update Effectiveness, *U*, measures the probability that a change will propagate successfully for a given SU, i.e., $t_{U(n)} < D$. More formally, assuming definitions from 4.3.1 hold, let *X* be the number of runs (30 here) during which a particular topology is observed under identical conditions. Recalling that *N* is the total number of SUs in a topology, define the number of SUs observed under identical conditions as: $O = X * N$. Define *U*, the probability that $t_{U(n)} < D$, as: $U = 1 - P(F)$, where $P(F) = (\Sigma i \Sigma j$ (one if $R_{i,j}$ equals 0 and zero otherwise))$/O$ and where $i = 1...X$ and $j = 1...N$.

4.3.3 Update Efficiency. Given a specific service-discovery topology, examination of the available architectures (two-party and three-party) and consistency-maintenance mechanisms (polling and notification) reveals a minimum number of messages, *M*, that must be sent to propagate a change to all SUs. In our topologies, *M* ($M = 7$) occurs when using notification to propagate information in a three-party architecture with one SCM. Update Efficiency, *E*, can be defined as the ratio of *M* to the actual number of messages observed. More formally, let *S* be the number of messages sent while attempting to propagate a change from a SM to SUs in a given run. Define average *E* as: $E_{avg} = (\Sigma k(M/S_k))/X$, where $k = 1..X$.

5. Results and discussion

In this section, after showing results from our experiments, we consider the relative performance of our models and propose reasons for these differences.

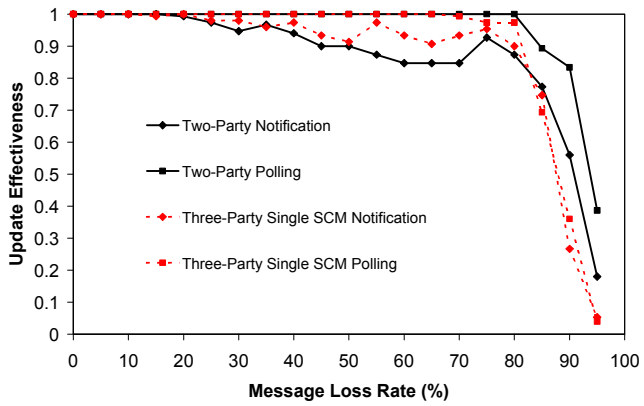
5.1. Results

In a series of six graphs, which have identical abscissas (message-loss rate, increasing from 0% to 95% in increments of 5%) and ordinates (an appropriate metric ranging between 0 and 1), we plot selected measurements generated from our models. Each graph compares four of the configurations in Table 3 against one of the metrics: update responsiveness (average), effectiveness, or efficiency (average). Figure 3(a) compares effectiveness from our two-party model against that from our single-SCM, three-party model, for both polling and notification. Figure 3(b) provides a similar comparison, but substitutes the results from our dual-SCM, three-party model in place of results from our one-SCM, three-party model. Figures 3(c) and 3(d) compare update responsiveness using the same combinations. Figures 3(e) and 3(f) use the same combinations, but compare update efficiency. The graphs reporting measures of effectiveness and responsiveness depict a system undergoing a phase transition from peak performance (where changes propagate quickly) to non-performance (where changes fail to propagate). Regarding efficiency, the graphs show a system that begins at its best efficiency (without interfering message losses) and then asymptotically approaches zero efficiency as the message-loss rate increases toward 100%. Because the graphs can be difficult to interpret, we compute summary statistics (see Table 5) for each of our six combinations. Each summary statistic reflects the mean of a particular metric, when averaged across all message-loss rates, for a specified configuration.

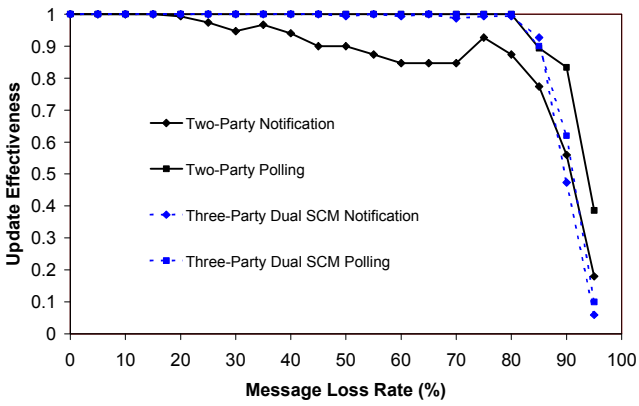
5.2. Relative performance

Below, we discuss the results for each of our three metrics. The reader should note that engineering trade-offs exist among: effectiveness, responsiveness, and efficiency.

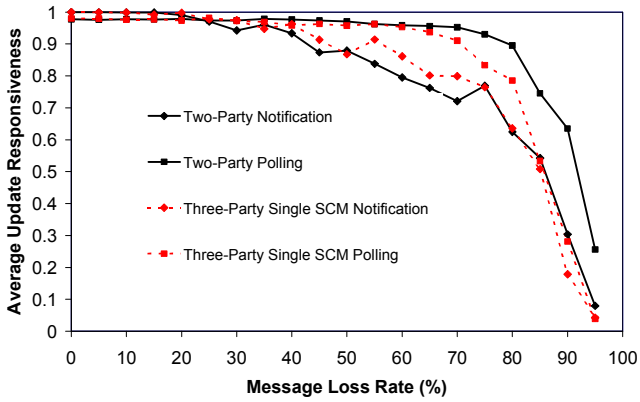
5.2.1 Effectiveness. Figs. 3(a) and 3(b) show that all combinations of architecture, topology, and consistency maintenance strategy exhibit update effectiveness of 0.85 or better up to a message-loss rate of 85%, after which they decline sharply. This similarity in effectiveness among the combinations can be attributed to commonality in the recovery behaviors of the discovery protocols, as implemented in our models. We require each SU (and the SM in the three-party case) to discard discovered information after a break in communications (recall Table 2) and then to initiate rediscovery. In the two-party model, periodic (120 s) *Msearch* queries by each SU (aggressive-discovery procedures) lead to rediscovery. Similarly, in the three-party case, periodic (120 s) announcements by each SCM (lazy-discovery procedures) lead to rediscovery.



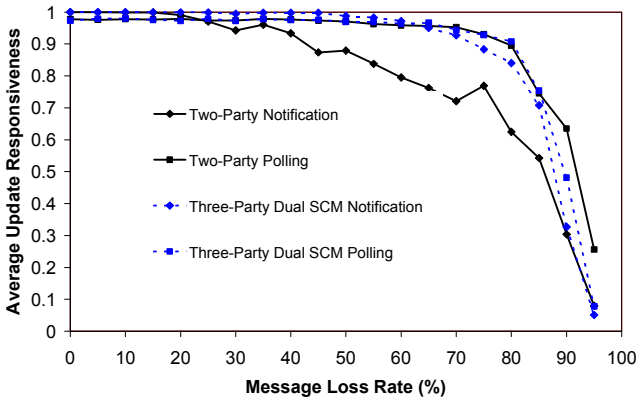
(a) Update effectiveness of two-party vs. three-party (single-SCM)



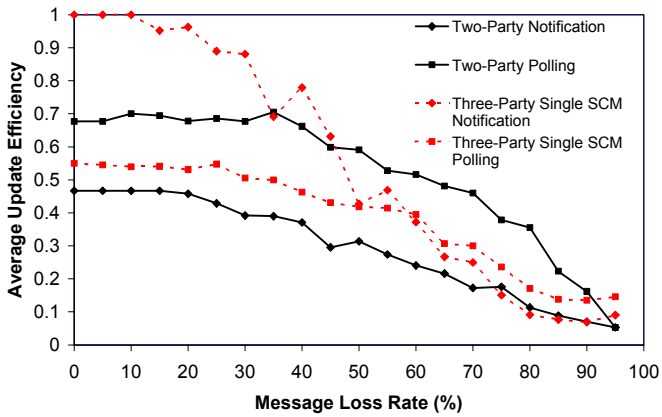
(b) Update effectiveness of two-party vs. three-party (dual-SCM)



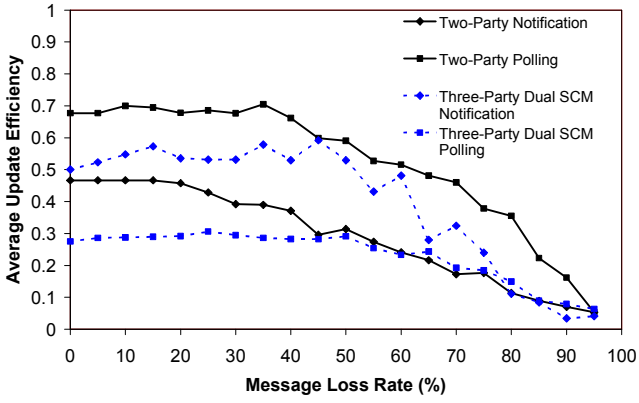
(c) Average update responsiveness of two-party vs. three-party (single-SCM)



(d) Average update responsiveness of two-party vs. three-party (dual-SCM)



(e) Average update efficiency of two-party vs. three-party (single-SCM)



(f) Average update efficiency of two-party vs. three-party (dual-SCM)

Figure 3. Graphs comparing combinations of architecture, topology, and consistency-maintenance mechanism.

After rediscovering a discarded node, the SU or SM re-establishes lost registrations, as appropriate for the consistency-maintenance strategy: notification registration for SUs and service registrations for the SM (three-party cases). In the process of restoring this distributed state information, each SU may obtain and cache a consistent copy of the SD maintained by the SM. As message-loss rate increases beyond 50%, this rediscovery machinery tends to dominate the effectiveness results.

	Mean (across all message-loss rates)		
	Update Effectiveness	Average Update Responsiveness	Average Update Efficiency
Two-Party Notification	0.867	0.799	0.296
Two-Party Polling	0.956	0.901	0.525
Three-Party Notification (Single SCM)	0.870	0.807	0.552
Three-Party Polling (Single SCM)	0.902	0.846	0.391
Three-Party Notification (Dual SCM)	0.921	0.881	0.400
Three-Party Polling (Dual SCM)	0.931	0.887	0.233

Table 5. Summary statistics (mean across all message-loss rates) computed for each curve in the graphs shown in Figures 3(a) through 3(f).

Despite rough similarity, certain combinations do show slightly better effectiveness than others (see Figs. 3(a) and 3(b) and the first column of Table 5). We attribute these differences to the consistency-maintenance strategy (polling or notification), and to differences in the recovery actions taken by the application software while implementing a particular strategy. Architecture and topology play a secondary role. In general, polling should lead to better effectiveness than notification, and our results support this in all architecture-topology combinations. Polling has built-in robustness from issuing periodic requests. On the contrary, notifications are issued only once with no further action by the sender in response to a REX (recall Table 2). Therefore, in notification, effectiveness suffers from situations where the notice is lost but where the notification registration and the node (SM or SCM) discovery are not lost. In these situations, there is no opportunity for recovery mechanisms to regain a lost node (SM or SCM) and to register for notification. Without such recovery, the SU might never obtain a copy of a changed SD. However, in three-party notification with dual SCMs, the effects of architecture and topology also come into play. Here, a replicated SCM provides an additional path for the SM to propagate the update, thus increasing the effectiveness of notification almost to the level of polling.

Beyond a rough similarity with distinguishable differences, the curves for effectiveness in two-party notification and in three-party single-SCM notification also include some irregularities, where effectiveness first drops and then improves as the message-loss rate increases. We used Rapide analysis tools to investigate the reasons underlying these dips. For both cases, we found that as the failure rate increases beyond 40%, the rate of recovery of the lost SM and lost registrations also increases. Recall that notification has no built-in robustness, relying instead on recovery mechanisms in TCP. Thus, to regain consistency when TCP recovery fails, notification must rely on recovery mechanisms in the discovery protocols, which provide opportunities to propagate previously lost updates. The higher the recovery rates, the greater the number of opportunities to regain consistency. As the message-loss rate increases, the recovery rate increases, and the effectiveness improves, up to a limit. Once the message-loss rate reaches 80%, the ability of the discovery protocols to effect recovery becomes impaired, leading to an inevitable decline in effectiveness. We also note that between 40% and 80% message-loss rate one of the notification combinations (three-party single-SCM) provides better effectiveness than the other (two-party). We suspect this occurs because the recovery actions of the SM (regaining the SCM discovery and registering the SD) provide additional opportunities (not available in the two-party case) to propagate the updated SD. Also recall that in Jini (the basis for behavior in our three-party models) notification includes the SD, while in the two-party case, based on UPnP, the SU must invoke separate operations to retrieve a copy of the SD. This provides additional opportunities for message loss to interfere with the restoration of consistency in the two-party case. These somewhat surprising dips in the effectiveness curves for notification also appear under conditions of node interface failures, discussed in a companion paper [2].

5.2.2 Responsiveness. Results in Figs. 3(c) and 3(d) and the second column of Table 5, show that three combinations of architecture and behavior (two-party polling, three-party polling with dual SCMs, and three-party notification with dual SCMs) exhibit similar responsiveness. Below 70% message-loss rate, three-party polling with a single SCM also exhibits similar responsiveness, but then declines more steeply than the others. For each architecture-topology combination, Table 5 shows that polling leads to better overall responsiveness than notification. However, Figs. 3(c) and 3(d) show that notification is more responsive at lower message-loss rates, where the periodicity of polling incurs a greater lag time. As message-loss rate increases, polling becomes more responsive than notification, which must rely on recovery mechanisms in the discovery protocols to recover from failure to transfer notifications (recall 5.2.1), whereas the built-in robustness of polling overcomes failures in lower protocol layers. In the three-party case with dual SCMs, notification achieves a similar

responsiveness to polling because notifications are sent over redundant paths, which mitigate the effect of transmission failures.

At high message-loss rates, under both polling and notification, restoring consistency depends largely upon recovery mechanisms in the discovery protocol. For responsiveness, as for effectiveness, our models of these recovery mechanisms ensure a degree of similarity in the results for three cases: two-party polling, three-party polling, and three-party notification with dual SCMs. In the case of three-party polling with a single SCM, responsiveness declines more rapidly at higher message-loss rates because, lacking a redundant SCM, fewer opportunities exist to recover a copy of the updated SD. Finally, for reasons already addressed (see 5.2.1), between 40% and 90% message-loss rates, both two-party notification and three-party notification with a single SCM prove considerably less responsive than the other combinations.

5.2.3 Efficiency. For a given combination of architecture and topology, we expect notification to be more efficient than polling. We also expect the two-party architecture to be more efficient than the three-party architecture, and the single-SCM topology to be more efficient than the dual-SCM topology. In general, our results support these expectations. However, there are a few twists. First, the three-party, single-SCM architecture with notification proves more efficient than the two-party architectures because in Jini the SD arrives with the notification, while in UPnP the notifications indicate only that a change has occurred, requiring a SU to exchange a request-response message pair to obtain the updated SD. Second, each SU must periodically refresh notification requests deposited on the SM (two-party case) or SCM (three-party case). As the message-loss rate increases, failure to transfer refresh messages leads to REXs, which stimulate retry procedures: every 120 s until 540 s of continuous REX (three-party case) or every 120 s until a SM is purged (two-party case). For this reason, efficiency decreases for notification as the message-loss rate increases.

6. Conclusions

Emerging service-discovery protocols provide the foundation for software components to discover each other, to organize themselves into a system, and to adapt to changes in node connectivity. While likely suitable for small-scale commercial applications, questions remain regarding the performance of such protocols at large scale, and during periods of high volatility and duress, such as might exist in military and emergency-response applications. In this paper, we used architectural models to characterize the performance of selected combinations of system topology and consistency-maintenance mechanism during severe message loss. Further, we used behavioral analysis to investigate the causes of observed performance. Our initial investigations show significant differences in performance can be obtained by varying aspects of the design (architecture, topology, consistency-maintenance mechanism, and recovery strategies).

7. Acknowledgments

The work described benefits from financial support provided by the National Institute of Standards and Technology (NIST), the Defense Advanced Research Projects Agency (DARPA), and the Advanced Research Development Agency (ARDA). In particular, we acknowledge the support of Susan Zevin from NIST, Doug Maughan and John Salasin from DARPA, and Greg Puffenbarger from ARDA. We also thank Stefan Leigh and Scott Rose of NIST and the anonymous WAMS reviewers for insightful comments that helped us to improve the manuscript.

8. References

[1] G. Bieber and J. Carpenter, "Openwings A Service-Oriented Component Architecture for Self-Forming, Self-Healing, Network-Centric Systems," on the web site: http://www.openwings.org.

[2] Dabrowski, C., Mills, K., and Elder, J. "Understanding Consistency Maintenance in Service Discovery Architectures during Communication Failure", *Proceedings of the 3rd International Workshop on Software Performance*, ACM, Rome, Italy, July 24-26, 2002.

[3] Ken Arnold et al, The Jini Specification, V1.0 Addison-Wesley 1999. Latest version is 1.1 available from Sun.

[4] Universal Plug and Play Device Architecture, Version 1.0, Microsoft, June 8, 2000.

[5] Dabrowski, C. and Mills, K., "Analyzing Properties and Behavior of Service Discovery Protocols Using an Architecture-Based Approach", *Proceedings of Working Conference on Complex and Dynamic Systems Architecture*, Brisbane, Australia, December 2001.

[6] Luckham, D. "Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Ordering of Events," http://anna.stanford.edu/rapide, August 1996.

[7] Salutation Architecture Specification, Version 2.0c, Salutation Consortium, June 1, 1999.

[8] Specification of the Home Audio/Video Interoperability (HAVi) Archiecture, V1.1, HAVi, Inc., May 15, 2001.

[9] Service Location Protocol Version 2, Internet Engineering Task Force (IETF), RFC 2608, June 1999.

[10] Specification of the Bluetooth System, Core, Volume 1, Version 1.1, the Bluetooth SIG, Inc., February 22, 2001.

Understanding Self-healing in Service-Discovery Systems

C. Dabrowski
U.S. National Institute of Standards and Technology
Gaithersburg, MD 20899
+1 (301) 975-3249
cdabrowski@nist.gov

K. Mills
U.S. National Institute of Standards and Technology
Gaithersburg, MD 20899
+1 (301) 975-3618
kmills@nist.gov

ABSTRACT

Service-discovery systems aim to provide consistent views of distributed components under varying network conditions. To achieve this aim, designers rely upon a variety of self-healing strategies, including: architecture and topology, failure-detection and recovery techniques, and consistency maintenance mechanisms. In previous work, we showed that various combinations of self-healing strategies lead to significant differences in the ability of service-discovery systems to maintain consistency during increasing network failure. Here, we ask whether the contribution of individual self-healing strategies can be quantified. We give results that quantify the effectiveness of selected combinations of architecture-topology and recovery techniques. Our results suggest that it should prove feasible to quantify the ability of individual self-healing strategies to overcome various failures. A full understanding of the interactions among self-healing strategies would provide designers of distributed systems with the knowledge necessary to build the most effective self-healing systems with minimum overhead.

Categories and Subject Descriptors

D.1.3 **[Concurrent Programming]**: Distributed programming

General Terms

Algorithms, Measurement, Performance, Design, Reliability, Experimentation.

Keywords

Architecture, Self-Healing Systems, Self-Repairing Systems, Service Discovery.

1. INTRODUCTION

Growing deployment of wireless communications, implying greater user mobility, coupled with proliferation of personal digital assistants and other information appliances, foretell a future where software components can never be quite sure about the network connectivity available, about the other software services and components nearby, or about the state of the network neighborhood a few minutes in the future. In extreme situations, as found for example in military applications [1], software components composing a distributed system may find that cooperating components disappear due to physical or cyber attacks or due to jamming of communication channels or movement of nodes beyond communications range. In such volatile environments, service discovery protocols enable distributed components to rediscover lost components or to find other components that provide essential services needed to accomplish critical tasks. To do this, service discovery systems include self-healing strategies to mitigate, detect, and recover from failures.

Service discovery protocols rely on several self-healing strategies. Architecture, which defines the logical components and relationships that compose a system, coupled with topology, which specifies the number and placement of components in a system, can be used in combination to mitigate the effects of failures by increasing system redundancy. Failure detection techniques, which typically include monitoring of periodic announcements and bounded retries (and resulting exceptions), allow components to estimate uncertainty regarding the state of cooperating components or regarding the intervening network path. Recovery techniques, which include application-level persistence and soft state, define actions a component can take to address suspected failures. Consistency-maintenance mechanisms, which include notification and polling, provide a means to maintain synchronized state among distributed components by propagating state changes to remote components.

In previous work, we used architectural models to investigate the behavior of various service-discovery systems under increasing communication failure [6] and message loss [7]. Our investigations yielded quantitative measures for the effectiveness, responsiveness, and efficiency of alternate system designs. We considered various combinations of architecture, topology, and consistency-maintenance mechanisms, however we did not vary failure recovery techniques.

In this paper, we extend our approach to quantify the contribution of failure recovery techniques in order to provide a more complete picture of the actions of individual self-healing strategies within

service discovery systems. We focus our investigation on four combinations of failure-detection and recovery technique while limiting other variables to include only two architecture-topology combinations and one consistency-maintenance mechanism. We examine system behavior under increasing communication failure. We use the same Rapide [4] models of service-discovery systems that we used in our previous research. Our models are based on two specifications: Jini™ Networking Technology [2] and Universal Plug-and-Play [3]. We adapted self-healing strategies from these specifications.

The remainder of the paper is organized as four sections. In the first section, we provide an overview of the self-healing strategies used in service-discovery systems. The second section gives a quantitative summary of the overall effectiveness of various combinations of self-healing strategy, when used to maintain consistent state among distributed components as the duration of communication failures increases. In the third section, we investigate and quantify the contribution of failure detection and recovery techniques to overall system effectiveness. In the conclusions, we discuss the feasibility and desirability of gaining a full understanding of the interactions among self-healing strategies for adaptive distributed systems.

2. DISCOVERY SYSTEMS AND SELF-HEALING

Service discovery systems enable distributed software components to discover each other, and to determine if discovered components meet specific requirements. Discovery protocols include *consistency-maintenance mechanisms*, which can be used to disseminate changes in component availability and status, and to maintain, within some time bounds, a consistent view of components in a network. *Failure-detection and recovery techniques* enable components to detect and react to network changes by restoring communications with remote components or by locating alternate components. A number of different designs have been proposed for service-discovery systems. For example, a team at Sun Microsystems designed Jini Networking Technology, a general service-discovery mechanism atop Java™. As another example, a group from Microsoft and Intel conceived Universal Plug-and-Play (UPnP) to provide plug-and-play components for distributed systems.

2.1 Architecture and Topology

Our analysis of six distinct discovery systems revealed that most designs use one of two underlying architectures: two-party and three-party. A two-party architecture consists of two components types: service manager (SM) and service user (SU). Figure 1 shows a two-party architecture deployed in a six-component topology: one SM and five SUs. A three-party architecture adds a third component type: service cache manager (SCM). The three-party architecture allows for multiple SCMs to mitigate the effect of failures (passive self-healing). Figure 2 shows a three-party architecture with one SM, five SUs, and up to two SCMs. A SM maintains a database of service descriptions (SDs), where each SD encodes the essential characteristics of a particular service. A SU seeks SDs maintained by SMs that satisfy specific requirements. Where employed, the SCM operates as an intermediary, matching advertised SDs of SMs to SD requirements provided by SUs.

To animate our two-party model, we incorporated behaviors from the UPnP specification. Upon startup, each SU and SM seeks to discover other, relevant components within the network neighborhood. In a lazy-discovery process, each SM periodically announces the existence of its SDs over the UPnP multicast group. Upon receiving these announcements, SUs with matching requirements request copies of the desired SDs from the SM. The SU stores SD copies in a local cache. Alternatively, the SU may engage in an aggressive-discovery process by transmitting SD requirements, as Msearch queries, on the UPnP multicast group.

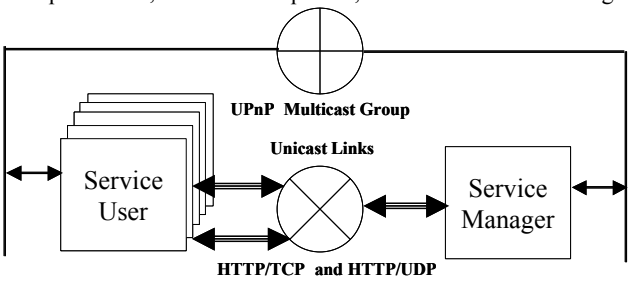


Figure 1. Two-party service-discovery architecture with five service users (SUs) and one service manager (SM).

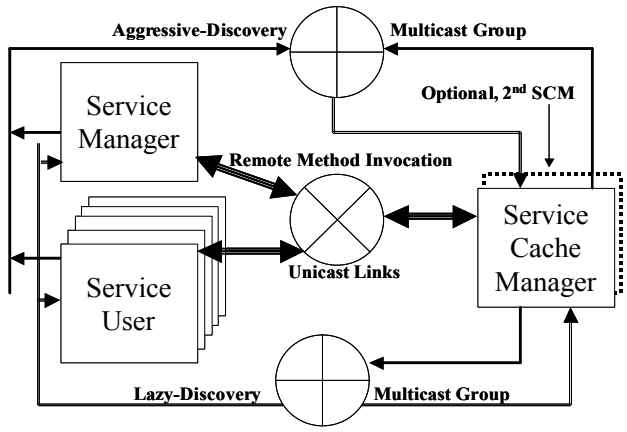


Figure 2. Three-party service-discovery architecture with five service users (SUs), a service manager (SM), a service cache manager (SCM), with an optional 2nd SCM.

Any SM holding a SD with matching requirements may respond directly to the SU. The SU may then request a copy of the relevant SDs, caching them locally. To maintain a SD in its local cache, a SU expects to receive periodic announcements from the relevant SM at a specified interval, known as a Time-to-Live, or TTL (or it must receive replies to its Msearchs within the TTL). Otherwise, the SU may discard the SD.

To animate our three-party model, we chose behaviors described in the Jini specification. In Jini, the discovery process focuses upon discovery by SMs and SUs of any intermediary SCMs that exist in the network neighborhood. Upon initiation, a Jini component enters aggressive discovery, where it transmits probes on the aggressive-discovery multicast group at a fixed interval for a specified period or until it has discovered a sufficient number of SCMs. Upon cessation of aggressive discovery, a component

enters lazy discovery, where it listens for announcements sent at intervals by SCMs. Once discovery occurs, a SM deposits a copy of the SD for each of its services on the discovered SCM for a specified length of time, or TTL. To maintain a SD on the SCM beyond the TTL, a SM must refresh the SD; otherwise it is purged. The SCMs match SDs provided by SMs to SU requirements, and forward matches to SUs.

2.2 Consistency-Maintenance Mechanisms

After initial discovery and information propagation (through SDs), SUs can use consistency-maintenance mechanisms to obtain updates to SDs for discovered services. We consider two basic mechanisms: notification and polling. In polling, a SU periodically sends queries to obtain up-to-date information about a previously discovered SD. In a two-party architecture, the SU issues the query directly to the SM from which the SD was obtained, and receives a response. In a three-party architecture, polling consists of two processes: 1) a SM propagates an updated SD to each SCM where the SD was originally cached and 2) each SU periodically queries relevant SCMs.

In notification, immediately after an update occurs, a SM sends events that announce a SD has changed. To receive events about a SD, a SU must first register for this purpose. In the two-party architecture, the SU requests registration with a SM. The request, if accepted, is retained for a TTL, which may be refreshed with subsequent requests from the SU. In a three-party architecture, a SU registers with a SCM to receive updates. The SCM grants event registrations for a TTL, which may be refreshed. When a SD is updated, the SM first propagates the update to all SCMs on which it deposited the SD; each SCM then forwards the event to all SUs registered to receive updates to the SD.

2.3 Failure-Detection Techniques

In a hostile military or emergency response environment, faults may arise due to enemy jamming or other interference, congestion, physical severing of cables, improperly configured or sabotaged routing tables, or multi-path fading as nodes move across a terrain. In this paper, we consider communication failure. Node communication may fail fully (both transmit and receive) or partially (either transmit or receive). All outbound messages from an interface will be lost when the transmitter fails, while all inbound messages will be lost when the receiver fails.

To detect failures, discovery systems use a combination of two techniques: monitoring periodic announcements and bounded retries (and resulting exceptions). Discovery protocols specify periodic transmission of key messages. Listeners can monitor these messages; much in the same way a heartbeat is monitored to assess the health of a patient. For example, as described above, both Jini and UPnP provide for periodic announcements of the availability of essential resources. Failure to receive scheduled announcements may indicate that the announcing entity has failed or that the network path is blocked. In other situations, software components send messages using reliable communication protocols, which persistently resend unacknowledged messages up to some bound, issuing a remote exception (REX) if the bound is exceeded. Failure detection allows components to employ recovery techniques.

2.4 Recovery Techniques

Discovery systems generally support two recovery techniques: soft-state and application-level persistence. Periodic announcements convey *soft* information about essential state, which a receiver can cache for a period of time, consistent with the expected announcement or heart rate. Each new re-announcement, or heartbeat, may convey updated state information; thus, the receiver overwrites previously cached state with state arriving in the latest announcement, or heartbeat. When the heartbeat fails, the receiver discards the cached state. When the heartbeat resumes, the receiver recovers the latest state. For example, upon failure of heartbeat messages sent by Jini SMs to refresh SDs cached on SCMs, the SD is discarded. The same occurs upon failure of periodic refreshes of notification registrations in both Jini and UPnP. Similarly, UPnP SUs may commence periodic *Msearch* queries after failure by a SM to refresh a SD within the TTL, which causes the SU to discard knowledge of the SM. Once a SU regains its desired SD, the related *Msearch* queries cease. This method is also employed when, after an initial aggressive discovery phase, Jini SCMs enter lazy discovery where they announce themselves every 120s. This ensures rediscovery of the SCM by SMs and SUs within 120s after a fault is rectified.

When failure-detection leads to a REX, discovery systems generally expect application software to initiate recovery, guided by an application-level persistence policy. In our models, depending on the situation, we implement three different persistence policies: (1) ignore the REX, (2) retry the operation for some period, and (3) discard knowledge. A SU can ignore a REX received in response to an attempted poll, because the query recurs periodically. In our models, two-party SMs and three-party SCMs also ignore a REX received as a result of attempted notifications. This behavior, which is described in both the Jini and UPnP specifications, depends upon reliable lower-layer protocols to provide robustness for events. In other cases, the retry policy attempts to recover from transient failures by resending a message (for which it has received a REX) after a nominal delay. The discard policy, which occurs following repeated failure of a retry, relies upon monitoring periodic soft-state announcements to recover from more persistent failures. As indicated above, in the two-party model, the SU discards the SM and its related SDs after failure to receive announcements from the SM within the TTL. In Jini, the specification states that a discovering entity *may* discard a SCM with which it cannot communicate. In our three-party model, a SM or SU deletes a SCM if it receives only REXs after attempting to communicate with the SCM over a 540-s interval. After discarding knowledge of a SM (UPnP) or SCM (Jini), all operations involving the node cease until it is rediscovered by monitoring periodic announcements (through either lazy or aggressive discovery).

3. EFFECTIVENESS OF SELF-HEALING

In previous work, we investigated the effectiveness of selected self-healing strategies when attempting to maintain synchronized state among distributed components during communication failure [6] and message loss [7]. We compared combinations of two- and three-party architectures and topologies (as shown in Figures 1 and 2), together with different consistency-maintenance mechanisms (notification or polling). In each combination, we used the same failure-detection (monitoring periodic

announcements and bounded-retries) and recovery (soft state and application-level persistence) techniques (see Table 1). We measured effectiveness (as the probability that a node achieves state synchronization) for increasing failure rates. Here we summarize our findings for effectiveness in the face of communications failure. Figure 3 shows effectiveness for each combination as communication-failure rate increases to 75%.

Failure Detection and Recovery Technique		Two-Party Architecture (UPnP)	Three-Party Architecture (Jini)
Heartbeat and Soft State	Lazy Discovery	SM: announces with n (3+2d+k) messages every 1800 s	SCM: announces every 120 s
	Aggressive Discovery	SU: issues *Msearch* every 120 s (after purging SD)	SU and SM: issue seven probes (at 5 s intervals) only during startup
Bounded Retries and Application-Level Persistence Policy	Ignore REX	SU: Pull Query (After Initial Pull) SM: Push Event	SU: Pull Query SCM: Push Event
	Retry after REX	SU: Initial Pull Query retry in 180 s (retries ≤ 3) Push Event Registration retry in 120s	SM: depositing or refreshing SD copy on SCM retry in 120s SU: Push Event registration or refresh on SCM retry in 120 s
	Discard Knowledge	SU: purge SD after failure to receive SM announcement within 1800 s	SU and SM: purge SCM after 540 s of continuous REX

Table 1. Summary of self-healing strategies included in our models.

Our previous papers provided qualitative explanations (based on analysis of execution traces) regarding the contributions of each self-healing strategy to measured differences in effectiveness. Here, we summarize our main findings for communications failure. Figure 3 indicates a rough similarity in effectiveness for all combinations; however, within these ranges, there are also significant differences. We attribute similarity in effectiveness to the fact that we employ similar failure-detection and recovery techniques in all combinations. The graph contains several eccentricities, in the form of saw-tooth behaviors. For example, two-party notification suffers a significant drop in effectiveness between 5% and 25% failure rate. This occurs because notifications rely on underlying reliable communication protocols to achieve robustness. When these protocols fail (as would be likely in case of communication failure), notifications are lost. The application software then relies upon detection of failure of periodic announcements (heartbeat) and restoration through initiation of recovery actions. Unfortunately, in UPnP the lazy-discovery announcement occurs no more frequently than every 1800s. Between 5% and 25% failure rate, there exists a substantial likelihood that communication failure is corrected prior to the next announcement. In such cases, an aggressive-discovery announcement (120-s interval) is not initiated, and state contained in the notification remains lost. As the failure rate increases, coincidence of announcement failure and notification failure becomes more probable, leading to initiation of the aggressive-discovery announcements, which eventually recovers state contained in the lost notification. Jini does not suffer as much from this phenomenon for two reasons. First, in Jini the lazy-discovery announcements occur at a 120-s interval. Second, Jini SMs exhibit some persistence when attempting to propagate SDs to SCMs. In selected cases, this persistence causes the SCM to periodically retry notifications.

Despite the dominance of failure-detection and recovery techniques, our results show that certain combinations of architecture, topology, and consistency-maintenance mechanism contribute to differences in effectiveness. For instance, each SD copy must propagate over either one link (two-party case) or two links (three-party case). For this reason, the three-party architecture (single SCM) can prove more vulnerable to communication failures (two links must be operational). This suggests that the two-party architecture will be more effective under severe failures, and our results support this. On the other hand, the three-party architecture allows replication of SCMs, which provides a greater number of paths through which information can propagate. This suggests (and our results agree) that the three-party architecture with dual SCM provides superior effectiveness over the single-SCM, three-party architecture. Our results also indicate that the dual-SCM three-party architecture yields effectiveness close to that of the two-party architecture.

Regarding consistency-maintenance mechanism, we conclude that polling, with its built-in persistence, should lead to better effectiveness than notification, where events are issued only once with no further action by the sender in response to a REX. Our results support this analysis for the two-party architecture and for the three-party architecture with a single SCM. However, notification appears slightly more effective than polling for the three-party architecture with dual SCM. We suspect this may be because notifications require only that the SCM-to-SU link be operational, while polling also requires the SU-to-SCM link.

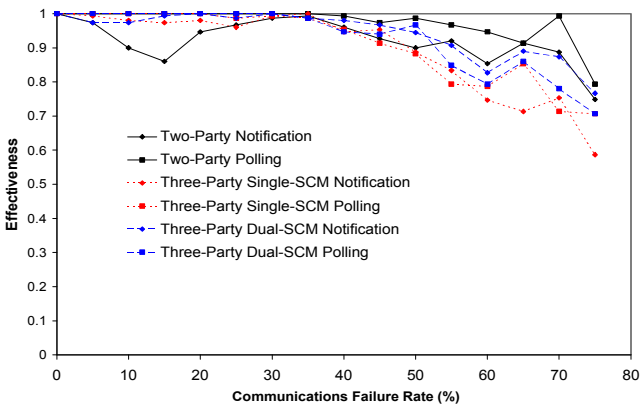


Figure 3. Effectiveness for various combinations of architecture, topology, and consistency-maintenance mechanism, as failure rate increases.

4. DISSECTING RECOVERY STRATEGIES

To further dissect recovery strategies, we decided to factor recovery techniques into four cases: 1) no recovery, 2) soft state only, 3) application persistence only, and 4) both soft state and application persistence.[1] We believe that this finer degree of factoring will enable us to quantify the contribution of various self-healing strategies to overall system effectiveness. Further, we expect that such factoring might reveal interactions among self-healing strategies, and help to identify situations where strategies are redundant, complementary, or conflicting. To explore these

[1] When a failure recovery technique is factored out of an experiment, the related failure detection technique (see Table 1) is also factored out. Eliminating soft state implies that the related heartbeat is ignored, while eliminating application-level persistence implies that the related REX (after bounded retries) is ignored.

ideas, we applied our approach to investigate the contribution of recovery techniques, given various architecture-topology combinations, in the case of one consistency-maintenance mechanism (notification) and one fault type (communication failure).

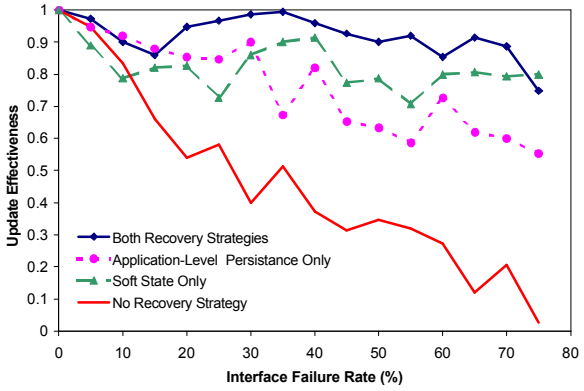


Figure 4. Update effectiveness of two-party notification with soft state, application persistence, and no recovery shown separately.

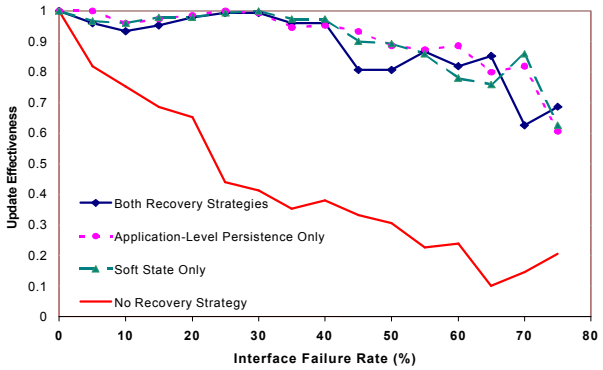


Figure 5. Update effectiveness of three-party (single SCM) notification with soft state, application persistence, and no recovery shown separately.

Figure 4 shows effectiveness for two-party notification as communication failure increases to 75%. The curve representing the use of all recovery techniques was taken from Figure 3. The remaining three curves in Figure 4 depict effectiveness when selected recovery techniques are disabled. Where no recovery is employed, effectiveness decreases nearly linearly as failure rate increases, dropping below 10% when the failure rate reaches 75%. When soft-state recovery is enabled alone, effectiveness improves significantly. Similarly, when application-persistence is enabled alone, effectiveness also improves significantly. Further, Figure 4 shows that application-persistence contributes more to system effectiveness at lower failure rates (30% and below), while soft-state recovery contributes more at higher failure rates. For two-party notification, under communication failure, the two recovery techniques appear complementary.

Figures 5 and 6, which show the contribution of recovery techniques for three-party, single-SCM notification and three-party, dual-SCM notification, yield a different picture. Where all recovery techniques are disabled, effectiveness decreases nearly linearly as failure rate increases; however, the rate of decrease of the three-party dual-SCM architecture appears lower than for the two-party architecture, and effectiveness stays above 10% at the 75% failure rate.

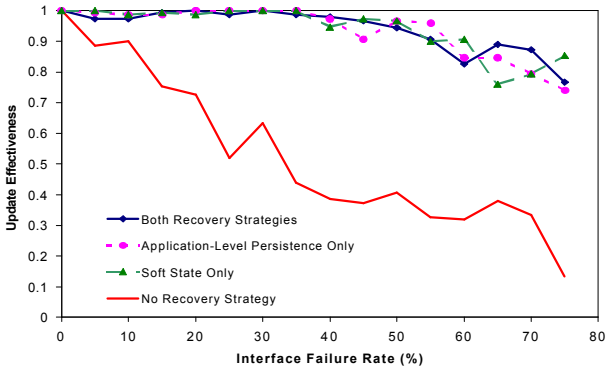


Figure 6. Update effectiveness of three-party notification (dual SCM) with soft state, application-recovery, and no recovery shown separately.

This suggests that increased robustness from a dual-SCM topology slightly mitigates the effects of communication failures. The three-party, single-SCM architecture with no recovery provides the poorest level of performance, reflecting the need to propagate the notification across two links without the alternative path provided by the second SCM. Note, however, that once either recovery technique is enabled in both variants of the three-party architecture, effectiveness improves to the level observed when both recovery techniques are enabled. This result indicates that, for three-party, single and dual-SCM notification, the two recovery techniques (soft state and application persistence) are redundant. These results shown in figures 4 through 6 are summarized in computed summary statistics in Table 2.

	Both Recovery Strategies	Application Persistence Only	Soft-State Only	No Recovery Strategy
Two-Party Notification	**0.921**	**0.763**	**0.824**	**0.466**
Three-Party Notification (Single SCM)	**0.914**	**0.914**	**0.907**	**0.441**
Three-Party Notification (Dual SCM)	**0.942**	**0.938**	**0.942**	**0.533**

Table 2. Summary statistics (mean across all interface failure rates) computed for each curve in the graphs shown in Figure 4 through Figure 6.

5. CONCLUSIONS

Our preliminary results (in Figs. 4-6) show the desirability and feasibility of dissecting the quantitative contributions to system effectiveness of various recovery strategies. Further, our results show that interactions (such as redundancy and complementarity)

between various recovery techniques can be identified and quantified.

Emerging service-discovery protocols provide the foundation for software components to discover each other, to organize themselves into a system, and to adapt to changes in system topology. These capabilities can also be used to effect self-healing in distributed component systems. In this paper, we used architectural models to characterize how architecture, topology, consistency-maintenance mechanism, and failure-recovery strategy each contribute to self-healing during communication failure. Further, in the context of communication failure and using notification as a consistency-maintenance mechanism, we dissected the self-healing properties attributable to recovery techniques and to topology. Our results suggest that it should prove feasible to quantify the ability of individual self-healing strategies to overcome various types of failure. A full understanding of the interactions among self-healing strategies would provide designers of distributed systems with the knowledge necessary to build the most effective self-healing systems with minimum overhead.

6. ACKNOWLEDGMENTS

The work discussed in this paper was funded in part by DARPA, under the auspices of the FTN and DASADA programs.

7. REFERENCES

[1] G. Bieber and J. Carpenter, "Openwings A Service-Oriented Component Architecture for Self-Forming, Self-Healing, Network-Centric Systems," on the http://www.openwings.org web site.

[2] Ken Arnold et al, The Jini Specification, V1.0 Addison-Wesley 1999. Latest version is 1.1 available from Sun.

[3] Universal Plug and Play Device Architecture, Version 1.0, Microsoft, June 8, 2000.

[4] Luckham, D. "Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Ordering of Events," http://anna.stanford.edu/rapide, August 1996.

[5] Dabrowski, C. and Mills, K., "Analyzing Properties and Behavior of Service Discovery Protocols Using an Architecture-Based Approach", *Proceedings of Working Conference on Complex and Dynamic Systems Architecture*, Brisbane, Australia, December 2001.

[6] Dabrowski, C. Mills, K., and Elder, J. "Understanding Consistency Maintenance in Service Discovery Architectures During Communications Failure", Accepted at *Third Annual Workshop on Software Performance*, Rome Italy, July 2002.

[7] Dabrowski, C. Mills, K., and Elder, J. "Understanding Consistency Maintenance in Service Discovery Architectures In Response to Message Loss", Accepted at *Fourth Annual International Workshop on Active Middleware Services*, Edinburgh, Scotland, July 2002.

Performance of Service-Discovery Architectures in Response to Node Failures

C. Dabrowski, K. Mills, A. Rukhin
U.S. National Institute of Standards and Technology, Gaithersburg, MD 20899
{cdabrowski|kmills|arukhin}@nist.gov

Abstract

Current trends suggest future software systems will rely on service-discovery protocols to combine and recombine distributed services dynamically in reaction to changing conditions. We investigate the ability of selected designs for service-discovery protocols to support real-time distributed control applications by detecting and recovering from failure of remote services. We model two architectures (two-party and three-party) underlying most commercial service-discovery systems. We use simulation to quantify functional effectiveness achieved by the two architectures as the rate of failure increases for remote services. We further decompose non-functional periods into failure-detection delay and recovery delay. Our quantitative measurements suggest that a two-party architecture yields better robustness than a three-party architecture. We discuss the underlying causes for this outcome.

1. Introduction

Designs for distributed systems must consider the possibility that failures will arise, and must adopt specific failure detection and recovery strategies [1]. Much existing research surrounding failures in distributed systems focuses on providing fault-tolerant invocation of remote methods, either through parallel execution of replicated components or through automated checkpoint and restart procedures [2-4]. Fault-tolerant remote-method invocation typically relies upon a layer of mechanisms to detect and recover from failures without requiring application-specific awareness or action. While such application-transparent fault-tolerance appears appealing, many current distributed object systems, even large systems, employ simpler techniques that detect and report failures, requiring applications to decide upon appropriate recovery strategies [5-7]. In this paper, we investigate one such set of simpler techniques requiring application awareness and cooperation. These techniques encompass the fundamental failure detection and recovery strategies available in service-discovery systems [8-13].

In previous work, we investigated the ability of various service-discovery systems to propagate updates under communication failure [14] and message loss [15]. Our investigations yielded quantitative measures for the effectiveness, responsiveness, and efficiency of alternate system designs. In this paper, we investigate the effectiveness, efficiency and latency of service-discovery systems in detecting component failure and locating replacements. We model specific discovery strategies and failure-recovery techniques in combination with two major architectural variants found in service-discovery systems: two-party, where clients and services rendezvous directly, and three-party, where clients and services rendezvous through a directory. For the three-party architecture, we consider topologies that include directory replicas. Our models, which adapt discovery and recovery strategies from the Jini™[1] Networking Technology [10] and Universal Plug-and-Play [9] specifications, layer a real-time distributed control application above each of the discovery systems. We model application-level strategies that focus our experiments on the fundamental properties of service-discovery protocols; thus, we exclude a number of possible application choices, such as service caching. We measure functional effectiveness, defined as the proportion of time that a distributed application meets its requirements, or more precisely, as the proportion of time that a client component possesses an operational set of remote services needed to accomplish its task. To provide a clear picture of failure response, we also measure both failure-detection latency (time required to recognize that a remote service used by the client has failed) and failure-recovery latency (time required for the client to replace a failed service). We also measure overhead as the number of messages sent. Our models are written using Rapide [16], which records complete event traces that permit

[1] Certain commercial products or company names are identified in this report to describe our study adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the products or names identified are necessarily the best available for the purpose.

detailed analysis of system behavior, helping us to determine causes underlying quantitative performance.

2. Discovery and recovery

Service-discovery protocols enable networked components to rendezvous and to combine with discovered components into distributed applications meeting specific requirements. Discovery protocols include f*ailure-detection and recovery techniques* that enable components within distributed applications to detect and react to failures by restoring communications with remote components or by locating alternate components. A number of different designs have been proposed for service-discovery systems. For example, a team at Sun Microsystems designed Jini Networking Technology, a general service-discovery system atop Java™. As another example, a group from Microsoft and Intel conceived Universal Plug-and-Play (UPnP) to provide plug-and-play components for distributed systems.

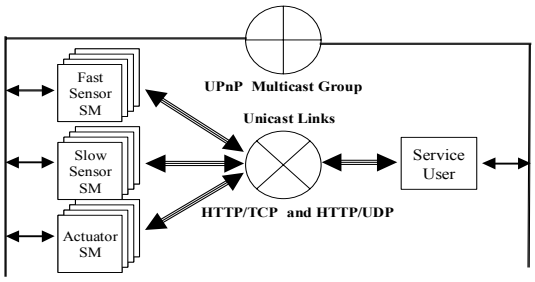


Figure 1. Two-party service-discovery architecture with one service user and 12 service managers

2.1. Service discovery

Our analysis of six discovery systems [8-13] revealed that most designs use one of two underlying architectures: two-party or three-party. A two-party architecture consists of two component types: service manager (SM) and service user (SU). The three-party architecture adds a third component type, service cache manager (SCM). Multiple SCMs can be used to mitigate the effect of SCM failure. In both architectures, service discovery occurs passively, via multicast announcements, and actively, via multicast queries. Each SM maintains a database of service descriptions (SDs), where each SD encodes the essential characteristics of a particular service provider (SP) managed by the SM. Each SU seeks SDs satisfying specific requirements. Where employed, the SCM operates as an intermediary, matching advertised SDs of SMs to SD requirements provided by SUs.

In this study, each SM manages one SP from among three service types: fast sensor, slow sensor, and actuator. Our experiment consists of four instances of each service type, whose roles are explained below. Figure 1 shows a two-party architecture deployed in our experiment topology with 12 SMs and one SU. To animate our two-party model, we incorporated discovery behaviors from the UPnP specification, as described elsewhere [14, 15]. Figure 2 shows the three-party architecture in our experimental topology: with 12 SMs, one SU, and up to three SCMs. To animate our three-party model, we chose discovery behaviors from the Jini specification, as described elsewhere [14, 15].

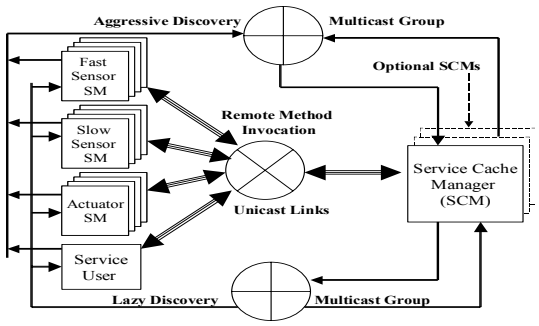


Figure 2. Three-party service-discovery architecture with one service user, 12 service managers, and up to 3 service cache managers

2.2. Failure-Detection Techniques

To detect failures, applications using discovery systems rely on a combination of two techniques: monitoring periodic transmissions and retrying ad hoc transmissions (where exceeding a retry bound causes an exception). Discovery protocols specify periodic transmission of key messages. In addition, components employing remote services may maintain regular contact to accomplish application-specific tasks. Components can listen for these recurring messages, much as a heartbeat can be monitored to assess patient health. For example, both Jini and UPnP periodically announce resource availability. Similarly, a sensor service may periodically issue readings to its clients. Failure to receive scheduled communications might indicate that the remote service has failed, or that the channel between client and service is blocked. In other situations, software components send messages using reliable communication protocols, which persistently resend unacknowledged messages up to some bound, issuing a remote exception (REX) if the bound is exceeded. For example, a client may attempt to invoke a method offered by a remote service that has failed. In the three-party architecture, a SU might attempt to query for a SD from a failed SCM, only to receive a REX. Failure

detection enables components to employ recovery techniques.

2.3. Failure-recovery techniques

Discovery systems generally support two recovery techniques: soft-state and application-level persistence. Periodic announcements issued by a component convey *soft* information about component state, which a receiver can cache for a period of time, consistent with the expected announcement rate. Each new announcement may convey updated state information; thus, a receiver overwrites previously cached state with state from newly arriving announcements. When an announcement fails to arrive, a receiver discards previously cached state, effectively eliminating knowledge about existence of the announcing component. When announcements resume, a receiver rediscovers the remote component and recovers the latest component state. Our application uses a modified form of soft state, which allows discarded components to be either rediscovered or replaced. For example, upon failure of heartbeat messages sent by UPnP SMs to refresh cached SDs, a SU discards knowledge of the SM and any associated SDs. Similarly, a SU may discard knowledge of a SM and SD for a remote sensor upon failure to receive sensor updates. To effect recovery, UPnP SUs may commence periodic multicast (*Msearch*) queries to search for a new instance of a required service. Once the SU regains a SD meeting requirements, the related queries cease. In Jini, loss of contact with a service may cause the SU to query a SCM for a replacement. In addition, service unavailability may be indicated by failure of heartbeat messages sent by Jini SMs to refresh SDs cached on SCMs, causing the SCM to discard the SD and to notify SUs that indicated interest in learning about service failures. Periodic announcements ensure rediscovery of the SCM by SMs within 120s after the SM recovers. The Jini SU can then receive the corresponding SD through notification or query. Of course, in Jini, SCMs could also fail. SCM startup announcements ensure discovery of a new or restarted SCM within about 30s.

When failures lead to a REX, discovery systems generally expect application software to initiate recovery, guided by an application-level persistence policy. The policy may require ignoring the REX, retrying the operation for some period, or discarding knowledge of the remote component. Since our experiment simulates a real-time control application, we chose not to persist after a REX, but instead to discard knowledge of the associated remote component, relying on periodic announcements and soft state to recover. This policy is also used in the three-party model when SCM failure is detected through a REX in response to a query (SU) or registration refresh (SU or SM). After discarding knowledge of a SM (UPnP) or SCM (Jini), all operations involving the remote component cease.

3. Experiment description

We investigate how effectively the two alternate service-discovery architectures, and associated failure detection and recovery mechanisms, provide clients with required services as nodes hosting the services fail and recover. We model the two- and three-party architectures using the four topologies shown in Figures 1 and 2. In all topologies, we deploy a single SU and twelve SMs, where each SM manages a specific type of SP: "fast" sensor, "slow" sensor, or actuator. The twelve SMs include four of each SP type. After discovery and activation by the SU, a "fast" sensor transmits a reading every two seconds and a "slow" sensor transmits a reading every 30 seconds. Once discovered and activated by the SU, an actuator can be invoked after the SU receives an appropriate combination of readings from a "fast" and "slow" sensor. In our experiment, we simulate actuation attempts using a uniform distribution with a mean of 60s. When the SU holds one SD for a SP of each type ("fast" sensor, "slow" sensor, and actuator) and each of the SPs is operational, then the application is considered *functional*. If the SU lacks SDs for one or more SP type or if one or more of the SDs held by the SU describes a SP that is not operational, then the application is considered *non-functional*. The experiment measures accumulated *functional time* in proportion to a duration *D* during which SMs and SCMs periodically fail and recover. To establish initial conditions, each topology is exercised until discovery completes, and the application becomes functional. To focus exclusively on failure detection and recovery processes, we do not cache services; the SU holds at most one SD for each SP type at any time. In the three-party architecture, some additional decisions are necessary. For each SD discovered and retained, the SU registers with the SCM for notification about failures. The SU refreshes notification registrations every 300s. Each SM registers with each discovered SCM, and refreshes every 60s (slow sensors/actuators) or 300s (fast sensors).

3.1. Failure model

During *D*, each SM (and SCM in the three-party case) fails randomly and independently, although at least one service of each type always remains active so that the application could become functional. We calculate a mean time to failure, *MTF*, from a failure rate *R*, varied from 0.1 to 0.9 of *D* in 0.1 increments, where $MTF = (1 - R) * D$. Node failure times are randomly chosen from a "stepped" normal distribution with three steps: a 0.15 probability that failure occurs before ($MTF - 0.2 * MTF$),

a 0.7 probability that failure occurs between (*MTF* - 0.2 * *MTF*) and (*MTF* + 0.2 * *MTF*), and a 0.15 probability that failure occurs between (*MTF* + 0.2 * *MTF*) and (2 * *MTF*). Failure time is distributed uniformly within each step.

When a SM or SCM fails, affected services become unavailable for a time. There are three failure classes, each with a different probability, *P*, and duration. Short failures occur with $P = 0.1$ for a fixed duration (135s); intermediate failures occur with $P = 0.7$ for a duration selected uniformly on the interval 180-300s, long failures occur with $P = 0.2$ selected uniformly on the interval 480-600s.

3.2. Metrics

We define non-functional time, *NF*, as accumulated time during which an application is in a non-functional state. Assuming we can measure *NF*, over a given duration *D*, then functional effectiveness, *F*, can be quantified as a ratio: $F = (D - NF)/D$. We define consistency conditions to measure *NF*, as explained below

A client in a distributed application may become non-functional due to failure of remote components but incur a delay before detecting the failure. We call this delay failure-detection latency. After detecting a non-functional state, the application may incur some delay while restoring required services. We call this delay failure-recovery latency. During periods when a client incurs either failure-detection or failure-recovery latency or both (the states can overlap when a client requires more than one remote service), the distributed application is non-functional. We accumulate such non-functional periods to *NF*.

We define two consistency conditions such that violation of one corresponds to failure-detection latency and violation of the other corresponds to failure-recovery latency. The following consistency condition requires each SD held by a SU to match a SD managed by a SM. More formally,

$$\forall[SM, SU, SD]$$
$$(SM, SD) \in discoveredServices_{SU}$$
$$\rightarrow \exists SM \mid SD \in managedServices_{SM}$$

In this condition (**CC-1**), $managedServices_{SM}$ denotes the database of SD(s) for services managed by a SM and $discoveredServices_{SU}$ denotes the (SM, SD) pairs a SU has discovered. **CC-1** is violated (and failure-detection latency commences) when a SM fails but the SU holds a SD provided by the SM. Once the SU discards the SD, or the SM recovers, consistency is restored (and failure-detection latency ends). A second consistency condition requires that available SDs matching SU requirements should be known to the SU. More formally,

$$\forall[SM, SU, SD]$$
$$SD \in managedServices_{SM} \wedge SD \in resourcesNeeded_{SU}$$
$$\rightarrow (SM, SD) \in discoveredServices_{SU}$$

This condition (**CC-2**) is violated (and failure-recovery latency begins) after the SU purges a SD for a failed service and commences search. Consistency returns (and failure-recovery latency ends) when the SU finds a SD matching its needs.

4. Results and discussion

For each of four topologies (two-party and three-party with one, two, and three SCMs), we set $D = 1800$s and executed multiple repetitions for each value of *R* using the failure model described in 3.1. We conducted separate experiment runs for cases where failed nodes (including SMs and SCMs) are discarded and *replaced* by new nodes, and for cases where failed nodes *restart*, maintaining persistent information in the manner specified by the protocols. For the replacement case, we ran a second variant of the experiment where all SMs for a resource type may fail. We recorded functional effectiveness, detection latency, recovery latency, and the total number of protocol messages exchanged in each run.

4.1. Effectiveness and efficiency

Figure 3 shows average functional effectiveness of the two-party and three-party architectures for the replacement case as *R* increases, and where one SM for each service type is always available (implying that the system could be functional for all of *D*). In examining Fig. 3, recall how failure detection occurs. In the two-party model, the SU may detect service unavailability by monitoring cyclical sensor readings or by monitoring notification registration refreshes. In the three-party model, the SCM notifies the SU if the SM fails to refresh service registrations. In both models, the SU may also detect unavailability when a REX occurs in response to attempted actuations. To become functional again, the SU must invoke appropriate recovery mechanisms to regain SDs to replace unavailable services. In the three-party architecture, at least one SCM must be operational for recovery to succeed. During periods when all SCMs fail, the SU is unable to recover needed services, increasing non-functional time.

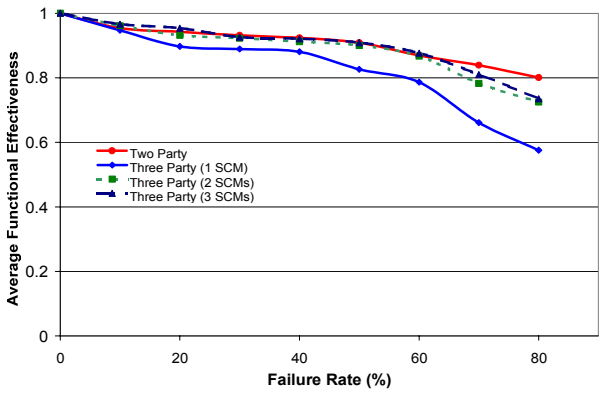


Figure 3. Functional effectiveness for four topologies under increasing *R* for the replacement case where at least one SM of each type is operational (60 reps/point)

Overall, the two-party architecture proves more effective above 60% failure rate, allowing the SU to remain functional for as much as 80% of *D* even when the failure rate reaches 80% (*MTF* = 360s). At rates below 60% the effectiveness of two-party is comparable to three-party with two and three SCMs. Fig. 3 also shows that effectiveness improves for the three-party architecture as the number of SCMs increase, though even with 3 SCMs, performance does not equal that of the two-party architecture. Adding SCMs improves effectiveness by lowering the incidence of concurrent failure of all SCMs.

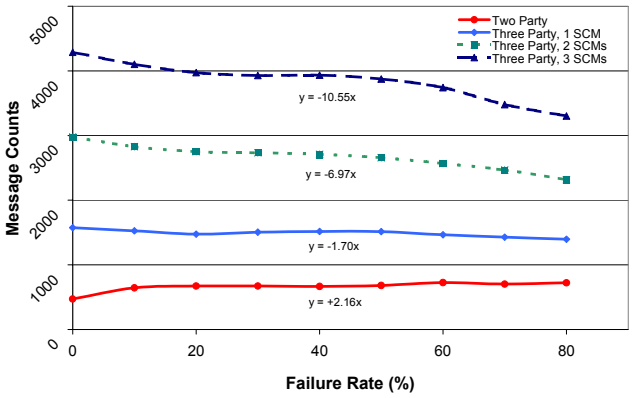


Figure 4. Average message counts for four topologies under increasing *R* for the replacement case where at least one SM of each type is operational (30 reps/point)

Message counts (Fig. 4) reveal the two-party architecture to be significantly more efficient than the three-party architecture. Note also that for the three-party architecture, total message counts decrease as failure rate increases, because SCMs remain down for longer periods; thus, requiring fewer registration refresh and SCM heartbeat messages. For the two-party model, message counts increase slightly at high failure rates because the SU invokes active recovery procedures after detecting failures. Fundamentally, the three-party architecture relies on redundancy of SCMs to improve functional effectiveness; thus, exacting a high overhead at low failure rates, but permitting overhead to diminish as failure rate increases. The two-party architecture relies on active recovery invoked by a SU; thus, at low failure rates overhead is lower because recovery procedures are not invoked often, but overhead increases with failure rate as recovery procedures are invoked more often.

4.2. Underlying causes

To better understand differences in effectiveness among the alternate architectures, we decomposed non-functional time to show the estimated proportion attributable to failure-detection latency and to failure-recovery latency. Figure 5 shows that detection latency is the dominant (~80%) component of non-functional time for the two-party model. Analysis of execution traces using the Rapide toolset showed most failures were detected through missed sensor readings (2s for fast sensors and 30s for slow sensors) or REXs received in response to failed actuations. We suspected that in the two-party architecture detection latency, and therefore non-functional time, could be reduced by increasing registration-refresh frequency; thus, decreasing the interval between heartbeats. Failed notification refresh attempts by the SU would permit detection of SM unavailability (and violation of **CC-1**) before non-receipt of slow sensor readings or failed actuation attempts. To test this theory, we lowered the registration refresh frequency from 300s to 30s in the two-party model, and reran the experiment The result was a 49% drop in detection latency leading to a 2.6% overall improvement in functional effectiveness (an increase in the mean effectiveness across all failure rates from 0.908 to 0.932). However, efficiency decreased 69%, with a rise in message count from an average of 662 to 1116. Similarly in the three-party architecture, we suspect increasing refresh frequency for service registrations would lead to earlier detection by the SCM of SM failure [see 17], and to earlier notification for the SU. Of course, increasing the heartbeat rate also would decrease efficiency.

Our data for the three-party architecture show that above 60% failure rate the incidence of concurrent failure of all SCMs increases steadily. This precludes finding available services meeting SU requirements; thus, leaving the system in violation of **CC-2**. To restore consistency and achieve operational functionality, a SCM must first recover, accept registrations for the SU and available

SMs, and then propagate matching SDs to the SU. Lacking an ability to directly discover SMs, the SU remains non-functional while awaiting recovery of at least one SCM. These effects are evident in Fig. 6, which shows the proportion of recovery latency increasing for the three-party model (3 SCMs) as the failure rate rises. This trend is more marked as the number of SCMs decreases (not shown here). We speculate that functional effectiveness might improve for the three-party model if SUs were permitted to discover SMs directly when no SCMs are available. We plan experiments along these lines using the Service Location Protocol (SLP) [12], which enables switching between the two- and three-party architecture as the situation warrants.

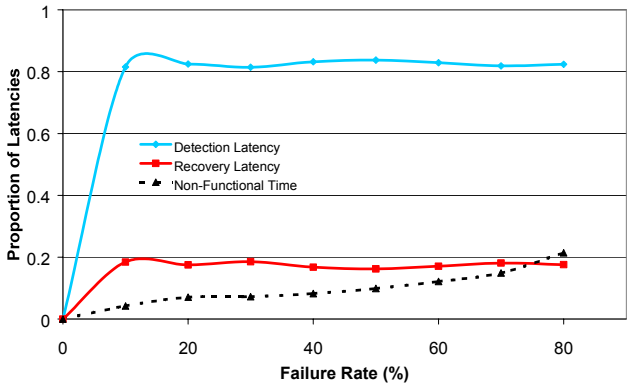


Figure 5. Detection and recovery latencies in two-party service-discovery model as a proportion of non-functional time (also shown) (60 reps/point)

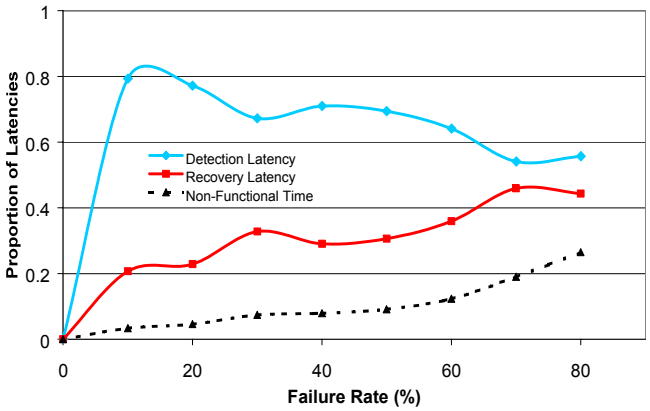


Figure 6. Detection and recovery latencies in three-party service-discovery model with 3 SCMs as a proportion of non-functional time (also shown) (60 reps/point)

4.3. Results for experiment variants

To confirm our findings, we varied the experiment in two respects. First, we changed node behavior to allow failed nodes to restart rather than be replaced by new nodes. In this case, three-party SCMs that recovered were allowed to retain previous, unexpired service registrations and notification registrations in accordance with the Jini protocol, while two-party SMs were permitted to retain notification registrations. The results showed no significant differences in performance between the restart and replacement cases, the graphs (not shown) were almost identical. This occurs in the three-party case because most of the persistent registrations expire by the time a failed SCM restarts. In the two-party case, where only notification registrations persist, the SU that registered the notification is likely to have discarded knowledge of the SM by the time it restarts. Since, in our experiment, restarting nodes derive little value from persistent information, functional effectiveness is mainly influenced by soft-state mechanisms, as in the replacement case.

Second, we varied the experiment to permit all SMs to fail, rather than to have at least one SM always available for each service type. The results, shown in Fig. 7., illustrate functional effectiveness for both the two- and three-party models decreases substantially above R = 60%, as the incidence of concurrent SM failures increases, resulting in extended periods when no SMs were available for a service type needed by the SU. Though the absolute functional effectiveness declined, the ranking of the curves remained the same as in the previous experiments, with the two-party model proving most effective followed by the three-party model with three-, two-, and one-SCM topologies, respectively. Thus, in all of our experiment variants, the two-party model achieved better functional effectiveness than the three-party model.

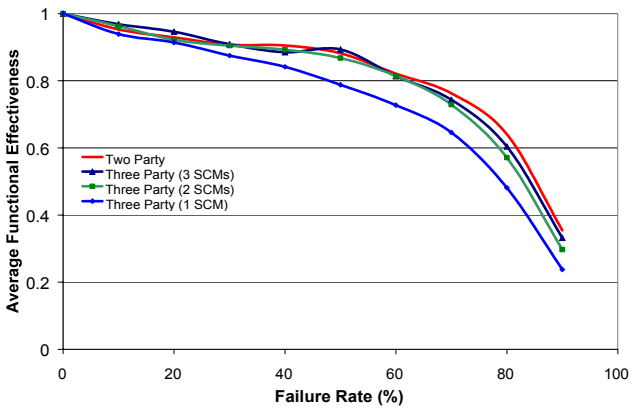


Figure 7. Functional effectiveness for four topologies under increasing *R* for the replacement case where all SMs of each service type are allowed to fail (30 reps/point)

5. Conclusion

This study provides an initial characterization of the performance of service-discovery architectures in response to node failures, which complements our previous studies of response to communication failures and message loss. The present study shows that in response to node failure, two-party systems exhibit better functional effectiveness and efficiency than three-party systems, with three-party SCMs being a potential point of vulnerability. Possible solutions to mitigate this vulnerability require further study. Similarly, further research is needed to verify that registration refresh rates or service caching could improve functional effectiveness. Finally, we need to verify that our conclusions hold in networks with large numbers of services.

6. Acknowledgements

The work discussed in this paper was funded in part by DARPA, under the auspices of the FTN and DASADA programs

7. References

[1] G. Bieber and J. Carpenter, "Openwings A Service-Oriented Component Architecture for Self-Forming, Self-Healing, Network-Centric Systems," http://www.openwings.org web site.

[2] *Fault Tolerant CORBA Specification*, *v1.0*, ptc/00-04-04, Object Management Group.

[3] C. Marchetti, A. Virgillito, and R. Baldoni, "Design of an Interoperable FT-CORBA Compliant Infrastructure," *Proceedings of the European Research Seminar on Advances in Distributed Systems* (ERSADS), 2001.

[4] D. Liang et al. "A Fault-Tolerant Object Service on CORBA," *The Journal of Systems and Software*, Vol. 48. 1996.

[5] Y.M. Wang, O.P. Damani, and W.J. Lee, "Reliability and Availability Issues in Distributed Component Object Model (DCOM)," *Proceeding of the International Workshop on Community Networking*, 1997, pp. 59-63.

[6] Felber, P. et al. "Failure Detectors as First Class Objects," *Proceedings of the International Symposium on Distributed Objects and Applications* (DOA'99), IEEE Computer Society Press, September 5-7, 1999, p. 132.

[7] Carey, R.W. et al. "Large-Scale Corba-Distributed Software Framework For Nif Controls," *Proceedings of the 8th International Conference on Accelerator & Large Experimental Physics Control Systems*, Stanford Linear Accelerator Center, November 27-30, 2001, p. 425.

[8] *Salutation Architecture Specification*, *V. 2.0c*, Salutation Consortium, June 1, 1999.

[9] *Universal Plug and Play Device Architecture*, *V. 1.0*, Microsoft, June 8, 2000.

[10] Ken Arnold et al, *The Jini Specification*, *V1.0* Addison-Wesley 1999. Latest version is 1.1 available from Sun.

[11] *Specification of the Home Audio/Video Interoperability (HAVi) Archiecture*, V1.1, HAVi, Inc., May 15, 2001.

[12] Guttman, E., Perkins, C., Veizades, J., and Day, M. *Service Location Protocol, V.2*, Internet Engineering Task Force (IETF), RFC 2608, June 1999.

[13] *Specification of the Bluetooth System, Core, Vol. 1*, Version 1.1, the Bluetooth SIG, Inc., February 22, 2001., 1999.

[14] Dabrowski, C. Mills, K., and Elder, J. "Understanding Consistency Maintenance in Service Discovery Architectures During Communications Failure," *Proceedings of the 3rd International Workshop on Software Performance*, ACM, July 2002, pp. 168-178.

[15] Dabrowski, C., Mills, K., and Elder, J. "Understanding Consistency Maintenance in Service Discovery Architectures In Response to Message Loss," *Proceedings of the 4th International Workshop on Active Middleware Services*, IEEE Computer Society, July 2002, pp. 51-60.

[16] Luckham, D. "Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Ordering of Events," http://anna.stanford.edu/rapide, August 1996.

[17] Bowers, K., Mills, K., and Rose, S. "Self-adaptive Leasing for Jini," *IEEE International Conference on Pervasive Computing and Communications 2003*, Dallas-Fort Worth, Texas, March 2003.

Understanding Failure Response in Service Discovery Systems

C. Dabrowski, K. Mills, S. Quirolgico
National Institute of Standards & Technology
Gaithersburg, Maryland 20899

ABSTRACT

Service discovery systems enable distributed components to find each other without prior arrangement, to express capabilities and needs, to aggregate into useful compositions, and to detect and adapt to changes. First-generation discovery systems can be categorized based on one of three underlying architectures and on choice of behaviors for discovery, monitoring, and recovery. This paper reports a series of investigations into the robustness of designs that underlie selected service discovery systems. The paper presents a set of experimental methods for analysis of robustness in discovery systems under increasing failure intensity. These methods yield quantitative measures for effectiveness, responsiveness, and efficiency. Using these methods, we characterize robustness of alternate service discovery architectures and discuss benefits and costs of various system configurations. Overall, we find that first-generation service discovery systems can be robust under difficult failure environments. This work contributes to better understanding of failure behavior in existing discovery systems, allowing potential users to configure deployments to obtain the best achievable robustness at the least available cost. The work also contributes to design improvements for next-generation service discovery systems.

Keywords: Distributed systems, robustness, service discovery

1. INTRODUCTION

Various teams designed and implemented a first generation of (competing) service discovery systems [1-6] that enable distributed components to find each other without prior arrangement, to express capabilities and needs, to compose into collections, and to detect and adapt to changes. Each specific design defines a system structure, along with protocols for discovery, monitoring, and recovery. Some designs [5,6] assume a specific underlying communication technology, and some designs [1,5] focus on one application domain. Three designs [2-4] were conceived to operate over Internet protocols and to support many applications.

In this paper, we investigate the architectures and behaviors underlying Jini Networking Technology[1] [2], Universal Plug and Play (UPnP) [3], and the Service Location Protocol (SLP) [4] when subjected to various failures. Elsewhere [7], we present a generic model encompassing the designs of these systems and we identify performance issues that could arise. While this previous work considers system behavior absent failures, here we explore the relative ability of discovery systems to cope with different types and intensities of failure.

We reported preliminary results in various conference papers [8-11]; however, this paper improves upon earlier work in two ways. First, we extend the scope of our results to cover three architectures (two-party, three-party, and adaptive), three failure scenarios (configuration restoration, service acquisition and maintenance, and consistency maintenance), four failure types (power failure and restart, node failure, communication failure, and message loss), and a set of failure detection and recovery techniques at three levels (transport protocols, discovery protocols, and application logic). Second, we increase the amount of data collected and analyzed to obtain better estimates for performance metrics at high failure rates.

This paper contributes to the understanding of service discovery systems. First, this paper characterizes robustness of discovery systems under difficult failure environments. This paper further identifies and discusses the most significant design and configuration decisions that influence robustness. Second, this paper identifies specific design and deployment decisions that could lead to diminished robustness. Third, this paper quantifies the relative cost associated with specific decisions. Overall, the information provided here should contribute to better understanding of failure behavior in existing discovery systems, allowing potential users to configure deployments to obtain the best achievable robustness at the least available cost. Further, results and discussions presented here could contribute to design improvements in the next generation of discovery systems.

This paper also contributes experimental methods to study robustness in distributed systems. First, we introduce and apply metrics to quantify relative robustness and cost at the application level for various scenarios. Second, we present a technique to decompose aggregate robustness into detection and recovery latency. Using this technique, we show how similar robustness can be achieved through different behaviors arising from particular design choices. Our methods can be adopted, adapted, or extended by other researchers to investigate failure response in distributed systems – a topic due for increased study.

We begin (in Section 2) with a synopsis of existing work comparing and contrasting service discovery systems. Most previous work focuses on functional comparisons [12-19], on means for translating among discovery systems [20-26], or on improving existing designs [27-37]. Our own related work [7, 38-42] attempts to unify designs for several existing discovery systems, and investigates performance problems arising when such systems are deployed at large scale.

[1] Certain commercial products or company names are identified in this paper to describe our study adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor to imply that the products or names identified are necessarily the best available for the purpose.

In Section 3, we survey the design and function of service discovery systems. We introduce a model to convey concepts across selected systems. Using our model, we describe how discovery operates under UPnP (a two-party architecture, where clients issue multicast queries to find services), Jini (a three-party architecture, where clients consult a directory to find services), and SLP (which is a three-party architecture that can adapt to become a two-party architecture). We also describe two mechanisms (polling and notification) used by discovery systems to maintain consistent information among distributed replicas. The architectures, discovery procedures, and consistency maintenance mechanisms described in Section 3 form the basis for scenarios, experiments, and results recounted in later sections.

In Section 4, we introduce selected types of failure that can impede a distributed system and we discuss selected techniques to detect and recover at three layers. At the lowest layer, transport protocols may include detection and recovery mechanisms (e.g., acknowledgments, retransmissions, and exceptions). In the middle layer, discovery protocols typically include some detection and recovery mechanisms (e.g., heartbeats and soft state). At the top layer, applications may take recovery actions in reaction to exceptions raised by transport protocols. Interactions among these detection and recovery techniques can become quite intricate and difficult to understand.

In Section 5, we describe our experiment methodology, consisting of six steps: (1) constructing (simulation) models reflecting structure, behavior, and deployments of selected service discovery systems, (2) incorporating failure models into the simulations (3) devising scenarios and related metrics to quantify robustness and cost, (4) simulating scenarios for selected configurations over a range of failure rates, (5) collecting, analyzing, and plotting data from simulations, and (6) investigating unexpected results and anomalies. In Section 6, we describe the design and results for our experiments: (1) restart after power failure, (2) service acquisition and maintenance impeded by node failures, and consistency maintenance impeded (3) by communication failures and (4) by message loss. We report results from these four experiments, which encompass 30 configurations. For each experiment, we explain the scenario and failure model, define metrics, present results, outline findings, and discuss unexpected outcomes. We close in Section 7 with a précis of our findings and contributions.

2. RELATED WORK

Emergence of various specifications for service discovery systems, coupled with the anticipated importance of discovery functionality in future distributed systems, has stimulated significant interest in understanding similarities and differences among competing designs. Most existing comparisons focus on architecture, features, and function. A few comparisons also consider programming differences, because most discovery systems are conceived as middleware to support distributed applications. Bettsletter and Renner [12] compare SLP, Jini, UPnP, and Bluetooth with respect to architecture, function, and features, and consider underlying requirements for programming languages, operating systems, and network protocols. The comparison is expressed using concepts and terminology specific to each discovery system, although the authors do identify three common aspects (support for searching on service attributes, inclusion of a directory, and use of leasing) for comparison. Richard [13] compares software architectures, along with system features and functions, for Jini, Bluetooth, Salutation, SLP, and UPnP. Elsewhere [17], Richard expands his comparison to include programming considerations by providing source code for clients and services in Jini, SLP, UPnP, and Bluetooth. Pascoe [15] outlines a brief architectural comparison of Jini, UPnP, and Salutation, and Rekesh [14] gives a similar comparison that appears to be based on Pascoe's work. In a subsequent paper [16], Pascoe amplifies his architectural comparison to include comparison of functions and features. O'Driscoll [18], when considering a wide range of home networking technology, provides descriptions of Bluetooth, HAVi (the Home Audio-Video interoperability specification), UPnP, and Jini. Though giving no direct comparison, O'Driscoll provides a summary of architecture, function, and features from which readers may infer a comparison. Olivier [19] provides a detailed description of Jini, but also includes a brief description of UPnP and a comparison between Jini and SLP. None of these comparisons considers performance or robustness.

Limitations in existing comparisons motivated our own work. Elsewhere [7], we provide a unified and general model for first-generation discovery systems and then show how our model can be used to represent Jini, UPnP, and SLP. Our unified model, conceived with neutral terminology, provides a basis for direct comparison among architectural, functional, and behavioral elements of designs. Our model also reveals limitations and open issues in existing designs and specifications, and includes a set of service guarantees that we believe discovery systems should attempt to satisfy. Further, we identify selected performance issues that may arise when deploying discovery systems at large scale, and we use our model to outline algorithms that might improve performance. While our previous work improves on existing comparisons, we did not consider robustness under various types of failure. The present paper extends our previous work by comparing failure response in the major designs for first-generation discovery systems (as represented by Jini, UPnP, and SLP).

As a natural extension to functional comparisons, some researchers conceive protocol translators in order to achieve interoperation among dissimilar service discovery systems. For example, the Open Services Gateway Initiative (OSGi) [20, and also chapter 17 in 18] defines a layer of middleware to bridge among Jini, UPnP, and Bluetooth. Miller and Pascoe [21] show how to map between the application-level programming interfaces of Salutation and Bluetooth. Allard et al. [22] and Sameh and El-Kharboutly [23] describe different techniques to bridge between Jini and UPnP, while Guttman and Kempf [24] consider techniques to bridge between Jini and SLP. Similarly, Yu et al. [26] define a software structure for middleware that can bridge among a diverse set of service discovery systems and distributed object systems. Ponnekanti and Fox [25] take a more general tact by defining a framework that clients may use to find candidate services and to automatically configure an appropriate set of proxies and stubs to allow a client to invoke a selected service. Only one [23] of these papers investigates performance, and none considers the effects of failures. While our paper does not consider translation among discovery systems, researchers could use our method to investigate and quantify robustness of various designs for bridges and translators.

Beyond first-generation systems for discovery of services operating in close proximity, researchers in industry and academe are investigating how to build discovery systems that scale over a wide area. An early proposal, known as Universal Description,

Discovery and Integration (UDDI) [36], defines well-known, web-accessible repositories, where service descriptions may be deposited so that clients may query for services of interest. The UDDI approach exhibits limited scalability because every service in a network must deposit its description with a central directory, or else with multiple replicas of a central directory. To overcome such limitations, researchers continue to propose a number of more flexible approaches. One early idea, E-speak [28], used an expanding-ring multicast search to discover directories that organized into a federated topology through which service descriptions permeated over time. A similar idea is contained in JXTA [29], where a peer-to-peer system is used to disseminate copies of service descriptions throughout a topology of caches, and in Neuron [32], a self-organizing and self-tuning topology of caches that can tolerate failures of nodes and communication links. Other self-organizing directories have also been proposed, including SRIRAM [31], NeuroGrid [34], and the Secure Service Discovery Service [27]. A somewhat different approach [30] forms a logical ring (based on node addresses) that helps individual nodes to bootstrap into various available overlay networks, each of which advertises services. Grid researchers have also proposed a design for wide-area service discovery [33], coupled with the ability to inject and disseminate real-time status information [35]. Most of these designs include provisions to detect and recover from failures or to mitigate failures; however, no comprehensive results exist that compare robustness among various designs. While this paper investigates robustness only for local discovery, we suspect that our method could be applied to quantify and compare robustness among designs for wide-area discovery.

3. MODELING SERVICE DISCOVERY SYSTEMS

Service discovery systems enable components in a network to discover each other, and to determine if discovered components meet specific requirements. Further, discovery systems include consistency-maintenance mechanisms, which can be used by applications to detect changes in component availability and status, and to maintain, within some time bounds, a consistent view of distributed components. Many diverse industry activities explore different approaches to meet such requirements, leading to a variety of proposed designs [1-6]. Some groups approach the problem from a vertically integrated perspective, coupled with a narrow application focus. Other groups propose more widely applicable solutions. For example, a team of researchers and engineers at Sun Microsystems designed Jini Networking Technology [2], a discovery system atop Java, which provides a base of portable software technology. As another example, a group of engineers at Microsoft and Intel conceived Universal Plug-and-Play (UPnP) [3] to extend plug-and-play from single computers to distributed systems. Similarly, the efforts of Sun Microsystems and other companies led to the Service Location Protocol (SLP) [4], aimed at providing service discovery for the Internet.

While these designs appear quite different, the systems share some common traits. First, they all assume availability of the Internet protocols as a base. Second, they all provide general approaches to describe the capabilities and status of services. Third, they all include mechanisms that can be used to detect and recover from failures. Jini, UPnP, and SLP differ in architecture, in approach to describing services, and in assumptions about how to use transport protocols. This interesting combination of similarities and differences led us to base our comparative study on Jini, UPnP, and SLP. Our main challenge was finding a means to clearly understand and represent similarities and differences among the three systems. To address this challenge, we developed a general model with common terminology and then mapped concepts from each specific system into our model.

3.1 A General Model of Service Discovery Systems

Our model provides a basis for comparative analysis of various discovery systems by representing major architectural components and concepts with a consistent and neutral terminology (see first column in Table 1). The main components in our model include: (1) *service user*, (2) *service manager*, and (3) *service cache manager*. A service user (SU) is a client in a service discovery system. A SU is concerned with discovering services from components within the distributed system, acquiring access to discovered services, and using discovered services. A service manager (SM) maintains a database of *service descriptions*, each of which encodes the characteristics of a particular *service provider* (i.e., the provider of the service). Each service description (SD) contains the identity, type, and attributes that characterize a service provider (SP). Each SD also includes the addresses of software interfaces (e.g., an application-programming interface or graphic user interface) to access a service. A SU seeks SDs satisfying specific requirements. A service cache manager (SCM) operates as an intermediary, matching advertised SDs from SMs to requirements provided by SUs. SCMs are optional components supported by some, but not all, discovery systems. Table 1 shows how these general concepts map to specific concepts from Jini, UPnP, and SLP.

Table 1. Mapping Concepts among Selected Service Discovery Systems.

Generic Model	Jini	UPnP	SLP
Service User	Client	Control Point	User Agent
Service Manager	Service or Device Proxy	Root Device	Service Agent
Service Provider	Service	Device or Service	Service
Service Description	Service Item	Device/Service Description	Service Registration
Identity	Service ID	Universal Unique ID	Service URL
Type	Service Type	Device/Service Type	Service Type
Attributes	Attribute Set	Device/Service Schema	Service Attributes
User Interface	Service Applet	Presentation URL	Template URL
Program Interface	Service Proxy	Control/Event URL	Template URL
Service Cache Manger	Lookup Service	not applicable	Directory Service Agent (optional)

The behaviors by which (Jini, UPnP, and SLP) SUs discover and maintain consistency in relevant SDs depend in part upon the system architecture and design and in part on the transport protocols used. Transport protocols are used for two kinds of message exchange: (1) multicast, in which transmitted messages are conveyed to all receivers that participate in a multicast group and (2) unicast, which is point-to-point communication directly between a pair of corresponding entities. Both Jini and UPnP use the UDP (User Datagram Protocol) for exchanging multicast messages and use the TCP (Transmission Control Protocol) for exchanging unicast messages. UPnP also uses UDP to unicast answers to multicast queries. SLP uses UDP for exchanging both multicast and unicast messages. The

differences in transport protocols become significant when considering approaches to detect and recover from failures; therefore, we defer (until Section 4) a more detailed discussion. Here, we focus on behavioral differences arising from variations in architecture and design.

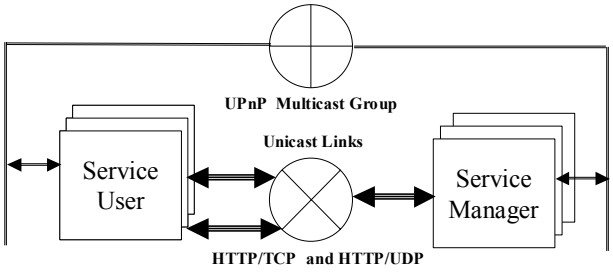


Figure. 1 Two-party service discovery system deployed in a topology with three service users (SUs) and thee service managers (SMs).

3.2 Modeling Service Discovery Architectures and Protocols

Our analysis of six distinct discovery systems revealed that most designs use one of two architectures: *two-party* or *three-party*. One discovery system we examined uses both architectures together. A two-party architecture consists of two major component types: SMs and SUs. Figure 1 illustrates a two-party architecture (configured for UPnP). Service discovery occurs through interactions between these two component types; SUs discover SMs and then query them for suitable SDs. A three-party architecture adds a third component type, the SCM, which contains a directory. Figure 2 illustrates a three-party architecture (configured for Jini). In a three-party architecture, both SMs and SUs first discover SCMs to serve as intermediaries. SMs deposit SDs with SCMs and SUs interact with SCMs to obtain suitable SDs. A third architectural variant (supported by SLP) employs both the two-party and three-party architecture and is capable of switching between them, depending on circumstances. We call this an *adaptive* architecture.

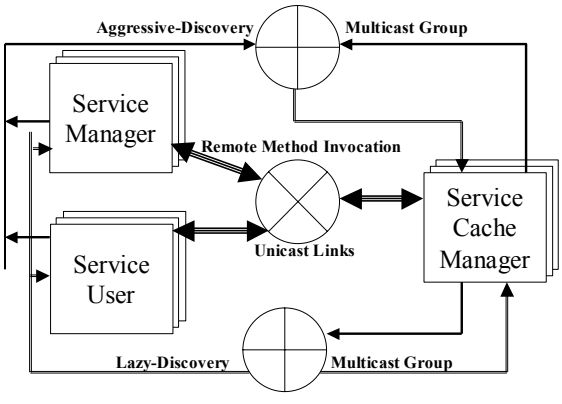


Figure 2. Three-party service discovery system deployed in a topology with three service users (SUs), three service manager (SMs), and three service cache manager (SCMs).

3.2.1 Discovery in Two-Party Architectures. Given a two-party architecture, we model the behavior of participating SMs and SUs. Upon startup, each SU and SM engages in a discovery process to locate other relevant components within the network neighborhood. We chose behaviors described in the specification for UPnP [3].

In a lazy-discovery process, each SM periodically announces existence of its SDs over a designated UPnP multicast group. Upon receiving these announcements, SUs with matching requirements use a HTTP (HyperText Transfer Protocol)/TCP unicast link to request, directly from the SM, copies of the SDs associated with relevant SPs. The request is made using an HTTP `GET` request. The SU stores SD copies in a local cache.

Alternatively, the SU may engage in an aggressive-discovery process, where the SU transmits SD requirements, as `Msearch` queries, on the UPnP multicast group. Any SM holding a SD with matching requirements may use a HTTP/UDP unicast link to respond (after a jitter delay) directly to the SU. Whenever a SM responds to an `Msearch` query (or announces itself), it repeats a sequence of messages, with separate messages for distinct devices and service types managed by the SM. For each appropriate response, the SU uses a HTTP/TCP unicast link to send an HTTP `GET` request for a copy of relevant SDs, caching them locally.

In UPnP, multiple HTTP `GET` requests are required to transfer the SD, because each SD consists of two parts. To maintain a SD in its local cache, a SU expects to receive periodic announcements from the relevant SM. In UPnP, the SM announces the existence of SDs at a specified interval, known as a Time-to-Live, or TTL (1800 s minimum recommended). Each announcement specifies a TTL value. If the SU does not receive an announcement from the SM within the TTL (or a periodic SU `Msearch` does not succeed within that time), the SU may discard the discovered SD.

3.2.2 Discovery in Three-Party Architectures. Given a three-party architecture, we model the behavior of participating SCMs, SMs, and SUs, which each engage in a discovery process upon startup. We chose behaviors described in the Jini specification [2], where SMs and SUs attempt to discover any intermediary SCMs that exist in the network neighborhood.

Upon initiation, a Jini component enters aggressive discovery, where it transmits probes on a designated aggressive-discovery multicast group at a fixed interval (5 s recommended) for a specified period (seven times recommended), or until it has discovered a sufficient number of SCMs. Upon cessation of aggressive discovery, a component enters lazy discovery, where it listens on a designated lazy-discovery multicast group for announcements sent at intervals (120 s recommended) by SCMs. Our three-party model implements both the aggressive and lazy forms of Jini multicast discovery.

Once discovery occurs, a SM deposits a copy of the SD for each of its services on the discovered SCM. The SCM caches this deposited state, but only for a specified length of time, or TTL. To maintain a SD on the SCM beyond the TTL, a SM must refresh the SD. In this way, if the SM fails, then the SCM can purge any SDs deposited by the SM. SUs may query discovered SCMs for SDs of interest. Alternatively, a SU may deposit a query with the SCM, which will attempt to match SDs provided by SMs to specifications of the deposited query. The SCM forwards any matching SDs on to the SU that deposited the relevant query.

3.2.3 Discovery in Adaptive Architectures. An adaptive architecture requires SMs and SUs to rendezvous through a SCM,

but allows direct SM-SU interaction when no SCM is available. If SMs and SUs interact directly and a SCM becomes available, then the architecture requires SMs and SUs to resume interacting through the SCM. We use the term *mode switching* to denote this ability to change architectural configurations (i.e., to switch between two- and three-party architectures). To model an adaptive architecture, we chose behaviors from the SLP specification [4].

SLP systems are configured by default to operate in three-party mode, switching to two-party mode when SCMs are unavailable. Like Jini, three-party SLP discovery requires that SMs and SUs first discover intermediary SCMs. Upon initiation, a SLP SM or SU enters aggressive discovery, where every 900 s it transmits six probes within a fixed interval of 15 s on a designated aggressive-discovery multicast group. On the other hand, a SLP SCM and SM component commences lazy discovery, where it emits announcements on a designated lazy-discovery multicast group at recommended intervals of 10800 s (once every three hours), which we lowered to 120 s in all experiments to provide more consistent behavior in the adaptive and three-party architectures. When operating in three-party mode, SLP SUs and SMs rendezvous through SCMs. After discovery, SLP SMs employ procedures (similar to Jini) to deposit SDs for relevant services on discovered SCMs for a specified TTL, and then to refresh deposited SDs. To make behavior as consistent as possible across our models, we decided to use the same TTLs (on a per experiment basis) for a SD to be cached by a SCM. We denote a specific choice of TTL when describing each experiment (see Section 6). SUs query SCMs for SDs matching their requirements. SCMs process queries, matching SDs against SU requirements, and forward matches to SUs. SUs can cache the response and contact the related SPs to obtain use of the service.

When SLP SUs and SMs fail to detect SCMs, they switch to two-party mode. In two-party mode, a SLP SU both listens for lazy announcements from SMs and transmits the aggressive-discovery six-message probe sequence at 900 s intervals, while SMs listen for probes and respond as appropriate. Upon receiving a lazy announcement or an aggressive-probe response, a SLP SU (in two-party mode) queries the SM for SDs matching its requirements. The SM responds with matching SDs, which the SU caches locally. In the meantime, SUs continue to search for a SCM, using both lazy and aggressive discovery. Upon finding a SCM, SLP requires the SU to switch to three-party mode and to cease direct contact with SMs discovered in two-party mode. All further contact with SMs must take place through SCMs.

3.3 Modeling Consistency Maintenance Mechanisms

Service discovery systems include *consistency-maintenance mechanisms* to ensure that changes to critical information about services can be propagated to interested SUs. Critical information could include service availability and capacity, and updates to descriptive information about service capabilities. Discovery systems that we analyzed provide one or both of two consistency-maintenance mechanisms: *polling* and *notification*. We discuss each in turn.

3.3.1 Polling. In polling, a SU periodically sends queries to obtain up-to-date information about a SD that was previously discovered, retrieved, and cached locally. In a two-party architecture, the SU issues the query directly to the SM from which the SD was obtained; thus, we model the UPnP HTTP `GET` request mechanism to poll the SM to retrieve a SD associated with a specific URL (Uniform Resource Locator). In response, the SM provides a SD containing a list of supported services, including relevant attributes.

Polling in a three-party architecture consists of two independent processes. In one process, a SM sends a request to propagate an updated SD to each SCM on which the SD was originally cached. In Jini, this request takes place through a `ChangeService` message, which causes the SCM to update the cached SD. In SLP, the SM re-registers the SD, which causes the SCM to replace the previously deposited SD with the new version and an updated TTL. In a second process, each SU polls relevant SCMs by periodically issuing a query for a copy of SDs that the SU has previously retrieved and cached. The SCM replies with matching SDs. In Jini, the poll is implemented with a `FindService` request and a `MatchFound` reply; SLP polls (SCMs in three-party mode and SMs in two-party mode) with `SrvRqst` and `SrvReply` messages, respectively. We adopted a 180 s polling interval for all architectures.

3.3.2 Notification. Notification requires that updates be transmitted to interested parties immediately after they occur. We model notification only for the two-party and three-party architectures (i.e., not for the adaptive architecture), because the SLP specification that we used does not include notification.

In two-party notification, a SM sends events to a SU that indicates a SD has changed. To receive events about a SD of interest, a SU must first register with the SM for this purpose. We model this procedure using the UPnP subscription mechanism, where the SU sends a `Subscribe` request, and the SM responds by either accepting or denying the request. The subscription, if accepted, is retained for a TTL, which may be refreshed with subsequent `Subscribe` requests from the SU. In our experiment, we chose 1800 s as TTL for subscriptions in both (the two- and three-party) architectures.

Three-party notification requires a two-step procedure, which we model as specified for Jini. First, SUs must register with SCMs to receive notification about SDs of interest. The SCM registers the notification request for a specified TTL, which may be refreshed. Second, a SM issues a `ChangeService` to propagate a SD update to all SCMs on which the SM has previously deposited the SD. When the SCM receives a `ChangeService` request from a SM for a SD it has cached, the SCM issues a `MatchFound` that propagates the updated SD to all SUs that have registered to receive such notifications.

4. MODELING FAILURE DETECTION AND RECOVERY TECHNIQUES

Interactions among distributed components may be impeded by failures; thus, such components must be prepared to detect failures and take recovery actions. In this section, we review the types of failure that can impede interactions and then we describe selected failure detection and recovery techniques. We explain how we incorporated the techniques into our models.

4.1 Failure Types

We classify failures into two general categories: process failures and communication failures. Process failures can be caused by cyber attacks, by programming errors, or by hardware failures. We can subdivide process failures into node and thread failures. During a catastrophic failure, processing in a node ceases, and the node must reinitialize before processing resumes. Some information maintained by the node may persist across the failure,

while other information may be lost. The nature and condition of persistent information could prove crucial to a node's behavior after processing resumes. Of course, the node might never reappear. Thread failures, while less catastrophic, can be more troublesome than node failures. A node might rely on certain long-running threads to react to events from other nodes. Failure of selected threads can interfere with the operation of the node, as well as other nodes. In some cases, a node can appear to be present, while being effectively inoperable. Since the effects of node and thread failure are similar, we focus only on node failure in this study, allowing the effects of thread failure to be inferred.

Communication failures can arise due to jamming, or other interference, due to congestion, due to denial of service attacks, due to physical severing of cables, due to improperly configured or sabotaged routing tables, or due to multi-path fading as nodes move across a terrain. We subdivide communication failures into three classes: interface failures, message loss, and path failures. A communication interface in a node may fail fully (both transmit and receive) or partially (either transmit or receive). All outbound messages from an interface will be lost when the transmitter fails, while all inbound messages will be lost when the receiver fails. Message loss, a less severe failure, implies that individual messages may be dropped, either sporadically or in bursts. Path loss appears as a blocked communication route between two nodes, or areas, in a network. A path can be blocked in one or both directions. Because effects of path failure are similar to interface failure, we studied only interface failure.

4.2 Failure Detection and Recovery Techniques

In service discovery systems, failure detection and recovery responsibilities are divided among three parties: (1) transport protocols, (2) discovery protocols, and (3) applications. The transport protocols support the discovery protocols and the application, while the application also relies on the discovery protocols. We first describe failure detection and recovery provided by transport protocols, such as TCP and UDP. We then discuss heartbeats and soft state — the main detection and recovery techniques implemented by discovery protocols. Subsequently, we discuss remote exceptions and retries, which are the main detection and recovery techniques available to applications and selected discovery processes. We describe how we model these techniques.

4.2.1 Recovery by Transport Protocols. Discovery protocols and applications use recovery services from three types of transport: (1) unreliable multicast protocols, (2) unreliable unicast protocols, and (3) reliable unicast protocols. We discuss each in turn.

Unreliable Multicast Protocols. Unreliable protocols, whether multicast or unicast, neither recover nor signal lost messages; thus, neither source nor destination will learn of a loss. Further, multicast protocols exchange messages along a tree of receivers. For this reason, a multicast message might be received by some nodes, but not by others. A failure near a multicast source prevents messages from being received by any node in the multicast tree, while a failure near a receiver prevents messages from being received by only a single node in the tree. Of course, failures at intermediate points in the tree could result in messages being lost to subsets of receivers. All three systems we studied (UPnP, Jini, and SLP) employ unreliable UDP multicast protocols.

When simulating UDP transmission, our models discard messages lost due to congestion and due to interface failures. During interface failure, the models discard all messages sent from a node with a failed transmitter, as well as all messages inbound for a node with a failed receiver. Neither sender nor receiver learns the fate of lost messages. Since unreliable protocols provide no guarantees, recovery must be provided by mechanisms at a higher layer.

Unreliable Unicast Protocols. Among the systems we studied, both SLP and UPnP use an unreliable unicast protocol. SLP uses unicast UDP to transmit `SrvRqst` messages, used for queries, and to transmit `SrvReg` messages for registrations and registration renewals. To improve reliability, SLP employs two additional procedures. First, SLP issues redundant `SrvRqst` messages; each request is sent four times within a 15 s interval. Second, SLP requires a waiting period (we used 15 s) to listen for a corresponding `SrvRply`. If no `SrvRply` is received within that time, then the message transmission is abandoned and a remote exception (REX) is declared so that a higher layer entity can decide upon an appropriate recovery action. Our SLP models incorporate this behavior.

UPnP uses unicast UDP to send responses to `Msearch` queries. To improve the reliability of these responses, UPnP requires that each UDP message be sent multiple (n) times. In our model, we set $n=2$.

Reliable Unicast Protocols. Reliable unicast protocols include mechanisms that attempt to ensure message delivery by detecting and re-transmitting lost messages. Of course, the reliability schemes may eventually give up if too many retransmissions are needed (which might indicate node or interface failure). In such cases, the reliable unicast protocol will signal to a higher layer that a message was (probably) not delivered. For example, Jini uses Remote Method Invocation (RMI) over TCP to invoke a method on a remote object, and to receive a response and UPnP uses TCP to submit HTTP requests and receive HTTP responses. Either the RMI layer (in Jini) or the TCP layer (in UPnP) can signal a remote exception (REX).

Our model unifies reliable unicast protocols into one set of procedures that simulate TCP in two phases: connection establishment and data transfer. The connection establishment phase consists of exchanging connection request and response messages. Both connection requests and responses may involve multiple retries before a connection is established. We simulate connection request retries with delays of 6 s, 24 s, and 24 s, before signaling the connection requester with a REX 24 s after the final retry (78 s after the initial request).

Successful connection establishment initiates a data-transfer phase, where the connection requester sends a data request and may await a data response. The data request and response may be subject to retransmissions. We compute a retransmission timeout (RTO) that is roughly the round-trip time, or RTT. We increase the RTO by 25% with each successive retransmission. Retries in the data-transfer phase continue until a time threshold (60 s) is reached, after which the transmission attempt is abandoned. Failure of a data request causes a REX to be issued to the requester. Failure of a data response causes a REX to be issued to both the requester and responder. The requester cannot determine whether a REX was caused by failure to transmit the request or by failure to receive a response. The responder has more information, as it does not receive a REX when an inbound request fails, but does receive a REX when its

Table 2. Summary of Recovery Mechanisms and Key Parameters.

Responsible Party	Recovery Mechanism	Two-Party Architecture (UPnP)	Three-Party Architecture (Jini)	Adaptive Architecture (SLP)
Transport Protocols	Multicast UDP	No recovery	No recovery	No recovery
	Unicast UDP	Redundant transmission $n = 2$ No recovery	Not Applicable	Redundant transmission $n = 4$ No recovery
	TCP	Issue REX in 78 s	Issue REX in 78 s	Not Applicable
Discovery Protocols	Heartbeat	SM sends n ($3+2d+k$) lazy announcements of SDs at interval varied by experiment. SU caches SD for TTL varied by experiment. (recommended 1800 s for announcement interval and TTL by UPnP)	SM registers SDs for TTL varied by experiment; SU registers notifications for TTL varied by experiment.	SM (in two-party mode only) sends lazy announcements at 120 s interval (recommended 10800 s by SLP); SM registers SDs for TTL varied by experiment.
	Soft-State Recovery	SU issues aggressive probe (UPnP *Msearch*) at interval after purging SD (set to 120 s).	SU and SM issue seven probes (at 5 s intervals) only during startup; SCM issues lazy announcements at interval (120 s).	SU and SM issue 6 probes within 15 s duration during startup and at 900 s interval; SCM sends lazy announcements at 120 s interval (SLP recommends 10800 s).
Application	Ignore REX	SU: *HTTP Get* Poll SM: Notification	SU: *FindService* Poll SCM: Notification	SU: *SrvRqst* Poll (Notification unsupported)
	Retry after REX	SU: *HTTP Get* after discovery retry (180 s with ≤ 3 retries); Registration request and refresh retry (120 s)	SM: depositing or refreshing SD on SCM retry; SU: registering and refreshing notification requests on SCM retry (120 s)	SU:*SrvRqst* after discovery retry (180 s with ≤ 3 retries); SM (three-party mode) depositing or refreshing SD on SCM retry (120 s)
	Discard Knowledge	SU purges SD after failure to receive SM announcement within TTL or after 3 retries of *HTTP Get*	SU and SM purge SCM after period of continuous REX (varied by experiment).	Three-party mode: SU and SM purge SCM after period of continuous REX Two-party mode: SU purge SM after period of continuous REX (varied by experiment).

outbound response fails. In essence, while reliable unicast protocols attempt to deliver messages in the face of various communication failures, ultimately the reliability mechanisms might prove insufficient, causing a higher-layer process to be notified of the failure. In such cases, the higher-layer process is free to determine an appropriate recovery strategy.

4.2.2 Recovery by Discovery Protocols. Components in a discovery system may also learn of failure by listening for recurring messages sent by remote components, much as a heartbeat is monitored to assess patient health. For example, UPnP SMs periodically multicast lazy announcements advertising SDs. Similarly, Jini and SLP SMs periodically refresh SD registrations on SCMs by sending unicast messages, and then listening for responses. Both lazy announcements and registration refresh messages convey *soft* state (or information) — in this case, the SD, which a receiver can cache for a period consistent with the associated TTL. When subsequent heartbeat messages fail to arrive within the TTL, a listener may assume failure of the SM and thus discard cached information about its related SD, effectively eliminating knowledge about existence of the related service.

Our models use a form of soft state that allows SDs for failed components to be discarded and then to be either rediscovered or replaced. For example in our two-party model, once a UPnP SU discards knowledge of a SM and any associated SDs, the SU commences periodic multicast (`Msearch`) queries to search for a new instance of the service. Once the SU regains a SD meeting its requirements, the related queries cease. SLP employs an analogous procedure when operating in two-party mode.

The process is more complicated in three-party situations. Here, failure of refresh messages causes SCMs to discard a service registration. A SU may monitor the status of the SD by periodically polling the SCM. When poll responses indicate the SD is no longer present on the SCM, the SU may then discard its cached copy of the SD. In Jini, SUs may also register with the SCM to be notified when the SCM discards the SD. When receiving such notification, a SU discards its cached copy of the SD and then attempts to find a replacement by querying the SCM for another SD that satisfies its requirements. Meanwhile, a SM for a SD discarded by the SCM might recover after failures are repaired. The SM may rediscover the SCM through aggressive or lazy discovery, and then reregister the lost SD. The SU, if it has not found a replacement, can then receive the original SD by querying the SCM (Jini and SLP) or through notification (Jini).

Table 2 summarizes the way in which we model heartbeat and soft state for each of our models. The table indicates values we adopted across all experiments (except as otherwise indicated in the table and discussed in Section 6). Though SCM discoveries could also be retained by SMs and SUs on a soft-state basis, the discovery systems we studied use an application-level technique to detect SCM failures.

4.2.3 Recovery by Applications. When failure detection leads to a REX, discovery systems generally expect application software to initiate recovery, guided by an application-level retry policy. In our models, depending on the situation, we implement three different policies: (1) ignore the REX, (2) retry the operation for some period, and (3) discard knowledge. The discard strategy, employed following repeated failure of the retry strategy, relies upon discovery mechanisms to recover from failures that are more persistent. These strategies (discussed below) are summarized in Table 2.

Ignoring the Remote Exception. In general, our models ignore any REX received when responding to a request, relying on the requester to retry. A SU can ignore a REX received when issuing a poll (e.g., `FindService`, `SrvRqst`, or HTTP `GET`) because the poll recurs at an interval. A Jini SCM (three-party model) or UPnP SM (two-party model) also ignores a REX received while attempting to issue a notification. This behavior, which is described in both the Jini and UPnP specifications, depends upon TCP to provide reliability for notifications. Notifications include sequence numbers that allow a receiving node to determine whether or not previous notifications were missed.

Retrying the Operation. In our models, we retry selected operations in the face of a REX. The UPnP specification separates the operation of discovering a service from obtaining a description of the service (Jini combines these operations). Without a description, a service cannot be used. For this reason, in the UPnP model, a SU must issue a HTTP `GET` to obtain a description. If no description arrives within 180 s, then our model retries the HTTP `GET`. If unsuccessful after three attempts, the SU purges the related SD and discards knowledge of the SM. Our three-party models, based on Jini and SLP, also contain a retry strategy, but associated with attempts to register or change a SD with a SCM. In these cases, the SM retries a `ChangeService` or `ServiceRegistration` 120 s after receiving a REX. Similarly, when a SU receives a REX (from either a SM or SCM) in response to a request to register for notification, the SU retries the registration in 120 s. These retries recur up to some time bound, after which the SM discards knowledge of the SCM.

Discarding Knowledge. Both the two-party and three-party models include the possibility that an application can discard knowledge of previously discovered nodes. After discarding knowledge of a SM or SCM, all operations involving that node cease until it is rediscovered, either through lazy or aggressive discovery.

In our UPnP model, SUs discard a SM (and any related SDs) after failure to receive announcements from a SM within a TTL or after three unsuccessful retries of a HTTP `GET`. In our SLP model (two-party mode), SUs do not discard SMs after failure to receive announcements. We took this decision because the SLP specification does not require SUs to discard a SM when missing a heartbeat.

In our three-party model (based on Jini), a SM or SU deletes a SCM after a period (varied by experiment) of receiving only REXs when attempting to communicate with a SCM. We adopt this behavior because the Jini specification states that a discovering entity *may* discard a SCM with which it cannot communicate. While the SLP specification is silent on these issues, we implemented our SLP model (in both two-party and three-party modes) so that SUs discard SMs after a period (varied by experiment) of continuous REXs. We took this decision to align this behavior among all our models.

5. EXPERIMENT METHODOLOGY

We adopted a common approach to modeling, to experiment design, and to metrics for analysis. Aspects of the approach seem suited to investigation of failure response in other classes of distributed systems. Below, we discuss our approach.

Model Construction. We created simulation models for the three architectures we found. Executable models enabled us to understand collective behavior among distributed components. We based the structure and behavior of our models (recall Section 3) on specifications for UPnP [3] (two-party architecture), Jini [2] (three-party architecture), and SLP [4] (adaptive architecture). Each model comprises a set of components (and relationships among them), interactions (as messages received by components), behavior (as actions taken in response to messages, including generating new messages), and variables (to represent internal state of components). Components communicate via a simulated transport service that represents multicast UDP and unicast UDP and TCP (as explained in Section 4.2.1). The transport service can be impeded by simulated message loss and interface failures. We used Rapide [43], an architecture description language and accompanying toolset developed at Stanford University, to implement models of Jini and UPnP; for SLP we used SLX, a simulation system developed by Wolverine Software [44]. We chose to use two different simulation systems in order to establish the generality of our approach. We note that the Rapide system automatically records causal event traces and provides tools to visual and analyze those traces.

Experiment Design. With simulation models in hand, we designed experiments to investigate failure response for selected configurations of components, where each configuration represents a distinct combination of architecture (two-party, three-party, or adaptive), number of deployed SCMs, and choice of behaviors for discovery, consistency maintenance, and recovery. We approached experiment design by focusing on the types of failures (recall Section 4.1) that might interfere with system operation. We decided to consider four failure types: (1) power failure and restart, (2) node failures, (3) interface failures, and (4) message loss. For each failure type, we constructed an application-level scenario to exercise simulated topologies. Our scenarios include: (1) recovering a previously discovered configuration (on restart after power failure), (2) maintaining operational capability in a distributed real-time control application (impeded by failure of nodes hosting needed components), and (3) maintaining consistency of distributed information (when communication is impeded by interface failures or message losses). For these scenarios, we simulated various configurations of our models with parameters selected to ensure that observed performance differences resulted only from differences in system architecture and protocol. For three scenarios (node failures, interface failures, and message loss), we subjected each configuration to increasing failure rates, while measuring system response. To focus on fundamental differences in the designs for discovery systems, we excluded a number of possible application-level choices, such as local caching of service descriptions and varying subscription lengths.

Metrics. To compare failure response among simulated configurations, we defined metrics specific to each scenario. Broadly these metrics fall into three categories: (1) *effectiveness*, which is the ability of a distributed system to exhibit a desired state, expressed as a probability that the state is reached or a

proportion of time a system is in the desired state; (2) *responsiveness*, which is the time taken, or latency, to reach the desired state; and (3) *efficiency*, which is the amount of effort, measured by the number of messages, required for a distributed system to complete a scenario. For most combinations of configuration and scenario, we conducted repeated simulations and then we plotted (on the *y*-axis) performance on a metric against increasing failure rate (on the *x*-axis). The graphs also include a table that summarizes performance by averaging a metric across all failure rates; this summarization of the plotted curves gives a quick comparison of relative performance. An exception to this general approach to measurement occurs for the scenario related to restart after power failure, where there is no increasing failure rate. In this case, we simply provide the average and variance of the latency before a configuration is restored. In selected cases, we analyzed event traces to understand how differences in architecture, topology, and behavior contribute to differences in performance.

6. EXPERIMENTS AND RESULTS

In this section, we describe our scenarios and exhibit results. For each scenario, we describe the related experiment, delineate the failure model and recovery parameters, define the metrics, display the results and discuss underlying causes. We begin in Section 6.1 with the power-failure-and-restart scenario and then consider in Section 6.2 the distributed real-time control scenario impeded by node failures. Subsequently (in Section 6.3), we discuss the consistency maintenance scenario impeded by communication failures of two types: interface failures and message losses.

6.1 Recovery After Power Failure

In this experiment, a distributed system establishes an initial configuration in which pairs of SUs and SMs rendezvous, so that each SU obtains one required service. Subsequently, a power failure causes all nodes to crash. Upon power restoration, each SU attempts to rediscover the previously acquired service. This experiment measures the latency until the initial configuration is restored.

6.1.1 Experiment Description. This experiment compares several system designs: a two-party model (based on UPnP), a three-party model (based on Jini), and an adaptive model (based on SLP). In the two-party case, the topology (recall Figure 1) consists of six nodes: three SUs and three SMs. We partition the nodes into three SU-SM pairs that attempt to rendezvous. In the three-party cases (Jini and SLP), the topology (recall Figure 2) adds three SCMs for a total of nine nodes; however, we use logical partitioning (Jini groups and SLP scopes) so the each SU-SM pair must discover each other through a different SCM; so that a previously discovered configuration may not be rediscovered until all nodes have restarted. We allow all SU-SM pairs to rendezvous, which establishes an initial configuration, and then we simulate a power failure lasting 40 s. We restore power and wait for SUs to rendezvous with the previously discovered SMs. Once the initial configuration is restored the scenario ends.

Each model includes parameters set to the values indicated in Table 3. The first three rows in Table 3 show parameters unique to specific discovery systems. These parameters include the pattern for aggressive-discovery probes and the interval for lazy-discovery announcements. Jini and UPnP allow SUs to register for notifications; we assume such registrations are lost on node failure. SLP does not allow notifications and thus requires SUs to poll SCMs to discover services. We instantiated the adaptive architecture with two different polling intervals: 31 s as recommended for SLP and 5 s in order to gain early acquisition of services. The fourth row of Table 3 shows parameters for which we selected common values across all models. In particular, note that each node has a restart delay, which in most cases is not defined in discovery specifications. Since the specification for Jini recommends a random delay distributed uniformly between 2 s and 15 s before commencing discovery operations, we decided to assign this same strategy to all of our models in order to eliminate this as a source of difference. The final row of Table 3 lists common transmission and processing delays that we used for each model.

Table 3. Parameters For Power Failure and Restart Experiment.

Parameter Class	Parameter	Value
UPnP Protocol Parameters	Probe Pattern	None used in experiment
	Announce Interval	1800 s
	Notification Requests	Purge on SM Failure
	Polling Interval	Not Applicable
Jini Protocol Parameters	Probe Pattern	7 Probes 5 s apart
	Announce Interval	120 s
	Notification Requests	Purge on SCM Failure
	Polling Interval	Not Applicable
SLP Protocol Parameters	Probe Pattern	4 Probes in 15 s
	Announce Interval	120 s
	Notification Requests	Not Applicable
	Polling Interval	5 s or 31 s
Common Protocol Parameters	Registration TTL	30 s
	Total Registration Duration	100 s
	Node Restart Delay	2 s – 15 s uniform
Delays Used in For All Models	Transmission Delay	1 us – 10 us uniform
	Processing Load Delay	10 us – 100 us uniform
	Per Item Processing	10 us (discovery DBs) 100 us (SCM cache)

6.1.2 Metrics. We defined two metrics to compare system performance: *restoration latency* and *efficiency*. Restoration latency measures the elapsed time from restoration of power until the initial configuration is reestablished. Since restoration latency depends upon the starting time of the last system component, we defined *restart delay* to measure the elapsed time from restoration of power until the final system component restarts. We defined efficiency as the total number of messages during restoration latency.

Table 4. Results For Power Failure and Restart Experiment.

Model Variant	Restart Delay (seconds)		Restoration Latency (seconds)		Efficiency (number of messages)	
	Mean	Variance	Mean	Variance	Minimum	Maximum
Two-Party	13.07	2.97	15.04	2.97	49	67
Three-Party	12.56	2.09	14.76	3.31	70	90
Adaptive (5 s polling interval)	13.13	1.57	16.2	4.25	55	77
Adaptive (31 s polling interval)	13.23	1.22	34.68	65.13	57	100

6.1.3 Results. Table 4 presents results, measured over 30 repetitions, for four different configurations. The metrics reveal that for most configurations, restart delay is the dominant component of restoration latency; the previous configuration is restored within about 2 s after all nodes have restarted. An exception arises when we configure the adaptive architecture with a 31 s polling interval. Here, the polling interval is the dominant component of restoration latency. This occurs in cases where a related SCM and SU both restart before the SM. Here the SU discovers and queries the SCM for services before the SM can find the SCM and register its service. In this situation, the SU must wait for the 31 s polling interval to elapse for issuing a second, successful query. Reducing the polling interval to 5 s brings restoration latency closer to that exhibited by the other architectures.

Regarding efficiency, Table 4 shows that architectures with more components exchange more messages during a restoration scenario, but those architectures with the same number of components tend to exchange more messages when the scenario takes longer to complete. The three-party architecture proves slightly less efficient than the adaptive architecture because Jini incurs messages related to registration, which SLP does not support.

One final point to note is the slightly better restoration latency of the three-party, as compared with two-party, architecture. This occurs because Jini delivers a service description in one step, concomitant with discovery, while UPnP requires a three-step process: discover the service, get the first part of the service description, and then get the second part of the service description. Should transmission delays increase, this factor would cause even greater difference in restoration latency.

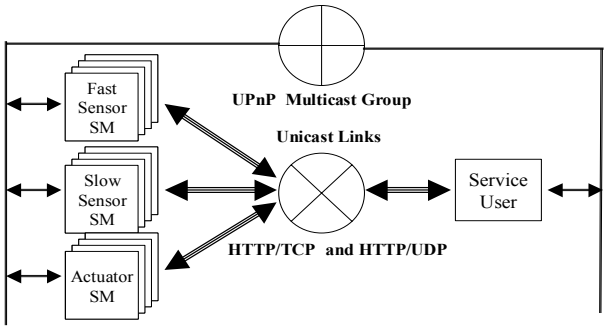


Figure 3. Two-party service discovery system with one service user and 12 service managers.

6.2 Service Acquisition and Maintenance Impeded by Node Failures

In this experiment, we investigate effectiveness and efficiency of service discovery systems in detecting component failure and locating replacements. We model a client for a distributed real-time control application that must discover two types of sensor and an actuator, then monitor sensor readings and control a process. The client has access to a population of sensors and actuators, each running on separate nodes that we allow to fail. The client, sensors, and actuators are supported by a discovery system, represented by configurations of the three architectural variants in our models: two-party (UPnP), three-party (Jini), and adaptive (SLP). Where applicable, the experiment topology may include one or more SCMs, which we also allow to fail. We compare configurations using *functional effectiveness*, measured as the proportion of time that the client possesses an operational set of sensors and actuators required to control the process. We also compare efficiency among configurations by the number of messages exchanged.

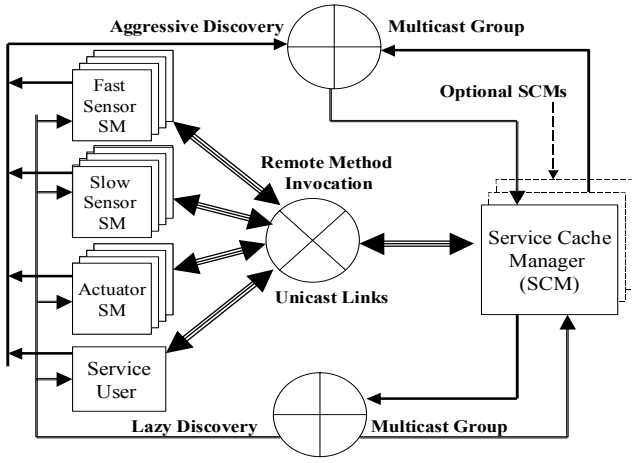


Figure 4. Three-party service discovery system with one service user, 12 service managers, and up to three service cache managers.

6.2.1 Experiment Description. Our experiment models a topology that includes one (client) SU and 12 SMs, composed of four instances each of three service types: "fast" sensor, "slow" sensor, and actuator. Figure 3 illustrates such a topology configured as a two-party architecture and Figure 4 shows the same topology configured as a three-party architecture (including one to three SCMs). We compare the performance of eight different configurations, enumerated in Table 5. Here, one configuration (**A0**) uses a two-party (UPnP) architecture and one (**C0**) uses an adaptive (SLP) architecture limited to two-party mode, three configurations (**B1**, **B2**, and **B3**) use a three-party (Jini) architecture, and three configurations (**C1**, **C2**, and **C3**) use an adaptive, three-party (SLP) architecture.

Table 5. Eight Configurations Compared in Node-Failure Experiment.

Configuration	Architecture	Behavior	SCMs
A0	Two-Party	UPnP	None
B1	Three-Party	Jini	One
B2			Two
B3			Three
C0	Adaptive	SLP	None
C1			One
C2			Two
C3			Three

To establish initial conditions, we exercise each configuration until discovery completes and the SU acquires one service of each of the three service types. We then fail nodes according to the failure model described below. In order to focus exclusively on failure detection and recovery processes, we do not allow the SU to cache backup services, so at any time the SU holds at most one SD for each service type. After activation, a "fast" sensor transmits a reading every two seconds and a "slow" sensor transmits a reading every 30 seconds. The SU invokes the actuator after receiving an appropriate combination of readings from a "fast" and "slow" sensor. We select actuation times randomly from a uniform distribution with a mean of 60 s, provided the SU receives the required sensor readings. When the SU holds one SD for a service of each type ("fast" sensor, "slow" sensor, and actuator) and when each of those services is operational, then the application is considered *functional*. If the SU lacks SDs for one or more service type or if one or more of the SDs held by the SU describes a service instance that is not operational, then the application is considered *non-functional*. When non-functional, the SU client must first detect what services have failed and then initiate recovery procedures to discover replacements. During each experiment repetition, we accumulate the periods when the client is non-functional as well as the time required for failure detection and recovery. We also record message counts of the underlying service discovery system for the experiment duration.

6.2.2 Failure Model. During the experiment duration T_D, each SM node (and SCM node in three-party configurations) fails randomly and independently, although at least one service of each type always remains active so that the application could become functional. We let λ be the node failure rate that varies from 0% to 80% in 10% increments (though no failures occur when $\lambda = 0$). The mean time to node failure is $t_{MF} = (1 - \lambda) \cdot T_D$. Node failure times are randomly chosen from a "stepped" normal distribution with three steps: a 0.15 probability of failure before $t_{MF} - 0.2t_{MF}$, a 0.7 probability of failure between $t_{MF} - 0.2t_{MF}$ and $t_{MF} + 0.2t_{MF}$, and a 0.15 probability of failure between $t_{MF} + 0.2t_{MF}$ and $2t_{MF}$. Failure times are distributed uniformly within each step. When a node fails, affected services become unavailable for a time, selected from three failure duration classes, each with a different probability and duration. Short failures occur with a probability of 0.1 for a fixed (135 s) duration; intermediate failures occur with a probability of 0.7 for a duration selected uniformly on the interval $[180, 300]$ s, long failures occur with a probability of 0.2 selected uniformly on the interval $[480, 600]$ s.

6.2.3 Failure Recovery Techniques. Table 6 gives common and configuration-specific parameters for failure recovery techniques we used in this experiment. We chose parameters that enable the SU to respond quickly to failure of remote services and to find replacements as soon as possible. We describe the recovery techniques employed in our model: first at the discovery level and then at the application level.

Discovery-Level Recovery. For the two-party (UPnP) architecture, we use a heartbeat and soft-state strategy, choosing a TTL of 600 s for refreshing cached SDs. If not refreshed within the TTL, the SU purges the SD and commences periodic (120 s) `Msearch` queries to find a replacement service. When we model SLP in two-party mode, the SU both listens for lazy announcements (120 s) from SMs and periodically issues multicast queries for SMs (900 s) to find replacements. In three-party configurations (both Jini and SLP), we model heartbeat monitoring through registration refreshes, choosing a refresh interval of 30 s for slow sensors and actuators and 300 s for fast sensors. If refreshes are missed, the SCM purges the SD. In the

Table 6. Recovery Parameters for Node-Failure Experiment.

	Configuration	Parameter	Value
Discovery-Level Recovery	Behavior for two-party UPnP configuration **A0**	Announce interval	600 s (lowered from recommended value)
		SU purges SD	At TTL expiration (600 s)
		Msearch query interval	120 s
	Behavior for two-party SLP configuration **B0**	Multicast query interval	120 s
	Behavior for three-party Jini and SLP configurations **B1**, **B2**, **B3**, **C1**, **C2**, **C3**	Refresh interval	30 s for slow sensors and actuators 300 s for fast sensors
		SCM purges SD	Immediately after a missed refresh
		SU-SCM query interval	180 s
		SU purges SD	Immediately after learning SD is unavailable
Application-Level Recovery	All configurations	Sensor interval	2 s for fast sensors 30 s for slow sensors
		SU purges SD	Immediately after missed sensor reading and after failing to receive an actuation response within 20 s
		SM or SU purges SCM	20 s after failure to receive response to request

three-party architecture, a SU that discovers a SD through a SCM polls that SCM every 180 s to learn if the SD has been purged; if so, the SU assumes failure of the related service and also purges the SD. In both three-party and adaptive architectures, SUs and SMs search for SCMs by listening for lazy announcements (120 s).

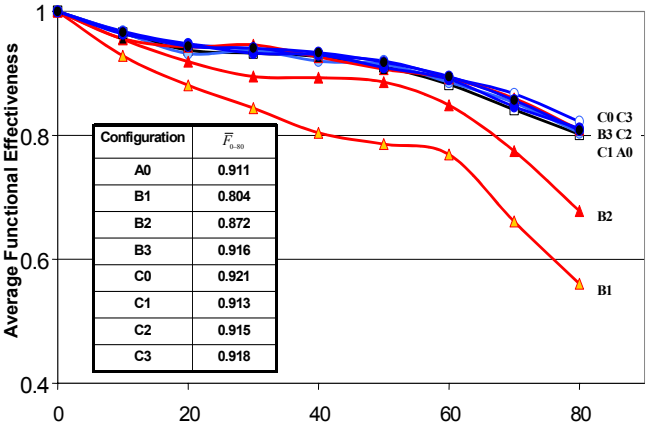
Application-Level Recovery. Across all models, we adopt an identical application-level recovery policy: upon failure to receive a scheduled sensor reading (every 2 s for fast sensors and 30 s for slow sensors) the SU immediately purges the related SD and commences search for a replacement. Similarly, failure to receive a response to an actuation attempt within 20 s causes the SU to purge the related SD and to commence search. A similar policy applies to detecting failed SCMs. If a SM does not receive a response when attempting to refresh a service registration, the SM assumes that the SCM has failed and begins searching for a replacement. Similarly, if a SU does not receive a response to a SCM query, the SU purges the SCM and begins to search.

6.2.4 Metrics. We define T_{NF} as accumulated time during which a client application is in a non-functional state. We compute the proportion of T_D that a client application is in a functional state, or the client's functional effectiveness, by the ratio $F = \left(T_D - T_{NF}\right)/T_D$. We compute the average functional effectiveness of a configuration at a particular failure rate λ for n experiment repetitions as

$$\overline{F}_\lambda = \frac{\sum_{i=1}^{n}\left(\left\langle T_D\right\rangle_i - \left\langle T_{NF}\right\rangle_i / \left\langle T_D\right\rangle_i\right)}{n}$$

We measure T_{NF} as follows. As indicated, a client that has become non-functional first incurs a delay before detecting the failure. We call this delay *detection latency*. After detecting a non-functional state, the client may incur some delay while restoring required services. We call this delay *recovery latency*. Detection latency commences when a SM fails but the SU holds a SD provided by the SM. Once the SU discards the SD, or the SM recovers, detection latency ends. Recovery latency begins after the SU purges a SD for a failed service and commences search. Recovery latency ends when the SU finds a SD matching its needs. During periods when a client incurs either detection or recovery latency or both (the states can overlap), the client is non-functional, and we accumulate such periods in T_{NF}.

6.2.5 Results. For each of the eight configurations in Table 5, we set $T_D = 1800$ **s** and executed 60 repetitions for each failure rate λ. Figure 5 shows average functional effectiveness $\overline{F}_\lambda$ for each configuration as λ increases. Figure 5 also includes a table that shows the summary statistic $\overline{F}_{0-80}$, which is $\overline{F}_\lambda$ averaged across all values of λ for each configuration. The results show that six of the eight configurations have similar curves for $\overline{F}_\lambda$ and a $\overline{F}_{0-80}$ of over 0.9. The three-party configuration with one SCM (**B1**) and two SCMs (**B2**) perform less well, because as λ rises, the incidence of failure of the single SCM in **B1** and concurrent failure of both SCMs in **B2** increases. With no SCM to query for services, the SU remains non-functional. Adding a third SCM (**B3**) reduces the probability of concurrent SCM failure sufficiently to raise $\overline{F}_{0-80}$ to a level comparable with other configurations. The adaptive architecture achieves a comparable $\overline{F}_{0-80}$ even with two or fewer SCMs, because when no SCMs can be found, the SU immediately switches to two-party mode to discover the available SMs. In the discussion below, we provide more detail on the effectiveness of these configurations by considering their comparative detection and recovery latencies.



Configuration	$\overline{F}_{0-80}$
A0	0.911
B1	0.804
B2	0.872
B3	0.916
C0	0.921
C1	0.913
C2	0.915
C3	0.918

Figure 5. Comparing average functional effectiveness $\overline{F}_\lambda$ for different configurations in response to increasing rate of node failures, where at least one SM of each type is operational (60 repetitions per data point). The table gives the $\overline{F}_{0-80}$, or functional effectiveness averaged across all values of λ for each configuration.

As revealed in Figure 6, efficiency varies markedly among the configurations. The two-party configurations **A0** and **C0** are notably more efficient than any three-party configuration. This occurs in part because more messages are needed for SUs and SMs to rendezvous through SCMs. These messages include heartbeats by the SCMs, registration and refresh of SDs by SMs, and polls of SCMs by the SU. In the three-party and adaptive architectures, differences in protocol also influenced efficiency. For equivalent configurations, the three-party architecture (**B1**, **B2**, and **B3**) proves more efficient than the adaptive architecture (**C1**, **C2**, and **C3**). This occurs, because in the former, Jini SCMs send lazy announcements at 120 s intervals, while Jini SUs and SMs employ aggressive search only at start-up. However, in the adaptive architecture, both SLP SCMs *and* SMs announce every 120 s, while SUs and SMs repeat a six-probe aggressive search sequence at regular intervals (900 s). We believe that with equivalent underlying behaviors, adaptive and three-party architectures would exhibit similar efficiency when configured with an equal number of SCMs.

One additional point is worth noting. In the two-party configurations (**A0** and **C0**), the message-count curves have increasing slope as λ increases, because the SU must search more frequently for replacement services. Note, however, that three-party configurations have message-count curves with decreasing slope as λ increases. The rate of message exchange decreases because SCMs fail more frequently and remain down for longer periods as λ rises, thus reducing the number of opportunities for SD refresh messages and SCM heartbeats.

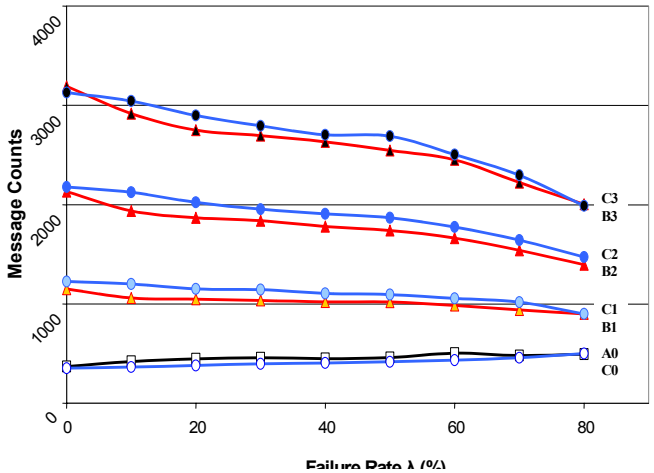
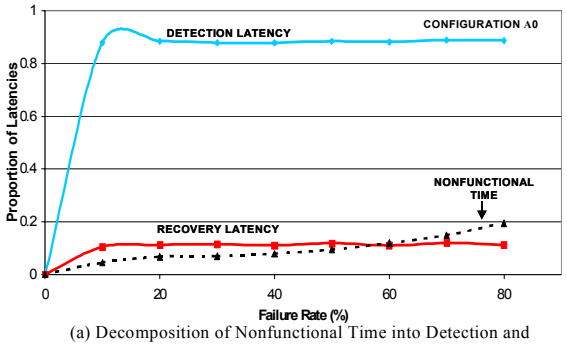


Figure 6. Comparing message counts for different configurations in response to increasing rate of node failures where at least one SM of each type is operational (60 repetitions per data point).

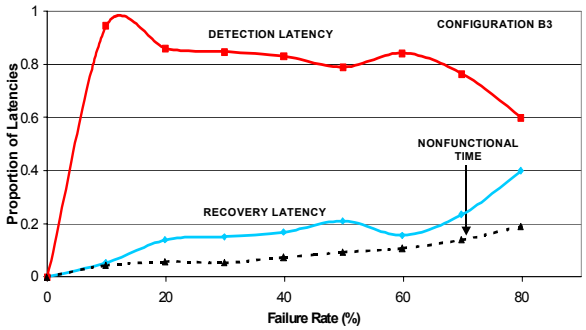
6.2.6 Discussion. While three-party configurations with three SCMs (**B3** and **C3**) yield comparable functional effectiveness to two-party configurations (**A0** and **C0**), our experiment reveals quite different underlying causes. Figures 7(a)-(c) display similar non-functional time (T_{NF}) under increasing failure rate for configurations **A0**, **B3**, and **C3**. The figures also decompose T_{NF} into the proportions attributable to detection latency and recovery latency. In the two-party configuration, reported in Figure 7(a), about 90 % of T_{NF} accrues while waiting to detect a failure; recovery occurs quickly. Analysis of execution traces showed most failures were detected through missed sensor readings or REXs received in response to failed actuations. In the three-party configuration, shown in Figure 7(b), the situation is different. Here, the largest component of T_{NF} is recovery latency. Execution traces for the three-party architecture show incidence of concurrent failure of all SCMs rising steadily with increasing λ. With no SCMs available, the SU is unable to find replacements for failed services until a SCM (1) recovers, (2) is discovered by the SU and SMs, (3) accepts registrations from available SMs, and (4) responds to queries from the SU. These factors dramatically increased the proportion of T_{NF} attributable to recovery latency. This trend is more marked with fewer SCMs (not shown here). In the adaptive configuration, as displayed in Figure 7(c), over 90 % of T_{NF} is again detection latency. Here, upon detecting failure, the SU switches to two-party mode when no SCMs can be found; thus, avoiding the delay incurred in waiting for a SCM to recover. Hence, the detection and recovery behavior of the adaptive configuration appears quite similar to the two-party configuration, which is also reflected in the similarity of Figures 7(a) and 7(c).

6.3 Consistency Maintenance Impeded by Communication Failures

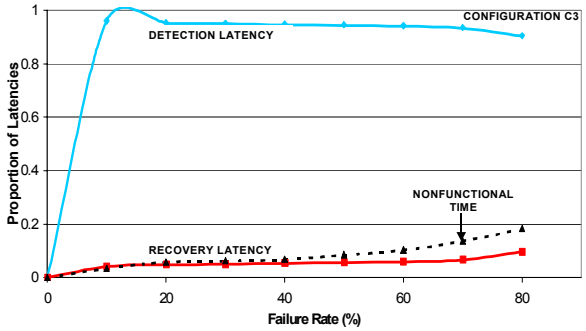
In this experiment, we investigate effectiveness and efficiency of service discovery systems in maintaining consistency of information replicated throughout a distributed system. We model five clients (SUs) that each discover the same service manager (SM) and obtain a copy of the service description (SD) managed by the SM. Subsequently, the SM updates its local copy of the SD, creating an inconsistency with the SDs replicated to the SUs. We measure the probability that each SD will receive an updated copy of the SD prior to a deadline, the latency incurred in receiving the updated SD, and the number of messages exchanged to convey the update. We consider effects from two types of communication failure, interface failures and message losses, which could impede dissemination of the updated SD. We also compare two alternate consistency maintenance mechanisms: polling (recall Section 3.3.1) and notification (recall Section 3.3.2), which are supported by selected discovery systems.



(a) Decomposition of Nonfunctional Time into Detection and Recovery Latency for Configuration **A0**.



(b) Decomposition of Nonfunctional Time into Detection and Recovery Latency for Configuration **B3**.



(c) Decomposition of Nonfunctional Time into Detection and Recovery Latency for Configuration **B3**.

Figure 7. Detection and recovery latencies of various configurations as a proportion of nonfunctional time (60 repetitions per data point).

6.3.1 Experiment Description. We compare performance of nine configurations, as enumerated in Table 7. One configuration (**A0p**) uses a two-party (UPnP) architecture (see Figure 8) with a polling regime to maintain consistency. Another configuration (**A0n**) combines the same architecture with notification. Four configurations (**B1p**, **B1n**, **B2p** and **B2n**) use a three-party (Jini) architecture (see Figure 9) with one or two SCMs and polling or notification. Three configurations (**C0p**, **C1p** and **C2p**) use an adaptive (SLP) architecture (with zero, one, or two SCMs) and polling (SLP does not include a notification mechanism).

Table 7. Nine Configurations Compared in Communication-Failure Experiments.

Configuration	Architecture	Behavior	Consistency-Maintenance Mechanism
A0p	Two-Party	UPnP	Polling
A0n	Two-Party	UPnP	Notification (with notification registration on SM)
B1p	Three-Party (One SCM)	Jini	Polling (with service registration on SCM)
B1n	Three-Party (One SCM)	Jini	Notification (with service registration and notification registration on SCM)
B2p	Three-Party (Two SCMs)	Jini	Polling (with service registration on SCM)
B2n	Three-Party (Two SCMs)	Jini	Notification (with service registration and notification registration on SCM)
C0p	Adaptive (no SCMs)	SLP	Polling
C1p	Adaptive (One SCM)	SLP	Polling (with service registration on SCM)
C2p	Adaptive (Two SCMs)	SLP	Polling (with service registration on SCM)

To establish initial conditions, we set aside an interval, up to time t_Q, for all SUs to discover the SM and obtain the SM's SD. We then activate interface failures or message loss according to the appropriate failure model described below. In addition, we establish a deadline t_D by which the change must propagate to all SUs, and then chose a time, randomly distributed on the uniform interval $\left[t_Q,\ t_D/2\right]$, to introduce a change in the SD on the SM. Here, we set $t_Q = 100$ s and $t_D = 5400$ s. Each experiment aims to restore consistency between the changed SD held by the SM and the cached copies of the SD held by the SUs. We recorded the time of change to the SD on the SM, the latency required to propagate the update to each SU prior to t_D (or failure to do so) and the number of messages exchanged.

6.3.2 Failure Models. We conducted separate experiments for interface failure and message loss. Table 8 summarizes relevant parameters for each failure model.

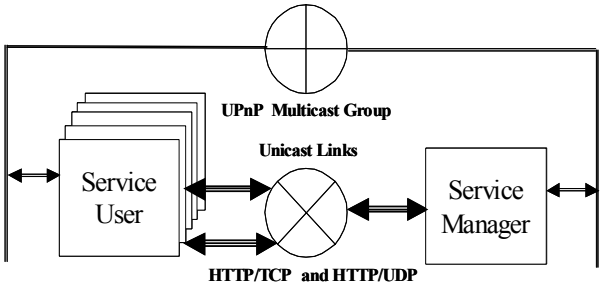


Figure 8. Two-party service discovery system deployed in a six-node topology: five service users and one service

Interface Failure. In the interface-failure experiment, we let λ be the interface failure rate. During the experiment, each node suffers an interface failure at a time, randomly distributed on the uniform interval $\left[t_Q,\ t_Q - \left(t_Q \cdot \lambda\right)\right]$. When activating each interface failure, there is an equal likelihood that the transmitter, receiver, or both fail. Once activated, each failure remains in effect for the duration of $t_D \cdot \lambda$, after that the failure is remedied. During a failure interval, no messages are sent from a node with a failed transmitter, and a node with a failed receiver does not receive messages. For each configuration simulated, we varied λ from 0 to 90 % in increments of 5 %.

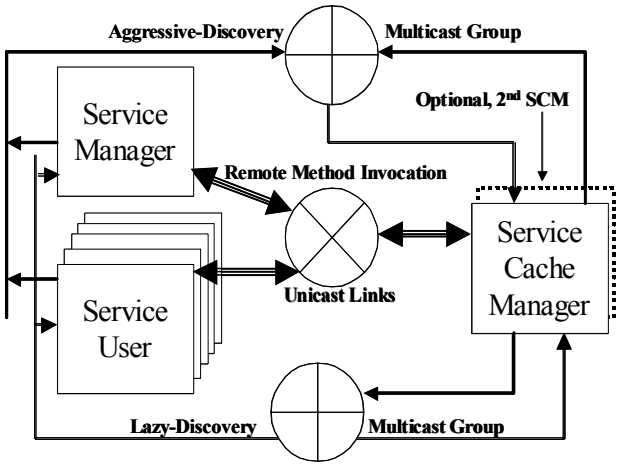


Figure 9. Three-party service discovery system deployed in a seven- or eight-node topology: five service users, a service manager, and one or two service cache managers.

Message Loss. In the message-loss experiment, we let λ be the message-loss rate. For each attempt to transmit a message, whether on a reliable or unreliable channel, a uniform random real number is selected from the unit interval $\left[0,1\right]$. If the number is less than λ, the message is discarded. Loss of a message sent on a reliable channel stimulates a retransmission after an appropriate timeout. We varied λ as in the interface-failure experiment.

Table 8. Parameters for Interface Failure and Message Loss Models.

Failure	Parameter	Value
Interface Failure	Failure incidence	Once per run for each node
	Failure scope	Transmitter, receiver, or both with equal likelihood
	Failure duration	5% increments of 5400 s from 0 to 90%
Message Loss	Failure incidence	Each transmission may fail with probability equal to message loss rate from 0 to 90%.
	Failure scope	Individual message transmission
	Failure duration	Individual message transmission

6.3.3 Failure Recovery Techniques. We model recovery techniques at three levels: transport protocols, discovery protocols, and application. Recovery techniques for the transport protocols are described in Section 4.2.1. Table 9 shows the recovery techniques and related parameters we adopted for the discovery and application levels.

Discovery-Level Recovery. In the two-party (UPnP) architecture, we use a heartbeat and soft-state strategy where SUs discarded SDs not refreshed within a TTL (of 1800 s). To enable rediscovery of SMs (and SCMs, where applicable) we adopt a discovery behavior consistent with the specific protocol (UPnP, Jini, or SLP) being modeled. In all configurations (except **A0p**, which does not employ registration), we chose the same TTL (of 1800 s) after which registrations would be discarded if not renewed. For REXs received in response to registration or refresh attempts, to ad-hoc queries, or to change-service operations, the retries occur at intervals of 120 s (but only up to a maximum of 540 s). To comply with the Jini and UPnP specifications, there are no retries after a REX when attempting to issue notifications.

Table 9. Key Model Parameters for Communication-Failure Experiments.

	Configuration	Parameter	Value
Discovery-Level Recovery	**A0p** and **A0n** (UPnP)	Announce interval	1800 s
		Msearch query interval	120 s
		SU purges SD	At TTL expiration
	B1p, **B1n**, **B2p** and **B2n** (Jini)	Probe interval	5 s (7 times)
		Announce interval	120 s
	C0p, **C1p**, and **C2p** (SLP)	Probe interval	Variable (4 probes in 15 s)
		Announce interval	900 s
	A0n, **B1p**, **B1n**, **B2p**, **B2n**, **C1p** and **C2p**	Registration TTL	1800 s
	A0n, **B1p**, **B1n**, **B2p**, **B2n**, **C0p**, **C1p** and **C2p**	Time to retry after REX	120 s
Application-Level Recovery	**A0p**, **B1p**, **B2p**, **C0p**, **C1p** and **C2p**	Polling interval	180 s
	A0p, **A0n**, and **C0p**	SU purges SD	After 540 s with only REX
	B1p, **B1n**, **B2p**, **B2n**, **C1p** and **C2p**	SM or SU purges SCM	After 540 s with only REX

Application-Level Recovery. For configurations (**A0p**, **B1p**, **B2p**, **C0p**, **C1p**, and **C2p**) that use polling, we set the polling interval to 180 s. In (UPnP) configurations (**A0p** and **A0n**), SUs discard a SD after (HTTP `GET`) queries to the SM result in nothing but REXs for a total of 540 s. In other configurations, SUs discard a SCM after receiving nothing but REXs over 540 s while attempting to interact with the SCM.

6.3.4 Metrics. We evaluate update effectiveness, responsiveness, and efficiency. Update effectiveness measures the probability that a change to a SD will propagate to a given SU before the deadline t_D. We let n be the number of repetitions of an experiment, m be the number of SUs in a topology, and t'_{ij} be the time that an updated SD is propagated to SU j, $1 \le j \le m$, in experiment repetition i, $1 \le i \le n$. Then, we define update effectiveness for the failure rate λ over n repetitions as

$$U_\lambda = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} chg_{ij}}{n \cdot m}$$

where

$$chg_{ij} = \begin{cases} 1 & \textbf{if } t'_{ij} < t_D, \\ 0 & \textbf{otherwise} \end{cases}$$

defines whether a change in a SD was propagated to the jth SU during the ith repetition (i.e., 1 if true, 0 if false).

Update responsiveness measures the latency in propagating the SD update. We let t'_i be the time the SD change occurred on the SM in experiment repetition i. Update responsiveness $\tilde{R}_\lambda$ is the median of all $1 - p_{ij}$ at a particular value of λ where

$$p_{ij} = \frac{t'ij - t'_i}{t_D - t'_i}$$

is the proportion of time required to propagate an update to the jth SU in the ith repetition t'_i at λ.

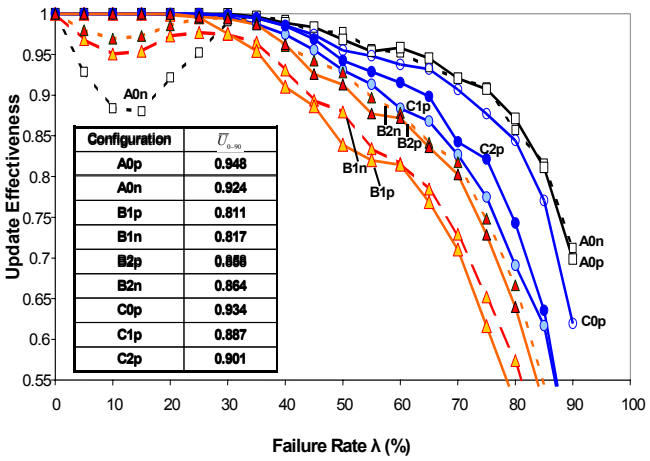
Update efficiency measures the effort required to (attempt to) maintain consistency. Analysis of our experiment configurations revealed a minimum number of messages, x, that must be sent to propagate a change to all SUs. This minimum ($x = 7$) occurred for the three-party configuration with notification and one SCM (**B1n**)[2]. We define update efficiency based on the ratio of x to the actual number of messages observed. We let y be the number of messages sent while attempting to propagate a change from the SM to the SUs in a given repetition. Then, for n number of experiment repetitions, we define average update efficiency at a particular failure rate λ as

$$\overline{E}_\lambda = \frac{\sum_{i=1}^{n}\left(x / y_i\right)}{n}.$$

6.3.5 Interface-Failure Results. For each configuration in Table 7, we executed $n = 1000$ repetitions at each interface-failure rate λ. Figure 10 shows update effectiveness U_λ for the configurations as λ increases. The figure also includes a table with mean update effectiveness $\overline{U}_{0-90}$, which is U_λ averaged across all values of λ for each indicated configuration. Overall, these results show that a two-party architecture, or an adaptive architecture that has a two-party mode, provides superior effectiveness to a three-party architecture (at least given topologies limited to one or two SCMs). This occurs because each updated SD must propagate over only one channel (SM to SU) in two-party cases, but over two channels (SM to SCM and SCM to SU) in three-party cases. For both three-party and adaptive architectures, $\overline{U}_{0-90}$ improves with the number of SCMs due to the reduction in the incidence of joint failure of both channels. We note that polling yields better effectiveness than notification. For example, when comparing three-party polling with one SCM (**B1p**) against three-party notification with one SCM (**B1n**), the advantage of polling appears as λ exceeds 35 % because when

[2] Recall that the two-party (UPnP) architecture requires a multiple-message exchange to convey SDs.

notifications fail, SD updates are propagated by recovery mechanisms, which activate only after some delay. On the other hand, polling persists with retries after receiving a REX. We note that configurations using notification also exhibit anomalous behavior when λ is in the range $[5, 25]$ %; we discuss the reasons for this below in Section 6.3.7.

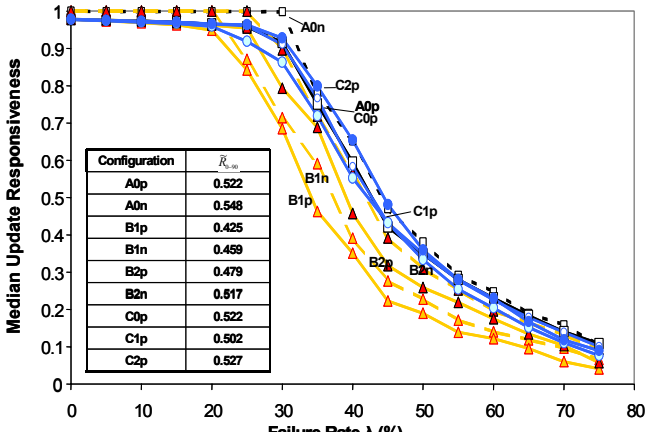


Configuration	$\overline{U}_{0-90}$
A0p	0.948
A0n	0.924
B1p	0.811
B1n	0.817
B2p	0.858
B2n	0.864
C0p	0.934
C1p	0.887
C2p	0.901

Figure 10. Comparing update effectiveness (U_λ) for different configurations in response to increasing rate of interface failures (1000 repetitions per data point). The table gives $\overline{U}_{0-90}$, or U_λ averaged across all values of λ for each configuration.

Figure 11 shows median update responsiveness $\widetilde{R}_\lambda$ for all configurations as λ increases. Generally, the ranking of architectures for responsiveness is similar to effectiveness. Where employed, notification exhibits better responsiveness than polling, which incurs increased latency from the 180 s polling interval. Figure 11 also shows a steep drop-off in $\widetilde{R}_\lambda$ for all configurations as λ increases beyond the $[20, 30]$ % range, where failures prevent initial propagation of the updated SD, forcing invocation of recovery actions that cannot succeed until paths are restored. Thus, even though some configurations achieved effectiveness of over 0.9 as λ reaches 70% (see Figure 10), responsiveness for all configurations approaches zero. Three-party configurations experience longer delays at high values of λ as paths to SCMs become increasingly unavailable.

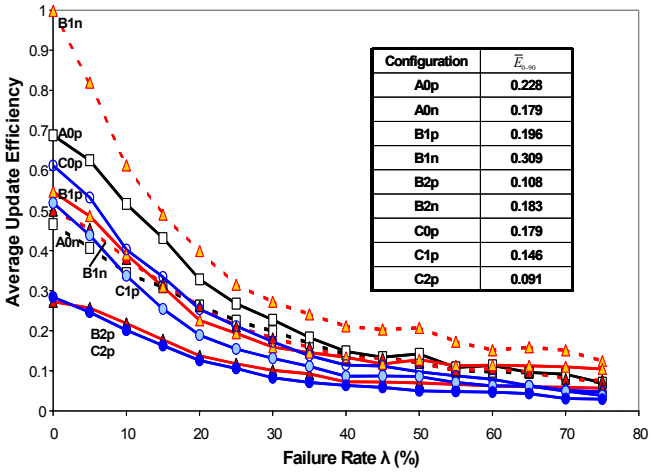
Figurre 12 shows average efficiency $\overline{E}_\lambda$ for experiment configurations as λ increases. The table included in Figure 12 shows $\overline{E}_{0-90}$, which is $\overline{E}_\lambda$ across all values of λ for each indicated configuration. Here, $\overline{E}_\lambda$ declines for all configurations as λ increases. This reflects a rising number of messages generated when recovery strategies are invoked more frequently as λ rises. Configurations using more SCMs are less efficient (but more effective) than similar configurations with fewer SCMs. The adaptive architecture appears less efficient than the three-party architecture with an equivalent number of SCMs for the reasons described above in section 6.2.5. Again, we expect the use of equivalent underlying behaviors would yield comparable efficiencies.



Configuration	$\widetilde{R}_{0-90}$
A0p	0.522
A0n	0.548
B1p	0.425
B1n	0.459
B2p	0.479
B2n	0.517
C0p	0.522
C1p	0.502
C2p	0.527

Figure 11. Comparing median update responsiveness ($\widetilde{R}_\lambda$) for different configurations in response to increasing rate of interface failures (1000 repetitions per data point). The table gives $\widetilde{R}_{0-90}$, which is $\widetilde{R}_\lambda$, averaged across all values of λ for each configuration.

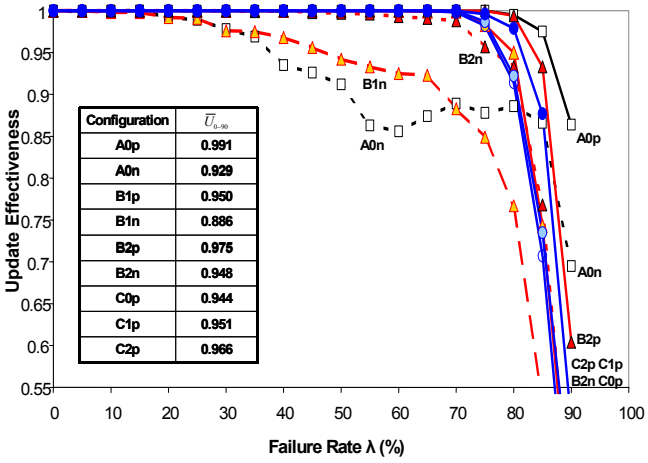
Some other points seem worth noting. The three-party configurations using notification (**B1n** and **B2n**) are more efficient than similar configurations using polling (**B1p** and **B2p**) because in Jini each SU poll to a SCM involves a request followed by a reply, while a Jini SCM notification is a single message. However, for $\lambda < 40$ %, two-party (UPnP) notification (**A0n**) appears less efficient than two-party polling (**A0p**). This occurs because when UPnP notifications are lost, recovery strategies must often be used, thus prolonging the time to propagate the updated SD and increasing message counts.



Configuration	$\overline{E}_{0-90}$
A0p	0.228
A0n	0.179
B1p	0.196
B1n	0.309
B2p	0.108
B2n	0.183
C0p	0.179
C1p	0.146
C2p	0.091

Figure 12. Comparing average update efficiency ($\overline{E}_\lambda$) for different configurations in response to increasing rate of interface failures (1000 repetitions per data point). The table gives $\overline{E}_{0-90}$, which is $\overline{E}_\lambda$ averaged across all values of λ for each configuration.

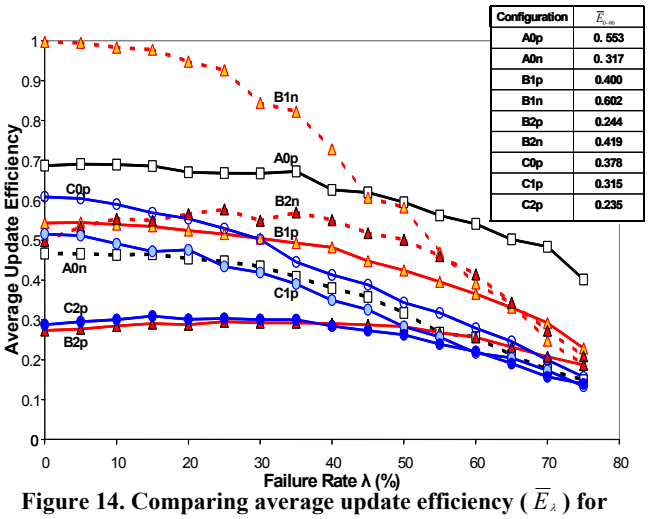
6.3.6 Message-Loss Results. For each configuration in Table 7, we executed $n = 200$ repetitions at each message-loss rate λ. Figure 13 shows update effectiveness U_λ for the configurations as λ increases. Figure 13 also includes a table that shows U_{0-90} across all values of λ for each indicated configuration. Overall, these results show that most configurations provide an effectiveness of 0.95 or better until λ exceeds 80 %. Overall, effectiveness under message loss conditions is higher than under interface failure conditions. This is because interfaces fail for protracted periods at higher values of λ, increasing the probability that channels remain blocked until t_D, so updates never get through. In contrast, message loss affects only individual transmissions, allowing recovery strategies more opportunities to propagate the update before t_D. Polling continues to yield better effectiveness than notification. The two-party configuration with polling (**A0p**) achieves a mean effectiveness of 0.99, due to the combined advantages of using polling with just two parties (which requires transiting one channel rather than two). We note that the two-party configuration with notification (**A0n**) and the three-party notification with one SCM (**B1n**) exhibit anomalous behavior and reduced effectiveness as λ surpasses 20 %; we discuss the reasons for this below in Section 6.3.7. Responsiveness (not shown here) exhibits a steep decline after $\lambda > 80$ %, compared with $\lambda > 30$ % for interface failure. The higher responsiveness under message loss conditions occurs for the same reasons as higher effectiveness. Under message loss, notification also continues to provide better responsiveness than polling.



Configuration	$\overline{U}_{0-90}$
A0p	0.991
A0n	0.929
B1p	0.950
B1n	0.886
B2p	0.975
B2n	0.948
C0p	0.944
C1p	0.951
C2p	0.966

Figure 13. Comparing update effectiveness (U_λ) for different configurations in response to increasing rate of message loss (200 repetitions per data point). The table gives $\overline{U}_{0-90}$, or U_λ averaged across all values of λ for each configuration.

Figure 14 shows average efficiency $\overline{E}_\lambda$ for experiment configurations as λ increases and includes a table for $\overline{E}_{0-90}$ for for each configuration. As in the case of effectiveness and responsiveness, all configurations prove more efficient under message loss conditions than under interface failure for the reasons given above. The better efficiency is also reflected in the overall more gradual decline in the message loss efficiency curves. Otherwise, the general ordering of efficiencies for the various configurations appears similar under both interface failure and message loss. We note the reduced efficiency of the two-party (UPnP) notification (**A0n**) above $\lambda = 20$ % in comparison with two-party polling (**A0p**). In **A0n**, efficiency suffers from cases where notifications are lost and recovery procedures are required to propagate the update (taking more time and requiring more messages). The combination of lost notifications and use of recovery also causes a sharp decline in the efficiency of the three-party notification with a single SCM (**B1n**), which at low values of λ, generates the fewest (7) messages to propagate updates. Another exception is the three-party configuration using notification with two SCMs **(B2n)**, which exhibits increasing efficiency over the failure rate range $[5, 35]$% and overtakes the three-party configuration using polling with one SCM (**B1p**). This counterintuitive result occurs because in some repetitions, lost messages cause the SM or SUs to discover only one of the two SCMs; thus, messages that would normally be duplicated to both SCMs are not.



Configuration	$\overline{E}_{0-90}$
A0p	0. 553
A0n	0. 317
B1p	0.400
B1n	0.602
B2p	0.244
B2n	0.419
C0p	0.378
C1p	0.315
C2p	0.235

Figure 14. Comparing average update efficiency ($\overline{E}_\lambda$) for different configurations in esponse to increasing rate of message loss (200 repetitions per data point). The table gives $\overline{E}_{0-90}$, which is $\overline{E}_\lambda$ averaged across all values of λ for each configuration.

6.3.7 Discussion. The notification mechanism included in UPnP and Jini (and other distributed systems) proved unexpectedly ineffective at disseminating updates under certain conditions. Foremost, under low interface-failure rates (in the range $[5, 30]$%) our results exhibit saw-tooth phenomena for configurations using notification. The dip is most pronounced (nearly 15%) for the two-party (UPnP) configuration (**A0n**) and less pronounced (around 5%) for the three-party (Jini) configurations (**B1n** and **B2n**). In the two-party case, analysis of execution traces showed a large number of notifications were lost when either the SM transmitter was inoperable (causing notifications to all SUs to be lost) or when SU receivers were inoperable (causing lost notifications to individual SUs). Since neither UPnP nor Jini require notification senders to retry after a REX, updated information must be disseminated through a

recovery mechanism. At low failure rates, a notification can be lost to an interface failure, which is repaired prior to the next announcement or registration-refresh attempt. Under such conditions, recovery mechanisms are not invoked and the SU does not obtain an updated SD. Polling proves more effective because the SU checks periodically (180 s intervals) and persistently for updated information and retrieves the SD when indicated.

A similar sequence of events occurs in the three-party case, but the effects are more modest. The three-party configurations require a SM to first propagate a change to a SCM. Failure to propagate a change results in a REX that causes the SM to retry the change for up to 540 s, during which time the interface failure may be repaired. If still unconfirmed after 540 s, the SM purges the SCM and initiates aggressive discovery. After rediscovering the SCM, the SM propagates the change, and the SCM then notifies registered SUs. Even with this redundancy, there still is some chance that a SU receiver is blocked and thus unable to receive notification. The redundancy does, however, increase the probability that an updated SD reaches a SU.

Notification (as specified for UPnP and Jini) also appears less effective under message loss. Lack of application-level retries to deliver notices leads to significant decline in update effectiveness above $\lambda = 20\,\%$. This appears for the relevant two-party (UPnP) configuration (**A0n**) and three-party (Jini) configuration (**B1n**), both of which use notification. Above $\lambda = 20\,\%$, the incidence of undelivered notifications increases and, unless recovery is stimulated, the updated SD is not disseminated. In configuration **A0n**, as λ exceeds 60 %, lost registration-refresh requests trigger recovery procedures with increasing frequency, which causes propagation of the updated SD when a registration is reestablished. This process slightly improves and then maintains effectiveness within the failure rate range $\left[60,\ 80\right]\%$, causing this curve to echo the saw-tooth feature in the update effectiveness curve for **A0n** under interface failure. Above $\lambda = 80\,\%$, lost messages effectively close the channel, and effectiveness collapses for all configurations.

For the three-party configuration (**B1n**), loss of change requests (from the SM) as well as registration refreshes (from the SM and SUs) also stimulate recovery procedures that partly compensate for lost notifications. When a second SCM is added (configuration **B2n**) update effectiveness improves because the SM now has two paths through which to disseminate updates to SUs.

7. CONCLUSIONS

Overall, we found designs for first-generation discovery systems can be robust under difficult failure environments. Across all experiments, most configurations exhibited an effectiveness of better than 0.9 in obtaining services or propagating updates for failure rates approaching (often exceeding) 80 %. Configurations proved ineffective only when all essential nodes failed or were unreachable, or when recovery actions were not activated (as occurred in response to lost update notifications). Similarly, extensive delays in propagating updates depended on the duration of path outages.

For our scenarios and metrics, two-party configurations (or three-party configurations that could adapt to two-party mode) appeared more robust than three-party configurations (where robustness improved with the number of replicated directories). Deploying three directory replicas yielded robustness equal to two-party configurations. In tradeoff, increasing the number of directory replicas lowers system efficiency by increasing the number of messages exchanged. In most cases, we found the adaptive architecture with one directory achieved robustness comparable to other configurations, while providing better efficiency than configurations with replicated directories.

To disseminate updates, we found polling more effective than notification. Our polling regime used persistent retries, while our notification regime depended only on reliable transport protocols, falling back to alternate recovery mechanisms when notifications could not be delivered. The alternate recovery mechanisms were not always activated at lower failure rates. This anomaly appeared in effectiveness plots for configurations using notification. Notification generally conveyed updates with less delay than polling. In the two-party architecture, polling was more effective, so scenarios tended to end earlier and require fewer messages.

Beyond our methodology and comparisons, we identified and discussed the most significant design and configuration decisions that influence robustness and efficiency in first-generation discovery systems. We showed how available architectural alternatives, as well as choices for consistency maintenance and recovery strategies, lead to robustness-efficiency tradeoffs. We also showed how faulty assumptions regarding recovery strategies could unexpectedly degrade robustness and efficiency. The information provided should convey a better understanding of failure behavior in existing discovery systems, allowing potential users to configure deployments for high robustness at low cost. The discussions presented here could also help to improve designs for future discovery systems.

8. ACKNOWLEDGMENTS

We received generous funding support from Susan Zevin, as acting director of the NIST Information Technology Laboratory, Douglas Maughan, as manager of the Defense Advanced Research Projects Agency (DARPA) Fault-Tolerant Networks Program, John Salasin, as manager of the DARPA program in Dynamic Assembly for System Adaptability, Dependability and Assurance, and James Puffenbarger of the Advanced Research and Development Activity (ARDA).

9. REFERENCES

[1] Salutation Architecture Specification, Version 2.0c, Salutation Consortium, June 1999.

[2] K. Arnold, et al, The Jini Specification, Version 1.0, Addison-Wesley, 1999.

[3] Universal Plug and Play Device Architecture (UPnP), Version 1.0, Microsoft, Inc., 2000.

[4] E. Guttman, C. Perkins, J. Veizades, and M. Day, Service Location Protocol, Volume 2, Internet Engineering Task Force (IETF), RFC 2608, 1999.

[5] Specification of the Home Audio/Video Interoperability (HAVi) Architecture, Version 1.1, HAVi, Inc., 2001.

[6] Specification of the Bluetooth System, Core, Version 1.1, Volume 1, the Bluetooth SIG, Inc., 2001.

[7] C. Dabrowski, K. Mills, and S. Quirolgico, A Model-based Analysis of First-Generation Service Discovery Systems, Special Publication 500-260, National Institute of Standards and Technology, 2005.

[8] C. Dabrowski and K. Mills, "Analyzing Properties and Behavior of Service Discovery Protocols Using an Architecture-Based Approach," Proceedings of Working Conference on Complex and Dynamic Systems Architecture, Brisbane, Australia, December 2001.

[9] C. Dabrowski, K. Mills, and J. Elder, "Understanding Consistency Maintenance in Service Discovery Architectures During Communications Failure," Proceedings of the 3rd International Workshop on Software Performance, Rome, Italy, July 2002, pp. 168-178.

[10] C. Dabrowski, K. Mills, and J. Elder, "Understanding Consistency Maintenance in Service Discovery Architectures In Response to Message Loss," Proceedings of the 4th International Workshop on Active Middleware Services, Edinburgh, United Kingdom, July 2002, pp. 51-60.

[11] C. Dabrowski, K. Mills, and A. Rukhin, "Performance of Service-Discovery Architectures in Response to Node Failure," Proceedings of the International Conference on Software Engineering Research and Practice, Las Vegas, NV, June 2003, pp. 95-104.

[12] C. Bettstetter and C. Renner, "A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol," Proceedings of the Sixth EUNICE Open European Summer School: Innovative Internet Applications, Open EUNICE 2000, Twente, Netherlands, September 2000.

[13] G. Richard, "Service Advertisement and Discovery: Enabling Universal Device Cooperation," *IEEE Internet Computing*, Volume 4, Number 5, pp. 18-26, 2000.

[14] J. Rekesh, UPnP, Jini and Salutation - A look at some popular coordination framework for future network devices, Technical Report, California Software Lab, 1999.

[15] R. Pascoe, "Salutation Architectures and the newly defined service discovery protocols from Microsoft and Sun: How does the Salutation Architecture stack up," Salutation Consortium white paper, 1999.

[16] R. Pascoe, "Building Networks on the Fly," IEEE Spectrum, Volume 38, Issue 3, pp. 61-65, 2001.

[17] G. Richard, Service and Device Discovery: Protocols and Programming, McGraw-Hill, 2002.

[18] G. O'Driscoll, Essential Guide to Home Networking Technologies, Prentice-Hall Trading Company, 2000.

[19] B. Olivier, "Jini: a platform for building adaptive integrated learning environments," Report from the Centre for Learning Technology (CeLT), University of Wales Bangor, United Kingdom, December 2000.

[20] D. Bushmitch, W. Lin, A. Bieszczad, A. Kaplan, V. Papageorgiou, and A. Pakstas, "A SIP-Based Device Communication Service for OSGi Framework," Proceedings of the 2004 IEEE Consumer Communications And Networking Conference, Las Vegas, NV, January 2004, pp. 453-458.

[21] B. Miller and R. Pascoe, "Mapping Salutation Architecture APIs to Bluetooth Service Discovery Layer," Version 1.0, Bluetooth SIG white paper, July 1999.

[22] J. Allard, V. Chinta, S. Gundala, and G. Richard, "Jini Meets UPnP: An Architecture for Jini/UPnP Interoperability," Proceedings of the 2003 International Symposium on Applications and the Internet (SAINT 2003), Orlando, FL, January 2003, pp. 268-275.

[23] A. Sameh and R. El-Kharboutly, "Modeling Jini-UPnP Bridge using Rapide ADL," Proceedings of the IEEE/ACS International Conference on Pervasive Services (ICPS'04), Beirut, Lebanon, July 2004, p. 237.

[24] E. Guttman and J. Kempf, "Automatic Discovery of Thin Servers: SLP, Jini and the SLP-Jini Bridge," Proceedings of the 25th Annual Conference of the IEEE Industrial Electronics Society (IECON 99), Volume 2, San Jose, CA, December 1999, pp. 722-727.

[25] S. Ponnekanti and A. Fox, "Application-Service Interoperation without Standardized Service Interfaces," Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom 2003), Fort Worth, TX, March 2003, pp. 30-39.

[26] M. Yu, A. Taleb-Bendiab, D. Reilly, and W. Omar, "Multi-Standard Service Interoperation Protocol through Polyarchical Middleware," Proceedings of the PostGraduate Networking Conference (PGNet), Liverpool, United Kingdom, June 2003, pp.143-148.

[27] S. Czerwinski, et al, "An Architecture for a Secure Service Discovery Service," Proceedings of the Fifth Annual International Conference on Mobile Computing and Networks (MobiCom '99), Seattle, WA, August 1999, pp. 24-35.

[28] S. Frolund, et al., "Building Dependable Internet Services with E-speak," Hewlett Packard Laboratories Technical Report HPL-2000-78, 2000.

[29] JXTA v2.0 Protocols Specification, Sun Microsystems, 2004, http://spec.jxta.org/v1.0/docbook/JXTAProtocols.html.

[30] M. Castro, et al., "One Ring to Rule them All: Service Discovery and Binding in Structured Peer-to-Peer Overlay Networks," The Proceedings of the Tenth ACM SIGOPS European Workshop, ACM, Saint-Émilion, France, September 2002.

[31] D. Verma, et al., "SRIRAM: A scalable resilient autonomic mesh," *IBM SYSTEMS JOURNAL*, Volume 42, Number 1, pp. 19-28, 2003.

[32] H. Hsiao and C. King, "Neuron – A Wide-Area Service Discovery Infrastructure," Proceedings of the International Conference on Parallel Processing (ICPP '02), Vancouver, British Columbia, August 2002, p. 455.

[33] A. Iamnitchi and I. Foster, "On Fully Decentralized Resource Discovery in Grid Environments," Proceedings of an IEEE International workshop on Grid computing, Denver, CO, November 2001.

[34] S. Joseph, "NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks," Proceedings of the International Workshop on Peer-to-Peer Computing, Pisa, Italy, May 2002.

[35] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing," Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10), San Francisco, CA, August 2001, pp. 181-194.

[36] UDDI Technical White Paper, published by the members of *uddi.org*, September 2000.

[37] V. Sundramoorthy, M. Speelziek, G. van de Glind, and J. Scholten, "Service Discovery with FRODO," 12th IEEE International Conference on Network Protocols (ICNP), Berlin, Germany, October 2004, pp. 24-27.

[38] K. Bowers, K. Mills, and S. Rose, "Self-adaptive Leasing for Jini," Proceedings of the IEEE International Conference on

Pervasive Computing and Communications (PerCom 2003), Fort Worth, TX, March 2003, pp. 539-542.

[39] K. Mills and C. Dabrowski, "Adaptive Jitter Control for UPnP M-Search," Proceedings of 2003 IEEE International Communications Conference, Anchorage, AK, May 2003.

[40] S. Rose, K. Bowers, S. Quirolgico, and K. Mills, "Improving Failure Responsiveness in Jini Leasing," Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX-III 2003), Volume 2, Washington, DC, April 2003, pp. 103-105.

[41] K. Mills, S. Rose, S. Quirolgico, M. Britton, and C. Tan, "An Autonomic Failure-Detection Algorithm," Proceedings of the 4th International Workshop on Software Performance (WoSP 2004), San Francisco, CA, January 2004, p. 79.

[42] C. Tan and K. Mills, "Performance Characterization of Distributed Algorithms for Replica Selection in Distributed Object Systems," Accepted for Fifth International Workshop on Software Performance (WoSP 2005), Palma de Mallorca, Spain, July 2005.

[43] D. Luckham, "Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Ordering of Events," http://anna.stanford.edu/rapide, 1996.

[44] J. Henriksen, "An Introduction to SLX™," Proceedings of the 1997 Winter Simulation Conference, ACM, Atlanta, GA, December 1997, pp. 559-566.

A Model-based Analysis of First-Generation Service Discovery Systems[1]

Christopher Dabrowski, Kevin Mills, and Stephen Quirolgico
National Institute of Standards & Technology
Gaithersburg, Maryland 20899

Information technology is undergoing a paradigm shift from desktop computing, where isolated workstations connect to shared servers across a network, to pervasive computing, where myriad portable, embedded, and networked information appliances continuously reconfigure themselves individually and collectively to support the information requirements of mobile workers and work teams. This shift will not occur overnight, nor will it be achieved without solving a range of new technical and social problems. Still, this inexorable change should yield many economic opportunities for the global information technology industry, and for the increasing swath of businesses that depend on information. The potential value of pervasive computing motivated the NIST Information Technology Laboratory (ITL) to establish a five-year program of research to help the information technology industry identify and solve some looming technical roadblocks that seemed likely to slow development and acceptance of the new paradigm. The ITL Pervasive Computing program addressed three general areas: human-computer interaction, programming models, and networking. Service discovery systems, which reside in an intersection between programming models and networking, cover a key aspect of pervasive computing. For this reason, researchers in ITL decided to study various industry designs for service discovery systems that could play a key part in future technology to enable pervasive computing applications. This special publication provides an analysis of a first generation of designs for service discovery systems.

Over the period from about 1998 to 2000, industry developed a first generation of competing architectures and protocols for device and service discovery. Such a plethora of incompatible approaches might impede the interoperability required by a market for pervasive computing. Is the existence of so many different service discovery systems justified? NIST researchers analyzed various technical approaches and developed a model to unify the features, functions, and processes provided. The goal of this modeling effort was threefold: (1) to understand the essential service-discovery functionality provided by the industry, (2) to reveal any technical deficiencies in existing service-discovery specifications, and (3) to define the technical bounds achievable from this first-generation of service-discovery systems. The result of this modeling effort is reported in this special publication.

The fact that numerous competing designs have appeared indicates a substantial industry interest in using dynamic service discovery as a means to deploy and evolve component-based systems. But why have so many different designs appeared? Are the designs sufficiently different to warrant multiple solutions? What elements are contained within the various designs? What problems should service discovery systems solve? What are the shortcomings of the first-generation of service discovery systems? What open issues do first-generation designs for service discovery systems leave for implementers to solve? These are the questions that motivate the work reported in this publication.

[1] Due to its scope and length, this paper was published as a separate, companion NIST Special Publication (SP 500-260). Only the execution summary of NIST SP 500-260 is reproduced here.

Based on careful analyses of selected specifications for service discovery architectures and protocols, we present a generic model that represents the key elements, relationships, and behaviors of a service discovery system. Our model consists of two parts: a meta-model that defines the context in which service discovery systems operate and a generic, object-oriented model that represents the fundamental structure and behavior of service discovery systems. We also identify some open issues or limitations in existing designs for first-generation service discovery systems. We demonstrate how our generic model can be used to represent specific service discovery systems.

Beyond an analysis of the structure and behavior of first-generation service discovery systems, we consider two other problems. First, the current generation of service discovery systems can lead to some system-wide performance issues, unless implementers and users exercise due care. We identify three classes of performance issues that might arise, and we suggest a range of solutions that implementers might adopt to solve each issue. A second problem relates to service guarantees. None of the service discovery systems we analyzed defined any expectations about the guarantees, or even the goals, that the design aimed to satisfy. We propose a set of service guarantees that we believe service discovery systems should aim to achieve, and we explain the qualifications associated with such guarantees. In other work, we have used our proposed service guarantees to assess the performance and correctness of specific designs for service discovery systems.

In summary, this special publication makes three specific contributions – intended to inform a future generation of designs and to improve the performance of implementations for the current generation of designs. First, we provide a generic model of the structure and behavior of first-generation service discovery systems, and we show how our model can represent the designs for several, specific service discovery systems. Our model unifies the common elements and behaviors in modern service discovery systems. Should an industry standards group choose to develop a unified specification for service discovery, our model could provide helpful input to the process. We also identify issues that designers should attempt to resolve in the next generation of service discovery systems. Second, we propose a set of service guarantees that we believe service discovery systems should strive to satisfy, along with an analysis of the factors that might interfere with meeting service guarantees. Such service guarantees could be cast into test assertions that serve to evaluate the behavior or measure the performance of designs and implementations of service discovery systems. Third, we identify and suggest possible solutions to performance issues that can arise in dynamic service discovery systems. Identifying possible performance issues can alert users to the potential for unexpected behavior when service discovery technology is deployed at large scale. Further, implementers of service discovery systems can consider our suggested solutions when developing software to embody related processes in a service discovery system. Our three contributions should help to improve the quality of the next generation of service discovery systems on which the service-oriented architectures of tomorrow appear likely to depend.

PERFORMANCE IMPROVEMENT TECHNIQUES FOR DISCOVERY SYSTEMS

Initial investigation and measurement of prototype service discovery systems revealed an interesting property. The performance of such systems depends primarily on a set of tunable parameters, while the optimum values for the parameters depend upon the number of elements present in the system. Unfortunately, the nature of service discovery systems is for the number of elements to fluctuate significantly over a wide range of time scales, many of which prove too short for system administrators to detect and respond with appropriate parameter changes. On the other hand, service discovery systems include mechanisms to monitor system elements and to detect changes in system composition. NIST researchers conceived the idea that self-adaptive algorithms could be developed to use the underlying protocols of a service discovery system to monitor system composition and to automatically adjust various parameter settings to achieve improved performance. The papers in this section of the special publication describe and report the performance properties of several self-adaptive algorithms developed by NIST researchers and applied to various service discovery systems.

In Paper #22, "Adaptive Jitter Control for UPnP M-Search", Mills and Dabrowski investigate various self-adaptive algorithms that could be used to mitigate response implosion, which can occur in systems where a client multicasts a query to an unknown population of potential respondents all of whom may respond, overrunning the client's receive buffer space. While response implosion may occur in any multicast-query system, the case of UPnP is particularly compelling because each respondent is required to send n $(3+2d+k)$ messages in response to each multicast query, where n is a redundancy factor, d is the number of devices contained by a respondent, and k is the number of unique service types contained by a respondent. To mitigate response implosion, UPnP multicast queries carry a value, M, such that each respondent chooses a time to response by drawing a uniformly distributed random number from the interval $0…M$. UPnP clients have no specific information to help in choosing a reasonable value for M. For example, this paper shows that a performance tradeoff exists between increased discovery latency (if M is chosen too large) and decreased discovery effectiveness (if M is chosen too small). The paper also shows that even when M is chosen to be the theoretically correct value, the random nature of responses leads to situations where the receive buffer becomes overly full; a situation that persists and leads to discovery effectiveness of only around 80%-85%. The paper proposes self-adaptive algorithms in two classes: random and scheduled. All the proposed algorithms rely on the fact that respondents likely have a picture of the state of the system, which they can use to determine when to reply. Further, respondents can feedback information to the client, allowing selection of a better value for M. Interestingly, the paper shows that the self-adaptive random schemes all exhibit the potential to overfill the receiver buffer, leading to lower than expected discovery effectiveness (82%-90%). On the other hand, the self-adaptive scheduled algorithms all achieve 100% discovery effectiveness at the cost of increased memory and processing usage at each respondent. The paper quantifies the estimated costs. The paper also suggests some alternatives to UPnP M-Search, which might lead to more effective discovery at lower cost and in larger networks.

In Paper #23, "Self-Adaptive Leasing for Jini", Bowers, Mills, and Rose analyze the performance of the Jini leasing system, which is one of several similar subscription-and-renewal functions included in many service discovery systems (an other distributed systems). Leasing systems require a client interested in using a remote resource to register for such use and then to periodically renew the registration. Failure to renew the registration will result in the client losing access to the remote resource. Such schemes allow client failures to be detected so that resources may be redirected to other clients. The paper defines relationships between lease period, detection-failure latency, and overhead (in bandwidth or processing time) for the Jini leasing system. The paper also defines and analyzes two self-adjusting leasing algorithms (called *adaptive* and *inverted*), which permit a lease granter to select a lease time that provides the lowest feasible failure detection latency while respecting limits on the overhead devoted to renewing leases. The analysis shows that the adaptive algorithm detects failures in ½ the time taken by the inverted algorithm. The paper also presents simulation results that confirm the analysis.

In Paper #24, "Improving Failure Responsiveness for Jini", Rose, Bowers, Quirolgico, and Mills show how a self-adjusting leasing algorithm (the adaptive algorithm from Paper #23) can be incorporated into a Jini lookup service to provide the best possible failure-detection latency within the limits of the resources that can be dedicated to lease renewal. This paper extends Paper #23 by describing how NIST researchers incorporated the algorithm into "reggie", the SUN Microsystems Java implementation of a Jini lookup service. The implementation discussed in this paper provides the basis for a demonstration of the algorithm. In Paper #25, "Self-Managed Leasing for Distributed Systems" Bowers, Mills, Quirolgico, and Rose recast the results from Paper #24 in the context of self-managed systems.

In Paper #26, "An Autonomic Failure-Detection Algorithm", Mills, Rose, Quirolgico, Britton, and Tan demonstrate how the self-adaptive leasing algorithm first introduced in Paper #23 can be applied to several functions in service discovery systems. The example applications include: (1) leasing in Jini, (2) subscriptions in UPnP, (3) service registration in SLP, and (4) polling in SLP. The paper provides analytical and simulation results for all the example applications and also adds measured empirical results for the Jini leasing application.

In Paper #27, "Performance Characterization of Decentralized Algorithms for Replica Selection in Distributed Object Systems", Tan and Mills survey key concepts related to replica selection and then use simulation to characterize performance (response time, server latency, selection error, probability of server overload) for four common replica-selection algorithms (*random*, *greedy*, *partitioned*, *weighted*) when applied in a decentralized form to client queries in a distributed object system deployed on a local network. The researchers introduce two new replica-selection algorithms (*balanced* and *balanced-partitioned*) that give improved performance over the more common algorithms. The paper finds the weighted algorithm performs best among the common algorithms and the balanced algorithm performs best among all those considered. The paper also discusses the limits of applicability for the algorithms as presented, and suggests modification that might extend the range of applicability. The work reported in this paper should prove applicable to service discovery systems (such as Jini and SLP) that require service cache managers to be maintained as replicated directories of available services. Given a set of replicated directories and a population of clients, the algorithms investigated in this paper can be used to select a directory for to receive each client query.

Adaptive Jitter Control for UPnP M-Search

Kevin Mills and Christopher Dabrowski
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

Abstract – **Selected service-discovery systems allow clients to issue multicast queries to locate network devices and services. Qualifying devices and services respond directly to clients; thus, in a large network, potential exists for responses to implode on a client, overrunning available resources. To limit implosion, one service-discovery system, UPnP, permits clients to include a jitter bound in multicast (M-Search) queries. Qualifying devices use the jitter bound to randomize timing of their responses. Initially, clients lack sufficient knowledge to select an appropriate jitter bound, which varies with network size. In this paper, we characterize the performance of UPnP M-Search for various combinations of jitter bound and network size. In addition, we evaluate the performance and costs of four algorithms that might be used for adaptive jitter control. Finally, we suggest an alternative to M-Search for large networks.**

I. INTRODUCTION

Selected service-discovery systems allow clients to issue multicast queries to locate network devices and services [1, 3]. Qualifying devices and services respond directly to clients; thus, in a large network, potential exists for responses to implode on a client, overrunning available resources. This implosion problem also arises in other protocols that support multicast queries and responses [4-7]. To limit implosion, one service-discovery system, Universal-Plug-and-Play[1] (UPnP) permits clients to include a *jitter bound* (*MX*) in multicast (M-Search) queries. Each qualifying device jitters its response time by randomly selecting a delay up to *MX*. Initially, clients lack sufficient knowledge to select an appropriate jitter bound, which varies with network size.

In this paper, we model the UPnP M-Search mechanism and characterize performance for various combinations of jitter bound and network size. The resulting performance curves should help designers of UPnP clients to understand the effects of selecting particular jitter bounds. We also consider four algorithms that might be used to adaptively control jitter in UPnP M-Search. We compare the performance of the adaptive algorithms against each other and against a fixed jitter bound. We discuss the costs associated with adaptation. These costs lead us to suggest an alternative approach to M-Search for large networks. The insights we provide should help designers of service-discovery systems to create architectures that can scale across a variety of network sizes, while achieving effective and efficient performance.

The remainder of this paper is organized as follows. Section II describes the UPnP M-Search mechanism, defines an experiment and related metrics to characterize M-Search performance, and illustrates M-Search performance for varying jitter bounds and network sizes. Section III outlines four algorithms that might be used to adaptively adjust M-Search jitter bounds, and compares the performance of the algorithms against each other and against a fixed jitter bound. Section III also discusses the costs and assumptions underlying adaptation. Section IV suggests an alternative to M-Search for use in large networks. Section V gives our conclusions.

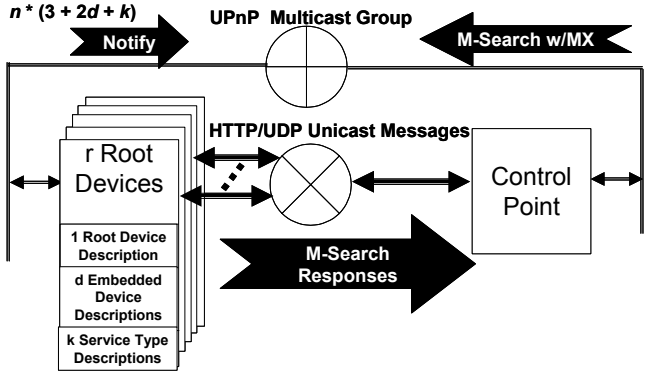


Fig. 1. General Operation of UPnP Discovery

II. CHARACTERIZING M-SEARCH PERFORMANCE

Fig. 1 depicts the general operation of device and service discovery in UPnP. UPnP consists of two main elements: root devices (servers) and control points (clients). A UPnP network may contain $r \geq 0$ root devices. Each root device contains $d \geq 0$ embedded devices and $k \geq 0$ unique service types, where each device and service has a specific type. Each root device also contains a hierarchical description that defines the capabilities of $1 + d + k$ elements: the root device and each of its embedded devices and unique service types. The description can be rather lengthy; thus, UPnP provides a two-step process for obtaining descriptions. A control point first discovers devices or services of interest by type or identity, and then requests the related descriptions.

[1] Certain commercial products and standards are identified in this paper to describe our study adequately. The National Institute of Standards and Technology neither recommends nor endorses these products or standards as the best available for the purpose.

UPnP provides two discovery modes: lazy and aggressive. Lazy discovery uses periodic announcements sent by each root device on the UPnP multicast group (Notify in Fig. 1). At each announcement interval, each root device sends n (3 + 2d + k) Notify messages to identify the root device (and its identity and type), each embedded device (by identity and type), and each unique service type (by type). The UPnP specification recommends a duplicate transmission factor, n, "due to the unreliable nature of UDP" (user-datagram protocol) [1]. Control points listen for announcements to discover the existence of various devices and services. The UPnP specification sets the announcement interval at 30 min. or more. For this reason, control points may use aggressive discovery to get an immediate picture of available services.

Aggressive discovery commences when a control point multicasts an M-Search query, which specifies an interest (that can include specific devices, device types, service types, or all) and a jitter bound (*MX* in Fig. 1). Root devices listen for M-Search queries to determine if any contained items are of interest. Each root device sends 3n responses if the root device qualifies, and 2n and n responses respectively for each qualifying embedded device and service type. If the query asks for everything (SSDP_ALL), each root device responds with the same n (3 + 2d + k) messages used in lazy discovery. To mitigate a potential implosion of responses, each root device waits a random time, uniformly distributed in the range 0..*MX*s, before transmitting its responses in a burst.

A. Experiment Definition

To characterize M-Search performance, we used SLX™ [8] to construct a simulation model representing the topology shown in Fig. 1, deployed in a 10-Mbps Ethernet. Since the UPnP specification allows implementation choices, we based those aspects of our model on UPnP software available publicly from Intel [9]. We allow the number of root devices, r, to vary from 10 to 200 by 10-step increments. Each root device includes an identical count of embedded devices (d = 2) and service types (k = 3). We set n = 2, the default value in the Intel implementation of UPnP. We allow *MX* to vary from 2 to 40 in 2-s increments. For each combination of r and *MX*, an M-Search task in a single control point issues a query requesting SSDP_ALL, which elicits n (3 + 2d + k) = 20 200-byte response messages from each root device; thus, aggregate implosion ranges from 200 (r = 10) to 4000 (r = 200) response messages. (To keep our graphs legible, we display results over only r = 10..100 and *MX* = 2..20.) We limit the M-Search task to buffer no more than 40 messages, dropping the excess. We allow the control point task to execute every 5 ms, processing one response message at each execution (200 messages/s maximum rate). For each message, the task examines a cache to see if a new discovery occurs, adding items to the cache as required. The task takes c ms to process a message, where c varies with the cache size. When finished, the task reschedules itself to execute in 5 – c ms. If $c \geq 5$, the task executes immediately.

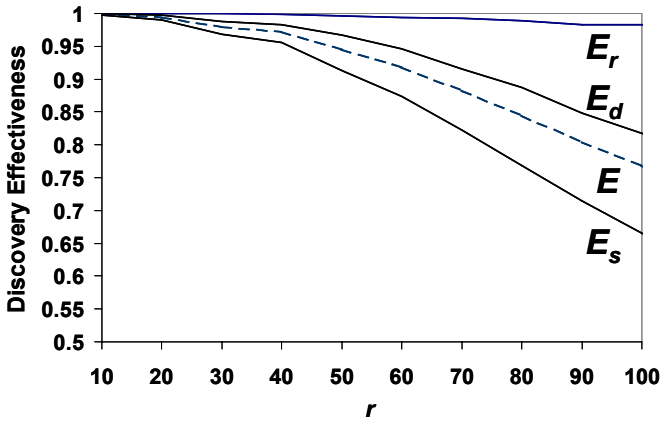


Fig. 2. Overall discovery effectiveness (E) compared against discovery effectiveness by entity type: root devices (E_r), embedded devices (E_d), and services (E_s). [*MX* = 10 s]

B. M-Search Performance

We measure system performance with four metrics: discovery effectiveness (E), discovery latency (L), buffer utilization (B), and processor usage (P). Given a network comprising $e = r + rd + rk$ entities and assuming that a control point discovers $f \leq e$ entities from responses to an M-Search, then $E = f / e$. We can also track discovery effectiveness by entity type, root devices (E_r), embedded devices (E_d), and services (E_s) as shown in Fig. 2. In the Intel implementation, each root device sends M-Search responses in the same order (3n then 2dn then kn) and since responses earliest in the sequence are more likely to find buffer space available at the control point, root devices are more likely to be discovered than either embedded devices (next most likely) or services (least likely).

Fig. 2 also reveals that randomly jittering responses does not ensure E = 1, even when *MX* is set to a seemingly suitable value. When r = 100, a total of 2000 response messages will implode on the control point, which processes 200 messages/s, suggesting that 10s (2000/200) might be a suitable value for *MX*. Unfortunately, since each root device picks a random time to respond and then sends a burst of 20 response messages, collision periods can occur during which receive buffers are overrun in the control point. Fig. 3 displays the problem.

Even at *MX* = 20s, collisions occur with sufficient frequency that E decays significantly beyond r = 50. Collisions lead to increased buffer occupancy (Fig. 4), which leads to increased likelihood of message drops. Periods of high buffer occupancy (and therefore message loss) tend to persist, as incoming messages arrive in bursts at random intervals, while the M-Search task reduces the buffer backlog at a steady rate.

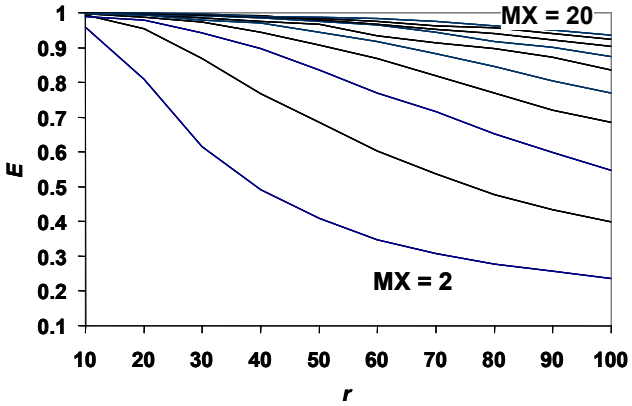


Fig. 3. Discovery effectiveness (*E*) for various values of *MX* (2s to 20s in 2s increments) as the number or root devices (*r*) increases.

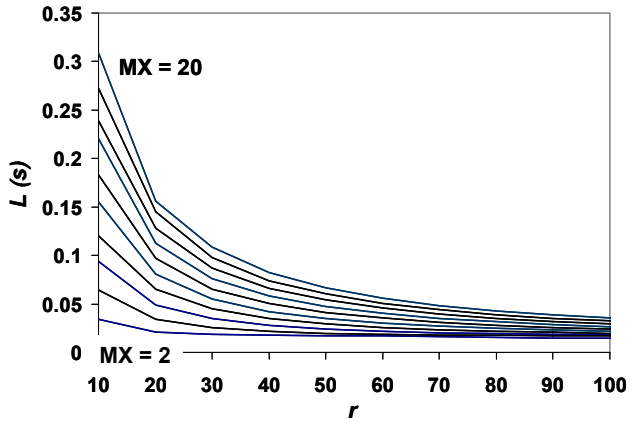


Fig. 4. Average buffer occupancy (*B*) as a percentage of available buffers (40 messages in this case) for various values of *MX* (2s to 20s in 2s increments) as the number or root devices (*r*) increases.

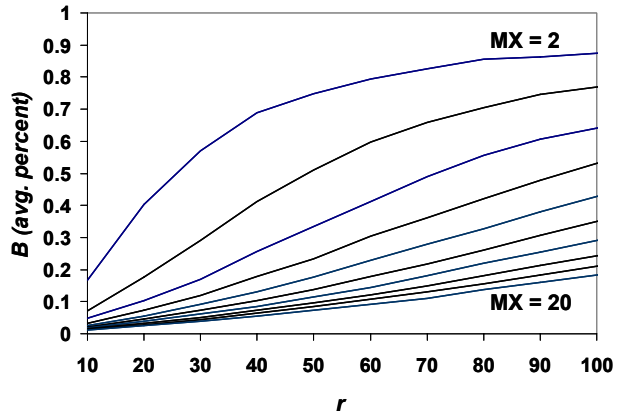


Fig. 5. Average discovery latency (*L*) for various values of *MX* (2s to 20s in 2s increments) as the number or root devices (*r*) increases.

Buffer size at the control point can be augmented to accommodate additional responses; however a suitable buffer size may be difficult to determine given the random nature of response jitter (and unknown network size). Instead, a control point could increase *MX*; but then, as Fig. 5 shows, discovery latency will grow.

We define discovery latency (*L*) as the time that elapses between successive discoveries of new entities in the network. As Fig. 5 shows, when *MX* is large compared to network size the gap between new discoveries grows for a control point. An increased *MX* also leads to fewer buffer overruns, which increases the discovery effectiveness for a control point. As discovery effectiveness increases, the discovery cache in the control point increases in size, which causes the M-Search task to spend more processor cycles examining each response message (Fig. 6). This increase occurs because the M-Search task must look through more cache entries to determine if a new entity has been discovered, and to insert a related cache entry if needed. For relatively large values of *MX*, processor utilization increases linearly with network size, though this would change if we modeled more efficient search algorithms. For relatively small values of *MX*, growth in processor utilization levels off with the size of the discovery cache maintained by the M-Search task.

III. ADAPTIVE JITTER CONTROL

We propose four algorithms for adaptive-jitter control, and then illustrate the performance arising from each. We also discuss the costs and assumptions underlying the algorithms. Some other algorithms to address multicast query-response implosion can be found in the literature [10,11].

A. Four Adaptive Jitter-Control Algorithms

In adaptive-jitter control, each root device independently estimates the time it will take for all root devices to respond to each M-Search query. Each root device then uses its estimate to determine a time to send its own responses (if any). Included in each response message is a value recommending how long the control-point M-Search task should listen for responses. With this approach, the M-Search task need not guess an appropriate *MX* value.

Each root device listens on the UPnP multicast group for Notify messages (which include a caching time, or *max-age*) sent by all root devices, and builds a map (*NM*) of devices and services in the network. For each root device, *NM* includes: the identity and type of the root device and all embedded devices and unique service types, a *max-age*, and an estimate of the redundant transmission factor (*n*). Estimates of *n* exploit the fact that in the Intel implementation each message is sent *n* times before the next message. Listening root devices apply a time threshold to identify duplicate messages and then compute an average *n* for each announcing root device.

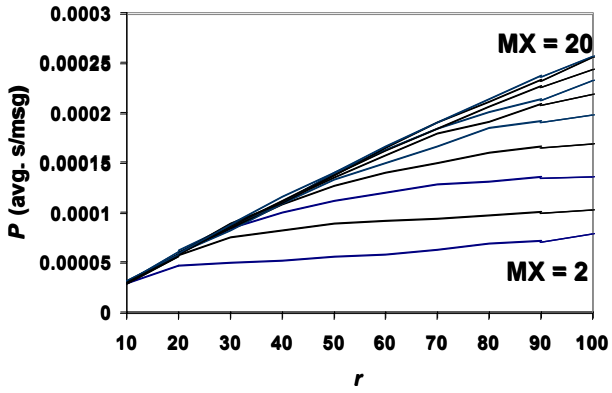


Fig. 6. Average processor time (*P*) in seconds/message for a Control Point M-Search task to examine a response for various values of *MX* (2s to 20s in 2s increments) as the number or root devices (*r*) increases.

M-Search queries issued by the M-Search task include a rate, *R*, at which the task can consume messages, and also an $MX \geq 0$. Upon receiving an M-Search query, a root device cycles through its *NM* to estimate how many response messages will be sent by all root devices, using *R* to also estimate when the last set of responses should commence (*Jstart*) and finish (*Jend*) – under an assumption that messages will be sent consecutively at rate *R*. During this process, a root device can also note a time (*Stx*) when it should send its own responses – under an assumption that root devices will send messages sequentially in the ascending order of their unique identities. Using this information, we devised four adaptive jitter-control algorithms: Random Burst (RB), Random Paced (RP), Scheduled Burst (SB), and Scheduled Paced (SP).

In the random algorithms (RB and RP), a root device selects a time, T_r, randomly distributed uniformly on the interval [0,*Jstart*], to send its response messages. The root device includes *Jend* in each response so that the M-Search task will learn an appropriate time interval to listen. The root device will not respond if $0 < MX < T_r$. In the RB variant of the algorithm, the root device bursts its response messages. In the RP variant, the root device paces its responses at rate *R*.

In the scheduled algorithms (SB and SP), a root device sends its response messages at *Stx*; however, the root device will not respond if $0 < MX < Stx$. Response messages are sent in a burst (SB) or at rate *R* (SP). The root device includes *Jend* in each response message.

B. Performance of Adaptive Jitter Control

Fig. 7 illustrates discovery effectiveness (*E*) for each adaptive jitter-control algorithm as the number of root devices (*r*) increases from 10 to 300. For comparison, we include the performance of a fixed *MX* = 33s, which is the *Jstart* value estimated by each root device when *r* = 300.

Scheduling transmissions achieves full effectiveness (*E* = 1). On the other hand, randomizing transmissions leads to collisions in the receive buffers, and then to buffer overflows and lost discoveries. Pacing responses (RP) results in fewer buffer overflows, but fails to eliminate them. While RP more closely matches arrival rate with service rate, Fig. 8 indicates a nearly identical average buffer occupancy for RP and RB.

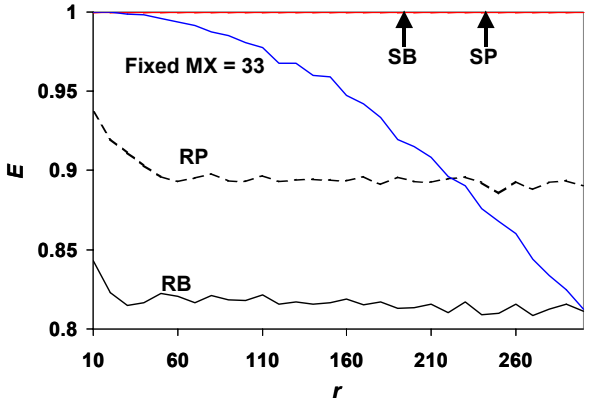


Fig. 7. Discovery effectiveness for four adaptive jitter control algorithms and one fixed jitter bound as the number of root devices increases

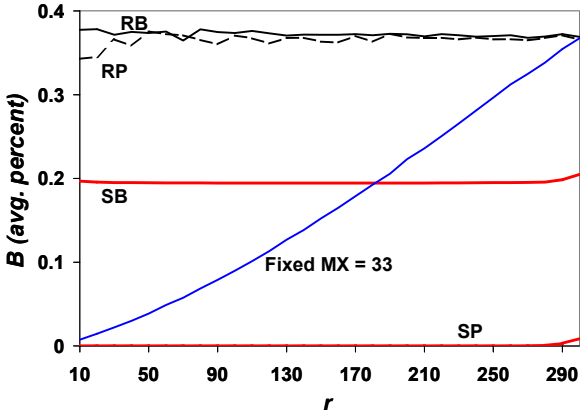


Fig. 8. Average buffer occupancy for various jitter-control algorithms as the number of root devices increases.

Buffer utilization is very low for SP because scheduling eliminates collisions and responses arrive at the rate at which the M-Search task can process them. While SB avoids collisions, responses arrive in (20-message) batches, leading to a higher average buffer utilization. Fig. 8 also shows that each of the adaptive jitter-control algorithms yields a nearly stable average buffer utilization (but at different occupancy levels), while buffer utilization for a fixed *MX* varies with the relationship between *MX* and *r*.

Fig. 9 illustrates that all the adaptive jitter-control algorithms provide consistently low average discovery latency, *L*, despite variation in network size, which is not the case for a fixed *MX*, where latency varies with the relationship between *MX* and *r*. The scheduled algorithms (SB and SP) perform slightly better than the random algorithms (RB and RP) because buffer overflows resulting

from jitter randomization cause some discoveries to be lost, which tends to lengthen the time between new discoveries.

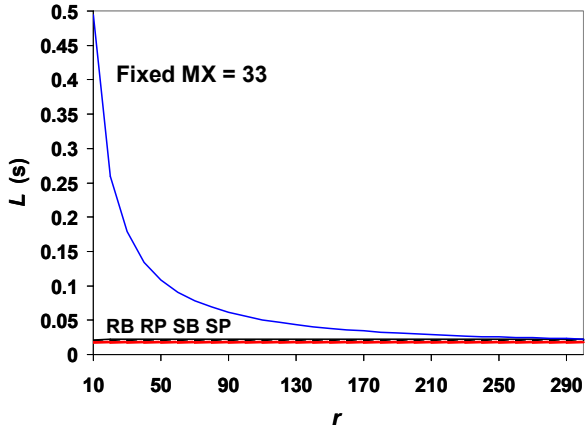


Fig. 9. Average discovery latency of various jitter-control algorithms as the number of root devices increases

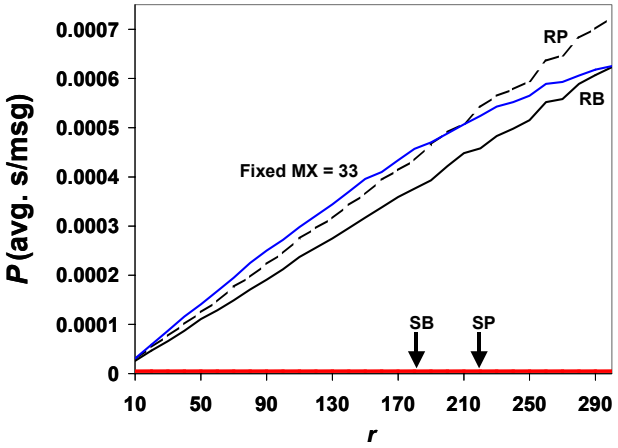


Fig. 10. Average processor seconds per message used by the M-Search task for various jitter-control algorithms and increasing network size.

The scheduled algorithms also lead to some serendipitous effects on processor utilization in the M-Search task (Fig. 10). Since scheduled responses arrive in order, the M-Search task need not conduct a search of its discovery cache for each response message. Instead, the M-Search task checks to see if a response can be inserted into the cache at the current insertion point. Only if this is not the case does the M-Search task need to search its cache. In our experiments all scheduled responses arrived in the expected order; thus, Fig. 10 shows that both SB and SP consume a small, fixed amount of processor time for each message.

Fig. 10 also shows that the cache search required by the random algorithms causes processor utilization to increase as the number of discovered entities increase. Processor utilization for RP always exceeds that for RB because the RP algorithm discovers a greater percentage of entities. Random jitter with a fixed *MX* = 33s uses more processor time than either RP or RB up until about *r* = 200, where RP proves more effective and thus requires more processor time. The rate of increase in processor utilization for the fixed *MX* continues to decline, reaching the same value as RB when *r* = 300 (and *Jstart* = *MX* = 33s).

C. Costs and Caveats

Adaptive jitter control comes with two costs: memory and processing time in root devices. Each root device creates, stores, and maintains a network map (*NM*) of size

$$S = h + \sum_{i=0}^{r} [p + q \cdot (d_i + k_i + t_i)], \qquad (1)$$

where h is the cache-header size, p is the root-device header size, q is per-entry content size, r is the number of root devices, and d_i, k_i, and t_i represent respectively the number of embedded devices, service types, and device types maintained by root device i. In our experiments, S varies from about 1.2 (r = 10) to 37 (r = 300) Kbytes.

To process an M-Search query, a root device must scan *NM* to estimate the likely number of responses that will be issued by all root devices. During the scan, a root device also purges stale entries. Thus, for each M-Search query a root device uses processor time

$$C = \sum_{i=0}^{r} [x \cdot (1 + d_i + k_i)] + (y \cdot O), \qquad (2)$$

where x is processor time to scan one entry, y is processor time to purge one root-device, and O is the number of stale root-device entries found during the scan. In our experiments, for SSDP_ALL queries with no stale entries, C varies between 0.3 (r = 10) and 9 (r = 300) ms.

In addition to memory and processing costs, the scheduled algorithms assume that each root device has the same knowledge about network state (*NM*). Absent this assumption, root devices would schedule collisions, leading to lower discovery effectiveness. This same-*NM* assumption should hold in steady state, where all root devices have had a chance to announce themselves and where changes occur infrequently. Of course, when a root device enters a network it must acquire *NM* to participate effectively in adaptive jitter control.

IV. DISCOVERY IN LARGE NETWORKS

Most discovery protocols provide for recurring announcements at a known interval. For example, the Jini protocol recommends announcements every 120s [11]. Recurring announcements permit a network device to listen for a period of time over which a reasonably complete *NM* might be constructed. Unfortunately, the minimum announcement interval specified for UPnP is 30 min., which might prove too long a period for a device to wait before participating on the network. To compensate for this lengthy announcement interval, UPnP provides the M-Search

mechanism so that network devices can attempt to gain a sense of the *NM* on demand. We have shown, though, limitations of the M-Search as a means to find all devices and services on the network. Some other discovery protocols [2,3] include feedback mechanisms within their multicast queries in order to provide a means of dampening responses. Using such dampening mechanisms, a short repeated burst of multicast queries (for example, Jini recommends seven queries at intervals of five seconds) might be used to obtain a reasonably complete *NM* (ignoring the possibility of temporary node and channel failures). Unfortunately, UPnP includes none of these mechanisms; thus, acquiring the *NM* needed to permit effective participation in adaptive jitter control for M-Search queries seems to require using the regular UPnP M-Search. Since we have already shown UPnP M-Search to be ineffective for this purpose, we propose an alternative to M-Search for *NM*-bootstrap and for general use in large networks.

Suppose that on startup a root device initiates a network mapping (*NM*) service with probability *W*. In that case, the network will contain only *rW NM* services. Then a control point, or a newly starting root device, can use M-Search (in fixed or adaptive form) to query only for instances of *NM* services. Each qualifying *NM* service can respond with the count of root devices, embedded devices, and service types known to it. Using this information, a querying node can select one *NM* service and use http-GET (HyperText Transfer Protocol) to retrieve its *NM*. Alternatively, the querying node may issue http-GETs to multiple *NM* services, and then merge the results into a signal *NM*. After retrieving a *NM*, a root device should be sufficiently bootstrapped to participate in adaptive M-Search. For a control point, querying for *NM* services will reduce (or eliminate) the need to issue SSDP_ALL M-Search queries.

A further advantage of using *NM* services can accrue as network volatility increases. As the need arises, due to increase in load or in network or node failures, a root device can choose to start a *NM* service to increase redundancy or to share the load from an increasing number of client queries. Similarly, as volatility diminishes or as network size decreases, root devices with a running *NM* service can elect to terminate the service in order to reduce network overhead.

V. CONCLUSIONS

Given the UPnP M-Search mechanism, we illustrated relationships among network size (*r*), jitter bound (*MX*), discovery effectiveness (*E*) and latency (*L*), and buffer (*B*) and processor (*P*) utilization. Specifically, we showed how an inappropriate jitter bound (*MX* value) in UPnP M-Search queries could significantly reduce discovery effectiveness or increase discovery latency. We outlined four algorithms that might be used for adaptive jitter control, and we explained the storage and processing costs associated with adaptation. We compared the performance of the adaptive algorithms against each other and against a fixed *MX* value. The random paced (RP) algorithm yielded increased discovery effectiveness over random burst (RB). Both scheduled algorithms (SB and SP) led to better performance than either random algorithm. In particular, the scheduled paced (SP) algorithm achieved optimal performance for all metrics. We explained, however, that the performance of the scheduled algorithms would deteriorate if all root devices do not share the same picture (*NM*) of the network state.

We outlined an approach to enable root devices to bootstrap their *NM*. We suggested that control points might also use this approach to replace M-Search SSDP_ALL queries, thus avoiding the potential for an implosion of M-Search responses. Further, we hinted that the *NM*-bootstrap mechanism might be adapted to modulate redundancy and load sharing in support of aggressive discovery in UPnP networks. Further exploration of these ideas remains for future work. We also suggest that these algorithms should be investigated under various types and rates of failure.

REFERENCES

[1] Universal Plug and Play Device Architecture, Version 1.0, Microsoft, June 8, 2000.

[2] Erik Guttman and James Kempf. Service Location Protocol, Version 2, RFC 2608bis, January 10, 2002.

[3] Ken Arnold et al, The Jini Specification, V1.0 Addison-Wesley 1999. Latest version is available from Sun.

[4] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, Session Invitation Protocol, RFC 2543, March 1999.

[5] Stuart Cheshire, DNS-based Service Discovery, Internet Draft, December 20, 2002.

[6] O Catrina, D. Thaler, B. Aboba, and E. Guttman, Zeroconf Multicast Address Allocation Protocol, Internet Draft, October 22, 2002.

[7] Stuart Cheshire, Performing DNS queries via IP Multicast, Internet Draft, December 20, 2002.

[8] James O. Henriksen, "An Introduction to SLX™" *Proceedings of the 1997 Winter Simulation Conference*, ACM, Atlanta, Georgia, December 7-10, 1997, pp. 559-566.

[9] Preston Hunt and Ulhas Warrier, "UPnP Applications Enhance Mobile Functionality", *Intel Developer Update Magazine*, Intel, April 2002, pp. 1-5. (Intel UPnP Software Development Kit available from: http://www.intel.com/labs/connectivity/upnp/index.htm)

[10] T. Imieli´nski and S. Goel, "Dataspace - querying and monitoring deeply networked collections of physical objects," Tech. Rep. DCS-TR-381, Rutgers University, July 1999.

[11] B. R. Badrinath and Pradeep Sudame. "Gathercast: The design and implementation of a programmable aggregation mechanism for the Internet", *Proceedings of IEEE International Conference on Computer Communications and Networks* (ICCCN), October 2000.

Self-Adaptive Leasing for Jini

Kevin Bowers
Renssalaer Polytechnic Institute
bowerk@rpi.edu

Kevin Mills and Scott Rose
National Institute of Standards and Technology
{kmills, srose}@nist.gov

Abstract

Distributed systems require strategies to detect and recover from failures. Many protocols for distributed systems employ a strategy based on leases, which grant a leaseholder access to data or services for a limited time (the lease period). Choosing an appropriate lease period involves tradeoffs among resource utilization, responsiveness, and system size. We investigate these issues for Jini Network Technology. First, we establish quantitative tradeoffs among lease period, bandwidth utilization, responsiveness, and system size. Then, we consider two self-adaptive algorithms that enable a Jini system, given a fixed allocation of resources, to vary lease periods with system size to achieve the best responsiveness. We compare performance of these self-adaptive algorithms against each other, and against fixed lease periods. We find that one of the self-adaptive algorithms proves easy to implement and performs reasonably well. We anticipate that similar procedures could add self-adaptive capability to other distributed systems that rely on leases.

1. Introduction

Distributed systems require strategies to detect and recover from failures. One commonly used strategy employs a leasing mechanism, where a node grants a leaseholder access to a resource for a limited time (the lease period). If the resource is needed beyond the original lease period, then the leaseholder can renew the lease by requesting additional lease periods. Once the resource is no longer needed, the leaseholder may relinquish its lease. If the leaseholder does not renew a lease before expiration of the lease period, the lease grantor assumes leaseholder failure and terminates the lease to prevent resource leaks. Since originally proposed by Gray and Cheriton for consistency maintenance in a distributed file cache [1], leases have become widely used in a range of applications [2-6].

In any leasing system, questions arise regarding how to select the lease period. Choosing an appropriate lease period requires consideration of tradeoffs among resource utilization, responsiveness, and number of leaseholders. We investigate these issues in the context of service-discovery protocols, which allow distributed software components to discover each other and compose themselves into assemblies that cooperate to meet application needs. Though several service-discovery protocols currently exist [e.g., 5-8], we selected Jini Network Technology [5] for our study because leasing plays a central role in registering Jini services. We base our modeling and analysis on the Jini specification [7].

We investigate self-regulating algorithms for achieving the best available responsiveness from a leasing system as system size varies, while respecting a constraint on resources devoted to leasing. We begin by establishing quantitative tradeoffs among responsiveness, resource consumption, and system size. Then, we propose two different self-regulating algorithms for varying lease periods in response to changing system size. We use simulation to compare the effectiveness of the algorithms against each other and against fixed lease periods. We consider whether one of the algorithms might be used to improve performance of Jini leasing and discuss using the algorithm in other service-discovery protocols, such as Universal Plug-and-Play (UPnP) [6].

2. Jini Leasing

Jini defines an architecture that enables clients and services to rendezvous through a third party, known as a lookup service. A Jini service registers a description of itself with each discovered lookup service. A Jini client may register a request to be notified by a lookup service of arriving or departing services of interest, or of changes in the attributes describing services of interest.

Figure 1 illustrates message exchanges for some typical Jini leasing scenarios. A registering component requests registration for a duration (L_R), which may be accepted at time T_G for a granted lease period $L_G \leq L_R$. L_R may be *any*, which allows any value for L_G. To extend registration beyond L_G, registering components must renew the lease prior to an expiration time $T_E = T_G + L_G$; otherwise, registration is revoked. This cycle continues until a Jini component cancels or fails to renew a lease. Lookup services assign L_G within a configured range, $L_{MIN} \leq L_G \leq L_{MAX}$. While a granted lease may not be revoked prior to T_E, lookup services may deny any lease request. Jini components must adopt strategies for selecting values for L_R. Similarly, lookup services must determine algorithms for assigning values for L_G, L_{MIN}, and

L_{MAX}; and for deciding when to deny leases. We identify some relevant relationships.

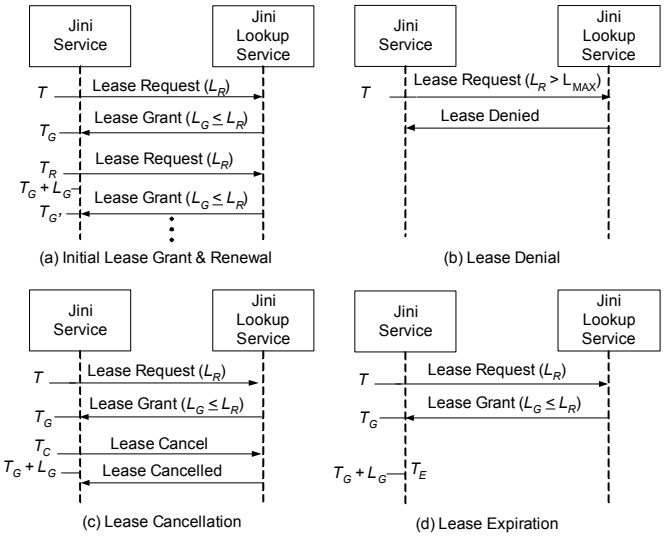


Fig. 1. Message exchanges for four Jini leasing scenarios.

Let S_R be lease-request size, S_G be lease-grant size, and N be the number of leaseholders. Typically, a leaseholder and lookup service exchange one request-grant pair per renewal cycle, with rate $1/L_G$ Hz. Assuming identical L_G assigned for each lease, bandwidth use (B) can be estimated as: $B=(N/L_G)\cdot(S_R+S_G)$. Assuming constant S_R and S_G, B increases linearly with N and decreases exponentially with L_G. Another metric, responsiveness, R, measures the latency with which lookup services can detect leaseholder failure. Assuming uniformly distributed failure times, then expected responsiveness is $R = L_G/2$; thus, R is independent of N, but B and R are related through L_G.

These relationships can be used to constrain and predict behavior of a leasing system. For example, assume known requirements for R and B. The responsiveness equation can be rewritten to determine L_G [i.e., $L_G = 2R$]. Then, using L_G, the bandwidth equation can be transformed to find maximum system size [i.e., $N_{MAX} = (B \cdot L_G)/(S_R + S_G)$]. With this information, lookup services could grant lease periods $\le L_G$ to ensure required responsiveness, deny requested leases that would consume an excess share of bandwidth, and deny requests for leases once N reaches N_{MAX}.

3. Two Self-adaptive Leasing Schemes

We consider two techniques to vary L_G with N; thus, using available bandwidth (B) to achieve the best possible responsiveness (R) for a given value of N. One technique restricts lease requests to L_R = *any*. The second technique inverts the leasing process, permitting lookup services to poll leaseholders at a variable interval.

Restricting L_R. Assuming a leasing system must consume at most bandwidth B and guarantee minimum average responsiveness R_{MIN}, a lookup service can grant a maximum lease period $L_{MAX} = 2R_{MIN}$. Given B, S_R, and S_G, we can determine a maximum lease-renewal rate $G = B / (S_R + S_G)$. For minimum system size, $N_{MIN} = 1$, the lookup service can grant a minimum lease period $L_{MIN} = 1/G$. While this value for L_{MIN} respects the bandwidth constraint, other factors should be considered. For example, at $L_{MIN} = 1/G$ leaseholder processing burden might prove unacceptable. Instead, a leasing system might constrain maximum responsiveness (R_{MAX}), giving a minimum lease period $L_{MIN} = 2R_{MAX}$. Knowing N, a lookup service may select a suitable granted lease period from a range ($L_{MIN} \le L_G \le L_{MAX}$) using a simple algorithm. First, compute $L_G = N/G$. If $L_G > L_{MAX}$, then deny the lease; otherwise, if $L_G < L_{MIN}$, then set $L_G = L_{MIN}$. Assigning L_G with this algorithm permits a leasing system to constrain B and guarantee minimum average responsiveness (R_{MIN}), while providing the best responsiveness achievable (up to R_{MAX}) as N varies over $1..N_{MAX}$.

Inverted Leasing. As an alternative, we could invert the leasing process so that a lookup service polls periodically on a multicast channel, where all leaseholders listen. Figure 2 illustrates some associated message exchanges. To obtain a lease, a leaseholder sends (via reliable unicast) a lease request to the lookup service, which returns a time (T_P) when the leaseholder should expect to hear a multicast poll.

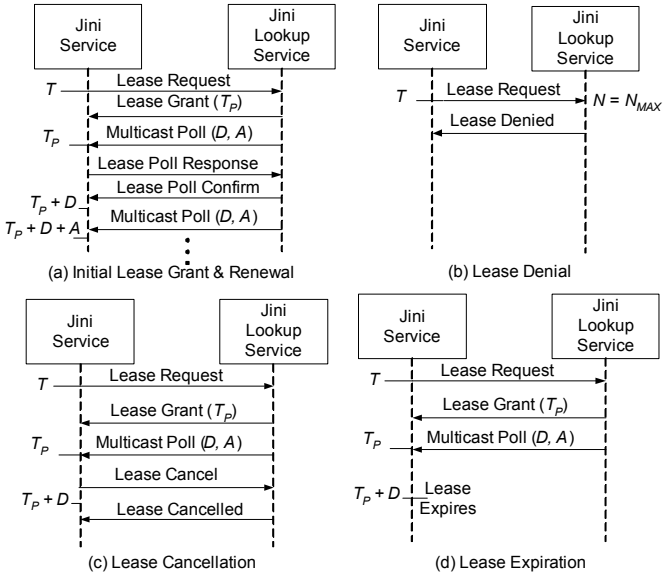


Fig. 2. Message exchanges for inverted leasing mechanism.

Each poll includes two values: the duration (D) over which the lookup service will listen for leaseholders to respond and the additional time ($A \ge 0$) beyond D within which leaseholders can expect the next poll. Each leaseholder chooses a random time (distributed uniformly over 0..D) to respond to the lookup service, which confirms each response.

The lookup service cancels a lease if the leaseholder does not respond within D. Similarly, failing to receive a poll within $D + A$ after the previous poll, causes a leaseholder to request a new lease. The main issue is selecting values for D and A in each poll.

Assuming the polling interval is bounded by $L_{MIN} \leq D + A \leq L_{MAX}$, the lookup service computes $D = \mathbf{max}(N/G, L_{MIN})$. A rapidly expanding system might benefit from deferring the next poll until $D + A$ to accommodate increases in N during D. Choosing an appropriate value for A depends on system growth expected during D. In our experiments, we set A as a percentage of D. Recall, though, that $D + A \leq L_{MAX}$, so A may be reduced below its computed value. When $A = 0$, the leasing system has reached maximum capacity. To ensure this, the lookup service must deny lease requests that will cause N to exceed N_{MAX}, where $N_{MAX}=L_{MAX}/G$.

When using inverted leasing, a lookup service limits bandwidth usage according to $B = S_P + ((N/P)\cdot(S_{PR} + S_{RC}))$, where P is the polling interval ($D < P \leq D + A \leq L_{MAX}$) and S_P, S_{PR}, and S_{RC} represent respectively the size of poll, poll-response, and response-confirm messages. Inverted leasing achieves system responsiveness of $R = D$, which is only ½ as responsive as simple adaptive leasing. To understand this difference, consider the following analysis.

Assume failure times are distributed uniformly on D. Failures may occur either before or after a leaseholder responds to a poll. For leaseholders that fail before a poll, expected failure-detection latency is $D/2$. For leaseholders that fail after a poll, expected failure-detection latency increases to $(D/2) + D$. Assuming that failures are equally likely before or after a poll, then $R = 1/2\cdot(D/2) + 1/2\cdot(3D/2)$, which reduces to $R = D$.

4. Simulation Results and Discussion

We used simulation to investigate dynamic behavior of our self-adaptive algorithms. We coded an SLX discrete-event simulation [9] model of Jini. To confirm our analysis and to verify our simulation, we conducted simulation experiments, varying N from 10..200 and L_G from 15..300 s in 15-s increments. We used S_R = 128 bytes and S_G = 32 bytes. Figure 3 shows simulated results for average B and R when L_G = 15 s, 60 s, and 120 s. Our simulation confirms our analyses: (1) B increases linearly with N for a given L_G and decreases exponentially with L_G for a given N and (2) $R = L_G/2$, independent of N.

Next, we created model variants to implement the self-adaptive leasing algorithms described in Section 3. One variant (Adaptive) replaces fixed L_G with our simple adaptive algorithm; the other variant (Inverted) substitutes our inverted procedures for Jini leasing. We measured B under increasing and decreasing N. We measured the control variable (L_G for Adaptive and D for Inverted) under increasing N, and we measured R under decreasing N. We set L_{MIN} = 15 s, $L_{MAX} = \infty$, and G = 3. For experiments involving Inverted, we set $A = 0.2\cdot D$.

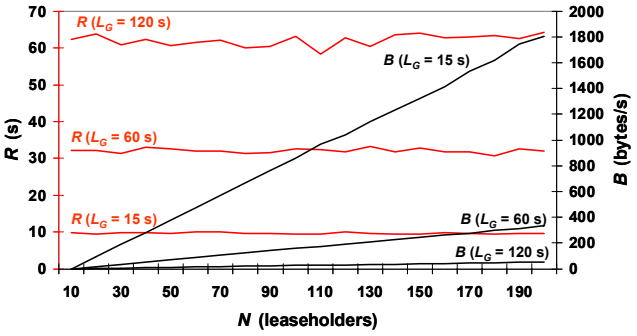


Fig. 3. System responsiveness (R) – left-hand y-axis – and bandwidth usage (B) – right-hand y-axis – for three granted lease periods (L_G = 15 s, 60 s, and 120 s) as system size increases (N = 10 to 200 leaseholders).

Figure 4 depicts both Adaptive and Inverted under increasing N. While the control variables change in a similar fashion, change in B exhibits two obvious differences. First, B increases more steeply under Adaptive than under Inverted. Second, Inverted begins to constrain B earlier than Adaptive, which leads to a higher peak bandwidth usage. Inverted affects all leaseholders with each adjustment in the control variable, while Adaptive affects leaseholders one-by-one, and only as each lease is renewed.

Figure 5 plots average R achieved by each self-regulating scheme as N decreases. Inverted begins to reduce B sooner than Adaptive. For R, the results tell two stories. First, as indicated by a steeper negative slope, Inverted adapts R more quickly than Adaptive. Unfortunately, Inverted achieves only ½ the responsiveness of Adaptive. Implementing Inverted would require profound changes in Jini. Adaptive can be implemented easily within Jini lookup services, and might apply to domain-wide leasing.

Each Jini service is required to register its service description with each appropriate lookup service that it discovers; thus, a service may be maintaining leases on N_D different lookup services. System-wide leasing demands will vary with N_D. Assuming a known network-wide resource budget for leasing, e.g., either aggregate bandwidth (B_D) or renewal rate (G_D), then each lookup service can compute its share (either B_D/N_D or G_D/N_D). Jini facilitates monitoring N_D by requiring each lookup service to announce itself periodically. By monitoring announcements, each lookup service can increment and decrement N_D as lookup services come and go, and continuously adjust its share of resources.

Our results might also apply to a number of leasing schemes outside of Jini. For example, UPnP devices manage variables for which they may offer subscriptions to control points. UPnP subscription procedures, and associated

parameters, appear quite similar to those defined in Jini. We are confident our adaptive leasing algorithm could be applied to UPnP, yielding performance properties similar to those we report for Jini.

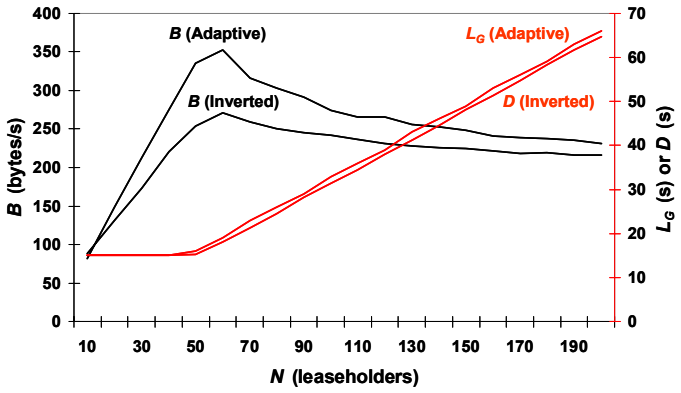


Fig. 4. Bandwidth usage (*B*) – left-hand y-axis – and control variable (L_G for Adaptive and *D* for Inverted) setting – right-hand y-axis – as system size increases (*N* = 10 to 200 leaseholders). L_{MIN} = 15 s, *G* = 3 renewals per second, L_{MAX} = ∞, and (for Inverted) *A* = 0.2*D*.

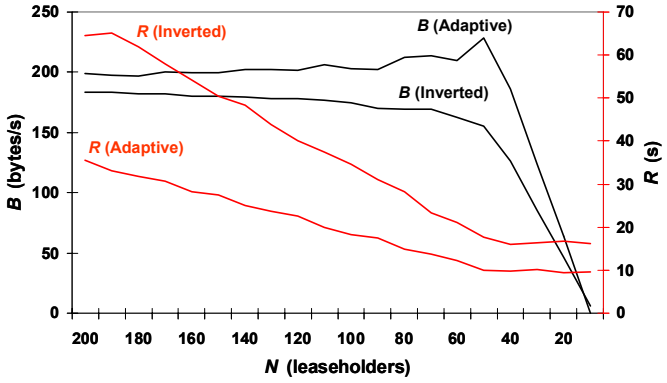


Fig. 5. Bandwidth usage (*B*) – left-hand y-axis – and system responsiveness (*R*) – right-hand y-axis – as system size decreases (*N* = 200 to 0 leaseholders). L_{MIN} = 15 s, *G* = 3 renewals per second, L_{MAX} = ∞, and (for Inverted) *A* = 0.2*D*.

5. Conclusions

We investigated Jini leasing procedures, establishing quantitative tradeoffs among responsiveness, resource consumption, system size, and granted lease period. We suggested an approach to bound bandwidth use, while guaranteeing a minimum level of responsiveness in detecting leaseholder failures. We also showed a simple adaptive leasing algorithm that bounds bandwidth consumption, while achieving the best available responsiveness as system size varies. We described an alternate algorithm that inverts the leasing process, and we showed that inverted leasing achieves only half the responsiveness guaranteed by the simple adaptive algorithm. We used simulation to show that inverted leasing adapts responsiveness more quickly and constrains bandwidth consumption better than our simple adaptive algorithm. Given the performance tradeoffs and implementation costs, we conclude that our simple adaptive leasing algorithm can yield useful performance properties with little cost. We outlined a simple technique for allocating a domain-wide resource budget among multiple lease grantors. We expect our analyses can be used to deploy Jini systems with understood leasing behavior, and we hope our ideas for adaptive leasing can provide improvements over static strategies. We argued that our adaptive leasing algorithm and related analyses should also apply in similar leasing systems, such as event subscriptions offered by UPnP.

6. References

[1] C. Gray and D. Cheriton. "Leases: an efficient fault-tolerant mechanism for distributed file cache consistency", *ACM SIGOPS Operating Systems Review, Proceedings of the Twelfth ACM symposium on Operating systems principles*, November 1989, Volume 23 Issue 5.

[2] Charles E. Perkins and Kevin Luo. "Using DHCP with computers that move", *Wireless Networks*, March 1995, Volume 1 Issue 3.

[3] Anoop Ninan, Purushottam Kulkarni, Prashant Shenoy, Krithi Ramamritham, and Renu Tewari. "Performance: Cooperative leases: scalable consistency maintenance in content distribution networks", *Proceedings of the eleventh international conference on World Wide Web*, May 2002.

[4] Jacob Harris and Vivek Sarkar. "Lightweight object-oriented shared variables for distributed applications on the Internet", *ACM SIGPLAN Notices*, *Proceedings of the conference on Object-oriented programming, systems, languages, and applications*, October 1998, Volume 33 Issue 10.

[5] Jim Waldo. "The Jini™ architecture for network-centric computing", *Communications of the ACM*, July 1999.

[6] Universal Plug and Play Device Architecture, Version 1.0, 08 Jun 2000 10:41 AM. © 1999-2000 Microsoft Corporation. All rights reserved.

[7] Ken Arnold et al, The Jini Specification, V1.0 Addison-Wesley, 1999. The latest version is available on the web from Sun.

[8] Service Location Protocol Version 2, Internet Engineering Task Force (IETF), RFC 2608, June 1999.

[9] James O. Henriksen, "An Introduction to SLX™" *Proceedings of the 1997 Winter Simulation Conference*, ACM, Atlanta, Georgia, December 7-10, 1997, pp. 559-566.

Improving Failure Responsiveness in Jini Leasing

Scott Rose, Kevin Bowers, Steve Quirolgico, and Kevin Mills
National Institute of Standards and Technology
srose@nist.gov

Abstract

Distributed systems require strategies to detect and recover from failures. Many protocols for distributed systems employ a strategy based on leases, which grant a leaseholder access to data or services for a limited time (the lease period). Choosing an appropriate lease period involves tradeoffs among resource utilization, responsiveness, and system size. We explain these tradeoffs for Jini Network Technology. Then, we describe an adaptive algorithm that enables a Jini system, given a fixed allocation of resources, to vary lease periods with system size to achieve the best responsiveness. We anticipate that similar procedures could improve failure responsiveness in other distributed systems that rely on leases. We describe how we implemented our adaptive algorithm in "reggie", a publicly available implementation of the Jini lookup service. We can use our implementation to demonstrate how adaptive leasing provides the best available responsiveness as network size varies.

1. Introduction

Distributed systems require strategies to detect and recover from failures. One commonly used strategy employs a leasing mechanism, where a node grants a leaseholder access to a resource for a limited time (the lease period). Once the resource is no longer needed, the leaseholder may relinquish its lease. If the resource is needed beyond the original lease period, then the leaseholder can renew the lease by requesting additional lease periods. If the leaseholder does not renew before expiration of the lease period, the lease grantor assumes leaseholder failure and terminates the lease.

Choosing an appropriate lease period entails tradeoffs among resource utilization, responsiveness, and number of leaseholders. We explore these issues in the context of service-discovery protocols, which allow distributed software components to discover each other and compose themselves into assemblies. Though several service-discovery protocols currently exist [e.g., 1-4], we selected Jini Network Technology [1] to demonstrate our ideas, because leasing plays a central role in registering Jini services. We base our analysis on the Jini specification [2].

2. Jini Leasing

Jini defines an architecture that enables clients and services to rendezvous through a third party, known as a lookup service. A Jini service registers a description of itself with each discovered lookup service. A Jini client may register a request to be notified by a lookup service of arriving or departing services of interest, or of changes in the attributes describing services of interest.

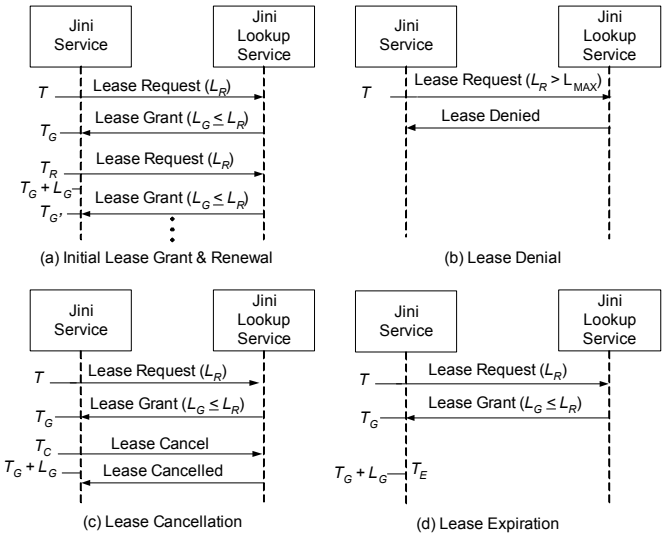


Fig. 1. Message exchanges for four Jini leasing scenarios.

Figure 1 illustrates message exchanges for some typical Jini leasing scenarios. A registering component requests registration for duration L_R, which may be accepted at time T_G for a granted lease period $L_G \le L_R$. L_R may be *any*, which allows any value for L_G. To extend registration beyond L_G, registering components must renew the lease prior to an expiration time $T_E = T_G + L_G$; otherwise, registration is revoked. This cycle continues until a Jini component cancels or fails to renew a lease. Lookup services assign L_G within a configured range, $L_{MIN} \le L_G \le L_{MAX}$. While a granted lease may not be revoked prior to T_E, lookup services may deny any lease request.

We can analyze performance of a Jini leasing system. Let S_R be lease-request size, S_G be lease-grant size, and N be

number of leaseholders. Typically, a leaseholder and lookup service exchange one request-grant pair per renewal cycle, with rate $1/L_G$ Hz. Assuming identical L_G assigned for each lease, bandwidth use (B) can be estimated as: $B=(N/L_G)\cdot(S_R+S_G)$. Assuming constant S_R and S_G, B increases linearly with N and decreases exponentially with L_G. Another metric, responsiveness, R, measures the latency with which lookup services can detect leaseholder failure. Assuming uniformly distributed failure times, then expected responsiveness is $R = L_G/2$; thus, R is independent of N, but B and R are related through L_G.

These relationships can be used to constrain and predict behavior of a leasing system. For example, assume known requirements for R and B. The responsiveness equation can be rewritten to determine L_G [i.e., $L_G = 2R$]. Then, using L_G, the bandwidth equation can be transformed to find maximum system size [i.e., $N_{MAX} = (B \cdot L_G)/(S_R + S_G)$]. With this information, lookup services could grant lease periods $\leq L_G$ to ensure required responsiveness, deny requested leases that would consume an excess share of bandwidth, and deny requests for leases once N reaches N_{MAX}.

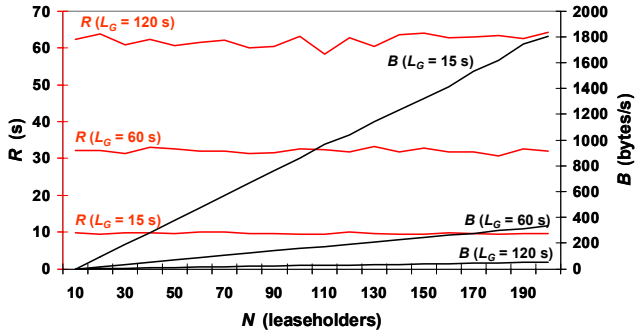


Fig. 2. System responsiveness (*R*) – left-hand y-axis – and bandwidth usage (*B*) – right-hand y-axis – for three granted lease periods (*LG* = 15 s, 60 s, and 120 s) as system size increases (*N* = 10 to 200 leaseholders).

3. A Self-adaptive Algorithm for Jini Leasing

We propose an algorithm that restricts lease requests to L_R = *any*. Assuming a leasing system must consume at most bandwidth B and guarantee minimum average responsiveness R_{MIN}, a lookup service can grant a maximum lease period $L_{MAX} = 2R_{MIN}$. Given B, S_R, and S_G, we can determine a maximum lease-renewal rate $G = B / (S_R + S_G)$. For minimum system size, $N_{MIN} = 1$, the lookup service can grant a minimum lease period $L_{MIN} = 1/G$. While this value for L_{MIN} respects the bandwidth constraint, other factors should be considered. For example, at $L_{MIN} = 1/G$ leaseholder processing burden might prove unacceptable. Instead, a leasing system might constrain maximum responsiveness (R_{MAX}), giving a minimum lease period $L_{MIN} = 2R_{MAX}$. Knowing N, a lookup service may select a suitable granted lease period from a range ($L_{MIN} \leq L_G \leq L_{MAX}$) using a simple algorithm. First, compute $L_G = N/G$. If $L_G > L_{MAX}$, then deny the lease; otherwise, if $L_G < L_{MIN}$, then set $L_G = L_{MIN}$. Assigning L_G with this algorithm permits a leasing system to constrain B and guarantee minimum average responsiveness (R_{MIN}), while providing the best responsiveness achievable (up to R_{MAX}) as N varies over 1..N_{MAX}.

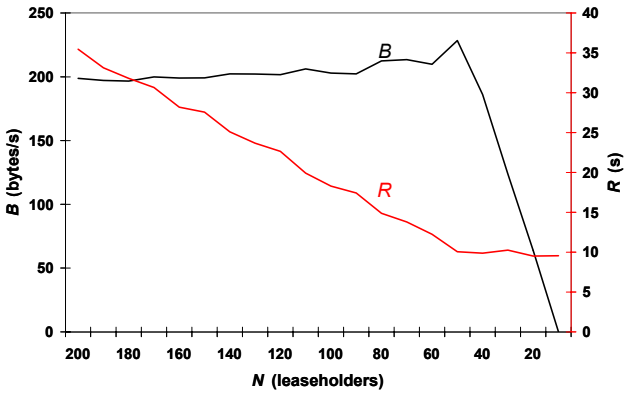


Fig. 3. Responsiveness (*R*) – left-hand y-axis – and bandwidth usage (*B*) – right-hand y-axis –as system size decreases (*N* = 200 to 0 leaseholders).

4. Simulation Results

We coded an SLX discrete-event simulation [5] model of Jini to confirm our analysis and to investigate dynamic behavior of our self-adaptive algorithm. We conducted simulation experiments, varying N from 10..200 and L_G from 15..300 s in 15-s increments. We used S_R = 128 and S_G = 32 bytes. Figure 2 shows simulated results for average B and R when L_G = 15 s, 60 s, and 120 s. The simulation confirms our analyses: (1) B increases linearly with N for a given L_G and decreases exponentially with L_G for a given N and (2) $R = L_G/2$, independent of N. Next, we simulated our adaptive leasing algorithm. Figure 3 illustrates how the algorithm constrains B while improving R as system size decreases. These promising results led us to implement our adaptive algorithm in "reggie", a publicly available implementation of a Jini lookup service.

5. Implementation in "reggie"

We base our adaptive-leasing implementation on the "reggie" lookup server provided with the Sun Microsystems Jini release. The "reggie" server implements the Jini specification for a lookup service, and includes additional extensions to allow remote administration of the lookup server through a service proxy. Administrative actions occur through the *RegistrarAdmin* interface, which is not part of the

Jini core specification, but a Sun extension to Jini (**com.sun.jini.reggie** package). The *RegistrarAdmin* interface allows basic monitoring, configuration, and control of an operational lookup server just as if it were any other type of Jini-enabled service. Using the *RegistrarAdmin* interface, an administrator can also perform some basic manipulation of minimum and maximum granted lease periods for services and events maintained on the lookup server. However, this method requires constant human supervision to optimize leasing performance in a Jini network. Such human supervision would prove impractical in a large network where numerous services may join and leave.

To implement adaptive leasing, we modified the "reggie" server implementation to assign lease grant times (L_G) based on the required failure responsiveness (R) of the system and the bandwidth (B) allocated to lease renewal transactions. We added a collection of access methods to the *RegistrarAdmin* interface, callable via remote-method invocation (RMI), allowing a Jini client to view: the current L_{MIN}, L_{MAX}, and L_G, the number of leaseholders (N) on the server, the instantaneous average bandwidth (B_{AVG}) consumed by lease renewals, and the instantaneous average failure responsiveness (R_{AVG}). Since values for granted lease periods can be adjusted from changes to the allocated bandwidth and target responsiveness, we added methods to set B and R in the *RegistrarAdmin* interface.

The lookup server (**com.sun.jini.reggie.RegistrarImpl**) starts with default values for L_{MIN} and L_{MAX}. A Jini client can use the *RegistrarAdmin* interface to adjust target responsiveness and allocated bandwidth. Based on these adjustments, the lookup server computes new values for L_{MIN} and L_{MAX}. At regular intervals, the lookup service samples average bandwidth use (B_{AVG}) and the number of leaseholders, adjusting granted lease periods (L_G) accordingly. Current bandwidth usage is calculated by multiplying the number of lease transactions (RMI calls) by the size of messages involved in the transaction. Currently, the lookup service records these values once every sixty seconds of operation, or when an administrator changes R or B.

When a Jini service registers with the lookup service, it may either request a specific lease interval or use Jini's *LEASE.ANY* constant to allow the lookup server to select an appropriate lease period for the service. In our implementation, if the service requests the *LEASE.ANY* constant, the lookup service uses the current value for L_G as the granted lease period. Otherwise, if the service requests a lease period in the range of L_{MIN} and L_{MAX}, it is granted. The lookup service rejects requests for leases outside this range.

Figure 4 shows a snapshot of a Jini client graphical user interface (GUI) that uses the modified "reggie" *RegistrarAdmin* interface. The left-hand column plots values for L_G, for bandwidth used (B_{AVG}), and for average responsiveness (R_{AVG}) over time. These graphs display values returned to the client from regular polling of access methods *RegistrarAdmin* interface, which uses RMI to call the corresponding method in the lookup server. The right-hand column of Figure 4 lists leaseholders using the lookup service, and displays their current status in the Jini network. Note that the GUI displays only the current *known* status, as a service may have left the network, but a proxy could still be registered with the server. The proxy would be purged when its lease expires. The GUI in Figure 4 does not include the *RegistrarAdmin* GUI used to adjust allocated bandwidth or target responsiveness in the lookup service.

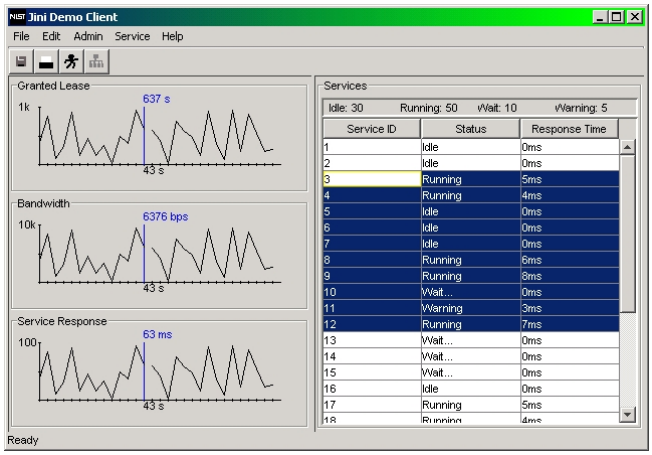


Fig. 4. Sample Graphical User Interface for a Jini client monitoring a lookup server that implements adaptive leasing

6. Acknowledgments

The work described in this paper is funded in part by the DARPA Fault-Tolerant Networks program and the NIST Pervasive Computing program. We gratefully acknowledge support from Dr. Doug Maughan of DARPA and Dr. Susan Zevin of NIST.

7. References

[1] Jim Waldo. "The Jini™ architecture for network-centric computing", *Communications of the ACM*, July 1999.

[2] Universal Plug and Play Device Architecture, Version 1.0, 08 Jun 2000 10:41 AM. © 1999-2000 Microsoft Corporation. All rights reserved.

[3] Ken Arnold et al, The Jini Specification, V1.0 Addison-Wesley, 1999. The latest version is available on the web from Sun.

[4] Service Location Protocol Version 2, Internet Engineering Task Force (IETF), RFC 2608, June 1999.

[5] James O. Henriksen, "An Introduction to SLX™" *Proceedings of the 1997 Winter Simulation Conference*, ACM, Atlanta, Georgia, December 7-10, 1997, pp. 559-566.

Self-Managed Leasing for Distributed Systems

Kevin Bowers
Renssalaer Polytechnic Institute
bowerk@rpi.edu

Kevin Mills, Steve Quirolgico, and Scott Rose
National Institute of Standards and Technology
{kmills, steveq, scottr}@nist.gov

ABSTRACT
We describe an adaptive algorithm that enables a distributed Jini enabled system, given a fixed allocation of resources, to vary lease periods to achieve the best responsiveness.

Keywords
algorithm, self-managing, performance, Jini, leasing

1. INTRODUCTION
Distributed systems require strategies to detect and recover from failures. One commonly used strategy employs a leasing mechanism, where a node grants a leaseholder access to a resource for a limited time (the lease period). Once the resource is no longer needed, the leaseholder may relinquish its lease. If the resource is needed beyond the original lease period, then the leaseholder can renew the lease by requesting additional lease periods. If the leaseholder does not renew before expiration of the lease period, the lease grantor assumes leaseholder failure and terminates the lease.

Choosing an appropriate lease period entails tradeoffs among resource utilization, responsiveness, and number of leaseholders. We explore these issues in the context of service-discovery protocols, which allow distributed software components to discover each other and compose themselves into assemblies. Though several service-discovery protocols currently exist [e.g., 1-3], [5] we selected Jini Network Technology [1] to demonstrate our ideas, because leasing plays a central role in registering Jini services.

2. JINI LEASING
Jini defines an architecture that enables clients and services to rendezvous through a third party, known as a lookup service. A Jini service registers a description of itself with each discovered lookup service.

A registering component requests registration for duration L_R, which may be accepted at time T_G for a granted lease period $L_G \leq L_R$. L_R or may be *any*, which allows any value for L_G. To extend registration beyond L_G, registering components must renew the lease prior to an expiration time $T_E = T_G + L_G$; otherwise, registration is revoked. This cycle continues until a Jini component cancels or fails to renew a lease. Lookup services assign L_G within a configured range, $L_{MIN} \leq L_G \leq L_{MAX}$. While a granted lease may not be revoked prior to T_E, lookup services may deny any lease request.

We can analyze performance of a Jini leasing system. Let S_R be lease-request size, S_G be lease-grant size, and N be number of leaseholders. Typically, a leaseholder and lookup service exchange one request-grant pair per renewal cycle, with rate $1/L_G$ Hz. Assuming identical L_G assigned for each lease, bandwidth use (B) can be estimated as: $B=(N/L_G)\cdot(S_R+S_G)$. Assuming constant S_R and S_G, B increases linearly with N and decreases exponentially with L_G. Another metric, responsiveness, R, measures the latency with which lookup services can detect leaseholder failure. Assuming uniformly distributed failure times, then expected responsiveness is $R = L_G/2$; thus, R is independent of N, but B and R are related through L_G.

These relationships can be used to constrain and predict behavior of a leasing system. For example, assume known requirements for R and B. The responsiveness equation can be rewritten to determine L_G [i.e., $L_G = 2R$]. Then, using L_G, the bandwidth equation can be transformed to find maximum system size [i.e., $N_{MAX} = (B \cdot L_G)/(S_R + S_G)$]. With this information, lookup services could grant lease periods $\leq L_G$ to ensure required responsiveness, deny requested leases that would consume an excess share of bandwidth, and deny requests for leases once N reaches N_{MAX}.

3. A SELF-ADAPTIVE ALGORITHM FOR JINI LEASING
Assuming a leasing system must consume at most bandwidth B and guarantee minimum average responsiveness R_{BEST}, a lookup service can grant a maximum lease period $L_{MAX} = 2R_{BEST}$. Given B, S_R, and S_G, we can determine a maximum lease-renewal rate $G = B / (S_R + S_G)$. For minimum system size, $N_{MIN} = 1$, the lookup service can grant a minimum lease period $L_{MIN} = 1/G$. While this value for L_{MIN} respects the bandwidth constraint, other factors should be considered. For example, at $L_{MIN} = 1/G$ leaseholder processing burden

might prove unacceptable. Instead, a leasing system might constrain maximum responsiveness (R_{WORST}), giving a minimum lease period $L_{MIN} = 2R_{WORST}$. Knowing N, a lookup service may select a suitable granted lease period from a range ($L_{MIN} \leq L_G \leq L_{MAX}$) using a simple algorithm. First, compute $L_G = N/G$. If $L_G > L_{MAX}$, then deny the lease; otherwise, if $L_G < L_{MIN}$, then set $L_G = L_{MIN}$. Assigning L_G with this algorithm permits a leasing system to constrain B and guarantee minimum average responsiveness (R_{BEST}), while providing the best responsiveness achievable (up to R_{WORST}) as N varies over $1..N_{MAX}$.

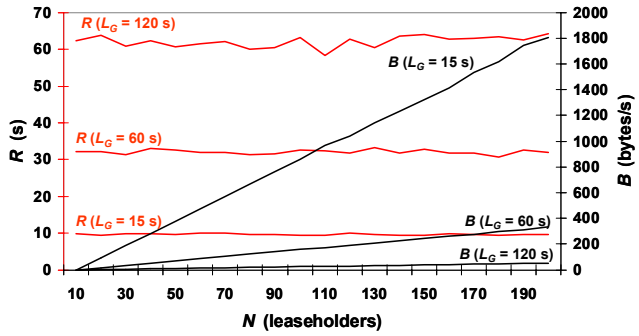


Fig. 1. System responsiveness (*R*) – left-hand y-axis – and bandwidth usage (*B*) – right-hand y-axis – for three granted lease periods (*LG* = 15 s, 60 s, and 120 s) as system size increases (*N* = 10 to 200 leaseholders).

4. SIMULATION RESULTS

We coded an SLX discrete-event simulation [5] model of Jini to confirm our analysis and to investigate dynamic behavior of our self-adaptive algorithm. We conducted simulation experiments, varying N from 10..200 and L_G from 15..300 s in 15-s increments. We used $S_R = 128$ and $S_G = 32$ bytes. Figure 2 shows simulated results for average B and R when L_G = 15 s, 60 s, and 120 s. The simulation confirms our analyses: (1) B increases linearly with N for a given L_G and decreases exponentially with L_G for a given N and (2) $R = L_G/2$, independent of N. Next, we simulated our adaptive leasing algorithm. Figure 3 illustrates how the algorithm constrains B while improving R as system size decreases.

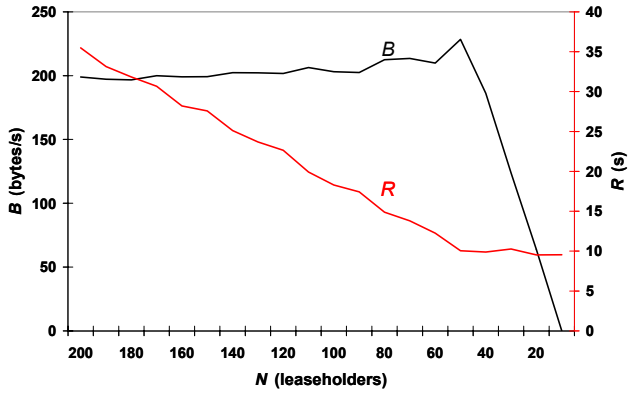


Fig. 2. Responsiveness (*R*) – left-hand y-axis – and bandwidth usage (*B*) – right-hand y-axis –as system size decreases (*N* = 200 to 0 leaseholders).

5. IMPLEMENTATION

These promising results led us to implement our adaptive algorithm in "reggie", a publicly available implementation of a Jini lookup service. Administrative actions occur through the *RegistrarAdmin* interface, which is not part of the Jini core specification, but a Sun extension to Jini. The *RegistrarAdmin* interface allows basic monitoring, configuration, and control of an operational lookup server just as if it were any other type of Jini-enabled service.

To implement self-managed leasing, we modified the "reggie" server code to assign lease-grant times (L_G) based on an administrator-assigned policy specified by two target values: worst-case average failure responsiveness (R_{WORST}) and average bandwidth (B) allocated to lease renewal transactions. We added a collection of access methods to the *RegistrarAdmin* interface, allowing a Jini client to view: the current L_{MIN}, L_{MAX}, and L_G, the number of leaseholders (N) on the server, the instantaneous average bandwidth (B_{AVG}) consumed by lease renewals, and the instantaneous average failure responsiveness (R_{AVG}).

Results from our live experiment are similar to the results we obtained from simulations. For example, the behavior of B_{AVG} (Bandwidth) and R_{AVG} (Responsiveness) were similar to the simulation results when services were added, then removed from the network.

6. FUTURE WORK

Given the performance tradeoffs and implementation costs, we conclude that our simple adaptive leasing algorithm can yield useful performance properties at little cost. We argue that our adaptive leasing algorithm should also apply to similar systems that employ leasing for resources, such as UPnP event subscriptions [2]. The modifications are done on the resource provider side, and any system that allows for flexible lease times should be able to take advantage of this algorithm.

7. REFERENCES

[1] Jim Waldo. "The Jini™ architecture for network-centric computing", *Communications of the ACM*, July 1999.

[2] Universal Plug and Play Device Architecture, Version 1.0, 08 Jun 2000 10:41 AM. © 1999-2000 Microsoft Corporation. All rights reserved.

[3] Ken Arnold et al, The Jini Specification, V1.0 Addison-Wesley, 1999. The latest version is available on the web from Sun.

[4] James O. Henriksen, "An Introduction to SLX™," *Proceedings of the 1997 Winter Simulation Conference*, ACM, Atlanta, Georgia, December 7-10, 1997, pp. 559-566.

[5] A. Ninan, P. Kulkarni, P. Shenoy, K. Ramamiritham, and R. Tewari. "Performance: Cooperative Leases: Scalable Consistency Maintenance in Content Distribution Networks", *Proceedings of the eleventh International Conference on World Wide Web*, May 2002.

An Autonomic Failure-Detection Algorithm*

K. Mills, S. Rose, S. Quirolgico
NIST
Mail Stop 892
Gaithersburg, Maryland 20899
1-301-975-3618
{kmills, scottr, steveq}@nist.gov

M. Britton
Southern Methodist University
1265 Columbine Lane
Salina, Kansas 67401
1-214-641-9914
mbritton@mail.smu.edu

C. Tan
Montgomery Blair High School
403 Branch Drive
Silver Spring, MD 20901
1-301-593-8132
cetan@mbhs.edu

ABSTRACT

Designs for distributed systems must consider the possibility that failures will arise and must adopt specific failure detection strategies. We describe and analyze a self-regulating failure-detection algorithm that bounds resource usage and failure-detection latency, while automatically reassigning resources to improve failure-detection latency as system size decreases. We apply the algorithm to (1) Jini leasing, (2) service registration in the Service Location Protocol (SLP), and (3) SLP service polling.

1. INTRODUCTION

Recent research on failure detection and recovery in distributed systems reports non-functional periods comprising two distinct phases: periods when a system is unaware of a failure (failure-detection latency) and periods when a system attempts to recover from a failure (failure-recovery latency)[1]. Depending on system architecture and assumptions about failure characteristics of components, the study found failure-detection latencies covered from 55% to 80% of non-functional periods. The study also revealed failure detection can consume substantial overhead. These findings suggest distributed systems could benefit from failure detection algorithms that exhibit definite bounds on latency and overhead. We define and analyze such an algorithm, and then apply it to Jini leasing and to service registration and polling in SLP.

2. AUTONOMIC FAILURE-DETECTION

Figure 1 illustrates a two-way heartbeat failure-detection technique, where N *monitorables* each issue a rising heartbeat (every H_P seconds) to one *monitor*, which replies with a falling heartbeat. Assuming rising and falling heartbeat messages of known size (S_R and S_F, respectively), the system consumes network bandwidth $B = N (S_R+S_F)/H_P$. The monitor must process N/H_P heartbeat messages per second. The monitorable must process $1/H_P$ heartbeat messages per second.

Figure 2 defines the period of inconsistency when a monitorable fails between heartbeats. Should a monitorable fail immediately after receiving a falling heartbeat from a monitor, then the maximum failure-detection latency (L_{MAX}) is defined by the heartbeat period, i.e., $L_{MAX} = H_P$. Assuming a monitorable is equally likely to fail at any time, the average failure-detection latency (L_{AVG}) is half the heartbeat period, $L_{AVG} = H_P/2$.

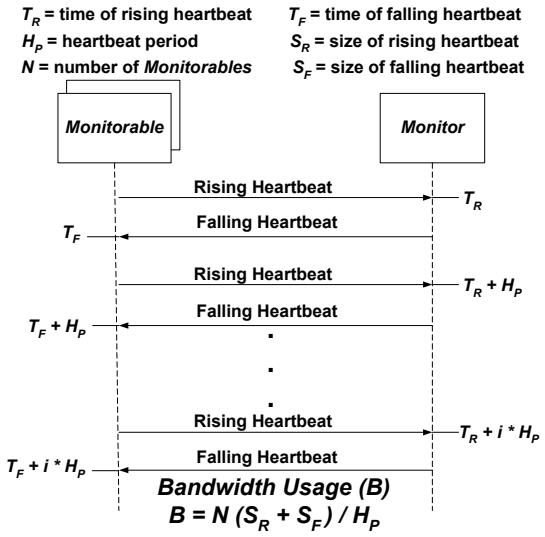


Figure. 1. Fundamental outlines of a two-way, heartbeat-based, failure-detection technique.

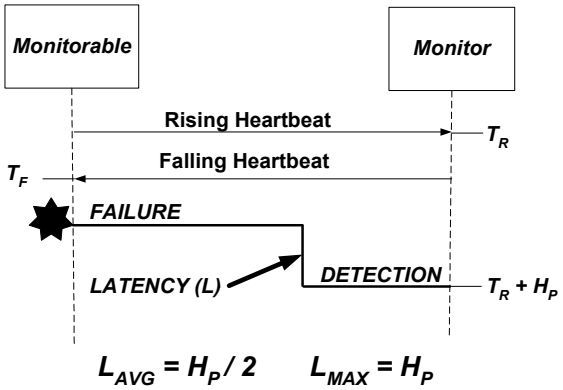


Figure 2. Defining failure-detection latency for heartbeat-based failure-detection techniques.

For monitor failure, where detection occurs when the monitor does not respond with a falling heartbeat, the situation differs slightly. A monitorable may wait for some timeout period (T_O) before concluding the monitor has failed; thus, the maximum detection latency for monitor failure is $H_P + T_O$, and the average detection latency is $H_P/2 + T_O$.

We define an autonomic algorithm to limit bandwidth usage to an allocated capacity (B_A) and to limit average failure-detection

latency (to L_{WORST}), while reducing average failure-detection latency ($L_{AVG} < L_{WORST}$) to some lower bound (L_{BEST}) when the number of monitorables (N) falls below system capacity (N_{MAX}). We modify the two-way heartbeat technique so that the monitor includes a heartbeat period (H_P) in each falling heartbeat. The monitorable uses H_P to determine when to issue the next rising heartbeat. The monitor may vary H_P with each falling heartbeat to maintain an operating range defined by three policy goals: the average failure-detection latency in the worst (L_{WORST}) and best (L_{BEST}) cases and the allocated bandwidth (B_A).

```
if new monitorable then N++;
H_P = N / C;
if  H_P > H_MAX
        then  N--;
              raise capacity exception;
        elseif H_P < H_MIN
              then H_P = H_MIN;
        endif
endif
```

Figure 3. Autonomic algorithm to vary heartbeat period.

Assuming N monitorables, the monitor varies the heartbeat period ($H_{MIN} \leq H_P \leq H_{MAX}$) using the algorithm in Figure 3.The maximum heartbeat period (H_{MAX}) is set to twice the worst-case average failure-detection latency ($H_{MAX} = 2\ L_{WORST}$). Given H_{MAX} and the monitor's capacity [$C = B_A/(S_R + S_F)$], a monitor can watch at most $N_{MAX} = H_{MAX}\ C$ monitorables. Assuming a monitor watches at least one monitorable, a natural choice for H_{MIN} would be $1/C$; however, this heart rate might place too great a load on individual monitorables. Instead, we choose a best-case goal (L_{BEST}) for average failure-detection latency and set $H_{MIN} = 2\ L_{BEST}$. H_{MIN} establishes a lower bound on failure-detection latency.

Below, we report analytical results as time-series plots (time increases from 0 to 400) with N first increasing to 200, and then decreasing back to 0. In all plots (Figure 4) we assume the same heartbeat sizes (S_R = 128 bytes and S_F = 64 bytes) and policy goals (L_{WORST} = 30 s, L_{BEST} = 7.5 s, and B_A = 576 bytes/s). The top plot shows our algorithm limits monitor workload to N_{MAX}, while the second plot illustrates our algorithm adjusting H_P between H_{MIN} and H_{MAX} as N varies. The third plot shows how average bandwidth increases and decreases but never exceeds allocated bandwidth (B_A). The bottom plot illustrates how our algorithm improves average failure-detection latency (L) as N decreases (ticks 200 to 400), while average failure-detection latency never exceeds the worst (L_{WORST}) and best (L_{BEST}) cases.

3. SAMPLE APPLICATIONS

We apply our algorithm to selected functions in two service-discovery protocols: Jini [2] and the Service Location Protocol (SLP) [3]. ***Jini*** enables clients and services to rendezvous through a third party, known as a *lookup service*. Each Jini service registers a description of itself with each discovered lookup service, and requests a lease duration (L_R), which may be accepted at time T_G for a granted lease period $L_G \leq L_R$. L_R may be "any", which allows a lookup service to assign any value for L_G. To extend registration beyond L_G, services must renew the lease prior to an expiration time $T_E = T_G + L_G$; otherwise, registration is

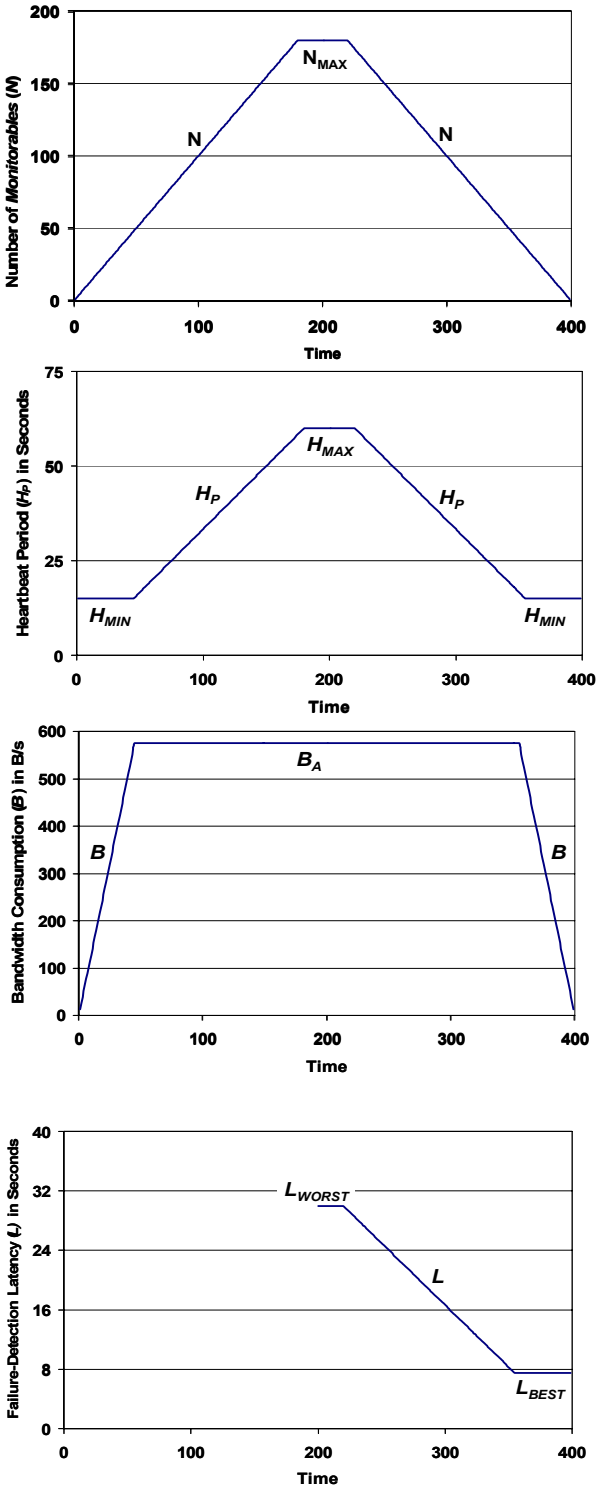


Figure 4. Time-series plots from an analytical model of the proposed autonomic failure-detection algorithm.

revoked. This cycle continues until a service cancels or fails to renew a lease. Lookup services assign L_G within a configured range, $L_{MIN} \leq L_G \leq L_{MAX}$. While a granted lease may not be revoked prior to T_E, lookup services may deny any lease request.

We apply our algorithm to enable Jini lookup services to vary L_G within a bounded range ($L_{MIN} \leq L_G \leq L_{MAX}$) while limiting resource consumption associated with lease renewal. The mapping is straightforward. Assuming three policy goals, bandwidth allocated (B_A) and worst (L_{WORST}) and best (L_{BEST}) failure-detection latencies, we compute $L_{MIN} = H_{MIN} = 2\ L_{BEST}$ and $L_{MAX} = H_{MAX} = 2\ L_{WORST}$. Knowing the size of the rising (lease request) and falling (lease grant) heartbeats (S_R and S_F, respectively), leasing capacity (C) is computed as before. Knowing the number of registered services (N), a Jini lookup service uses the algorithm in Figure 3 to compute H_P and then uses that value as the granted lease period ($L_G = H_P$). If the new lease would exceed system capacity, then the lookup service issues a *LEASE_DENIED* exception.

To verify our analysis, we implemented our algorithm in a Jini simulation and compared simulation results against analytical predictions, given a selected set of policy goals and known sizes for Jini messages. We subsequently implemented our algorithm in a publicly available implementation of the Jini lookup service. We modified the lookup service code to accept our policy goals and to measure and report average bandwidth usage (B), the number of registered services (N), and the value for L_G. We deployed our modified Jini lookup service in a test bed built to control and monitor thousands of Jini services. We coded a measurement client to detect service arrivals and departures, computing average failure-detection latency (L). We measured behavior of a live Jini system using the same policy goals selected for analysis and simulation. We report our results in Figure 5 as four time-series plots, where we used the same protocol parameters (S_R = 350 bytes and S_F = 350 bytes) and policy goals (B_A = 2100 bytes/second, L_{BEST} = 7.5 seconds and L_{WORST} = 1200 seconds, and so N_{MAX} = 7200) for the analysis, the simulation (1000 repetitions per data point), and the live system (20-30 repetitions per data point).

SLP Service Registration. SLP enables clients, called user agents (UAs), and services, called service agents (SAs), to rendezvous through a third party, known as a directory agent (DA). A SLP SA registers a description of itself with each discovered DA. A UA may query any discovered DAs to find services of interest and to obtain attributes that describe services.

A SA requests registration for a time-to-live (TTL_R), which may be accepted by a DA at time T_G. To extend registration beyond TTL_R, the registering SA must renew the registration prior to an expiration time $T_E = T_G + TTL_R$; otherwise, the DA revokes the registration. This cycle continues until the SA cancels or fails to refresh a registration. While an accepted registration may not be revoked prior to T_E, a DA may deny any registration request. A DA will always deny a registration request when TTL_R is too small, as determined by comparing the TTL_R against a minimum-refresh interval (RF_{MIN}) included within advertisements multicast by the DA at a periodic rate (DA_{BEAT}). We apply our algorithm to provide SLP service registration with rapid feedback, eliminating the need for RF_{MIN}.

We add a field (TTL_G) to the *SrvAck* message, used by DAs to acknowledge service-registration (*SrvReg*) messages from SAs. Then, a DA can ignore TTL_R and instead compute a granted time-to-live (TTL_G), which can vary dynamically within a bounded range ($TTL_{MIN} \leq TTL_G \leq TTL_{MAX}$) while limiting resource consumption associated with refreshing service registrations. The

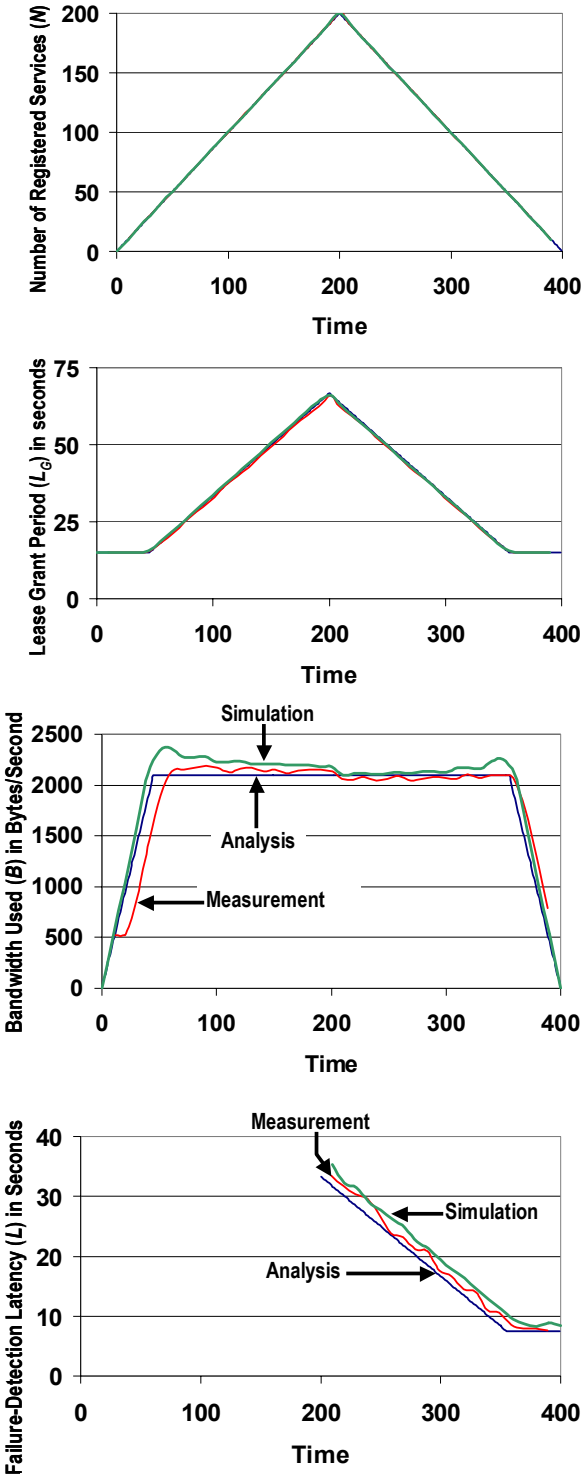


Figure 5. Time-series plots showing application of autonomic failure-detection to Jini leasing procedures

mapping is straightforward. Assuming our three policy goals, bandwidth allocated (B_A) and worst (L_{WORST}) and best (L_{BEST}) failure-detection latencies, we compute $TTL_{MIN} = H_{MIN} = 2\ L_{BEST}$ and $TTL_{MAX} = H_{MAX} = 2\ L_{WORST}$. Knowing the size of the rising (*SrvReg*) and falling (*SrvAck*) heartbeat messages (S_R and S_F, respectively), registration capacity (C) is computed as before.

Knowing the number of registered services (N), a DA can use the algorithm in Figure 3 to compute H_P and then use that value as the granted time-to-live ($TTL_G = H_P$). If a new registration would exceed system capacity, then the DA issues the *SrvAck* with a status code of *DA_BUSY_NOW*.

To verify our analysis, we implemented our algorithm in a SLP simulation and compared simulation results against analytical predictions. We report our results in Figure 6 as four time-series plots, where we used the same protocol parameters (S_R = 76 bytes and S_F = 56 bytes) and policy goals (B_A = 396 bytes/s, L_{BEST} = 7.5 seconds and L_{WORST} = 500 seconds) for the analysis and the simulation. Setting L_{WORST} = 500 seconds and B_A = 396 bytes/s provides a maximum system capacity of N_{MAX} = 3000 registered services. In the main, Figure 6 shows a close correspondence between analytical predictions and simulation results; however, the bandwidth-usage simulation plot (as well as the bandwidth-usage simulation plot for Jini leasing procedures - recall Figure 5) illustrates a hysteresis within the control loop of our proposed algorithm. During periods of increasing system size, the algorithm typically assigns a heartbeat period that immediately becomes too small for the now increased system size, and will only be able to reduce the heartbeat period one monitorable at a time, as each previously assigned heartbeat expires. This lag causes the algorithm to slightly overshoot the allocated bandwidth. The larger the heartbeat message size the greater the overshoot. For example, the Jini plot (700 bytes per heartbeat) overshoots allocated bandwidth more than the SLP plot (132 bytes per heartbeat). However, the downward slope in the bandwidth-usage simulation plots (as system size increases from 50 to 200) suggests that the algorithm will stabilize bandwidth usage at the allocated bandwidth once the system size stabilizes.

SLP UA Polling. SLP UAs must poll DAs periodically to learn about service arrivals and departures or about changes in attribute values of service descriptions. SLP includes no mechanisms through which DAs can control the polling rate of UAs. We can modify DA procedures to determine which UAs are polling a DA, and then we can apply our algorithm to assign polling intervals to those UAs. First, we explain the modified DA procedures.

When a UA queries a DA, either using a service-request (*SrvRqst*) or attribute-request (*AttrRqst*) message, we modify the DA procedures to lookup the UA in a local DA cache. If the UA is not found, then the DA creates a new cache entry for the UA; otherwise, the DA uses the existing cache entry. The DA grants the cache entry a polling interval (P_G), which can vary dynamically within a bounded range ($P_{MIN} \leq P_G \leq P_{MAX}$) while limiting resource consumption associated with UA polling. We modify the format of the appropriate reply message, either the service reply (*SrvRply*) or the attribute reply (*AttrRply*), to include a field to hold P_G for return to the UA. If the UA fails to issue another query to the DA by the time P_G expires, then the DA purges the associated entry from the local cache of UAs. Upon receiving P_G in the reply message, the UA schedules its next poll (if any) of the DA to occur slightly before P_G expires.

The DA can use our algorithm to determine a suitable value for P_G. The mapping is straightforward. Assuming our three policy goals, bandwidth allocated (B_A) and worst (L_{WORST}) and best (L_{BEST}) failure-detection latencies, we compute $P_{MIN} = H_{MIN} = 2\ L_{BEST}$ and $P_{MAX} = H_{MAX} = 2\ L_{WORST}$. Estimating an average size for the rising (*SrvRqst* or *AttrRqst*) and falling (*SrvRply* or *AttrRply*) heartbeat messages (S_R and S_F, respectively), registration capacity (C) is computed as before. Knowing the number of polling clients

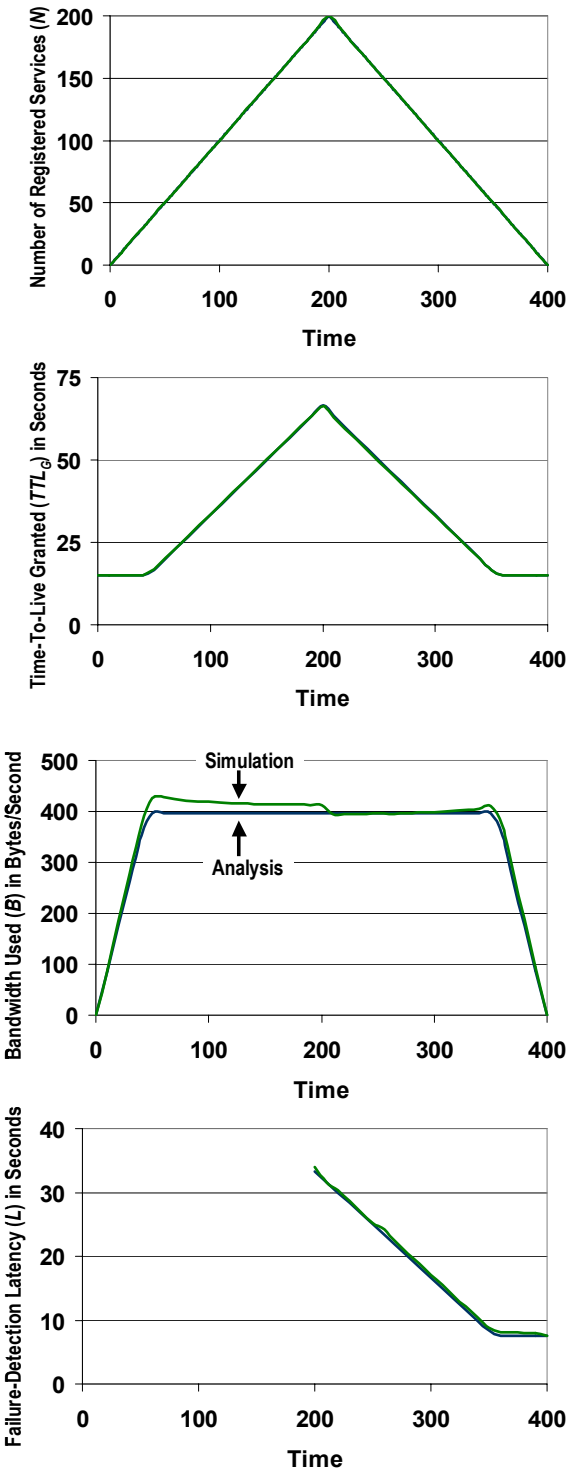


Figure 6. Time-series plots showing application of autonomic failure-detection to SLP service-registration refresh procedures

(N), a DA can use the algorithm in Figure 3 to compute H_P and

then use that value as the assigned polling interval ($P_G = H_P$). If a new polling UA would exceed system capacity, then the DA issues the *SrvRply* or Attr*Rply* with a status code of *DA_BUSY_NOW*.

To verify our analysis, we implemented our algorithm in a SLP simulation and compared simulation results against analytical predictions, given a selected set of policy goals and estimated sizes for SLP messages. We devised a specific polling algorithm. Upon discovering a DA, a UA first issues one *SrvRqst* message (receiving a *SrvRply* from the DA) and then an *AttrRqst* message (receiving a *AttrRply* from the DA). The DA grants a P_G only for each *AttrRply* message, and the UA polls only with *AttrRqst* messages. In other words, on initial discovery a UA and DA exchange four messages (*SrvRqst-SrvRply-AttrRqst-AttrRply*), and then the UA and DA periodically exchange two messages (*AttrRqst-AttrRply*). We modified our analytical model to account for these polling procedures.

We report our results in Figure 7 as four time-series plots, where we used the same protocol parameters (average S_R = 77 bytes and average S_F = 128 bytes) and policy goals (B_A = 615 bytes/s, L_{BEST} = 7.5 seconds and L_{WORST} = 500 seconds) for the analysis and the simulation (1000 repetitions per data point). Setting L_{WORST} = 500 seconds and B_A = 615 bytes/s provided a maximum system capacity of N_{MAX} = 3000 registered services. For the simulation, we sampled individual message sizes from a distribution for each *AttrRqst* and *AttrRply*. The distribution parameters for an *AttrRqst* were: 4 bytes minimum, 256 bytes maximum, 77 bytes average, and 138 bytes variance. The distribution parameters for an *AttrRqst* were: 64 bytes minimum, 224 bytes maximum, 128 bytes average, and 25 bytes variance.

Figure 7 shows a close correspondence between analytical predictions and simulation results, and also again illustrates the hysteresis associated with the bandwidth-allocation control loop. Here, the overshoot is worse than for SLP service registration because the SLP polling heartbeat message sizes are greater (205 bytes on average compared with 132 bytes). The bandwidth-usage overshoot is lower than for Jini leasing, however, because the SLP polling heartbeat messages are smaller (205 bytes on average compared with 350 bytes). Further, the overshoot for SLP polling is somewhat exaggerated because the initial four-message exchange prior to the polling heartbeats is included in the bandwidth-usage during periods of increasing system size. This can also be seen in the results from our modified analytical model, which overshoots the allocated bandwidth (615 bytes/s). This effect diminishes as the granted polling interval (P_G) increases, as can be seen in the downward slope in both the analytical predictions and simulation results.

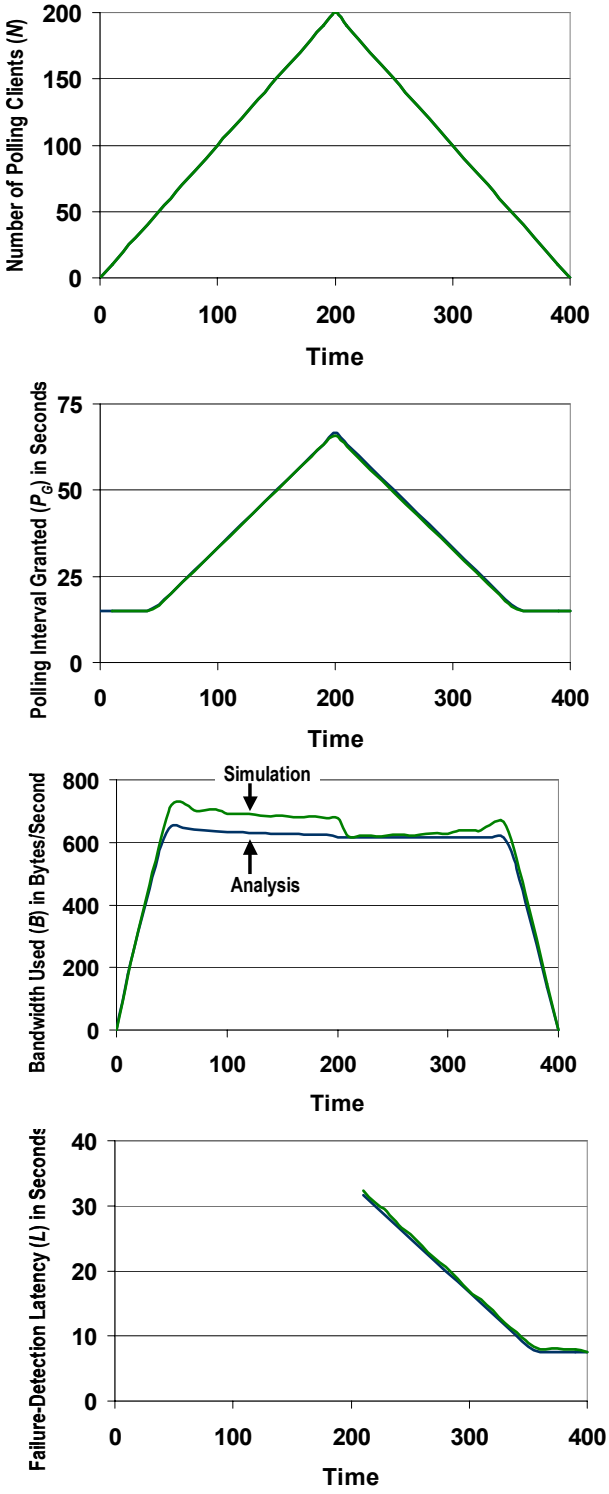


Figure 7. Time-series plots showing application of autonomic failure-detection to SLP UA polling

4. REFERENCES

[1] C. Dabrowski, K. Mills, and A. Rukhin A. Performance of Service-Discovery Architectures in Response to Node Failures, *Proceedings of the 2003 International Conference on Software Engineering Research and Practice* (SERP'03), CSREA Press, June 2003, pp. 95-101.

[2] K. Arnold et al, The Jini Specification, V1.0 Addison-Wesley 1999. Latest version is 1.1 available from Sun.

[3] Service Location Protocol Version 2, Internet Engineering Task Force (IETF), RFC 2608, June 1999.

Performance Characterization of Decentralized Algorithms for Replica Selection in Distributed Object Systems*

Ceryen Tan
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
1-617-253-1000
ctan@mit.edu

Kevin Mills
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
1-301-975-3618
kmills@nist.gov

ABSTRACT
Designers of distributed systems often rely on replicas for increased robustness, scalability, and performance. Replicated server architectures require some technique to select a target replica for each client transaction. In this paper, we use simulation to characterize performance (response time, selection error, probability of server overload) for four common replica-selection algorithms (random, greedy, partitioned, weighted) when applied in a decentralized form to client queries in a distributed object system deployed on a local network. We introduce two new selection algorithms (balanced and balanced-partitioned) that give improved performance over the more common algorithms. We find the weighted algorithm performs best among the common algorithms and the balanced algorithm performs best among all those we considered. Our findings should help designers of distributed object systems to make informed decisions when choosing among available replica-selection algorithms.

Categories and Subject Descriptors
D.1.3 [**Concurrent Programming**]: Distributed Programming

General Terms
Algorithms, Design, Measurement, Performance

Keywords
Distributed Object Systems, Replica Selection.

1. INTRODUCTION
Designers of distributed systems often rely on replicas for increased robustness, scalability, and performance. Replication appears in a growing range of applications, such as web services [1-15], distributed object systems [16-20], grid systems [21-22], and content distribution networks [25-26]. Replication systems require that each client transaction be assigned to a specific server replica for processing. Selection (or assignment) algorithms aim to minimize client response time, to balance server load, or to achieve a combination. Typical commercial systems for server replication [10-15] allow a designer to choose among several alternate selection algorithms; however, the designer is given little quantitative information to aid in choosing. At best, commercial systems outline heuristics to differentiate among available algorithms. Even academic papers [e.g., 1-9] do not give comprehensive quantitative results.

In this paper, we aim to help designers understand quantitative performance differences (and underlying causes) among the most common algorithms (*random*, *greedy*, *partitioned*, and *weighted*) for replica selection. We also introduce two new algorithms (*balanced* and *balanced-partitioned*), and compare performance with the more common algorithms. We consider three performance characteristics: average client response time, probability of selection error, and probability of server overload.

Section 2 surveys common selection algorithms typically implemented in commercial systems and identifies some algorithms proposed by researchers. Section 3 explains the design of our experiment, including performance metrics. Section 4 presents simulation results, which are discussed in Section 5. We conclude in Section 6.

2. REPLICA SELECTION
Our literature survey revealed two classes of replica-selection algorithms. One class encompasses heuristically based, statically configured algorithms. One static algorithm uses a *round robin* approach [10,13,15] to rotate client transactions in turn among replicas. A similar algorithm (using a uniform distribution) *randomly assigns* [10,11,13] each client transaction to one of the available replicas. These two algorithms assume that each replica has similar processing power available and that the mix of transaction types is congruent among the client population. Absent these assumptions, the round robin and random algorithms could perform poorly; however, no dynamic measurements are needed for either algorithm. A third approach uses a *proportional* algorithm [13,14], which distributes client transactions among replicas in proportion to relative power ratings assigned by a system administrator. This accounts for variation in processing power when a server population consists of heterogeneous platforms. Here, some information must be collected (off-line) and encoded for use by the algorithm, which cannot adapt should configuration information prove inaccurate or transient. Our experiments investigate algorithms that dynamically adjust assignment of client transactions based on measured conditions; thus, we do not consider statically configured approaches. We do simulate random assignment as a baseline case.

A second class of selection algorithms dynamically assigns client transactions based on measured conditions. The most common approach, *greedy* selection, [3,7,9-11,14,18, 23-25] assigns each transaction to the replica estimated to give best performance against some metric (different systems adopt

WOSP'05, July 12–14, 2005, Palma, de Mallorca, Spain.
ACM 1-59593-087-6/05/0007

different metrics). Greedy selection exhibits a well-known undesirable behavior where transactions oscillate in groups among available replicas. To combat this "thundering herd" effect, some systems incorporate a *weighted* algorithm [1,8,9,11,15] that first estimates the performance of each replica against a selected metric and then distributes client transactions in proportion to the likelihood that each replica will provide acceptable performance. Some systems first *partition* [1,2,5,15-17,20] replicas (based on estimated performance against some metric) into two groups, available and unavailable, and then, using greedy [2,15,20], weighted [1], random [5,16], or multicast [17] selection, assign client transactions among replicas in the available group. Multicast selection sends a transaction to every replica in the available set and uses the first returned result. Our experiment investigates the performance of three, common dynamic replica-selection algorithms: greedy, weighted, and partitioned (with random assignment).

Most replica-selection systems that we examined adopt a selection metric from one of two classes: client response time or server load. Estimated response time, an ideal selection metric from the client perspective, can be decomposed [17] as the sum of communications delay (C_D), server queuing delay (S_Q), and server processing time (S_P). C_D is important when clients access replicas through the Internet. S_Q is salient when a server is heavily loaded. S_P can dominate when transactions are computationally intensive. From a server perspective, estimated server load is an ideal selection metric. An alternative is estimated server latency (S_L), which can be decomposed as $S_Q + S_P$, yielding a convenient relationship between response time and server load. When C_D is similar among all clients, S_L provides a reasonable approximation of relative response time. When highly variable, C_D should be measured independently. Our experiments use S_L as an estimator for client response times because we simulate a distributed object system deployed on a local network, where clients experience similar communication delays.

3. EXPERIMENT DESIGN

We designed an experiment to meet the following objective: Given a set of r replicas deployed in a local network and queried periodically by c clients, characterize and compare performance of alternate selection algorithms. Our experiments exhibit the following constraints: (1) client-director pairs are deployed in a decentralized architecture (see Figure 1), (2) replicas are implemented as Jini lookup services, (3) each replica executes on a distinct, but similar, server, (4) each server is shared with other applications, and (5) replica state is piggybacked on existing Jini multicast announcements. Below, we provide details about the experiment architecture, key parameters, our technique to vary processor availability, selection metric and algorithms, and performance metrics.

3.1 Experiment Architecture

Figure 1 outlines the experiment architecture, which implements five replicas, each simulating a Jini [27] lookup service and a set of unrelated applications. Using Jini discovery and registration procedures all Jini services (not shown) register a service description with each replica. Each client periodically queries its local director (that uses some selection algorithm) to determine the address of a replica, and then queries the selected replica for service descriptions. Each client query is initiated 30 s after receiving a reply to the previous query (the first query is issued after a random startup delay). Each replica periodically (every 60 s) multicasts a Jini announcement extended to include two elements of replica state: (1) the number (N) of pending queries and (2) the current query processing rate (Q).

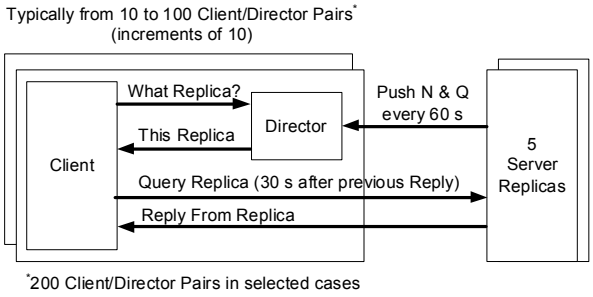


Figure 1. Experiment Architecture

Table 1. Key Experiment Parameters

Component Quantities	Servers	5	Per run constant
	Directors	10 to 100* in increments of 10 (*200 in selected cases)	Clients and Directors are paired
	Clients		
Component Startup Delays	Servers	0…15 s	Randomly selected using a uniform distribution
	Directors		
	Clients	60..75 s	
Component Workloads	Servers	Background Load is 25% to 99% (i.e., capacity for queries is 75% to 1%)	Per Server - varied every 60 s
	Directors	5 updates per minute	Each Server pushes an update every 60 s, but startup delays cause update offsets
	Clients	Client issues next query 30 s after receiving reply from previous query	Startup delays cause offset in Client queries

3.2 Key Experiment Parameters

Table 1 summarizes key parameters in three classes: component quantities, startup delays, and workloads. In most instances, an experiment considers an increasing population of clients from 10 to 100 (in increments of 10); however, the balanced and balanced-partitioned algorithms require 200 clients to distinguish their performance. The (uniformly distributed) random startup delays for servers and directors are required by Jini, while higher startup delay for clients allows Jini discovery and registration to complete before initiating client queries. Each server reserves a minimum of 1% of its processing capacity for client queries; however, as much as 75% may be used for client queries, depending upon the server's background load, which we vary every 60 s.

3.3 Processor Availability

Table 2 exhibits parameters controlling variation in processor availability. Each server reserves a minimum (BL_{MIN}) and maximum (BL_{MAX}) percentage (25% to 99%) of its capacity to process a background workload, which also defines a maximum (C_{MAX}) and minimum (C_{MIN}) capacity (75% to 1%) each server can devote to processing client queries. An unloaded server can process Q_{RATE} = 4 queries/s (assuming a query can be processed in QP_{TIME} = 250 ms), which means that a loaded server's query

processing rate may vary from a minimum (Q_{MIN}) of 0.04 queries/s to a maximum (Q_{MAX}) of 3 queries/s.

Table 2. Parameters Controlling Query Processing Rate

	Parameter	Value	Explanation
Bounds on Server Background Load	BL_{MIN}	0.25	A minimum of 25% of each Server is reserved for processing a background workload
	BL_{MAX}	0.99	Up to 99% of each Server may be allocated to process the background workload
Bounds on Server Query Capacity	C_{MAX}	0.75	1 - BL_{MIN} defines the maximum % of each Server that can be allocated to process Client queries
	C_{MIN}	0.01	1 - BL_{MAX} defines the minimum % of each Server that can be allocated to process Client queries
	QP_{TIME}	250 ms	Time to process a single query on an unloaded Server
	Q_{RATE}	4 queries/s	1/ QP_{TIME} defines the rate (in queries per second) at which an unloaded Server can process queries
	Q_{MAX}	3 queries/s	$Q_{RATE} \cdot C_{MAX}$ defines the rate at which a minimally loaded Server can process queries
	Q_{MIN}	0.04 queries/s	$Q_{RATE} \cdot C_{MIN}$ defines the rate at which a maximally loaded Server can process queries
Variation in Server Query Capacity	C_D	-20	The maximum % that a Server's query capacity can decrease between updates
	C_I	+20	The maximum % that a Server's query capacity can increase between updates
	dC	−0.2 to +0.2	Selected every 60 s from a discrete uniform distribution, dC = discrete_uniform(C_D, C_I) · 0.01
	C_t	$C_{t-1} + dC$	Computed every 60 s from new dC and previous C, but constrained as follows: $C_{MIN} \le C_t \le C_{MAX}$
	Q_t	$Q_{RATE} * C_t$	Computed every 60 s, after selecting dC and computing C_t(note that $Q_{MIN} \le Q_t \le Q_{MAX}$)

Every 60 s each server updates capacity (C_t) for processing queries, subject to a constraint that capacity may not change by more than 20% (C_D and C_I bound the maximum percentage of decrease and increase, respectively) from the previous capacity (C_{t-1}). The updated capacity determines the current query-processing rate (Q_t). Figure 2 displays a two-hour time series depicting the relationship between changes in available capacity (C_t) – left-hand y-axis and query-processing rate (Q_t) – right-hand y-axis.

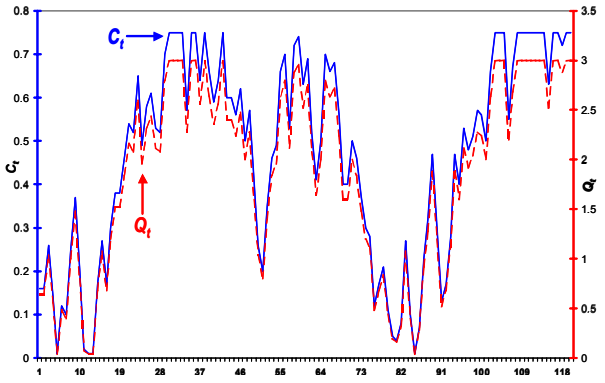


Figure 2. Variations in C_t causing Variations in Q_t

We assume each query requires similar processing, i.e., transactions are homogeneous. We also assume query-processing rate remains stable between announcements because the schedulers in the server operating systems allocate portions of processor time to specific processes and periodically (each minute here) adjust that allocation. We further assume that communication delays will be insignificant (and similar) because we simulate deployment in a local network.

3.4 Selection Metric and Algorithms

Directors select replicas based on estimated latency for each server r (S_{Lr}). $S_{Lr} = N_r/Q_r$, where N_r and Q_r are the number of queries pending and the query processing rate, respectively, received in the most recent announcement from server r. Table 3 defines key elements of the notation we use in the following description of our replica-selection algorithms.

Table 3. Notation for Defining Selection Algorithms

Notation	Explanation
S	Set of Servers
n	Number of Servers
s_i	ith Server
N_i	Number of queries backlogged at Server i
Q_i	Number of queries/second that Server i can process
T_{QAVAIL}	Server i is available when $N_i / Q_i \le T_{QAVAIL}$
A	Set of available Servers
a	Number of available Servers
W	Set of Servers with weights
w_i	Weight assigned to ith Server
K	Normalization factor
T_{QREF}	Maximum estimated Server latency
D	Set of Servers with additional queries needed to match T_{QREF}
d_i	Number of additional queries needed for ith Server to match T_{QREF}

Our baseline algorithm is *random selection*, where a director selects one of the known replicas, with each replica having an equal selection probability. Let S be the set of known server replicas, and n be the number of known server replicas. The director selects server replica s_i, where i is an integer selected uniformly on the interval [1..n]. The *greedy* algorithm requires a director to select the replica s_i with the lowest estimated server latency (N_i/Q_i).

In the *partitioned* (random selection) algorithm, a director first uses the state information cached for each replica to subset S into a set (A) of a available replicas with estimated server latencies at or below a threshold (T_{QAVAIL}). The director then selects one replica (randomly) from A. If no replicas qualify for set A, then the director selects a replica randomly from set S.

In the *weighted* algorithm, a director assigns each replica a weight based upon the inverse of estimated server latency and apportions the unit interval according to the weights. The director then draws a random real number uniformly distributed on the unit interval and selects the replica assigned to the corresponding portion.

The greedy, partitioned, and weighted algorithms consider estimated server latency (N/Q) as a unified metric; however, replicas with similar server latency estimates could possess different capacities to absorb work. This observation led us to devise a *balanced* algorithm, where a director assigns each replica a weight, based on the number of queries (d_i) required for its server latency to reach the maximum estimated server latency among all replicas. The director then apportions the unit interval according to those weights and chooses a random real number uniformly distributed on the unit interval, selecting the replica assigned to the corresponding portion. Balanced selection entails some probability that client transactions may be assigned to overloaded replicas. For this reason, we devised a *balanced-partitioned* variant, which first partitions replicas into two subsets, available and unavailable, based on comparing estimated server latency against T_{QAVAIL}, and then uses the

balanced algorithm to select a replica from among the available subset. Where the available subset is empty, selection is made using the balanced algorithm.

3.5 Performance Metrics

To compare performance among selection algorithms, we define three metrics: average client response time (avg_{RT}), defined in Table 4, and probability of selection error ($prob_{SE}$) and server overload ($prob_{SO}$), defined in Table 5.

Table 4. Definition of Average Client Response Time

Notation	Definition
c	Number of Clients
q_i	Number of queries sent by Client i
$tq_{i,j}$	Time that Client i issued its jth query
$tr_{i,j}$	Time that Client i received a reply to its jth query
avg_{RT}	Average response time, computed as: $(\sum_{i=1}^{c}\sum_{j=1}^{q_i} tr_{i,j} - tq_{i,j}) / \sum_{k=1}^{c} q_k$

Table 5. Definition of Probability of Selection Error and Probability of Server Overload

Notation	Definition
s	Number of Servers
N_i	Number of queries backlogged at Server i
Q_i	Number of queries/second that Server i can process
T_{Qi}	Estimated time for Server i to clear its query backlog, computed as N_i/Q_i
T_{QMAX}	Server i is overloaded when $T_{Qi} > T_{QMAX}$ (here T_{QMAX} = 50 s)
T_{Oi}	Total time during which $T_{Qi} > T_{QMAX}$ for Server i (observed and updated upon arrival and departure of each query)
T_{Ui}	Total time during which Server i is up
N_{Oi}	Number of queries received by Server i when $T_{Qi} > T_{QMAX}$
N_{Ti}	Total number of queries received by Server i
$prob_{SO}$	Probability a Server is overloaded, computed as: $\sum_{i=1}^{s} T_{Oi} / \sum_{i=1}^{s} T_{Ui}$
$prob_{SE}$	Probability of a selection error, computed as: $\sum_{i=1}^{s} N_{Oi} / \sum_{i=1}^{s} N_{Ti}$

4. SIMULATION RESULTS

We implemented our experiment as an SLX™ [28] simulation of Jini lookup servers, services, clients, and directors, executing a set of runs that each considered an increasing population of clients, each supported by a director using one of the selection algorithms defined in Section 3.4. Each client in each run generated 1,000 queries, and each run was iterated 100 times; thus, each data point observes c x 10^5 replica selections. Below, we report results in two sets: the four common selection algorithms and the two algorithms we invented.

Figures 3(a)-(c) plot performance (each graph displays a different metric) under increasing load for the common selection algorithms. Figures 4(a)-(c) plot performance for the balanced and balanced-partitioned algorithms, where we increase beyond 100 clients in order to distinguish performance differences. Figures 4(a)-(c) also include for comparison weighted selection, the best performing of the common algorithms.

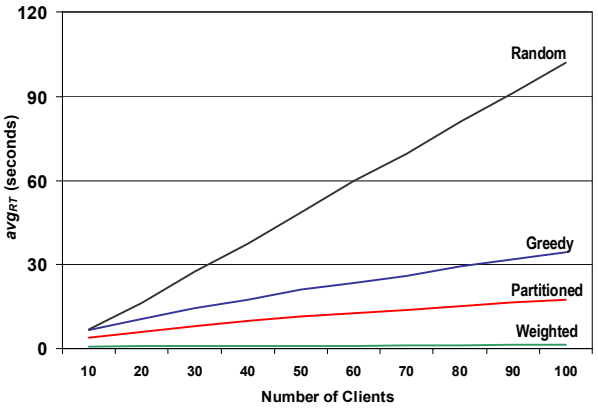


Figure 3(a). Average Client Response Time for Common Selection Algorithms

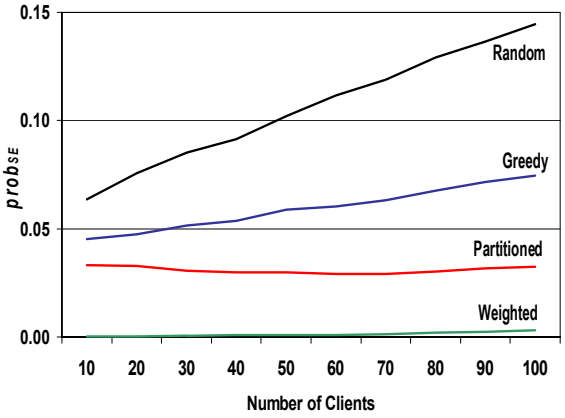


Figure 3(b). Probability of Selection Error for Common Selection Algorithms

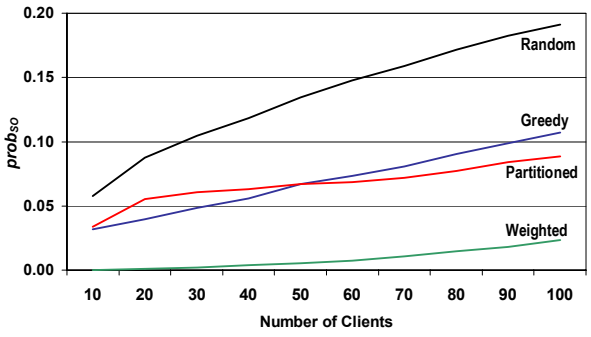


Figure 3(c). Probability of Server Overload for Common Selection Algorithms

5. DISCUSSION

Our results show that selecting replicas based on information yields superior performance over random selection, which may assign transactions to overloaded replicas; thus leading to higher response times and server latencies. One exception appears: the "thundering herd" effect induced by greedy selection causes higher variance in server latency (not shown), as transactions descend en masse upon the best performing replica,

transforming it to a poor performer. Bulk arrivals ensure that information on which decisions were based becomes outdated quickly.

Partitioning replicas into two sets (based on server latency) and then selecting randomly among the less loaded set, provides general improvement over greedy selection on all metrics. Further, the advantage of partitioned selection increases with client load. Spreading transactions evenly among replicas likely to provide good performance does not rapidly push one particular replica into overload. The general advantage of partitioned (random) over greedy selection exhibits one exception. Below 50 clients, servers have higher probability of being overloaded with partitioned (random) selection because the greedy algorithm assigns work in series – replica by replica – causing the number of overloaded servers to increase more slowly. Once all replicas reach saturation, the bulk arrival process of greedy selection creates larger backlogs, while partitioned selection spreads arrivals more evenly, allowing servers to spend less time in overload.

Among the common algorithms, weighted selection provides the best performance on all metrics. Weighted selection adapts to changes in replica state without inducing rapid or large fluctuations. Greedy selection stimulates large changes in workload, pushing a selected replica away from the state that led to its selection. The partitioned algorithm induces cyclic oscillation in replica workload, but at a somewhat slower frequency than greedy selection. Weighted selection tends mainly to react to changes in replica state, while the greedy and partitioned algorithms induce feedback that alters the state to which they are reacting. This difference leads weighted selection to exhibit more stable and desirable performance.

The balanced algorithm shares the reactive nature of weighted selection but improves performance for two reasons. First, balanced selection assigns more transactions to replicas with greater available processing capacity. Second, using the replica with the largest estimated server latency as the goal state reduces pressure for upward movement in system-wide server latency, and tends to reinforce downward movement. These reasons also explain why the balanced-partitioned algorithm performs well, up to a point. As the client population surpasses 100, performance degrades for the balanced-partitioned algorithm because the set of replicas available diminishes, forcing fewer replicas to receive more transactions. After load reaches saturation, partitioning creates a bulk-arrival process that pushes replicas into overload for longer periods. These results indicate that adding a partitioning step could diminish performance for an otherwise good selection algorithm.

6. CONCLUSIONS

We used simulation to characterize performance (response time, selection error, probability of server overload) for four common replica-selection algorithms (random, greedy, partitioned, weighted) when applied in a decentralized form to client queries in a distributed object system deployed on a local network. We introduced two new selection algorithms (balanced and balanced-partitioned) that give improved performance over the more common algorithms. We found that weighted selection performs best among the common algorithms and that balanced selection performs best overall. We explained why greedy and random algorithms should be avoided. We also provided evidence that preceding selection with a partitioning step can weaken an otherwise good selection algorithm.

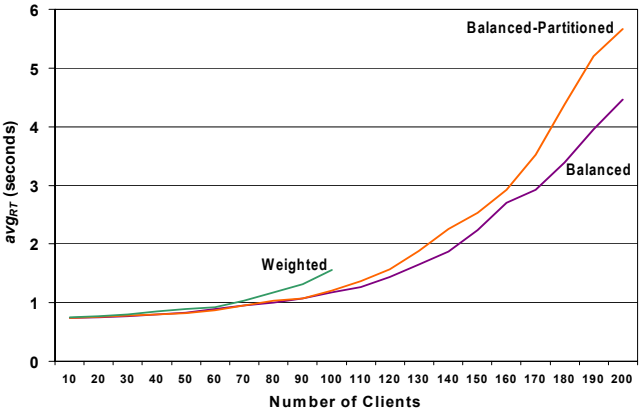


Figure 4(a). Average Client Response Time for New (and Weighted) Selection Algorithms

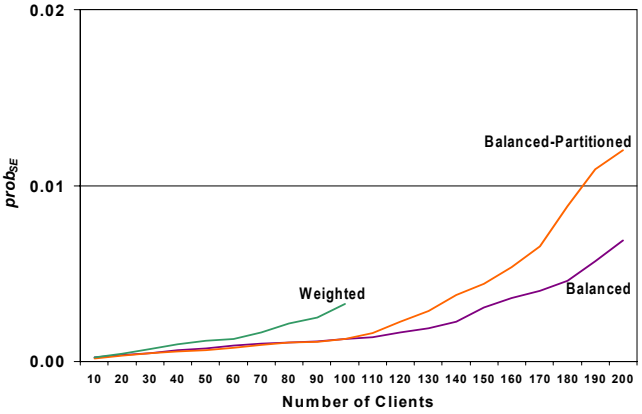


Figure 4(b). Probability of Selection Error for New (and Weighted) Selection Algorithms

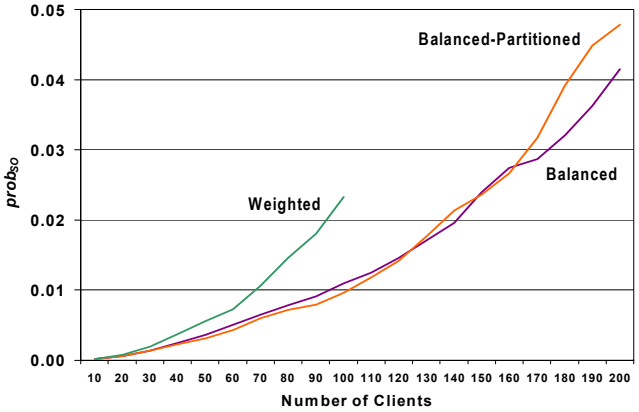


Figure 4(c). Probability of Server Overload for New (and Weighted) Selection Algorithms

7. REFERENCES

[1] Rabinovich, M., Xiao, Z., and Aggarwal, A. Computing on the Edge: A Platform for Replicating Internet Applications. In *Proceedings of the 8th International Workshop on Web Content Caching and Distribution*, (Hawthorne, New York, September 29 through October 1, 2003).

[2] Lewontin, S. and Martin, E. Client Side Load Balancing for the Web. In *Proceedings of 6th International World Wide Web Conference*. (Santa Clara, California, April 7-11, 1997).

[3] Vingralek, R., Breitbart, Y., Sayal, M., and Scheuermann, P. Web++: A System For Fast and Reliable Web Service. In *Proceedings of the USENIX Annual Technical Conference*. (Monterey, California, June 6-11, 1999). USENIX Association.

[4] Sayal, M., Scheuermann, P., and Vingralek, R. Content Replication in Web++. In *Proceedings 2nd IEEE International Symposium on Network Computing and Applications*. (Cambridge, Massachusetts, April 16 - 18, 2003). IEEE, p. 33.

[5] Fei, Z., Bhattacharjee, S., Zegura, E., and Ammar, M. A Novel Server Selection Technique for Improving Response Time of a Replicated Service. In *Proceedings IEEE INFOCOM 1998*. (San Francisco, California, March 1998). IEEE, pp. 783-791.

[6] Crovella, M. and Carter, R. Dynamic Server Selection in the Internet. In *Proceedings of the 3rd IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*. (Mystic, Connecticut, August 1995).

[7] Carter, R. and Corvella, M. Server Selection using Dynamic Path Characterization in Wide-Area Networks. In *Proceedings of INFOCOM 1997*. (Kobe, Japan, April 1997).

[8] Cardellini, V., Colajanni, M. and Yu, P. Request Redirection Algorithms for Distributed Web Systems. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 14, No. 4, April 2003, pp. 355-368.

[9] Sayal, M., Breitbart, Y., Scheuermann, P. and Vingralek, R. Selection Algorithms for Replicated Web Servers. In *Proceedings of the Workshop on Internet Server Performance*. (Madison, Wisconsin, June 1998).

[10] Connect Control Datasheet. Check Point Software Technologies Ltd. 2003.

[11] Load Balancing System, Chapter 6 in Intel Solutions Manual, Intel Corporation, pp. 49-67.

[12] Farrell, R. Review of Web server load balancers. Network World, September 27, 1997.

[13] Load Balancing in a Cluster, WebLogic Server 7.0, bea.

[14] Configuring application server load balancing, Tarantella.

[15] Server Load Balancing. TechBrief from Extreme Networks.

[16] Othman, O., O'Ryan, C. and Schmidt, D. The Design and Performance of an Adaptive CORBA Load Balancing Service. To appear in the "online" edition of the *Distributed Systems Engineering Journal*. February 2001.

[17] Krishnamurthy, S. Sanders, W., and Cukier, M. Performance Evaluation of a Probabilistic Replica Selection Algorithm. *In Proceedings of the Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable System*s. (San Diego, California January 07 - 09, 2002).

[18] Shen, K., Yang, T., and Chu, L. Cluster Load Balancing for Fine-Grained Network Services. In *Proceedings of the International Parallel and Distributed Processing Symposium* (IPDPS). (Fort Lauderdale, Florida April 15-19, 2002).

[19] Waldvogel, M., Hurley, P., and Bauer, D. Dynamic Replica Management in Distributed Hash Tables. IBM Research Report RZ-3502, July 2003.

[20] Ferdean, C. and Makpangou, M. A Scalable Replica Selection Strategy based on Flexible Contracts. In *Proceedings of the Third IEEE Workshop on Internet Applications*. (San Jose, California, June 23 - 24, 2003).

[21] Vazhkudai, S. Tuecke, S., and Foster, I. Replica Selection in the Globus Data Grid. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*. (Brisbane, Australia, May 15-18, 2001).

[22] Zhao, Y. and Hu, Y. GRESS – a Grid Replica Selection Service. In *Proceedings of the 15th International Conference Parallel And Distributed Computing and Systems*. (Marina Del Ray, California, November 3-5, 2003).

[23] Fu, Z. and Venkatasubramanian, N. Combined Path and Server Selection in Dynamic Multimedia Environments. In *Proceedings of the 7th ACM International Conference on Multimedia* (Part 1). (Orlando, Florida, 1999). ACM pp. 469-472.

[24] Guo, M. Ammar, M. Zegura, E. Selecting among Replicated Batching Video-on-Demand Servers. In *Proceedings of the 12th International Workshop on Network and Operating System Support for Digital Audio and Video*. (Miami, Florida, May 12-14, 2002).

[25] Huang, C. and Abdelzaher, T. Towards Content Distribution Networks with Latency Guarantees. In *Proceedings of the 12th International Workshop on Quality of Service*. (Montreal, Canada, June 7-9, 2004).

[26] Krishnamurthy, B. Wills, C. and Zhang, Y. On the Use and Performance of Content Distribution Networks. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*. (San Francisco, California, November 1-2, 2001).

[27] Arnold, K. et al, The Jini Specification, V1.0 Addison-Wesley 1999. Latest version is available from Sun.

[28] Henriksen, J. An Introduction to SLX™. In *Proceedings of the 1997 Winter Simulation Conference*. (Atlanta, Georgia, December 7-10, 1997), pp. 559-566.

SUMMARY OF CONTRIBUTIONS TO SERVICE DISCOVERY TECHNOLOGY

As part of the ITL research program in networking for pervasive computing, NIST researchers published the first generic model encompassing the structure and behavior of first-generation service discovery systems, and showed how that model can represent the designs for several, specific service discovery systems. The model provides a deep analysis of the common elements and behaviors in modern service discovery systems. NIST researchers also identified issues that designers should attempt to resolve in the next generation of service discovery systems. NIST researchers proposed a set of service guarantees that they believe service discovery systems should strive to satisfy, along with an analysis of the factors that might interfere with meeting service guarantees. Such guarantees could be cast into test assertions that serve to evaluate the behavior or measure the performance of designs and implementations of service discovery systems. NIST researchers also identified and suggested possible solutions to performance issues that can arise in service discovery systems. Identifying possible performance issues can alert users to the potential for unexpected behavior when service discovery technology is deployed at large scale. Further, implementers of service discovery systems can consider the suggested solutions when developing software to embody related processes in a service discovery system. All of the contributions reported in this special publication were provided to relevant standards bodies, consortia, and researchers in hopes of improving the quality of the next generation of service discovery systems on which the service-oriented architectures of tomorrow appear likely to depend.