A Java-based uniform workbench for simulating and executing distributed mobile applications[*]

Hannes Frey, Daniel Görgen, Johannes K. Lehnert, and Peter Sturm

University of Trier
Department of Computer Science
54286 Trier, Germany
E-mail: {frey|goergen|lehnert|sturm}@syssoft.uni-trier.de

Abstract. Spontaneous multihop networks with high device mobility and frequent fluctuations are interesting platforms for future distributed applications. Because of the large number of mobile devices required for any detailed analysis, it is nearly impossible to deploy prototype applications yet. In this paper, a comprehensive approach is presented which supports experiments ranging from pure simulation of several thousand mobile devices over hybrid scenarios with interaction among simulated as well as real life devices up to dedicated field trials. Part of this paper are also conclusions drawn from experiences with a first prototype version of a self-organized auction system for ad-hoc networks.

1 Introduction

Supporting mobile devices is central to any next generation networking technology. Future paradigms such as pervasive computing and ubiquitous computing [1] are characterized by countless numbers of small and thus mobile devices that communicate with neighboring peers using wireless transmission techniques. Groups of these mobile devices may form highly dynamic ad-hoc networks at any given time. Singlehop ad-hoc networks are state of the art, e.g. in the WLAN infrastructure mode by means of access points, and allow devices to reach a conventional wireful network with one hop of wireless communication. Spontaneous multihop networks with a high device mobility and frequent fluctuation require self-organization as a primary design principle to cope with the anticipated frequency of change. As a consequence, decisions within a mobile device can only be based on local information, e.g. the state of the device itself and its current neighborhood. In such an environment, most of the goals of a distributed mobile application are achieved by synergy through altruistic behavior of all involved devices.

Successfully evaluating self-organizing applications for multihop ad-hoc networks requires too many mobile devices as can be provided at the moment. Additionally, the number of volunteers needed to run these experiments is hard to

[*] This work is funded in part by DFG, Schwerpunktprogramm SPP1140 "Basissoftware für selbstorganisierende Infrastrukturen für vernetzte mobile Systeme".

obtain even in a university environment and they are even harder to orchestrate for the sake of reproducible scientific results. A promising approach is to provide a uniform workbench supporting experiments ranging from pure simulation of several thousand mobile devices, over hybrid scenarios with interaction among simulated as well as real life devices up to dedicated field trials as proofs of concept. From this uniform workbench, a development process can be derived that starts with implementing, testing, and evaluating algorithms and applications in a purely simulated environment first. In a second step, dedicated mobile devices can be cut out of a simulation run and be transferred to a real mobile device in order to deal with real user interaction and to evaluate the user experience. In a final phase, specific field trials can be defined and executed on a number of mobile devices. The hybrid nature of this approach leads to the additional requirement, that the simulator must be capable to achieve real-time execution behavior even in the case of several thousand mobile devices.

In the remainder of this paper, a Java-based implementation of such a uniform workbench for multihop ad-hoc networks is presented. The next section starts with a discussion of related work. The structure of the subsequent sections follows the aforementioned three phases of the development process. In section 3, the simulator part of the workbench is presented. Experiences gained during the evaluation of a test application show, that the required real-time behavior can be achieved for a sufficiently large number of mobile devices. Section 4 shows how dedicated mobile devices can be attached to a simulation run. The next section discusses the transfer of a simulated application on a number of mobile devices in order to carry out live experiments. Finally, in sections 6 and 7 conclusions drawn from experiences with a first self-organized auction system for ad-hoc networks are discussed.

2 Related work

This work presents a development environment for mobile applications running in a large scale mobile multihop ad-hoc network. The environment divides the design process of mobile applications in three parts, simulation, emulation and execution on real devices.

The CMU wireless extension to ns2 [2] and GloMoSim [3] are the commonly used tools for simulating protocols in mobile multihop ad-hoc networks. Both simulation environments focus on a detailed simulation of protocol layer 1 to 4. This high level of detail increases the computational complexity of the simulations. Thus, only a small number of devices can be simulated in acceptable time.

Emulation is used in order to allow user interaction and to allow existing applications to be tested in a simulation environment without the need of managing hundreds of real devices. An emulation platform using ns2 is proposed in [4]. It is an extension of the Vint/NS simulator [5] for emulation of wired networks using the wireless extension to ns2 [6]. This emulation platform scales well up to about 100 devices simulated by ns2. Other emulation approaches forego the simulation of mobile devices but emulate only node mobility and the resulting network behavior. Mobility data can be created synthetically as proposed in [7] or captured from real life test runs as done in [8].

Network experimentation using a testbed is useful to prove the applicability of a protocol or application in a real life scenario. Ad-hoc network examinations in such experiments were performed by [9] and [10] or [11] for single hop networks. An obvious disadvantage of using a testbed to examine protocol or application characteristics is that results are not reproducible and that it takes great efforts to manage the large number of real mobile devices. In order to allow more reproducible results [12] proposes users walking around by following the instructions of a scenario script running on each mobile device.

3 Simulation of mobile applications

The main focus of the simulation and evaluation platform presented in this paper is to ease the development and simulative evaluation of applications for mobile multihop ad-hoc networks. A high abstraction level based on an object-oriented design in Java enables the developer to concentrate on application design without worrying about technical details of the simulator. Instead of aiming to simulate the lower network layers as exactly as possible, the focus is on the simulation of the topological properties of the ad-hoc network. The combination of a high abstraction level and the limitation on the topological properties reduces the computational complexity of the simulation and allows the simulation of a high number of devices while maintaining short simulation times.

Additionally, the high abstraction level allows to implement the same abstractions in the simulation environment as well as on real hardware platforms. Applications developed and tested in the simulation environment can easily be transferred to the real hardware platform (see section 5). Thus, field trials are based on well-tested code. Traditional simulators require a complete rewrite of the application in order to port it to a real hardware platform.

The simulation environment is based on a three-layered architecture. The first layer abstracts the operating system of the mobile device, the positioning method and the wireless network technology and communication protocols. The application makes up the second layer, while the simulated user behavior controlling the application is placed on the top layer.

The operating system abstraction provides the application with information about the device, e.g. the current position and moving direction. The neighbor discovery service informs the application whenever another device enters or leaves the communication range of the device running the application. Additional information like the current position and direction of other devices is available from the neighbor discovery service, too.

Applications never access the network interface of a device directly, but use a collection of communication protocols instead. By using these existing communication protocols or defining new ones, applications can communicate with other applications running on different devices. Instead of bit-optimized messages simple Java objects are sent over these protocols. By defining a “simulated size” attribute for each message, the bandwidth used can still be simulated correctly while the development of algorithms and applications is easier. Sending objects as messages simplifies the reception of messages as well because the message objects “know” which particular method to call in the receiving application.

Mobility information of the devices in the simulation is generated from mobility models. The simulation environment defines mobility models in terms of three basic operations: devices enter and leave the simulation, and they move directly from one point to another with constant speed. Sequences of these base operations are sufficient to approximate all mobility models.

Given the network sending radii for all devices, the link calculator component computes the potential communication links between all devices for arbitrary mobility models. When a device starts a new movement, the link calculator can compute the potential communication links with all other devices for the *whole* movement. This reduces the number of comparisons with other devices significantly. The results of the link calculator can be saved and used in further simulation runs (even with a different application), thus resulting in an additional speed-up.

The network model of the simulation is exchangeable. The respective implementation has access to all simulated devices and receives the link information from the link calculator. Therefore, the network implementation can decide whether a potential link calculated by the link calculator will actually result in a communication link between devices. This allows to implement rather simple network models like an unbound network as well as complicated statistical models, which take into account the device density and other disturbances of the wireless network.

The simulation environment offers a comprehensive set of visualization tools. An application for example, may visualize its internal state, or the network implementation may draw a graph of all communication links. Visualization is based on a canvas supporting basic drawing operations like lines, rectangles, polygons, ellipses, images and text. Components use collections of geometric shapes drawing to this canvas to visualize themselves. Due to this generic approach the output of the visualization is not limited to windowing systems, but can be saved as video files or postscript figures. The current version provides canvas implementations using Java Swing/Java 2D, SWT[13], OpenGL[14, 15] and PostScript.

In order to allow a better evaluation of the scalability and performance of the simulation environment, a sample message dissemination application has been tested. This application was simulated with the Random Waypoint Mobility Model [6] and an unbound network implementation for 10 minutes. The sending radii of the mobile devices are uniformly distributed between 10 and 100 meters and the moving speed of the mobile devices is uniformly distributed between 0.8 and 1.0 meters per second.

Two series of measurements were done on a Pentium IIIm with 1.2GHz and Java 2 SDK 1.4.2. The first one measured the execution speed for an increasing number of devices on a fixed simulation area of $500m \times 500m$, thus increasing the average device density in each step. The second measurement increased the size of the simulation area while increasing the number of devices, thus keeping the average device density the same. Table 1 clearly shows that device density is an important factor for the performance of the simulation environment.

In case of a constant device density the simulation environment scales well with the number of devices: for a doubled number of devices the execution time is increased by a factor of 3.3 at most. With increasing device density the execution time for a doubled number of devices is increased by a factor of 5.2 at most. In both cases the real-time simulation of a high number of devices is possible: for the constant device density a simulation of 6400 nearly twice as fast as real-time is possible.

If the mobility and connectivity data is precomputed, the execution times decrease significantly. Especially in the case of constant device density the performance gain is dramatic. A simulation run with 12800 mobile devices is finished nearly 12 times as fast ($96.6s$ instead of $1139.9s$). For the measurement with increasing device density the execution time is reduced by 56% ($451.9s$ instead of $1030.4s$).

#devices	execution time	execution time (precomputed)
50	$0.5s$	$0.5s$
100	$0.8s$	$0.8s$
200	$2.5s$	$2.0s$
400	$10.4s$	$7.6s$
800	$44.4s$	$28.3s$
1600	$198.8s$	$113.2s$
3200	$1030.4s$	$451.9s$
6400	n/a[1]	n/a[1]
12800	n/a[1]	n/a[1]

(a)

#devices	simulation area	execution time	execution time (precomputed)
50	$250m \times 250m$	$0.6s$	$0.7s$
100	$354m \times 354m$	$1.0s$	$1.1s$
200	$500m \times 500m$	$2.3s$	$2.0s$
400	$707m \times 707m$	$6.4s$	$4.3s$
800	$1000m \times 1000m$	$15.9s$	$9.0s$
1600	$1414m \times 1414m$	$41.5s$	$16.2s$
3200	$2000m \times 2000m$	$122.1s$	$28.1s$
6400	$2828m \times 2828m$	$344.8s$	$49.3s$
12800	$4000m \times 4000m$	$1139.9s$	$96.6s$

(b)

Table 1. Execution times for the example simulation: (a) on a simulation area of $500m \times 500m$, (b) with an average device density of 0.0008 devices per m^2.

Using a Java2D canvas implementation, visualization in real-time (or even faster) is possible up to 1600 devices in the case of increasing device density. For constant device density the maximum number of devices for a real-time visual-

[1] The available RAM (384 MB) of the machine used for the measurement was not sufficient to run the simulation. This is due to the high device density and the resulting number of events.

ization is 3200. Further speed-ups are possible by using precomputed mobility and connectivity data.

4 Hybrid simulation platform

The hybrid simulation platform allows real mobile devices to be attached to a running simulation by using RMI [16] over a network connection. This enables users with dedicated external devices to interact with the simulated application.

The hybrid approach is advantageous over evaluating on real hardware solely, since no hundreds of real devices are needed and have to be configured/managed to run the application in a real scenario. Furthermore, the visualization component of the simulation environment allows visualization of the current global state regarding the considered application. This is profitable for debugging code and to get a better insight into the application behavior. Finally, it is possible to replay, start at a certain time or speed up the simulation. This allows easier evaluation and demonstration of the considered application. Even though the application is running in a simulation environment, nevertheless one gets a feeling about the application running in real life.

Figure 1 depicts the client/server architecture of the hybrid evaluation platform. Simulation server and simulation control are running on the server side. Simulation clients are running on real mobile devices communicating with the running simulation over a wireless connection or on traditional workstations using a stationary network.

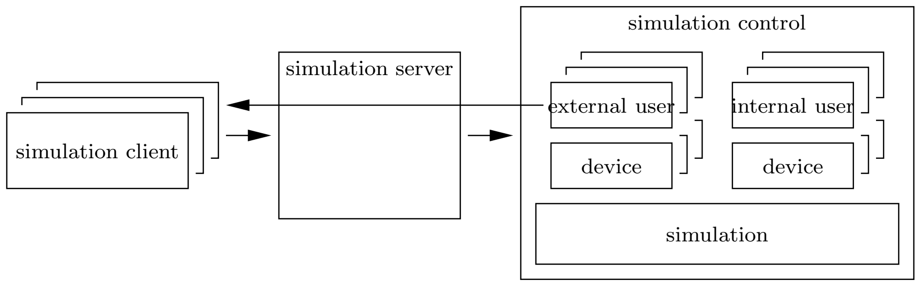


Fig. 1. Client server architecture of the hybrid simulation approach.

The simulation server is running as an independent thread registered as an RMI service. Any client message is processed by the simulation server first, i.e. the server waits for client logins and any user input which is passed to the simulation control in advance.

The task of simulation control is to run the simulation as a second thread on the server side. Initially, it creates a simulation instance with simulated users only. Each internal user controls one device. Additionally, simulation clients can be registered to the simulation control afterwards. The simulation control provides an additional device and an external user instance for the new client. By passing client input to the device, the external user controls the application in the same manner as an internal user. Since client input produces asynchronous events during the simulation run, synchronization regarding the event queue of the simulation is needed in order to achieve atomic processing of each event handle method. Additionally, any application output is passed to the client by using a callback object created by the simulation server. The core simulation is not aware of external clients, since by using the external user object, input and output regarding a certain client is treated in the same manner as for a common simulated user. Thus, there is no need to change any code from the pure simulation to the hybrid simulation platform.

Finally, the simulation client is responsible to provide a front-end for the client to control its assigned mobile device and application part in the current simulation run. In general, it consists of an application GUI replacing the simulated user behavior and providing information about the application state. When porting an application from the pure simulation to the hybrid simulation platform, this is the only part where additional code has to be written.

5 Mobile applications on real devices

Switching between a simulated environment and true hardware requires the same interfaces and the same platform behavior. Therefore a hardware abstraction layer is needed, which comprises all functionality of the simulated platform, such as network communication, neighbor discovery, positioning and the same event-based programming model including system timers and message events. Additionally, a user interface is needed to enable users to interact with the application. By following the proposed application development process, this GUI is already implemented for the hybrid testing environment.

The platform is implemented for devices such as PocketPC or notebooks running a JVM. These devices must be able to communicate with others over IEEE802.11b [17] and to identify their current position by using a GPS unit.

The network communication part is based on UDP/IP unicasts and broadcasts. One hop communication uses these primitives directly. All higher level protocols such as position-based routing or topology-based routing are based on these simple primitives and are already implemented for the simulation.

Beaconing is used to detect adjacent devices. The beacon contains the device address, its current position, its moving direction and additional device information. To detect bidirectional connections, devices answer broadcasted beacons with an empty unicast message. Neighbor information is cached and provided to the application and attach/detach events are communicated if needed.

To provide positioning information only GPS is supported up to now. Other positioning services such as [18, 19] are also possible. GPS provides spherical (longitude/latitude) coordinates but the application expects Cartesian coordinates.

The projection used in the platform is a variant of a Gauss-Kruger projection with a shifted zero-point.

Up to now, messages sent by the applications are directly passed to the send system without any bit optimization. The message objects are serialized by the standard serialization mechanism provided by Java. If needed, this could be replaced with a more efficient method.

Since the simulated applications are inherently event-driven, the real platform must provide the same event-driven programming model. This is achieved by multithreading and operating system timers (normally also implemented as threads). Due to the single-threaded nature of the simulation environment, the application code does not care about concurrency errors. To avoid application changes, precautions have to be made. The platform uses one thread to execute the application handlers, the events from the other threads are passed to it over an event queue. Contrary to the simulation, the platform events have no ordering, therefore no assumption should be made in case of contemporaneity or nearly contemporaneity of independent events.

In the simulated environment simulated user behavior is needed. On real hardware this simulated user behavior could still be used for testing purposes, e.g. to reproduce phenomena observable in simulations. The common uses for the implementation on real hardware are on the one hand field trials with real user interaction on the other the normal use of this application in a real environment. Therefore a GUI is needed allowing the user to interact with the application. Since a GUI for this application is already implemented and tested with the hybrid environment it can also be used for the real hardware. Porting the GUI is fairly easy done by omitting the remote link to the application achieved by RMI.

6 Case study: UbiBay

The development environment presented in this paper has been successfully used to implement UbiBay, a self-organizing auction platform for mobile multihop ad-hoc networks. UbiBay is based on SELMA[20], a middleware platform for mobile ad-hoc networks. This middleware platform uses the marketplace pattern [21] as its main communication method and provides applications with all the components needed to implement this pattern.

A marketplace is a small region of the ad-hoc network where a high device density is likely. The basic idea of the marketplace pattern is that users send their requests and offers to the marketplace where the negotiations take place. This increases the probability that matching peers are able to find each other and negotiate in an efficient manner. Requests, offers and negotiation results are sent to the marketplace and back using position-based routing. Negotiation results use a home zone of the user to find back to the originating device. This home zone is a small region in the ad-hoc network where the owner of a device will be found with a high probability, e.g. the user's house.

Communication at the marketplace is limited to the marketplace area and thus does not increase the network load outside of the marketplace. Due to the higher device density at the marketplace and its relatively small size, only a few hops are needed to communicate with other devices at the marketplace. Instead of simple flooding a neighbor knowledge broadcast[22] is used for marketplace communication. Additionally, devices at the marketplace can use topology-based routing for unicast communication.

UbiBay consists of three kinds of requests: auction, bid, and information requests. When a user wants to start an auction, he will send an auction request to the marketplace containing the description of the auction, the starting bid and the duration. The auction request stays at the marketplace until the auction is over. By sending information requests to the marketplace, a user gets information on all auctions currently running. In order to bid on an auction, the user sends a bid request with his maximum bid to the marketplace. At the marketplace, the bid request is matched with other bid requests for the auction. The bid request will stay at the marketplace until its maximum bid is exceeded or the auction is over. Both outcomes result in a notification message to the user.

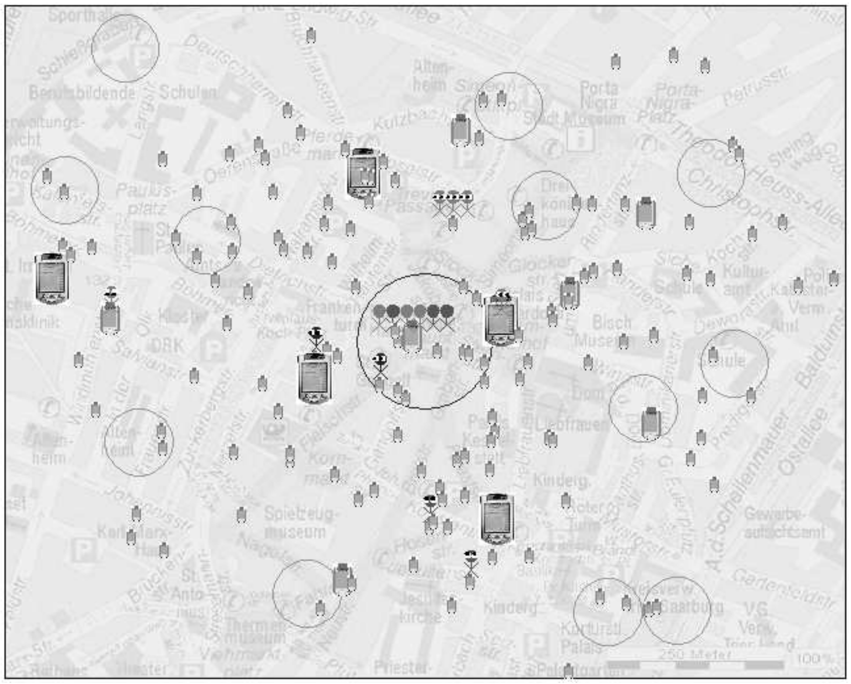
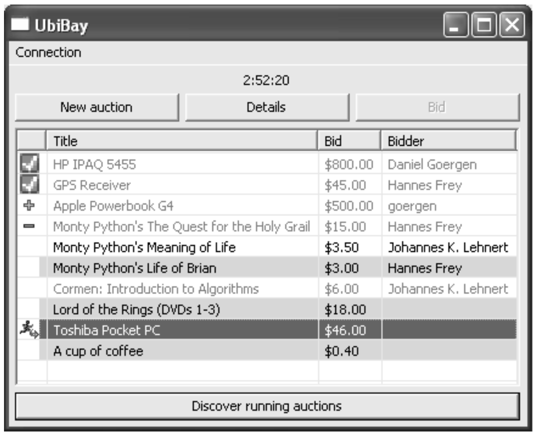


Fig. 2. Visualization of the simulated UbiBay application.

UbiBay was implemented as a simulated application using the simulation environment described in section 3. Using a simple simulated user behavior auctions where simulated and the visualization features of the simulation platform were used to debug the application during development and to get a better understanding of the application behavior (see Fig. 2).



	Title	Bid	Bidder
✓	HP IPAQ 5455	$800.00	Daniel Goergen
✓	GPS Receiver	$45.00	Hannes Frey
+	Apple Powerbook G4	$500.00	goergen
−	Monty Python's The Quest for the Holy Grail	$15.00	Hannes Frey
	Monty Python's Meaning of Life	$3.50	Johannes K. Lehnert
	Monty Python's Life of Brian	$3.00	Hannes Frey
	Cormen: Introduction to Algorithms	$6.00	Johannes K. Lehnert
	Lord of the Rings (DVDs 1-3)	$18.00	
🏃	Toshiba Pocket PC	$46.00	
	A cup of coffee	$0.40	

Fig. 3. Graphical user interface of UbiBay.

Based on this simulation application, a version for the hybrid simulation platform was developed. The application code of UbiBay did not change at all, but a suitable external user class was developed and plugged into the simulation server. Additionally, a graphical user interface was developed to allow the external clients to interact with the UbiBay application. See Fig. 3 for a screenshot. SWT was used as the widget set, because the graphical user interface was to be used on HP iPAQs in the next step. Other widget sets like AWT, Swing etc. are either not available on the mobile Java platform, do not allow rich user interfaces, or do not perform well enough.

The graphical user interface allows the user to send information requests to the marketplace. Results are displayed in a table together with a status icon. The user can view details of an auction, bid on an auction or start a new auction. Auctions are removed from the list if they are over and the user has not been involved.

Using the hybrid simulation platform with workstations in a fixed network proved to be very valuable. Testing an application by using a real graphical user interface is completely different from simulating an application with simple user behavior. Many (small) bugs emerged not until human users tested the application in the hybrid environment.

The last step in the development of UbiBay was the implementation of the application on real mobile devices. PocketPCs with Microsoft Pocket PC 2002, GPS receivers and the IBM J9 VM were used as the platform. Due to the fact that the execution platform provides the same abstractions as the simulation environment, the UbiBay application code was transferred “as is” to the execution platform. Transferring the graphical user interface to the mobile device required only minor modifications. The sizes of some GUI elements had to be tweaked and the RMI part of the client code was no longer necessary.

7 Conclusions

The proposed development platform for applications in mobile multihop ad-hoc networks takes advantage of three different environments. The simulative approach enables the testing of algorithms and applications with lots of devices. The hybrid approach allows testing with real user interaction in a simulated environment. Finally, the real platform implementation is needed to verify the simulation results in the real world and test the final application in field trials.

Using the development platform also leads to a development process for applications in mobile multihop ad-hoc networks. Starting with an application in the simulation environment, the complete application code can be reused in the hybrid platform, extended by a graphical user interface. Both the user interface and the application code again can be reused with minor modifications when testing the application on real devices. Following this development process reduces the implementation overhead to a minimum due to code reuse.

Both the development process and platform have been tested during the implementation of the UbiBay application and SELMA, a middleware platform for mobile multihop ad-hoc networks. Based on first simulation results the UbiBay application was tested both in the hybrid environment and on real hardware. Tests in the hybrid environment proved to be very valuable, because the user interface and behavior of the application could be thoroughly tested, which led to several improvements.

References

1. Weiser, M.: The computer for the 21st Century. Scientific American **265** (1991) 94–104
2. Fall, K., Varadhan, K.: The ns manual. The VINT Project – A collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC (1989-2003)
3. Zeng, X., Bagrodia, R., Gerla, M.: GloMoSim: A library for parallel simulation of large-scale wireless networks. In: Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS-98), Los Alamitos, IEEE Computer Society (1998) 154–161
4. Ke, Q., Maltz, D., Johnson, D.B.: Emulation of multi-hop wireless ad hoc networks. In: The 7th International Workshop on Mobile Multimedia Communications (MoMuC 2000). (2000)
5. Fall, K.: Network emulation in the VINT/NS simulator. Proceedings of the fourth IEEE Symposium on Computers and Communications (1999)
6. Broch, J., Maltz, D., Johnson, D., Hu, Y.C., Jetcheva, J.: A performance comparison of multi-hop wireless ad hoc network routing protocols. In: Proc. of the 4th ACM/IEEE Int. Conf. on Mobile Computing and Networking (MobiCom'98). (1998) 85–97

7. Zhang, Y., Li, W.: An integrated environment for testing mobile ad-hoc networks. In: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing. (2002) 104–111
8. Noble, B., Satyanarayanan, M., Nguyen, G.T., Katz, R.H.: Trace-based mobile network emulation. In: Proceedings of SIGCOMM 97. (1997) 51–61
9. Maltz, D., Broch, J., Johnson, D.: Lessons from a full-scale multihop wireless ad hoc network testbed. IEEE Personal Communications Magazine **8** (2001) 8–15
10. Ramanathan, R., Hain, R.: An ad hoc wireless testbed for scalable, adaptive qos support. In: Proceedings of IEEE Wireless Communications and Networking Conference (WCNC 2000). Volume 3. (2000) 998–1002
11. Agrawal, P., Asthana, A., Cravatts, M., Hyden, E., Krzyzanowski, P., Mishra, P., Narendran, B., Srivastava, M., Trotter, J.: A testbed for mobile networked computing. In: Proceedings of 1995 IEEE International Conference on Communications (ICC '95). (1995) 410–416
12. Lundgren, H., Lundberg, D., Nielsen, J., Nordström, E., Tschudin, C.: A large-scale testbed for reproducible ad hoc protocol evaluations. In: 3rd annual IEEE Wireless Communications and Networking Conference (WCNC 2002). (2002)
13. Eclipse Project - Universal Tool Platform: SWT: Standard widget toolkit (2003) Available from `http://www.eclipse.org/platform/index.html`
14. Eclipse Project - Universal Tool Platform: SWT experimental OpenGL plug-in (2003) Available from `http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/platform-swt-home/o%pengl/opengl.html`
15. Segal, M., Akeley, K.: The OpenGL graphics system: A specification (version 1.4) (2002) Available from `http://www.opengl.org/developers/documentation/version1_4/glspec14.pdf`
16. Sun Microsystems, Inc.: Java remote method invocation specification (revision 1.8) (2003) Available from `ftp://ftp.java.sun.com/docs/j2se1.4/rmi-spec-1.4.pdf`
17. International Standard ISO/IEC 8802-11: Information technology – telecommunications and information exchange between systems – local and metropolitan area networks – specific requirements – part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications (1999)
18. Bulusu, N., Heidemann, J., Estrin, D.: GPS-less Low Cost Outdoor Localization For Very Small Devices. IEEE Personal Communications Magazine **7** (2000) 28–34
19. Hamdi, M., Capkun, S., Hubaux, J.P.: GPS-free Positioning in Mobile Ad-Hoc Networks. In: Proceedings of HICSS, Hawaii (2001)
20. Frey, H., Görgen, D., Lehnert, J.K., Sturm, P.: Selma: A middleware platform for self-organizing distributed applications in mobile multihop ad-hoc networks. Submitted to PERCOM 2004 (2003)
21. Görgen, D., Frey, H., Lehnert, J., Sturm, P.: Marketplaces as communication patterns in mobile ad-hoc networks. In: Kommunikation in Verteilten Systemen (KiVS). (2003)
22. Williams, B., Camp, T.: Comparison of broadcasting techniques for mobile ad hoc networks. In: Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC). (2002) 194–205