Middleware to Support Sensor Network Applications

Wendi B. Heinzelman*†, Amy L. Murphy*‡, Hervaldo S. Carvalho*, and Mark A. Perillo*†

*Center for Future Health
University of Rochester
Rochester, NY 14627

†Dept. of Electrical and Computer Engr.
University of Rochester
Rochester, NY 14627

‡Computer Science Department
University of Rochester
Rochester, NY 14627

Abstract

Current trends in computing include increases in both distribution and wireless connectivity, leading to highly dynamic, complex environments on top of which applications must be built. The task of designing and ensuring the correctness of applications in these environments is similarly becoming more complex. The unified goal of much of the research in distributed wireless systems is to provide higher level abstractions of complex low-level concepts to application programmers, easing the design and implementation of applications. A new and growing class of applications for wireless sensor networks require similar complexity encapsulation. However, sensor networks have some unique characteristics, including dynamic availability of data sources and application quality of service requirements, that are not common to other types of applications. These unique features, combined with the inherent distribution of sensors and limited energy and bandwidth resources, dictate the need for the network functionality and the individual sensors to be controlled to best serve the application requirements.

In this paper, we describe different types of sensor network applications and discuss existing techniques for managing these types of networks. We also overview a variety of related middleware and argue that no existing approach provides all the management tools required by sensor network applications. To meet this need, we have developed a new middleware called MiLAN. MiLAN allows applications to specify a policy for managing the network and the sensors, but the actual implementation of this policy is affected within MiLAN. We describe MiLAN and show its effectiveness through the design of a sensor-based personal health monitor.

1 Introduction

For several decades, distributed computing has been both an enabling and a challenging environment in which to build applications. Initially, the major difficulty in implementing such systems was simply exchanging data across distances and among heterogeneous components. Today these problems are essentially solved, and research is turning its focus to higher level concerns such as improved fault tolerance through replication, optimal data access via distributed object placement, and methods of enabling high level communication abstractions such as event dispatching and remote invocation. The end result of this research into distributed systems is an expanding set of middleware platforms that reside above the operating system and below the application, abstracting lower level functionality such as network connectivity and providing a high-level coordination interface to the application programmer.

Often the combination of characteristics from the environment and application drive the design of the middleware. For example, consider the new class of applications for sensor networks with the following features.

- *Inherent distribution.* The sensors are distributed throughout a physical space, and are primarily connected wirelessly.
- *Dynamic availability of data sources.* Either mobility through space, addition of new sensors, or loss of existing sensors causes the set of available sensors to change over time.
- *Constrained application quality of service demands.* Sensor network applications require a minimum quality of service (QoS), and this level must be maintained over an extended period of time. There may be many ways to achieve the QoS (e.g., different sensors may offer data or services that meet the applications' QoS requirements). Furthermore, the required QoS and the means of meeting this QoS can change over time, as the state of the application or the availability of sensors changes.
- *Resource limitations.* Both network bandwidth and sensor energy are constrained. This is especially true when considering battery powered sensors and wireless networks.
- *Cooperative applications.* Sensor network applications share available resources (e.g., sensor energy, channel bandwidth, etc.) and either cooperate to achieve a single goal, or, at the very least, do not compete for these limited resources.

One unique feature of sensor network applications with these properties is that simply responding to the changing environment is insufficient to achieve the required QoS over time. Instead, the applications must be *proactive*, actively affecting the network. Most existing middleware systems do not support such a proactive approach with respect to the network, leaving reactivity as the only choice and sacrificing application quality over time. We believe that a middleware that enables applications to affect the network and the sensors themselves is needed to support this new and growing class of applications for sensor networks.

This paper presents an overview of the related research in the areas of sensor networks and middleware, highlighting how existing approaches to the management of sensor networks could benefit from a middleware abstraction and showing that existing middleware do not meet the specific needs of all sensor network applications. Based on this observation, we propose a new middleware for sensor networks called MiLAN (Middleware Linking Applications and Networks). MiLAN allows sensor network applications to specify their quality needs and adjusts the network characteristics to increase application lifetime while still meeting those quality needs. Specifically, MiLAN receives information from (1) the individual applications about their QoS requirements over time and how to meet these QoS requirements using different combinations of sensors, (2) the overall system about the relative importance of the different applications, and (3) the network about available sensors and resources such as sensor energy and channel bandwidth. Combining this information, MiLAN continuously adapts the network configuration (e.g., specifying which sensors should send data, which sensors should be routers in multi-hop networks, which sensors should play special roles in the network, etc.) to meet the applications' needs while maximizing application lifetime. Figure 1 shows a high-level diagram of a system that employs MiLAN.

Next we will describe several sensor network applications that could benefit from a middleware like MiLAN that proactively affects different characteristics of the network, and in Section 3 we will discuss existing sensor network management and middleware approaches. In Section 4 we will describe MiLAN and show how the design of a health monitor sensor application can be simplified using MiLAN.

2 Sensor Network Applications

As stated in the introduction, sensor network applications represent a new class of applications that are

- data driven, meaning that the applications collect and analyze data from the environment, and depending on redundancy, noise, and properties of the sensors themselves, the applications can assign a quality level to the data, and
- state based, meaning that the application's needs with respect to sensor data can change over time based on previously received data.

Typically sensors are battery-operated, meaning they have a limited lifetime during which they provide data to the application. A challenge of the design of sensor networks is how to maximize network lifetime while meeting application quality of service (QoS) requirements. For these types of applications, the needs of the application should dictate which sensors are active and the role they play in the network topology. To further illustrate this point, we discuss some specific sensor network applications and how they can benefit from this form of interaction.

2.1 Environmental Surveillance

Consider an environment where multiple sensors (e.g., acoustic, seismic, video) are distributed throughout an area such as a battlefield. A surveillance application can be designed on top of this sensor network to provide information to an end-user about the environment. The application may require a minimum percentage sensor coverage in an area where a phenomenon is expected to occur. The sensor network may consist of sensors with overlapping coverage areas, providing redundant information. If the application does not require all this redundant information, it would be desirable to conserve energy in some sensors by allowing them to sleep, thereby lengthening the lifetime of the network. For example, as sensors use up their limited energy, the application would like to use different sets of sensors to provide the required QoS (in this case, minimum sensor coverage area). This requires that the application manage the sensors over time. Such management can be as simple as turning sensors on and off, or as complex as selecting the routes for data to take from each sensor to the collection point in a multi-hop network. Furthermore, the needs of the surveillance application may change as a result of previously received data. For example, if the application determines that an intrusion has occurred, the application may assume a new state and require more sensors to send data to more accurately classify the intrusion. The implementation of these tasks can be complex, and they are difficult to incorporate into applications.

2.2 Home/Office Security

Home/office security systems are becoming increasingly complex, monitoring for not only intrusion into the space but also the occurrence of substances such as fire or carbon monoxide gas. To be able to monitor the application variables, the security system must obtain data from heterogeneous sensors such as acoustic, motion, heat, and vibration sensors scattered throughout the home/office. Making these sensors wireless and battery powered allows them to be easily placed in existing homes without major household modifications. To make the sensor network last as long as possible, the application may only want a subset of the sensors activated at any time. Once a sensor's activation has been triggered through some event, the application must analyze the data and decide how to change the configuration of active sensors. This can be modeled as the application changing state based on received data. For different application states, different sets of sensors should be activated to provide the greatest benefit to the security application. Thus, the application needs to be able to control which sensors are activated over time. At the same time, to allow the application to work as long as possible, the set of sensors activated for a given application state should be chosen wisely to reduce energy dissipation and maximize system lifetime. Furthermore, sensors whose data are very important to the application, such as the video sensor, should not be used as routers or control nodes, so that their energy is saved for sensing the environment and transmitting their data to the application. Performing such optimizations and controlling the sensors and network functionality from within the application would place an unreasonable burden on the application.

2.3 Medical Monitoring

As a final example, consider a personal health monitor application running on a PDA that receives and analyzes data from a number of sensors (e.g., ECG, EMG, blood pressure, blood flow, pulse oxymeter). The monitor reacts to potential health risks and records health information in a local database. Considering that most sensors used by the personal health monitor will be battery-operated and use wireless communication, it is clear that this application can benefit from intelligent sensor management that provides energy-efficiency as well as a way to manage QoS requirements, which may change over time with changes in the patient's state. For example, higher quality might be required for certain health-related variables during high stress situations such as a medical emergency, and lower quality during low stress situations such as sleep. We will return to the details of this application when describing our middleware system, MiLAN, in Section 4.

3 Sensor Network Management and Middleware Approaches

There has been considerable research into the development of low-level protocols to support sensor networks as well as high-level middleware systems to support the development of distributed computing applications by hiding environmental complexities. A recent trend includes the combination of these into middleware designed for sensor networks. In this section, we describe these developments and explain why they are insufficient for the unique style of many sensor network applications.

3.1 Sensor Networks

One of the distinguishing characteristics of sensor networks is their reliance on non-renewable batteries, despite their simultaneous need to remain active as long as possible. Therefore, initial work has been done to create network protocols tailored to sensor networks that extend network lifetime considering the energy constraints of the individual sensors. Figure 2 highlights the different protocols we discuss here and how they relate to different network services.

Some protocols make use of low-level node collaboration to reduce the energy cost of data transfer by aggregating data locally rather than sending all raw data to the application. For example, with LEACH [1], nodes form local clusters and all data within a cluster are aggregated by the cluster-head node before being transmitted to the base station. This limited form of low-level collaboration is also found in the query-based technique of Directed Diffusion [2], in which nodes collaborate to set up routes as *interests* for particular data are disseminated through the network.

Another approach to reducing energy dissipation is to turn nodes off whenever possible. As idle power can often be significant, this approach can greatly extend application lifetime. MAC-level protocols, such as PAMAS [3] and S-MAC [4] use this technique to reduce energy dissipation in the MAC protocol, often trading off latency in packet delivery for energy efficiency. Topology control protocols such as ASCENT [5], Span [6], and STEM [7] use a similar technique of turning on and off sensors to maximize network lifetime while keeping the network fully connected. Other topology control protocols such as Lint [8] aim to determine the minimum transmit power necessary for a fully connected network, whereas protocols such as those described in [9, 10] determine the optimal transmit power to minimize overall energy dissipation.

In addition to the above two techniques, considerable energy can be saved by tailoring the routing protocol to the characteristics of sensor networks, including the energy constraints of the sensors, the data-driven nature of these networks, and the many-to-one, many-to-some, or many-

to-many collection of the data. Sensor network routing protocols such as Rumor Routing [11], Directed Diffusion [2] and SPIN [12] provide lightweight, data-centric solutions tailored to typical sensor network traffic patterns.

Although these protocols are effective in extending the lifetime of sensor networks, the gap between the protocol and the application is often too large to allow the protocols to be effectively used by application developers.

3.2 Middleware

Middleware has often been useful in traditional systems for bridging the gap between the operating system (a low-level component) and the application, easing the development of distributed applications. Because wireless sensor networks share many properties with traditional distributed systems, it is natural to consider distributed computing middleware for use in sensor networks. Figure 3 shows a high-level view of the key relationships among the middleware we discuss here.

One of the most common middleware systems, Corba [13], hides the location of remote objects, simplifying the application's interactions with these remote objects by allowing all operations to appear local. Although this could be applied to sensor networks to provide access to the sensor data, by hiding the location of the object (e.g., the sensor), the context information (e.g., the location) of the sensor is similarly lost. Additionally, by providing individual sensor access through objects, the potential energy savings by aggregation is lost. Jini's [14] service discovery protocol and leasing mechanisms allow client applications to discover services and manage client-server connections as the set of available services changes. Service discovery is useful for dynamic sensor networks to know what sensors and/or services are available; however, access to services remains object-based, similar to Corba. The LIME middleware [15] focuses on a different API (application programming interface), namely a shared memory scheme for mobile ad hoc components through a Linda-like tuple space [16]. Neither Jini nor LIME consider the limited energy constraints of sensor networks, and their supporting protocols are heavyweight when compared to protocols tailored to sensor networks.

Some middleware acknowledge the changing properties of wireless networks and attempt to modify their own behavior to match the conditions detected within the network. For example, both Limbo [17] and FarGo [18] reorder data exchanges or relocate components to respond to changing network conditions such as bandwidth availability or link reliability. At a lower level, Mobiware [19] supports various levels of quality of service by adapting streams within the network

with active filters deployed in the routers. Other middleware systems provide hooks to allow the applications to adapt. For example, applications built on the Odyssey platform [20] can register for notification of changes in the underlying network data rate. Similarly, the Spectra [21] component of Aura [22] monitors the network conditions and the accessible computation resources, deciding where computation should be performed based on the network transmission required to complete them as well as the expense of the computation on mobile versus fixed nodes. These advances are applicable to wireless sensor networks; however, they do not integrate any of the specific data aggregation protocols of sensor networks, nor do they consider the details of the low-level wireless protocols.

Among existing distributed computing middleware, QoS-Aware Middleware [23] provides the closest example of a middleware that can support sensor network applications. This middleware is responsible for managing local operating system resources based on application requirements specified to the middleware. The application's QoS information is compiled into a QoS profile to guide the middleware in making resource use decisions.

3.3 Middleware for Sensor Networks

Recently, much work has targeted the development of middleware specifically designed to meet the challenges of wireless sensor networks, focusing on the long-lived and resource-constrained aspects of these systems.

Both the Cougar [24] and SINA [25] systems provide a distributed database interface to the information from a sensor network with database-style queries. Power is managed in Cougar by distributing the query among the sensor nodes to minimize the energy consumed to collect the data and calculate the query result. To support the database queries, SINA incorporates low-level mechanisms for hierarchical clustering of sensors for efficient data aggregation as well as protocols that limit the re-transmission of similar information from geographically proximate sensor nodes.

AutoSec [26], Automatic Service Composition, manages resources in a sensor network by providing access control for applications so that quality of service requests are maintained. This approach is similar to middleware for standard networks because resource constraints are met on a per-sensor basis, but the techniques for collecting the current resource utilization are tailored to the sensor network.

DSWare [27] provides a similar kind of data service abstraction as AutoSec, but instead of the service being provided by a single sensor, it can be provided by a group of geographically close

sensors. Therefore, DSWare can transparently manage sensor failures as long as enough sensors remain in an area to provide a valid measurement.

While these middleware for sensor networks focus on the form of the data presented to the user applications, Impala [28], designed for use in the ZebraNet project, considers the application itself, exploiting mobile code techniques to change the functionality of the middleware executing at a remote sensor. The key to energy efficiency for Impala is for the sensor node applications to be as modular as possible, enabling small updates that require little transmission energy.

Although each of these middleware is designed for efficient use of the wireless sensor network, they largely ignore the properties of the network itself. In other words, most of these approaches do not attempt to change the properties of the network in order to manage energy, and they are not flexible enough to support different protocol stacks or different applications' QoS requirements.

4 MiLAN Middleware

As the summary of related work in the previous section shows, most sensor network research has focused on designing new network-level protocols (e.g., MAC layer, routing layer, topology control, etc.), without considering existing standards or how applications use the protocols. We argue that sensor network applications may be built on top of existing protocols, and thus some coordination framework is needed to leverage the flexibility that exists in both standardized and non-standardized network protocols.

However, to make these protocols more useful, application designers would benefit from a middleware that encapsulates the protocols, providing a high-level interface. Although the middleware discussed provide reasonable APIs, they either invent their own energy management protocols or provide limited mechanisms to adapt to the constraints of the wireless network. We argue that additional savings can be achieved if the middleware varies the actual parameters of the network over time while simultaneously meeting the requirements of the application, thereby increasing the lifetime of the network.

We are developing a new middleware named MiLAN (Middleware Linking Applications and Networks) that receives a description of application requirements, monitors network conditions, and optimizes sensor and network configurations to maximize application lifetime. To accomplish these goals, applications represent their requirements to MiLAN through specialized graphs that incorporate state-based changes in application needs. Based on this information, MiLAN makes decisions about how to control the network as well as the sensors themselves to balance application

QoS and energy efficiency, lengthening the lifetime of the application.

Unlike traditional middleware that sits between the application and the operating system, MiLAN has an architecture that extends into the network protocol stack, as shown in Figure 4. As MiLAN is intended to sit on top of multiple physical networks, an abstraction layer is provided that allows network specific plug-ins to convert MiLAN commands to protocol-specific commands that are passed through the usual network protocol stack. Therefore, MiLAN can continuously adapt to the specific features of whichever network is being used for communication (e.g., determining scatternet formations in Bluetooth networks, coordinator roles in Span [6], etc.) in order to best meet the applications' needs over time.

Figure 5 shows an overview of the interactions among MiLAN, the applications, and the sensors, together with a partial API. This figure makes a distinction between the network plug-ins and the core of MiLAN, emphasizing the separation of computation that is specific to the selected network type versus the computation that always occurs, but the API specifies only the application and sensor level operations. To make the description of the MiLAN API and the network plug-in abstraction more concrete, we use the personal health monitor application from Section 2.3 as a running example.

4.1 Application Performance

Many sensor network applications are designed to receive data input from multiple sensors and to adapt as the available sensors change over time, either as new sensors come within range or as sensors go offline when they move away or run out of energy. We assume that application performance can be described by the QoS of different variables of interest to the application, where the QoS of the different variables depends on which sensors provide data to the application. For example, in the personal health monitor, variables such as blood pressure, respiratory rate, and heart rate may be determined based on measurements obtained from any of several sensors [29]. Each sensor has a certain QoS in characterizing each of the application's variables. For example, a blood pressure sensor directly measures blood pressure, so it provides a quality of 1.0[1] in determining this variable. In addition, the blood pressure sensor can indirectly measure other variables such as heart rate, so it provides some quality, although less than 1.0, in determining these variables. The quality of the heart rate measurement would be improved through high-level fusion of the blood pressure

[1]Quality is mapped to a specific reliability in determining the variable from the sensor's data, with 1.0 corresponding to 100% reliability.

measurements with data from additional sensors such as a blood flow sensor.

In order to determine how to best serve the application, MiLAN must know (1) the variables of interest to the application, (2) the required QoS for each variable, and (3) the level of QoS that data from each sensor or set of sensors can provide for each variable. Note that all of these may change based on the application's current state. As shown in Figure 5, during initialization of the application, this information is conveyed from the application to MiLAN via "State-based Variable Requirements" and "Sensor QoS" graphs. Examples of these graphs are shown in Figures 6 and 7, respectively. Figure 6a, an abstract State-based Variable Requirements Graph, shows the required QoS for each variable of interest based on the current state of the system and the variables of interest to the application, where these states are based on the application's analysis of previously received data. For a particular state (a combination of system state (level A) and variable state (level B)), the State-based Variable Requirements Graph defines the required QoS for each relevant variable. Because variables (level C) can be named in multiple variable states (level B), MiLAN must extract the maximum QoS for each selected variable to satisfy the requirements for all variable states. Figure 6b shows the State-based Variable Requirements Graph for the personal health monitor. This application has two states, a system state that includes the patient's overall stress level, as well as multiple states for each variable that can be monitored. The State-based Variable Requirements Graph specifies to MiLAN the application's minimum acceptable QoS for each variable (e.g., blood pressure, respiratory rate, etc.) based on the current state of the patient. For example, the figure shows that when a patient is in a medium stress state and the blood pressure is low, the blood oxygen level must be monitored with a quality level of .7 and the blood pressure must be monitored with a quality level of .8.

For a given application, the QoS for each variable can be satisfied using data from one or more sensors. The application specifies this information to MiLAN through the Sensor QoS Graph, Figure 7a. When multiple sensors are combined to provide a certain quality level to the variable, we refer to this as a single "virtual sensor." Figure 7b shows the Sensor QoS Graph for the personal health monitor. This graph illustrates the important variables to monitor when determining a patient's condition and indicates the sensors that can provide at least some quality to the measurement of these variables. Each line between a sensor (or virtual sensor) and a variable is labeled with the quality that the sensor (or virtual sensor) can provide to the measurement of that variable. For example, using data from a blood pressure sensor, the heart rate can be determined with a .7 quality level, but combining this with data from a blood flow sensor increases the quality level to

1.0.

Given the information from these graphs as well as the current application state, MiLAN can determine which sets of sensors satisfy all of the application's QoS requirements for each variable. These sets of sensors define the *application feasible set* $\mathcal{F}_\mathcal{A}$, where each element in $\mathcal{F}_\mathcal{A}$ is a set of sensors that provides QoS greater than or equal to the application-specified minimum acceptable QoS for each specified variable. For example, in the personal health monitor, for a patient in medium stress with a high heart rate, normal respiratory rate, and low blood pressure, the application feasible sets in $\mathcal{F}_\mathcal{A}$ that MiLAN should choose to meet the specified application QoS are shown in Table 1. MiLAN must choose which element of $\mathcal{F}_\mathcal{A}$ should be provided to the application. This decision depends on network-level information.

4.2 Network

The properties of specific network types as well as the current condition of the network can constrain the set of feasible sets to a subset of those in $\mathcal{F}_\mathcal{A}$. As shown in Figure 5, it is the network plug-in's job to determine which sets of nodes (sensors) can be supported by the network, as well as other protocol-specific information, such as what *role* each node must play.

MiLAN uses a service discovery protocol (such as SDP [30] or SLP [31]) to find new nodes and learn when nodes are no longer accessible (due to mobility or exhausting their energy resources). The service discovery protocol must return important information about the node, such as the type of data that can be provided by that node, the modes the node can operate in, the transmission power levels, and the current residual energy level. Using this information from each currently available node, the network plug-in must determine which sets of nodes can be supported by the network.

If we assume that all nodes are on a single-hop, centralized network, bandwidth constraints place limitations on the total amount of data that can be transmitted to the application. For example, if all nodes are on a Bluetooth piconet or an 802.11 network operating in infrastructure mode, all nodes transmit data directly to the application (residing at the master in Bluetooth or the Access Point in 802.11). Therefore, the network constraint is the total rate and schedulability of all data transmitted.

However, in more complex environments such as Bluetooth scatternets, 802.11 multi-hop networks, or hybrid networks, network topology plays an important role in determining network feasibility and power costs. For example, in Bluetooth it is necessary to choose a feasible scatternet

topology, where nodes selected in the feasible set allow the network to be fully connected. In addition to ensuring the feasibility of a network configuration, we must also consider how the power costs of nodes are affected by their roles in the network (e.g., piconet masters or bridge nodes in Bluetooth scatternets, data aggregators in Directed Diffusion [2], coordinators in Span [6]). The power cost of using a node is a combination of the power to run the device, the power to transmit its data, the power to forward the data of other nodes in the set, and the overhead of maintaining its role in the network. These costs can be influenced by MiLAN through techniques such as transmission power control, efficient traffic scheduling, and the setting of different sleep states. In multi-hop networks, routing data from nodes to the application also becomes an important factor. The plug-in should know all of the network's protocol-specific features that can be modified and choose how to set these features to make sets feasible and energy-efficient.

The subsets of nodes that can be supported by the network define a *network feasible set* $\mathcal{F}_\mathcal{N}$. As only sets in $\mathcal{F}_\mathcal{A}$ provide the required application QoS, we can combine these two constraints to get an overall set of feasible sets:

$$\mathcal{F} = \mathcal{F}_\mathcal{A} \cap \mathcal{F}_\mathcal{N} \tag{1}$$

For the personal health monitor, suppose that the sensors and processors communicate using an IEEE 802.11b network. As these networks can support overall throughput of nearly 11 Mbps, the network is able to support the transmission of all data from each of the sensor sets in $\mathcal{F}_\mathcal{A}$ from Table 1 in real-time. However, if other applications (e.g., video gait monitoring [32]) are running simultaneously on the network and the personal health monitor application can only utilize 100 kbps of the throughput, the network would not be able to support the transmission of data from the ECG sensor with either 3, 5, or 12 leads. Thus, the set of network feasible sets $\mathcal{F}_\mathcal{N}$ will only partially overlap with $\mathcal{F}_\mathcal{A}$. This overlap is the set of feasible sets $\mathcal{F}$ and consists of sets 1, 3, and 5 in Table 1. MiLAN must choose a set of sensors from one of the sets in $\mathcal{F}$ based on the tradeoffs discussed in the next section. If $\mathcal{F}$ is empty, MiLAN should raise an exception to the application, allowing it to decide the appropriate action.

4.3 Tradeoffs

Among the elements in $\mathcal{F}$, MiLAN chooses an element f_i that represents the best performance/cost tradeoff. How should "best" be defined? This depends on the application—the MiLAN framework supports any method of deciding how to choose an element of $\mathcal{F}$. In most sensor network applications, we want to allow the application to last as long as possible using the limited energy of each

of the sensors. Simple approaches to choosing sensor sets may yield the set f_i that consumes the least power or that will run for the maximum lifetime before the first sensor dies. However, if we want to ensure that the application can run at the required QoS level as long as possible, we should instead optimize the total lifetime by intelligently choosing *how long* to use each feasible sensor set [33]. In some cases, there are multiple ways to schedule sensors so that the same total network lifetime is achieved. In these cases, we may want to maximize the average quality of the sensor sets over time. For some applications, the goal may be to maximize some combination of lifetime and quality. MiLAN is flexible enough to incorporate any of these or other optimization criteria.

In Figure 5, we show this tradeoff computation occurring in the core MiLAN component. After the computation is complete and the first set of sensors is chosen, the MiLAN core informs the plug-in of the selection, and the plug-in configures the network accordingly, using information about the role each sensor should play.

5 Conclusions

Current research trends suggest the power of middleware to ease the application development task in complex environments. While conventional middleware operates above the networking layer, for sensor network applications that rely on multiple and varying sensors, it is not a viable approach to manage the network completely independently of the needs of the application. We have argued that the needs of the application should be integrated with the management of the network into a single, unified middleware system. Through this tight coupling, the middleware can trade application performance for network cost, while still retaining the separation between the policy specifying how to react to a dynamic environment (obtained from the application) and the mechanisms to implement the policy (performed in the middleware). We have shown that MiLAN, a sensor network middleware that we are developing to meet these goals, can aid the development of sensor network applications. For more details on the MiLAN project, including references to related papers, please visit the project web page at `http://www.futurehealth.rochester.edu/milan/`.

References

[1] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. *IEEE Transactions on Wireless Communication*, 1(4):660–670, Oct. 2002.

[2] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. *Proceedings of ACM Mobicom '00)*, Aug. 2000.

[3] S. Singh and C. Raghavendra. PAMAS: Power Aware Multi-Access Protocol with Signalling for Ad Hoc Networks. *ACM Computer Communication Review*, 28(3):5–26, July 1998.

[4] Y. Wei, J. Heidemann, and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, June 2002.

[5] A. Cerpa and D. Estrin. ASCENT: Adaptive Self-Configuring Sensor Network Topologies. In *Twenty-first International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, June 2002.

[6] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. *ACM Wireless Networks*, 8(5), September 2002.

[7] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava. Optimizing Sensor Networks in the Energy-Latency-Density Design Space. *IEEE Transactions on Mobile Computing*, 1(1):70–80, January 2002.

[8] R. Ramanathan and R. Rosales-Hain. Topology Control of Multihop Wireless Networks Using Transmit Power Adjustment. In *Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000)*, pages 404–413, March 2000.

[9] E. Lloyd, R. Liu, M. Marathe, R. Ramanathan, and S. Ravi. Algorithmic Aspects of Topology Control Problems for Ad Hoc Networks. In *Processings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2002)*, pages 123–134, June 2002.

[10] V. Rodoplu and T. Meng. Minimum Energy Mobile Wireless Networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1333–1344, August 1999.

[11] D. Braginsky and D. Estrin. Rumor Routing Algorithm for Sensor Networks. In *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications*, 2002.

[12] W. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In *Proc. of the 5th Annual ACM/IEEE Int. Conference on Mobile Computing and Networking (MobiCom '99)*, pages 174–185, August 1999.

[13] Object Management Group. *The Common Object Request Broker: Architecture and Specification Revision 2.2.* 492 Old Connecticut Path, Framingham, MA 01701, USA, 1998.

[14] K. Edwards. *Core JINI.* Prentice Hall, 1999.

[15] A.L. Murphy, G.P. Picco, and G.-C. Roman. LIME: A Middleware for Physical and Logical Mobility. In *Proceedings of the 21st International Conference on Distributed Computing Systems*, pages 524–533, April 2001.

[16] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.

[17] N. Davies, S. Wade, A. Friday, and G. Blair. Limbo: A tuple space based platform for adaptive mobile applications. In *Proceedings of the International Conference on Open Distributed Processing/Distributed Platforms (ICODP/ICDP '97)*, Toronto, Canada, May 1997.

[18] O. Holder, I. Ben-Shaul, and H. Gazit. System Support for Dynamic Layout of Distributed Applications. In *Proceedings of the 19th International Conference on Distributed Conputing*, pages 403–411, 1999.

[19] A.T. Campbell. Mobiware: QOS aware middleware for mobile multimedia communications. In *Proceedings of the 7th IFIP International Conference on High Performance Networking (HPN)*, White Plains, New York, USA, April 1997.

[20] B. D. Noble and M. Satyanarayanan. Experience with Adaptive Mobile Applications in Odyssey. *Mobile Networks and Applications*, 4(4):245–254, 1999.

[21] J. Flinn, D. Narayanan, and M. Satyanarayanan. Self-tuned remote execution for pervasive computing. In *Proceedings of the Eighth IEEE HotOs Conference*, Elmau/Oberbayern, Germany, May 2001.

[22] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, April–June 2002.

[23] K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li. QoS-Aware Middleware for Ubiquitous and Heterogeneous Environments. *IEEE Communications Magazine*, 39(11), 2001.

[24] P. Bonnet, J. Gehrke, and P. Seshadri. Querying the physical world. *IEEE Personal Communication*, 7:10–15, October 2000.

[25] C. Jaikaeo, C. Srisathapornphat, and C.-C. Shen. Querying and Tasking in Sensor Networks. In *SPIE's 14th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Control (Digitization of the Battlespace V)*, Orlando, Florida, April 24–28 2000.

[26] Q. Han and N. Venkatasubramanian. Autosec: An integrated middleware framework for dynamic service brokering. *IEEE Distributed Systems Online*, 2(7), 2001.

[27] S. Li, S. Son, and J. Stankovic. Event detection services using data service middleware in distributed sensor networks. In *Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks*, April 2003.

[28] T. Liu and M. Martonosi. Impala: A middleware system for managing autonomic, parallel sensor systems. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'03)*, June 2003.

[29] J.C.D. Conway, C.J.N. Coelho, D.C. da Silva, A.O. Fernandes, L.C.G. Andrade, and H.S. Carvalho. Wearable Computer as a Multi-parametric Monitor for Physiological Signals. In *Proceedings of the IEEE International Symposium on Bioinformatics and Bioengineering (BIBE)*, pages 236–242, 2000.

[30] S. Avancha, A. Joshi, and T. Finin. Enhanced Service Discovery in Bluetooth. *IEEE Computer*, 35(6):96–99, June 2002.

[31] Service Location Protocol (SLP). http://www.ietf.org/html.charters/svrloc-charter.html.

[32] S. L. Dockstader and A. M. Tekalp. Multiple Camera Tracking of Interacting and Occluded Human Motion. *Proceedings of the IEEE*, 89(10):1441–1455, Oct. 2001.

[33] Mark Perillo and Wendi Heinzelman. Simple Approaches for Providing Application QoS Through Intelligent Sensor Management. *Elsevier Ad Hoc Networks Journal*, 1(2-3):235–246, 2003.

Wendi B. Heinzelman is an assistant professor in the Department of Electrical and Computer Engineering at the University of Rochester. She received a B.S. degree in Electrical Engineering from Cornell University in 1995 and M.S. and Ph.D. degrees in Electrical Engineering and Computer Science from MIT in 1997 and 2000 respectively. Her current research interests lie in the areas of wireless communications and networking, mobile computing, and multimedia communication. She is an elected member of the Design and Implementation of Signal Processing Systems (DISPS) Technical Committee of the Signal Processing Society and a member of Sigma Xi, the IEEE, and the ACM.

Amy L. Murphy is currently an assistant professor in the Department of Computer Science at the University of Rochester in New York. She received a B.S. in Computer Science from the University of Tulsa in 1995, and M.S. and D.Sc. degrees from Washington University in St. Louis, Missouri in 1997 and 2000 respectively. Her research interests include the development of standard algorithms for mobility and the design, specification, and implementation of mobile middleware systems. These topics are integrated under the theme of enabling the rapid development of dependable applications for both physically and logically mobile environments. For more information, see http://www.cs.rochester.edu/ murphy/.

Hervaldo Sampaio Carvalho M.Sc., M.D., is a Ph.D. candidate in the Department of Computer Science at the University of Minas Gerais in Brazil. He is also a researcher in the Center for Future Health at the University of Rochester in New York and a professor of medicine/cardiology in the School of Medicine at the University of Brasilia in Brazil. Dr. Carvalho's research interests are in the area of data fusion algorithms and implementation in sensor networks and the design of a body-worn sensor network for personal long-term health monitoring.

Mark A. Perillo is a graduate student in the Department of Electrical and Computer Engineering at the University of Rochester. He received a B.S. degree in Electrical Engineering in 2000 and an M.S. degree in Electrical and Computer Engineering in 2002, both from the University of Rochester. His current research interests lie in the area of wireless ad hoc and sensor networks. He is a member of Tau Beta Pi and a student member of the IEEE.

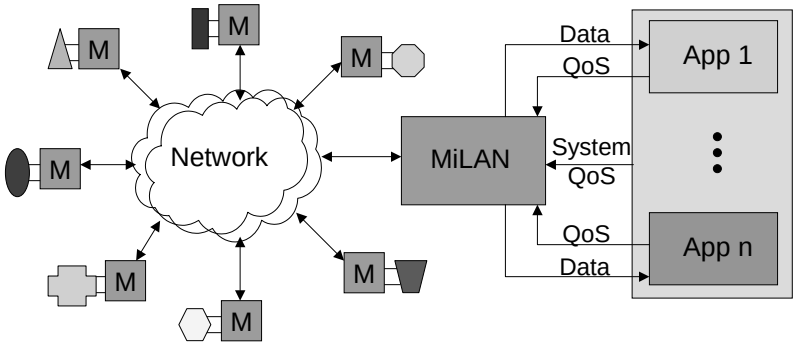


Figure 1: System that employs MiLAN. Each sensor runs a (possibly scaled-down) version of MiLAN. MiLAN receives information from applications about their QoS requirements, a system user about the desired interaction among the applications, and the network about available components and resources. MiLAN then decides how best to configure the network to support the applications.

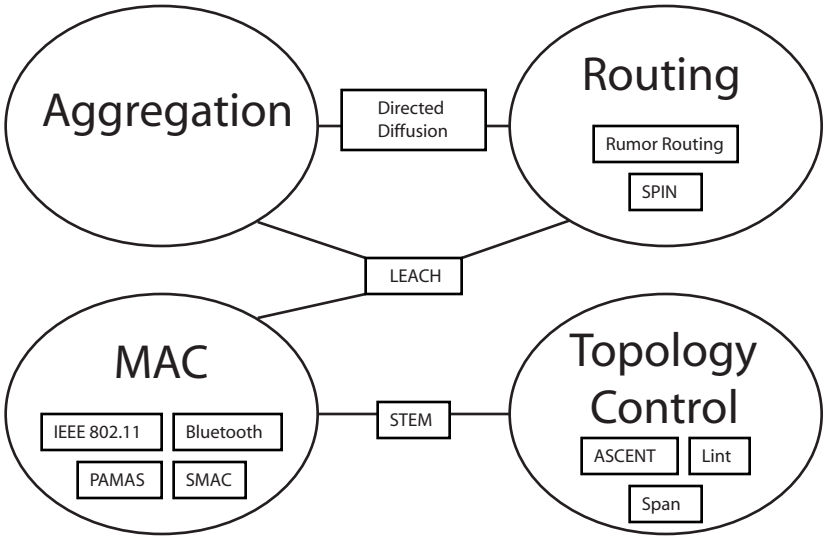


Figure 2: Relationships between different sensor network protocols and the network services they provide.

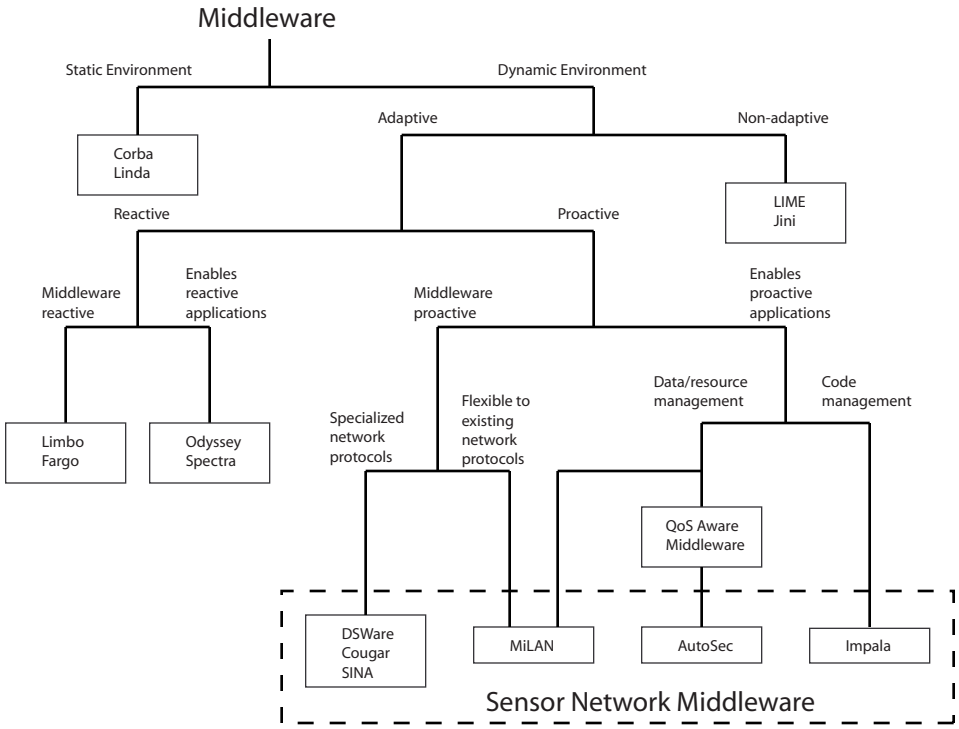


Figure 3: Relationships among different middleware. In this figure, "Middleware reactive" refers to middleware that reacts itself to changes in network behavior, whereas "Middleware proactive" refers to middleware that proactively changes the network functionality. Similarly, "Enables reactive applications" refers to middleware that provides hooks so that applications can react to changes in the environment, whereas "Enables proactive applications" refers to middleware that accepts information from the application about how to respond to changes in the network.

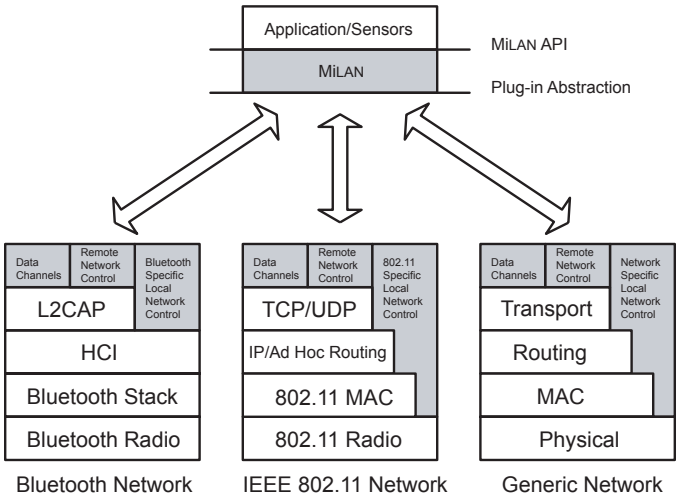
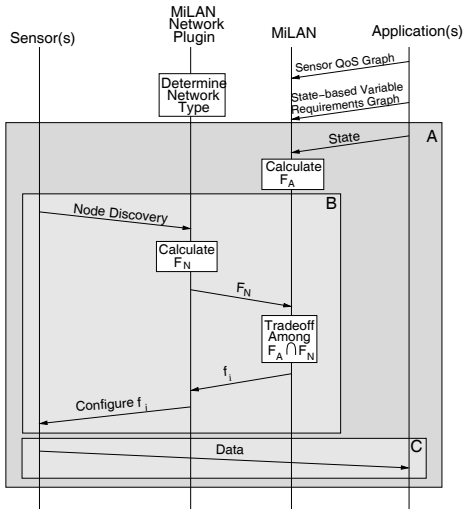


Figure 4: MiLAN components (shaded). MiLAN presents an API through which the application represents its requirements with regard to different sensors that may be available. MiLAN also presents an abstraction from the network-level functionality through which it issues commands to determine available sensors and configure the network.



(a)

```
/* application: set QoS */
int define_qos_graph(SQoS *sqos);

/* application: set variable requirements */
int define_variable_graph (SVRG *svrg);

/* sensor:  push data */
int send_data(int dest_milan_id, int data_length,
              char *data)

/* application: set system state */
int update_state(int state);

/* upcall - MILAN gives data to application */
int recv_data(int *src_milan_id, int *data_length,
              unsigned char *data, int *packet_type);
```

(b)

Figure 5: (a) High level overview of MiLAN operation. Segment A repeats when the application changes its state based on data received from the sensors. Segment B repeats when sensors arrive in the network. Segment C repeats as data arrives from each sensor, and represents the normal operation of MiLAN conveying information from the sensors to the application. (b) Partial MiLAN API. Applications represent their Sensor QoS Graph to MiLAN using the `SQoS` structure and the `define_qos_graph` function, and they represent their State-based Variable Requirements Graph to MiLAN using the `SVRG` structure and the `define_variable_graph` function. After initialization, sensors send data to the application using the `send_data` function and applications receive the data from MiLAN via an upcall from the `recv_data` function. Applications specify to MiLAN that they have changed state through the `update_state` function.

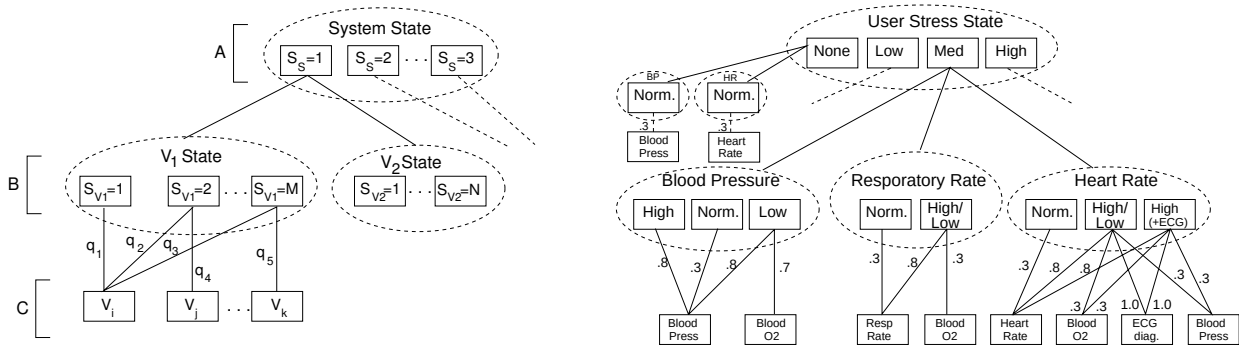
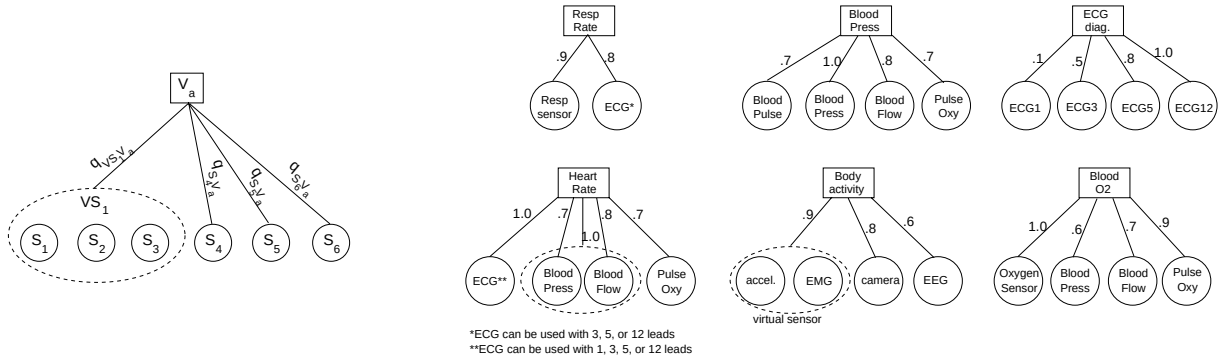


Figure 6: State-based Variable Requirements Graph for specifying the variables and the required QoS when the application is in various states. (a) Abstract example. (b) Example for the personal health monitor application. This graph illustrates only a subset of the application's possible states.



(a) (b)

Figure 7: Sensor QoS Graph for specifying which sensors, or sets of sensors, can provide what level of QoS for each variable. (a) Abstract example. (b) Example for the personal health monitor application. This graph illustrates only a subset of the variables that should be considered by the application.

Table 1: Feasible sets $\mathcal{F}_{\mathcal{A}}$ for the personal health monitor application for a patient in medium stress with high heart rate, normal respiratory rate, and low blood pressure.

Set #	Sensors
1	Blood flow, Resp. rate
2	Blood flow, ECG (3 leads)
3	Pulse oxymeter, Blood pressure, ECG (1 lead), Resp. rate
4	Pulse oxymeter, Blood pressure, ECG (3 leads)
5	Oxygen measurement, Blood pressure, ECG (1 lead), Resp. rate
6	Oxygen measurement, Blood pressure, ECG (3 leads)