

MiLAN: Middleware Linking Applications and Networks

Amy Murphy
University of Rochester
Center for Future Health and
Department of Computer Science
Rochester, NY 14627
murphy@cs.rochester.edu

Wendi Heinzelman
University of Rochester
Center for Future Health and
Dept. of Electrical and Computer Engr.
Rochester, NY 14627
wheinzel@ece.rochester.edu

November 13, 2002

Technical Report #xxx

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. This is also the goal of the proposed MiLAN middleware platform, but MiLAN's unique feature is its ability to continuously *control* the network functionality with respect to the application's changing demands.

Applications targeted by MiLAN are characterized by their ability to adapt to changing sets of available components, and their need to further constrain the active components for application-performance reasons. Physical resources (e.g., transmission distance, bandwidth) and minimum application performance limit the input to certain subsets of available components. It is the job of MiLAN to identify these *feasible sets* and determine which set optimizes the tradeoff between application performance and network cost (e.g., energy dissipation). MiLAN must then configure the network so that components in the selected feasible set are linked to the application. A key feature of MiLAN is the separation of the policy for managing the network, which is defined by the application, from the mechanisms for implementing the policy, which is effected within MiLAN. This report describes the initial design of MiLAN as well as our plans for future research.

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 an abstract coordination interface to the application programmer.

Often in distributed systems, the nature of the network has a major impact on the application performance. This is especially true when the underlying network is primarily wireless and/or the system endpoints are mobile and constrained by battery lifetime. In these environments, local network connectivity changes over time and must be closely monitored and managed to best serve the needs of the applications. However, multiple applications utilizing the network may have different criteria for how to best utilize the available resources. For example, consider a security application and an entertainment application accessing both audio and low-quality video from two different wireless sources. Suppose a new node joins the network offering high-quality video, but the network cannot support the transmission of all three streams. The goal of the security application may be to receive the high-quality video stream, while the goal of the entertainment application may be to keep the low-quality video and audio streams. This creates two problems that cannot be easily solved using conventional middleware systems: how should the needs of each application that utilizes the distributed network resources be implemented, and how should the conflict created by the different needs of the applications be resolved?

Generally, the scenario space we target is one with many low-level devices offering services in which some fundamental resource limitation is pushed, e.g., network bandwidth is exceeded and/or component energy is scarce. This brings out two core issues of distributed systems that are not addressed in conventional middleware. First, rather than an application *reacting* to the changing network environment, it is important for the application to actually *control* the environment to optimize its performance. Second, if there are multiple applications sharing a common set of network components, it is important to develop a method of resolving conflicting needs that maximizes an overall *utility*. While it is possible to build applications to address these issues in each unique instance, a properly designed middleware system will both speed the development of multiple applications in similar domains, as well as increase the dependability and reliability of applications by reusing the set of well-tested management components of the middleware.

The uniqueness of our approach is the integration of network control into the middleware, enabling application-directed network reconfiguration. For this reason, we call our system MiLAN: *Middleware Linking Applications and Networks*. From the application, MiLAN receives a set of performance specifications with respect to different system components as well as information specifying how different applications should interact. From the network, MiLAN monitors for available components and overall resources such as power consumption and bandwidth. Combining this information in some optimal manner, MiLAN continuously adapts the network configuration to best meet the applications' needs and balance performance for cost.

The remainder of this report explains the ideas behind MiLAN in detail. First, we describe several application scenarios that cannot easily be optimized using today's technology but that MiLAN will support. Then we outline the technical challenges that must be addressed in the design of a middleware that allows applications to control the network configuration, and we describe the impact we expect this middleware platform will have on application development. Finally, we discuss related work in this area and provide conclusions.

2 Application Scenarios

Although wireless networks are becoming common, the range and types of applications for wireless networks largely follow those of applications on wired networks, namely, setting up point-to-point connections between end-hosts. However, new application scenarios are being proposed that exploit some of the unique properties of wireless networks and distributed systems. For example, in sensor networks, the components typically cooperate to accomplish a common goal (e.g., monitoring a region), rather than compete for available resources. In multimedia applications, the components

can take advantage of the broadcast nature of radio technology in order to more efficiently distribute information to multiple recipients, rather than create independent connections. In applications of this new style, the applications themselves should dictate which nodes are active and how they access the channel. In this section, we discuss some specific 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. Suppose, however, that the channel linking all the sensors to the application software cannot handle the transmission of all the sensors' data to the end-user, due to bandwidth constraints of the wireless channel or energy constraints of the individual sensors. Based on data obtained from the sensors that are active at any time, the application knows which sensors' data it needs to maximize performance. For example, suppose initially one sensor in every square meter is sending data to the application and all other sensors are inactive. If the application discovers, through analyzing the data from the active sensors, that an event of interest is occurring in part of the area being surveyed, it needs the ability to control and activate other sensors in that area in order to maximize performance.

To accomplish this with today's technology, the application would need to track the location of all of the sensors, individually enable and disable sensors based on the collected data, as well as ensure that the enabled sensors do not send data at rates that are in excess of the network resources. We believe that through a well-defined API, an application should dynamically declare its needs (e.g., information from high activity areas) with respect to the low-level components (e.g., sensors), and the middleware should assume the task of managing the sensor components, thereby allowing independence of the application from whatever network it executes on top of.

2.2 Virtual Campus

Many ubiquitous computing efforts are focused on supporting university students and faculty with computer-based applications. These include digital whiteboards, note sharing, exam taking, viewing live and taped lectures, support for collaborative projects, etc. Many times, these applications exploit wireless connections among various battery powered devices. Our interest comes not from the design of the applications, but from their interactions on a common network. For example, if one student is watching and listening to a taped lecture (e.g., using an 802.11 WLAN) and a second student listening to a live lecture moves into the same area and must share the bandwidth, the network needs to be reconfigured to support both students' applications. Possible solutions include dropping the video but retaining the full quality audio for both students or delaying the taped lecture until resources become available. These system resource allocation choices require a careful balance between the user characteristics (e.g., students vs. faculty), the application characteristics (e.g., live vs. taped), and the current network resources.

Supporting the seamless integration of multiple applications and managing the network constraints is not easy with existing middleware platforms. We propose that distributed applications be built independently, but augmented with a minimal set of hooks that allow them to specify their needs to the middleware (e.g., first reduce video quality, but if the reduced quality video cannot be supported, then drop the video but keep the audio, otherwise pause the lecture, etc.). With these hooks and user-supplied utility metrics specifying the desired interaction of different applications, the middleware can manipulate the evolving system to best support faculty and students and the various applications running in the virtual campus.

2.3 Medical Monitoring

The Center for Future Health's *Smart Medical Home* provides another application domain. This Center at the University of Rochester is a consortium of groups from engineering, computer science, research and clinical medicine, and health services that was founded with the goal of utilizing technology to enable individuals to take care of their own health. Within the Smart Medical Home, the residents interact with several technological devices designed both to assist them in daily living tasks and to monitor and control medical conditions. This is a challenging environment in which dependable, flexible, interoperable applications must be designed. At the lowest hardware level, the home is composed of sensors (e.g., microphones, video cameras, thermometers etc.), actuators (e.g., speakers, video displays, medical devices, etc.), and computational units (e.g., processors attached to individual sensors, inside personal digital assistants, and within laptop and desktop computers). The computational units run applications that take the sensor data as input and produce control data for the actuators and/or other applications.

The key to the effectiveness of the Smart Medical Home is enabling seamless coordination among these devices to accomplish the high level application tasks with a required reliability. Consider a heart monitor application running on a PDA that records potential heart problems in a local database. Input to this monitor may come from a number of sensors such as a body position monitor, an electrocardiogram, a blood pressure sensor, a body temperature sensor, a pulse oximeter, etc. While ideally the heart monitor application would continuously obtain data from every sensor, this will not always be possible due to the wireless channel and battery-operated sensors placing constraints on: *sensor availability*, which will change over time due to sensors moving into and out of communication range, interference changing link quality, and sensors running out of energy, and *bandwidth limitations*, which dictate the maximum amount of data that can be provided to the application in real-time. Furthermore, as most sensors in the Smart Medical Home environment will be battery-operated, even if these network constraints were not an issue, it would be desirable to use only a subset of available sensors at a given time in order to achieve *energy-efficiency*. Using all sensors' data will greatly increase overall energy drain but may provide only minimal improvement to application performance compared with using only a subset of the sensors. Therefore, there is a tradeoff between application performance and energy dissipation that must be considered when configuring the network. Finally, applications in the Smart Medical Home must also account for the fact that the application focus may change over time. For example, the set of sensors that need to be activated when a possible emergency medical condition is detected may be different than the set of sensors activated when nothing is detected wrong with the user.

While our initial efforts focus on the heart monitor, we will eventually turn toward integrating many of the different applications of the Home, including an object recognition system that uses video cameras to monitor the movement of various objects on tables, including coffee cups, eyeglasses, and keys [10], a voice interaction system that converses with the home's resident about his or her medication needs [1], and a video system that monitors an individual's gait over time for possible signs of degenerative mobility [7].

2.4 Overall Goal

The unique feature of the applications described above is that they must be *proactive*, actively affecting the network when the environment changes, rather than *reactive*, simply responding to the changing environment. While possible, creating such proactive applications is neither straightforward nor simple using existing middleware platforms. Applications would need to continuously search for new low-level data sources that become available and keep track of data sources that are no longer available, know what network-level protocol is being employed, and monitor low-

level protocol information. This would greatly complicate the design of applications, changing the focus from one on the application to one on network resources and networking protocols. In essence, while the *policy* of how to manage and control the dynamic network should be left to the application, the *mechanisms* for implementing the policy should reside at the middleware. We are developing a generic middleware platform that provides this type of policy/mechanism split by allowing the application to specify its low-level data needs, how these needs vary over time, and how its performance is affected by different sets of input data. Knowing this information will allow the middleware to manipulate the network configuration, taking into account the tradeoffs between application performance, data availability, and energy consumption.

3 MiLAN Solution Strategy

To support applications that need to trade performance for energy cost, we are developing a new middleware, named MiLAN (Middleware Linking Applications and Networks), which accepts performance needs from the application, monitors network conditions, and optimizes the network configuration on behalf of the application. Our approach is based on a new technique of representing the application needs as a specialized graph, designing network feasibility templates for various underlying networks (e.g., Bluetooth and 802.11), and constantly analyzing current conditions and affecting the network to balance application utility and energy cost.

Unlike traditional middleware that sits between the application and the operating system, MiLAN, which intends to control the network, has an architecture that extends into the network protocol stack as shown in Figure 1. The interface to the application and the low-level components allows the application to provide a graph specifying the utilities of the low-level components that may be available over time, as well as a mechanism for MiLAN to control the low-level components. As MiLAN is intended to sit on top of multiple physical networks, an abstraction layer converts MiLAN commands to protocol-specific commands that are passed through the usual network protocol stack. The details of the application graph will be given in Section 3.1 and the details of the network control information in Section 3.2.

Our initial design explores the tradeoff between performance and network cost in the context of a single, centralized application that takes data from multiple distributed sensors. Specifically, we use the heart monitor as a test application, where the utility of the application is measured in terms of reliability of measuring certain variables. In this section we will go through the main ideas of MiLAN, including the definition of application performance, the network cost evaluation, the energy concerns, and the tradeoffs to be evaluated by the system.

3.1 Application Performance

Consider the heart monitor application described in Section 2.3, where several variables must be calculated based on data from sensors. For example, the heart rate, respiratory rate, blood oxygen level, blood flow, blood pressure, ECG signal, and position and activity of the person being monitored are all important variables in determining if the heart is healthy or developing problems [4]. If this application relies on a single sensor, for example a blood pressure sensor, it would have a certain *reliability* in characterizing each of the above-stated variables. For example, a blood pressure sensor directly measures blood pressure and therefore provides 100% reliability in determining this variable, but it indirectly measures variables like heart rate and blood flow and therefore provides less than 100% reliability in determining these parameters. The reliability of these variables would be improved by including data from additional sensors such as ECG, heart rate, and blood oxygen level sensors [4]. Figure 2 shows an example of the different parameters that

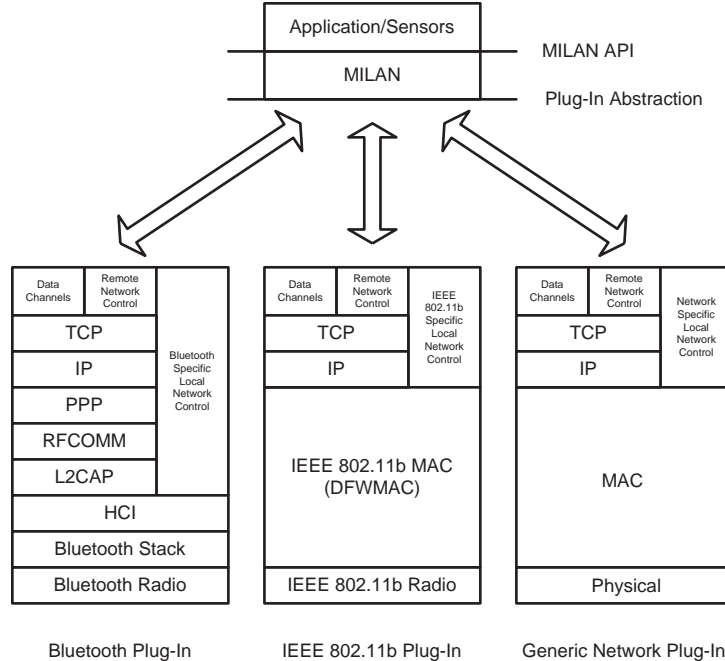


Figure 1: MiLAN protocol stack. MiLAN presents a well-defined API through which the application represents its desires with regard to low-level components. MiLAN also presents an abstraction from the network-level functionality through which it issues commands to determine available components and configure the network. The network-specific plug-ins convert MiLAN commands to the specific network protocol employed.

are important to monitor to determine the condition of the heart, as well as the sensors that can provide direct or indirect measurements of these variables. Lines between a sensor and a variable mean that the sensor can either directly or indirectly measure that variable, and the numbers on the lines represent example reliabilities in using that sensor’s data to measure the variable.

Ideally we would want to provide as much data to the heart monitor application as possible to increase its reliability¹. There are, however, several drawbacks to such an approach. For one, the lifetime of the system may be reduced if all sensors transmit all the time. Instead, the lifetime may be extended by requiring data from only a subset of available sensors, slightly decreasing the overall reliability but allowing some nodes to save energy. Furthermore, the network capacity may not support the transfer of all of the data for the heart monitor. Thus, it is important to devise an automatic way of dynamically choosing an appropriate subset of sensors to achieve the required reliability while minimizing cost and staying within network resource constraints.

In general, an application knows how it performs given data from different combinations of low-level components. This information must be transmitted to MiLAN. We propose using a graph-based approach to allow the application to specify this performance information. In the graph, nodes with links emanating from them represent the low-level components (e.g., sensors) and nodes with links ending at them represent the variables the application is trying to measure using sensor data. The weights of the links represent how accurately the sensor data at the tail of the link can determine the variable at the head of the link. Figure 2 shows an example of an application graph for the heart monitoring application.

This graph must be extended if sensor data can be *fused* and the fused data used to determine

¹Note that adding additional sensors does not always increase reliability. As the blood pressure sensor directly monitors blood pressure, obtaining data from other sensors does not affect reliability in measuring this parameter.

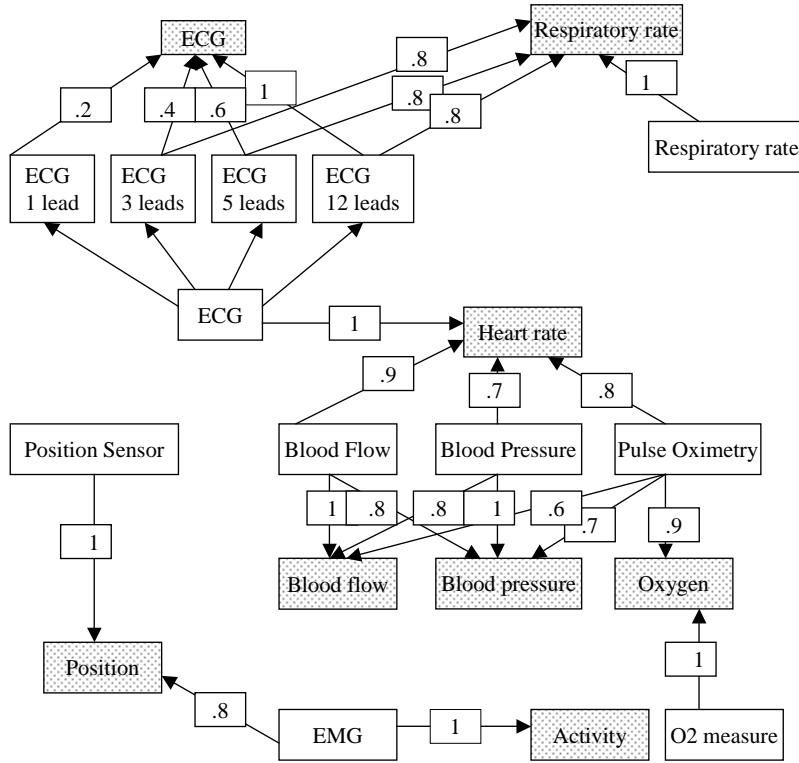


Figure 2: Example application performance graph for a heart monitoring application. For this application, sensors (unshaded boxes) take measurements that directly or indirectly measure different variables (shaded boxes) with a certain reliability (shown on the links).

the variables. In this case, we can add a node for every combination of sensors whose data can be fused. However, this approach adds considerably to the complexity of the graph presented by the application to MiLAN since the number of fused data sets can grow exponentially. Hence, we are currently exploring other ways to represent fused data.

Finally, it is possible that the contribution of each low-level component to the application performance will change over time with regard to the context of the application. This information must be conveyed to MiLAN to ensure the highest application performance over time. Initially for the heart monitor application we consider three states for the application corresponding to whether the user is healthy, unhealthy, or in between. The application provides reliability information for each of the three states (e.g., a different reliability graph for each state), and part of the MiLAN API allows the application to signal a change from one state to another.

In addition to the application graph(s), the application must specify minimum performance (e.g., reliability) bounds that guide MiLAN in choosing an appropriate set of sensors. Using the application graph that specifies the application variables' reliabilities and the application-specified minimum reliability at a given time, MiLAN can choose which sensors to use to meet or exceed the application's specified reliability while considering power costs and bandwidth constraints. For example, suppose the application sends MiLAN the following request for variable reliabilities:

- Heart rate must be measured with reliability ≥ 0.5

- Blood oxygen level must be measured with reliability ≥ 0.7
- Blood pressure must be measured with reliability ≥ 0.8

Based on the graph the application gives to MiLAN (see Figure 2), MiLAN can determine that the following combinations of sensors will satisfy the application requirements:

	Sensors (S_i)	Total Reliability ($R(S_i)$)
1	A,B,C	$1 + 0.9 + 1 = 2.9$
2	A,C,E	$1 + 1 + 1 = 3.0$
3	B,C	$0.8 + 0.9 + 1 = 2.7$
4	B,C,E	$0.8 + 1 + 1 = 2.8$
5	C,E	$0.7 + 0.9 + 1 = 2.6$
6	B,D	$0.9 + 0.9 + 1 = 2.8$
7	C,D,E	$0.9 + 1 + 1 = 2.9$

where A is the ECG sensor, B is the pulse oximeter sensor, C is the blood pressure sensor, D is the blood flow sensor, and E is the oxygen sensor. Here we use the sum of the individual variable reliabilities to represent the total reliability of the sensor set $R(S_i)$, but in general other total reliability functions may be employed (e.g., if $R(S_i, j)$ is the reliability obtained using sensors in set S_i to measure variable j , then total reliability $R(S_i) = \min_j R(S_i, j)$ may be more appropriate for some applications).

These sets of sensors define an *allowable* set $\mathcal{F}_{\mathcal{R}}$, a set of possible combinations of sensors that provide reliability greater than or equal to the application-specified minimum acceptable reliability r_j for each of the application variables j in set J .

$$\mathcal{F}_{\mathcal{R}} = \{S_i : \forall j \in J \quad R(S_i, j) \geq r_j\} \quad (1)$$

For the example above, $r_1 = 0.5$, $r_2 = 0.7$, and $r_3 = 0.8$ and $\mathcal{F}_{\mathcal{R}} = \{(A, B, C), (A, C, E), (B, C), (B, C, E), (C, E), (B, D), (C, D, E)\}$. All sets in $\mathcal{F}_{\mathcal{R}}$ are allowable, so the question becomes which set $S_i \in \mathcal{F}_{\mathcal{R}}$ should be chosen? This will depend on the other parameters for the system, namely network cost and power consumption, as will be described next.

3.2 Network Cost

One of the complexities of the environment is the inherent bandwidth limitations with transmitting data over a wireless network. While eventually we will consider heterogeneous (e.g., wired and wireless) networks, for our initial work we assume that all sensors are connected via a wireless network with bandwidth constraints. MiLAN must have a cost representation for using different sets of sensors given the particular networking protocol that is being employed. This is accomplished by defining network templates that specify feasible sets of nodes. These templates are described once for each network type and are exploited by MiLAN depending on the environment in which it is functioning.

If we assume all sensors are on a single, centralized network, bandwidth constraints place limitations on the total amount of data that can be transmitted to the application by all the sensors. In this case, the network template just specifies the maximum possible data rate. For example, if we assume nodes are all on a Bluetooth piconet or an 802.11 network operating in infrastructure mode, all sensors transmit data directly to the application (through the master in Bluetooth or the Access Point in 802.11). Therefore, the network constraint that must be met is keeping the total rate of all data transmitted to the application below a threshold.

However, in a more complex environment such as Bluetooth scatternets or 802.11 infrastructureless mode, capacity can be increased by employing spatial reuse and thus node location plays an important role in determining network costs. In this scenario, there will be dependencies among the cost of including different sensors, and this must be represented to MiLAN using a network template that defines the properties of the particular protocol. For example, a Bluetooth template should specify a maximum of 700 kbps and eight nodes per piconet with one node assigned the role of master. It may also be meaningful to include as one of the parameters in this template the proximity of the nodes to one another. For example, two nodes that are physically far apart cannot participate in the same Bluetooth piconet, or, in 802.11, lower transmit levels can be used for nodes that are close together. We are currently exploring how to create such network templates.

As a parallel concern, MiLAN must continuously monitor the network to find new nodes and learn when nodes are no longer accessible (due to mobility or running out of energy). When new nodes are discovered through the network-level discovery method (e.g., Bluetooth discovery), MiLAN must find out sensor-specific information about these nodes in order to know how to appropriately incorporate their availability. To do this, we use the IEEE 1451 standard that defines a Smart Transducer Interface for Sensors and Actuators. IEEE 1451 defines a transducer electronic datasheet (TEDS) that provides a standardized format for sensors to specify their type, attributes, operation and calibration information [21]. Thus if the sensors are all IEEE 1451 compliant, the TEDS information can be passed up to MiLAN and MiLAN can incorporate the specific functionality of the newly discovered sensor within the application framework.

As with the reliability analysis, it may be the case that only certain subsets of sensors meet the network constraints. These subsets define a *network feasible set* as follows:

$$\mathcal{F}_{\mathcal{N}} = \{S_i : N(S_i) \leq n_o\} \quad (2)$$

where $N(S_i)$ is the total “cost” of using all the sensors in S_i and n_o is the maximum cost supported by the network. For example, if a centralized network is assumed, n_o is the maximum data rate that the network can support (e.g., 700 kbps in Bluetooth, 11 Mbps in 802.11) and

$$N(S_i) = \sum_{s_j \in S_i} R_b(s_j) \quad (3)$$

where $R_b(s_j)$ is the bit rate for the j^{th} sensor. Only sets in $\mathcal{F}_{\mathcal{N}}$ can physically be supported by the network. From our analysis before, we know that only sets in $\mathcal{F}_{\mathcal{R}}$ are allowable from the application point of view. Thus, we can combine these two hard constraints to get an overall feasible set:

$$\mathcal{F} = \mathcal{F}_{\mathcal{R}} \cap \mathcal{F}_{\mathcal{N}} \quad (4)$$

MiLAN must choose a solution from one of the sets in \mathcal{F} .

For our initial investigation, we assume that all nodes have fixed data rates and MiLAN makes optimization decisions based on this value. In reality, however, some sensors can produce data with varying rates and degrees of accuracy, thus affecting application reliability as well as network cost. The MiLAN infrastructure provides a concentrated place for manipulating this aspect of the sensors as well. An initial idea is to include extra nodes in the application reliability graph that represent the different data rates, but this unreasonably increases the cost of the optimization computation. Therefore, we are exploring other ways of including variable data rate sensors into the MiLAN framework.

3.3 Energy

The final parameter to consider when deciding which sensors to include in the network is energy dissipation. There is an energy cost to a sensor whenever it transmits its data over the wireless

network. This cost may depend on the distance between the sensor and the intended receiver if we assume power control can be employed, and it may depend on which subset of sensors is chosen and what *role* the sensor must play in the network (e.g., master, gateway between multiple piconets, etc.). However, initially we will assume that the energy cost per node is independent of which set of sensors is chosen. Thus the total cost in terms of energy dissipation for each set S_i is:

$$C_P(S_i) = \sum_{s_j \in S_i} C_P(s_j) \quad (5)$$

where $C_P(s_j)$ is the power “cost” to node s_j if it were to be selected to transmit its data. This can either be a representation of energy dissipation or it can be a weighted metric where the “cost” is high for nodes that are running low on energy. For example, metrics similar to those used by Singh et al. [29] could be employed here as cost metrics.

3.4 Tradeoffs

Among the feasible sets in \mathcal{F} , MiLAN will choose a set S_i that represents the best performance/cost tradeoff as determined by the reliability of the set $R(S_i)$ and the cost of the set $C_P(S_i)$. How exactly should “best” be defined? This will depend on the application. One possible way to choose the appropriate set S_i at a given time is to minimize current energy drain by choosing the set from \mathcal{F} that has the lowest cost $C_P(S_i)$. While this is a good short-term solution, the application may also be interested in performance over long periods of time. Rather than choosing the set of sensors that is currently the best and operating using these sensors until a change must be made (e.g., one of the sensors dies), a better approach may be to look ahead and find the best *schedule of sets* that optimizes a criterion over time (e.g., finding a schedule of sets that all meet the reliability criterion and will last the longest). This optimization problem can be solved using a linear program and is thus solvable in polynomial time [20]. However, the time to solve this optimization problem may still be prohibitive due to the size of the problem. We are exploring the tradeoffs in terms of system lifetime and computational complexity for solving this optimization problem.

A further complication arises when we consider multiple applications working in the same network, sharing sensors. Returning to the sample applications presented in Section 2, multiple applications must operate within the virtual campus or the Smart Medical Home. In these examples, there is a “super-user” that knows how the different applications should interact and can thus specify the utility of each application to enable MiLAN to solve conflicts that may arise in the desires of different applications. For example, the super-user may specify that live lectures always have precedence over taped lectures or the heart monitor always has precedence over the video monitor for memory assistance. Using this information, MiLAN can determine which system components to connect to satisfy the super-user desires as well as the individual application desires. If there is no such super-user that knows how the different applications should interact, MiLAN must assume all applications have equal “weight” and give each application its fair share of resources. For example, for two non-cooperating applications sharing bandwidth in a wireless network, each application would be given half the total bandwidth and MiLAN would operate independently for each application, optimally using the total bandwidth allocated to that application.

Modeling systems using reliability graphs and network templates allows MiLAN to abstract away details of the underlying sensors and network configuration from the applications. In the medical example we used here to make the MiLAN functionality more concrete, application performance is measured in terms of reliability in measuring several different parameters using sensor data input. For other applications using MiLAN, a similar graph-based approach can be used where applications specify the “goodness” of data from low-level components in achieving one or more goals of the

application. For example, in the security application discussed in the introduction, the “goodness” of using high resolution video is higher than the “goodness” of using low resolution video plus audio, whereas for the entertainment application, the “goodness” of using low resolution video plus audio is higher. Each application must determine the relationship between its performance and all possible low-level components that may be available at any time and transfer this information to MiLAN through the API. MiLAN will then take this information, combined with the network-level information, and choose an appropriate set of sensors to connect over time, considering the utility specified by a super-user to resolve conflicts. We believe that this tight coupling between the functions of the middleware and the network layer will provide greater support to applications than traditional approaches where these levels are designed to be independent from each other.

3.5 Extending MiLAN

The previous sections described the basic MiLAN framework that allows application performance to be traded-off for energy savings while meeting network and application constraints within a centralized framework. Beyond this initial investigation, we will explore extensions that are necessary to allow for distributing the application and middleware among multiple processors.

Distributing application computation among multiple processors may reduce overall energy drain by trading local computation for communication. For example, the output of two of the sensors from the heart monitor can be combined on a processor that resides at one of the two sensors, with this result then provided to another part of the application on another processor. Essentially, this would create a new “virtual” sensor that produces data that is a combination of multiple low level sensors. This leads to changes in the communication patterns, restricting communication to smaller sets of sensors, yet increasing communication across these sets of sensors. A potential gain is in the exploitation of additional distributed processing power, parallelizing the application. How to represent this distributed processing within our formulation is an open question that we will explore.

Another interesting possibility is that of moving parts of the computation during the lifetime of the system. This can be as simple as moving part of the application processing to a specific sensor, exploiting one of the known benefits of mobile code for increasing locality of processing and data. This is particularly important if more than one application is processing the same data in the same manner (e.g., performing a standard image transformation on the output of a video camera). By doing this computation a single time at the source of the data, the overall system computation is reduced. Of course, it is non-trivial not only to determine which parts of the system computation are being repeated, but how they can be distributed dynamically. We must consider both the possible savings by concentrating the computation as well as the possible costs of moving this computation to a less powerful processor located on a sensor, the power consumption of this processor, and the cost of sending the results of the actual computation to the main application. These extensions will be explored.

In addition to distributing the application, it is also possible to distribute the middleware itself. In the centralized approach described thus far, an optimal decision is taken about which sensors will send data at a given time. If we relax the optimality constraint, it is possible to distribute the application reliability information to the network components. With this in hand, the sensors can monitor the environment and exchange basic information in a peer-to-peer manner, making independent decisions concerning their participation in the application. In addition to the tradeoff in optimality, we will also need to consider the additional network traffic and the additional energy cost of distributing the decision making.

4 Related Work

Our environment is characterized by applications that collaborate to achieve a common goal. This is very different than conventional applications that do not try to support other applications using the same low-level network components. For example, typical network-level protocols, such as media access control (MAC), are developed to give each network component its fair share of network resources. For the environments we consider, the network configuration should be designed to provide maximum support to the applications rather than trying to equally distribute network resources. To do this, we propose developing a middleware that knows the different application requirements and can configure the network to make the best tradeoff between the applications' performance and the network cost.

This type of application environment has been explored within the context of sensor networks, where the sensor nodes collaborate to provide maximum benefit to the application. Protocol architectures for sensor networks make use of this network collaboration to reduce data transfer by aggregating data locally. For example, in the LEACH protocol [12], nodes form local clusters and all data within a cluster is 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 [17], in which nodes collaborate to set up routes as *interests* for particular data are disseminated through the network. These architectures provide good performance for a specific type of application but cannot easily be generalized or react to changes in application desires. MiLAN will extend these ideas to create a general framework for allowing applications to specify their desires and appropriately configuring the network to meet these desires while minimizing cost.

Simplified sensor-level operating systems, such as TinyOS [14], provide general purpose software functionality for sensors, but do not address network management or the higher level application needs. While these approaches explore particular protocols that use collaborating low-level network components, they do not provide a unified approach to allow the application to specify exactly how the cooperation should be achieved.

4.1 Existing Middleware Solutions

To speed development of such collaborative applications, MiLAN formalizes the applications' dynamic goals and desires, and uses this information to configure the network. Most existing middleware systems do not make this tight integration between the network level concerns and the demands of the application. While the LIME middleware [24] is designed for coordination of mobile ad hoc network components, its focus is on the application model presented to the programmer. Specifically, LIME provides a Linda tuple space [9] whose contents change as the connectivity among components changes. While recent extensions to the model limit the scope of tuple space operations based on connectivity, no attempt is made to alter the detected network properties to benefit the application. Similarly, Corba [11] and other distributed object middlewares assume connections are reasonably stable, an assumption that cannot be made in the dynamic mobile environment. The FarGo programming environment [15] somewhat addresses these issues, allowing components to be relocated to respond to changing network conditions such as bandwidth availability or link reliability; but again, FarGo components cannot affect the network properties. By contrast, the applications we consider must not only adapt as the network changes but also affect changes in the underlying network to manage energy consumption and application performance.

4.2 Adaptation to Dynamic Network Components

Adaptation occurs at multiple levels in the system and has been the subject of prior research. First, network protocols, including MAC and routing protocols, adapt to new nodes and node failure. Service discovery protocols sit above the network and below the application, notifying the latter of changes in the former. Finally, applications themselves must be flexible enough to deal with the underlying changes in available services.

From the network perspective, solutions for mobile networking must adapt both their network-layer routing algorithms as well as their MAC protocols to the arrival and departure of nodes. For example, proactive routing algorithms for MANETs, such as Destination-Sequenced Distance-Vector (DSDV) [27], continuously monitor the network for the advertisement of new nodes, which are subsequently able to participate in message exchanges, and for link breakage that requires routes be re-established. These updates are flooded throughout the network to update each node's routing table. On the other hand, reactive MANET routing protocols, such as dynamic source routing (DSR) [18] and ad-hoc on-demand distance vector routing (AODV) [28], only set up routes when they are needed. Once they are established, routes are only changed if a link in the route breaks. Recently researchers have been studying how to update existing routes in order to minimize latency and traffic overhead when a link breaks [25, 30]. However, none of these approaches take into account application-level information beyond acceptable packet latency.

As another example of network node discovery, in Bluetooth, membership in a Personal Area Network (PAN) is continuously changed as new nodes become available and PAN members become inactive (either due to losing connectivity with the master or running out of energy) [3]. Since there are a maximum of 8 nodes allowed in the PAN (1 master and 7 slaves), the master of the PAN must keep track of which nodes are available to ensure that nodes can participate in the PAN and inactive nodes do not block other nodes from using the network. Similarly, the MAC layer must adapt to the available nodes in the network. In Bluetooth, responding to node availability requires that the master create a schedule that takes into account the traffic patterns of the slaves for optimal channel usage. On the other hand, 802.11 networks operating in infrastructureless mode automatically adjust to changing network traffic using a contention window backoff algorithm that does not require explicit notification of new nodes entering the network or nodes leaving the network.

An extension to the discovery of nodes at the network level is the discovery of services at the application level. Systems such as Jini [8] and SOAP [2] allow applications to specify a description of the service they are interested in (either as a Java interface in Jini, or an XML query in SOAP) and a discovery mechanism finds a matching, available provider of that service. Our application must also determine the set of available data sources (services); however, unlike SOAP and Jini, our initial application domain is a wireless environment where the network is already performing a very basic form of node discovery. Our goal is to take advantage of this low-level functionality to achieve the higher level function of service discovery. There is, however a difference between network discovery that exchanges network identification information, and service discovery that exchanges data descriptions. We propose minimal extensions to the network protocols that include higher level description information in discovery. This information may take the form of XML or DAML [5] descriptions, which are passed up to the middleware. Based on this input, the middleware can make the best decisions about the layout of the network including how much bandwidth is allocated to each source and the "role" each node is assigned in the network.

Work on adaptation at the application level comes from within the area of *Context Aware Computing*. The Odyssey platform [26] provides an interface that notifies applications of changes in the underlying network such as higher or lower rate network connections. The Rome [16] and comMotion [23] systems provide users with context-aware messaging services. Messages are created with the intent to be delivered when the recipient is in a certain location. For example, the grocery

list is delivered to a PDA when the user enters the grocery store. CybreMinder [6] takes this a step further, enriching the definition of context to include any facet of context including other people nearby, weather forecast, stock quotes, etc. These guides use some mechanism (typically GPS) to sense changes in context, then use contextual information to decide what information to display to the user. Again, the network is sensed and reacted to by the application, but never altered.

4.3 Application Controlled Network Adaptation

In our case, we propose that not only the network adapt to the presence of new nodes or that the application adapt to the changing network, but rather that the application simultaneously have control over the way the network adapts and force that adaptation as the needs of the application change. To a limited degree, this form of combined application and network integration is present in the On Demand Multicast Routing Protocol (ODMRP) for mobile ad hoc networks [22], which allows the higher layer protocols to define a condition for the inclusion of a link in the multicast tree. For example, an application can request that the multicast tree consist of only high bandwidth links. This coupling is also present in the distributed scheduling algorithm in [19] where information about packet priorities (as determined by an application-specified goal) is used by the 802.11 MAC protocol to control which nodes get access to the channel. This tie between application desires and network functioning has been explored in the design of the SPIN (Sensor Protocol for Information via Negotiation) family of protocols [13]. SPIN negotiates data transfer using application-defined *meta-data*, high-level descriptions of large data such as images and documents, to reduce the amount of data that must be transported in a dissemination network. The work we describe here is a much tighter coupling of the application goals with the underlying network functionality than simply assigning application-specific names to the data.

5 Summary

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 applications that rely on multiple and varying low-level data sources, it is not a viable approach to manage the network completely independent from the needs of the application. The core of the research presented in this report is the integration between the needs of the application and the management of the network into a single, unified middleware system called MiLAN. Through this tight coupling, MiLAN can tradeoff between application performance and network cost, while still retaining the separation between the policy specifying how to react to a dynamic environment and the mechanisms to implement the policy.

References

- [1] J. F. Allen, G. Ferguson, and A. Stent. An architecture for More Realistic Conversational Systems. In *Proceedings of Intelligent User Interfaces 2001 (IUI-01)*, Jan. 2001.
- [2] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. Technical report, W3C, May 2000.
- [3] J. Bray and C. Sturman. *Bluetooth 1.1: Connect without Cables*. Prentice Hall, 2001.
- [4] 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 *Pro-*

- ceedings of the IEEE International Symposium on Bioinformatics and Bioengineering (BIBE)*, pages 236–242, 2000.
- [5] DAML, the DARPA Agent Markup Language. <http://www.daml.org>, 2000.
 - [6] A. Dey and G. Abowd. CybreMinder: A Context-Aware System for Supporting Reminders. In *Proceedings of the Second International Symposium on Handheld and Ubiquitous Computing*, pages 172–186, September 2000.
 - [7] 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.
 - [8] K. Edwards. *Core JINI*. Prentice Hall, 1999.
 - [9] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.
 - [10] Isaac Green and Randal C. Nelson. Tracking Objects using Recognition. Technical Report 765, University of Rochester, Computer Science Department, February 2002.
 - [11] Object Management Group. *The Common Object Request Broker: Architecture and Specification Revision 2.2*. 492 Old Connecticut Path, Framingham, MA 01701, USA, 1998.
 - [12] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. *IEEE Transactions on Wireless Communication*, 2002 (to appear).
 - [13] 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.
 - [14] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System Architecture Directions for Network Sensors. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 93–104, Cambridge, MA, USA, November 2000.
 - [15] 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 Computing*, pages 403–411, 1999.
 - [16] A. Huang, B. Ling, S. Ponnkanti, and A. Fox. Pervasive Computing: What is it good for? In *Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 84–91, August 1999.
 - [17] 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.
 - [18] D.B. Johnson, D.A. Maltz, Y.-C. Hu, and J.G. Jetcheva. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR). Internet Draft, February 2002. IETF Mobile Ad Hoc Networking Working Group.
 - [19] V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, and E. Knightly. Distributed Multi-hop Scheduling and Medium Access with Delay and Throughput Constraints. In *Proceedings of ACM Mobicom '01*, Jul. 2001.

- [20] L. Khachiyan. A Polynomial Algorithm in Linear Programming (in Russian). In *Doklady Akademii Nauk SSSR 244*, 1979.
- [21] K. Lee. IEEE 1451: A Standard in Support of Smart Transducer Networking. In *IEEE Instruments and Measurements Technology Conference*, May 2000. <http://iee1451.nist.gov/IEEE1451.pdf>.
- [22] S.-J. Lee, W. Su, and M. Gerla. On-Demand Multicast Routing Protocol in Multihop Wireless Mobile Networks. *ACM/Baltzer Mobile Networks and Applications, special issue on Multipoint Communication in Wireless Mobile Networks*, 2000.
- [23] N. Marmasse and C. Schmandt. Location-Aware Information Delivery with comMotion. In *Proceedings of the Second International Symposium on Handheld and Ubiquitous Computing*, pages 157–171, September 2000.
- [24] 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.
- [25] A. Nasipuri, R. Castaneda, and S. Das. Performance of Multipath Routing for On-Demand Protocols in Mobile Ad Hoc Networks. *ACM/Baltzer Mobile Networks and Applications (MONET)*, 6:339–349, 2001.
- [26] B. D. Noble and M. Satyanarayanan. Experience with Adaptive Mobile Applications in Odyssey. *Mobile Networks and Applications*, 4(4):245–254, 1999.
- [27] C. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *Proceedings of the SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, August 1994.
- [28] C. Perkins and E. Royer. Ad-Hoc On-Demand Distance Vector (AODV) Routing. In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99)*, pages 90–100, 1999.
- [29] S. Singh, M. Woo, and C.S. Raghavendra. Power-Aware Routing in Mobile Ad Hoc Networks. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '98)*, October 1998.
- [30] S.-L. Wu, S.-Y. Ni, Y.-C. Tseng, and J.-P. Sheu. Route Maintenance in a Wireless Mobile Ad Hoc Network. In *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS '00)*, 2000.