Report Number: WUCSE-2003-57

2003-08-04

A Protocol for Supporting Context Provision in Wireless Mobile Ad Hoc Networks

Christine Julien and Gruia-Catalin Roman

The increasing ubiquity of mobile computing devices has made ad hoc networks everyday occurrences. In these highly dynamic environments, the multitude of devices provides a varied and rapidly changing environment in which applications must learn to operate. Successful end-user applications will not only learn to function in this environment but will take advantage of the variety of information available. Protocols for gathering an application's contextual information must be built into the network to function in a timely and adaptive fashion. This paper presents a protocol for providing context information to such applications. We present an implementation and show how... **Read complete abstract on page 2.**

A Protocol for Supporting Context Provision in Wireless Mobile Ad Hoc Networks

Christine Julien and Gruia-Catalin Roman

Complete Abstract:

The increasing ubiquity of mobile computing devices has made ad hoc networks everyday occurrences. In these highly dynamic environments, the multitude of devices provides a varied and rapidly changing environment in which applications must learn to operate. Successful end-user applications will not only learn to function in this environment but will take advantage of the variety of information available. Protocols for gathering an application's contextual information must be built into the network to function in a timely and adaptive fashion. This paper presents a protocol for providing context information to such applications. We present an implementation and show how it provides context information to mobile applications in an on-demand manner. We also provide a simulation analysis of the tradeoffs between consistency and range of context definitions in highly dynamic ad hoc networks.

Department of Computer Science & Engineering

Washington University in St. Louis
School of Engineering & Applied Science

2003-57

A Protocol for Supporting Context Provision in Wireless Mobile Ad Hoc Networks

Authors: Julien, Christine; Roman, Gruia-Catalin

August 4, 2003

Abstract: The increasing ubiquity of mobile computing devices has made ad hoc networks everyday occurrences. In these highly dynamic environments, the multitude of devices provides a varied and rapidly changing e environment in which applications must learn to operate. Successful end-user applications will not only learn to function in this environment but will take advantage of the variety of information available. Protocols for gathering an application's contextual information must be built into the network to function in a timely and adaptive fashion. This paper presents a protocol for providing context information to such applications. We present an implementation and show how it provides context information in an on-demand manner to mobile applications. We also provide an analysis of the protocol in highly dynamic ad hoc networks.

A Protocol for Supporting Context Provision in Wireless Mobile Ad Hoc Networks

Abstract

The increasing ubiquity of mobile computing devices has made ad hoc networks everyday occurrences. In these highly dynamic environments, the multitude of devices provides a varied and rapidly changing environment in which applications must learn to operate. Successful end-user applications will not only learn to function in this environment but will take advantage of the variety of information available. Protocols for gathering an application's contextual information must be built into the network to function in a timely and adaptive fashion. This paper presents a protocol for providing context information to such applications. We present an implementation and show how it provides context information to mobile applications in an on-demand manner. We also provide a simulation analysis of the tradeoffs between consistency and range of context definitions in highly dynamic ad hoc networks.

1 Introduction

A mobile ad hoc network is an opportunistically formed structure that changes rapidly in response to the movement of the hosts forming the network. To communicate, nodes in such a network commonly use ad hoc routing protocols (e.g., DSDV [14], DSR [3], AODV [15]) that deliver messages between a known source and destination using intermediate nodes as routers. Ad hoc multicast routing protocols require nodes to register as receivers for a specific multicast address. The network maintains a multicast tree [6, 8] or mesh [2, 12] for delivering messages to registered receivers.

Directly applying these routing techniques to gathering context information poses several drawbacks. In both unicast and multicast routing, the paths along which messages are delivered may extend across the entire ad hoc network. As the ubiquity of mobile devices increases, ad hoc networks may grow very large. Consider an ad hoc network composed of cars on a highway. Cars may be transitively connected for hundreds of miles, but it is generally not necessary or desirable to communicate at great distances. Many applications require only local interactions, e.g., gathering traffic information. In addition, for traditional routing protocols to function, senders and receivers require explicit knowledge of each other. Often, however, an application has no a priori knowledge about the hosts with which it will want to interact, since hosts in ad hoc networks move at will, and hosts that are encountered once may never be encountered again. Supporting context-aware applications in this unpredictable environment requires reevaluating what applications need from underlying protocols and providing solutions tailored to these needs.

Emerging applications for this environment focus on providing context information to the user. This context can be defined by physical properties of the host or surrounding hosts and by information available on them. For example, a context-aware tour guide [1, 5] may interact with nearby kiosks to display locally relevant tourist information. Cars on a highway may interact to gather traffic information about their intended routes. In any of these cases, devices cooperate to gather the information presented to the user. This information defines the operating context of the application, which differs for each application. The scope of interaction is driven by the instantaneous needs of applications, which change over time.

We focus on supporting an application's ability to specify what context information it needs from its environment and gathering that information in a manner that adapts to environmental changes. Because the network is constantly being reshaped, an application's requests must be evaluated in a timely fashion to ensure the freshness of the information. Previous work resulted in the Content-Based Multicast model (CBM) [21], which focuses on disseminating information collected by sensors. In general, this model is

tailored for distributing information about a (possibly mobile) threat to interested parties. The dissemination pattern in CBM is based on the relative movement patterns of the threat being sensed and the interested parties. Mobile nodes that sense the presence of a threat push information about the threat in the direction of its movement. At the same time, mobile components pull information about threats present in their direction of travel. This combination of both push and pull actions allows this multicast protocol to adjust to dynamic components with varying speeds.

While the CBM model addresses needs of context aware applications, it is tailored to a specific class of context-aware applications. It is a protocol tailored to dissemination of mobile threats to mobile parties. Our approach focuses on a more general treatment of context that caters to the varying and unpredictable needs of applications in heterogeneous mobile ad hoc networks. While traditional approaches to context-aware computing either deal with specific types of context (like CBM) or only context that can be sensed by the local host, we extend the notion of context to include information available in a region of the ad hoc network surrounding the host of interest. The protocol constructs and dynamically maintains a tree over a subnet of neighboring hosts and links whose attributes contribute to an application's specific definition of context. Here we present the first implementation of the *Network Abstractions* model presented in [17]. In this paper, we explore the protocol in detail, focusing on its practicality, implementation, and performance in an effort to quantify the guarantees that can be associated with extended contexts in ad hoc networks.

The remainder of this paper is organized as follows. Section 2 provides an overview of the Network Abstractions model and protocol. Section 3 discusses our implementation. Section 4 provides an analysis of the model through simulation. Discussions and conclusions appear in Sections 5 and 6 respectively.

2 Network Abstractions Overview

Ad hoc mobile networks contain many hosts and links with varying properties which define the context for an individual host in the network. An adaptive application's behavior depends on this continuously changing context. Because this definition of context includes information from across the network, it allows more flexible interactions between mobile applications and their environments. This approach, however, has the potential to greatly increase the amount of context information available, and so an application on a host must precisely specify its context based on the properties of hosts and links in the network. For example, an ad hoc network on a highway might extend for hundreds of miles, but a driver may be interested only in gas stations within five miles. The Network Abstractions approach discussed next allows a context specification to remain as general and flexible as possible while ensuring the feasibility of the protocol to dynamically compute the context. The model provides an application on a particular host, called the reference, the ability to specify a context that spans a subset of the network.

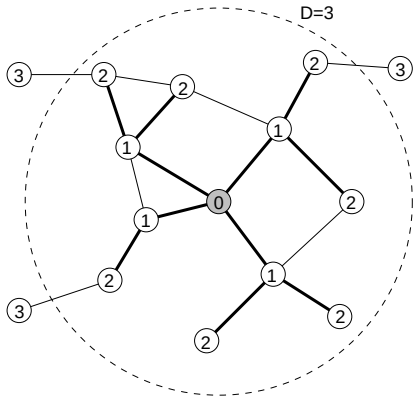


Figure 1. A Network Abstraction defined to include all hosts within three hops of the reference (shown in gray)

2.1 Model Overview

As discussed previously, an application in an ad hoc mobile network ideally operates only over a context tailored to its specific needs. The Network Abstractions model views this context as a subnet surrounding the application of interest. Consider the example network shown in Figure 1. In this network, the reference host where the application is running is shown in gray. The links shown are available communication links. This figure depicts the application's definition of a context that includes all hosts within fewer than three hops. The number inside each node is its shortest distance from the reference in terms of number of hops. The dashed line labeled "D=3" represents the application's bound on the context (three hops), while the darkened links indicate paths in a tree that define the context. By defining such a context, the application has restricted its operation to a subnet of the ad hoc network that is locally relevant to its desired functionality.

This example uses a simple definition of "distance"

(number of hops), but this approach can be generalized to include distance definitions tailored to unique applications. We will provide examples of more sophisticated distance metrics later in this section. In general, after providing its application-specific definition of distance and the maximum allowable distance, the reference host would like a list of hosts such that:

> *Given a host α and a positive D, find the set of all hosts Q_α such that all hosts in Q_α are reachable from α, and for all hosts β in Q_α, the cost of the shortest path from α to β is less than D.*

In the Network Abstractions model, the application specifies its distance metric with two components. The first defines the weight of a link in the network. In general, the weight on a link, w_{ij}, is a combination of properties of the link (e.g., latency, bandwidth, or physical distance) and properties of the two hosts (i and j) it connects (e.g., available power, location, or direction). The second component is a cost function evaluated over a series of weights. In the hop count example, the weight of all links is defined to be one, while the cost function simply adds the weights of links along the path.

The cost function determines the cost of a particular path in the network, defined by the series of nodes traversed by the path. Cost functions are defined recursively; this allows them to be computed in a distributed fashion. A path from reference host 0 to host k is represented as P_k. The cost function is defined as:

$$f_0(P_k) = Cost(f_0(P_{k-1}), w_{k-1,k})$$

where *Cost* indicates the application-specified function evaluated over the cost at the previous hop and the weight of the most recent link. To be able to successfully bound a context, we must require that the cost function strictly increases with the number of hops from the reference host. The reason will become more evident in the upcoming examples. Recursive evaluation of this cost function over a network path determines its cost. In a real network, multiple paths may exist between two nodes. Therefore, as shown by the darkened links in Figure 1, we build a tree rooted at the reference node that includes only the lowest cost path to each node in the network. Extensions to the Network Abstractions protocol [10] provide a mesh based abstraction for delivering queries and replies; throughout the presentation, we will focus on the tree-based abstraction.

To take maximum advantage of the abstraction provided by the cost function and its associated properties, an application bounds the maximum allowable cost. Nodes to which the cost is less than the bound are included in the context. This allows an application to restrict its operating context to a portion of the ad hoc network that exactly satisfies the application's needs. Nodes that lie outside the context's bound are never touched by the context's computation.

2.2 Example Metrics

Next we examine some example distance metrics. First we provide a metric that uses a sophisticated weight definition, then show a more complicated cost function.

2.2.1 Network Latency

Imagine field researchers studying the behavioral patterns of a group of animals. Each researcher is assigned a particular animal or animals to monitor. The researchers might carry wireless PDAs with attached cameras that automatically record their observations. If one researcher's subject moves behind a boulder, and the researcher can no longer see it from his location, he may want to use another's camera feed to observe the target. The context in this case will be bounded by network latency—only cameras within a certain end-to-end latency can provide a camera feed with a high enough frame rate to be useful. A link's weight is defined as:

$$w_{ij} = \frac{node\ latency_i}{2} + \frac{node\ latency_j}{2} + link\ latency_{ij}$$

where the first two components define the average time between when the node receives a packet and when it propagates the packet. We use only half of this number; otherwise we would count the node's latency twice if the node is in the middle of the path. This latency value will suffice under the assumption that a node's incoming latency is approximately equivalent to its outgoing latency. The third component of w_{ij} is the time required for a message to travel between two nodes.

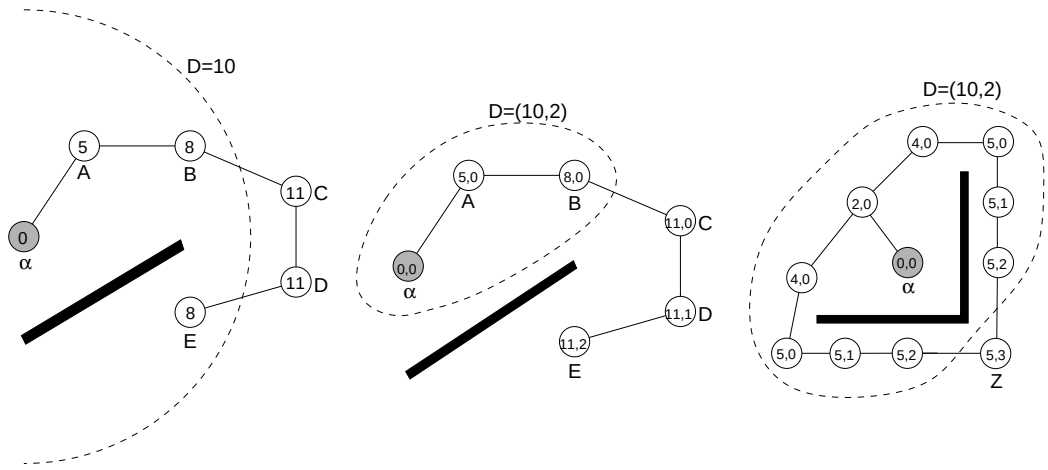
The application also provides a cost function; a simple one to use with this weight definition is the same as in the hop count example:

$$f_0(P_k) = f_0(P_{k-1}) + w_{k-1,k}$$

where the cost of the path from node 0 (the reference) to node k along path P_k is the sum of the cost to the previous node plus the weight of the new link. A bound on this cost function is defined by a bound on the total allowed latency.

2.2.2 Physical Distance

Next we present a general-purpose metric based on physical distance. Cars traveling on a highway collect information about weather conditions, highway exits,



(a) **(b)** **(c)**

$$f_0(P_k) = \begin{cases} (|f_0(P_{k-1}).\mathbf{V} + w_{k-1,k}|, f_0(P_{k-1}).C, f_0(P_{k-1}).\mathbf{V} + w_{k-1,k}) & \text{if } |\mathrm{f_0(P_{k-1}).\mathbf{V} + w_{k-1,k}}| > \mathrm{f_0(P_{k-1}).}\mathit{maxD} \\ (f_0(P_{k-1}).\mathit{maxD}, f_0(P_{k-1}).C + 1, f_0(P_{k-1}).\mathbf{V} + w_{k-1,k}) & \text{otherwise} \end{cases}$$

(d)

Figure 2. (a) Physical distance only; (b) Physical distance with hop count, restricted due to distance; (c) Physical distance with hop count, restricted due to hop count; (d) The correct cost function

accidents, traffic patterns, etc. As a car moves, it wants to operate over the information that will affect its immediate route, so the data might be restricted to information within a certain physical distance (e.g., within a mile). The context requirements for each car's tasks are likely to be different. The Network Abstractions model allows each application to tailor its context definition to its needs by defining a weight for each network link.

If the calculated context is based on the physical distance between the reference host and other hosts, a link's weight reflects the distance vector between two connected nodes, accounting for both the displacement and the direction of displacement between the two nodes: $w_{ij} = \vec{\mathrm{IJ}}$.

Figure 2(a) shows an example network where specifying distance alone causes the context to not be easily bounded. This results from the fact that a cost function based on distance alone is not strictly increasing as the number of hops from the reference host grows. To overcome this problem, the cost function should be based on both the distance vector and a hop count. The cost function's value at a given node consists of three values: (maxD, C, $\mathbf{V}$). The first value, maxD, stores the maximum distance seen on this path. This may or may not be the magnitude of the distance vector from the reference to this host. The second value, C, keeps the number of consecutive hops for which maxD did not increase. The final value, $\mathbf{V}$, is the distance vector from the reference host to this host.

Specifying a bound for this cost function requires bounding both maxD and C. A host is in the context only if both its maxD and C are less than the bound's values. Neither the value of maxD nor the value of C can ever decrease, and, if one value remains constant for any hop, the other is guaranteed to increase.

Figure 2(d) shows the cost function. In the first case, the new magnitude of the vector from the reference host to this host is larger than the current value of maxD; maxD is reset to the magnitude of the vector from the reference to this host, C remains the same, and the distance vector to this host is stored. In the second case, maxD is the same for this node as the previous node; maxD remains the same, C is incremented by

one, and the distance vector to this host is stored.

Figure 2(b) shows the same nodes as Figure 2(a) using this new cost function. The application specified bound shown in Figure 2(b) is $D = (10, 2)$ where 10 is the bound on *maxD* and 2 is the bound on C. This cost function can be correctly bounded, and no hosts that should qualify are missed. Figure 2(c) shows the same cost function applied to a different network. In this case, while the paths never left the area within distance 10, node Z still falls outside the context because the maximum distance remained the same for more than two hops.

2.3 Protocol Overview

An application desires the guarantee that any message it sends will be received only by hosts within the context and that it is received by all hosts within the context. Our protocol dynamically builds a tree over exactly the nodes in the context based on an application's specification, defining a single route from the reference host to all other hosts in the context. In this section, we provide an overview of the protocol in preparation for a discussion of its implementation and analysis. The details of the protocol were presented in [17].

The protocol provides two distinct functions. First it disseminates *one-time* queries over the context. This type of query serves applications that interact with the data in their contexts in a polling manner. Such queries may require replies from context members, but the context is built on the fly, and the structure is not maintained. This lack of maintenance benefits periodic activity because it removes the overhead that maintaining contexts introduces. In some cases, however, applications require constant interaction with the data in their environments. For this reason, the protocol also allows applications to register *persistent* queries on their contexts. These persistent queries require the context to be maintained as the hosts defining it move. Due to the maintenance cost involved, ideal interactions would extend one-time queries to larger contexts (e.g., traffic conditions for the next five miles), but only maintain smaller contexts (e.g., cars within potential collision range of my car).

2.3.1 Assumptions

The protocol relies on a few assumptions regarding the behavior of the underlying system. First, it assumes a message passing mechanism that guarantees reliable one-hop delivery with associated acknowledgments. These acknowledgments lie outside the concern of this protocol. The protocol also assumes that disconnection is detectable, i.e., when a link disappears, both hosts that were connected by the link can detect the disconnection. Finally, the protocol assumes that the underlying system maintains the weights on links in the network by responding to changes in the contextual information required by applications.

2.3.2 The Query Component

The protocol is on-demand in that a tree is built only when a data query is sent. Piggy-backed on this data message are the context specification and the information necessary for its computation. Specifically, the query contains the context's definition of link weight, the cost function, and the bound. The protocol uses this information to determine which hosts belong to the context and should receive this message.

2.3.3 Tree Building

Because any information required for computing another host's context arrives in a query, hosts need not keep information about the system's global state. An application with a data query to send bundles the context specification with the query. It then determines which of its neighbors are within the context and sends them the query. Due to the wireless nature of the network, this can be accomplished via one message transmission broadcast to any neighbor determined to be in the context. Neighbors in the context determine which (if any) of their neighbors are also in the context and rebroadcast the message. In the course of query propagation, every context member remembers the previous hop in its shortest path back to the reference host. A node only rebroadcasts a duplicate message if its cost has decreased since this may cause inclusion of additional nodes in the context. When the query reaches the bound, it will not be forwarded on; the query distribution stabilizes when every node in the context knows its shortest path to the reference host. Again the context is guaranteed to be boundable because the costs of paths strictly increase as the number of hops from the reference host grows.

2.3.4 Tree Maintenance

Some applications, for example one monitoring nearby cars for possible collisions, require constant monitoring of data in the context. The Network Abstractions model accomplishes this through the use of persistent queries. Contexts over which an application issues persistent queries require maintenance. The protocol for maintaining the context builds on the one-time query

protocol above. At a general level, this protocol is similar to an on-demand distance vector routing protocol like AODV [15]. The changes to the distance vector protocol are crucial to being able to maintain context-aware interactions in highly dynamic ad hoc networks. First, the link weights used to determine path costs are defined by arbitrary properties of the environment. Each application chooses which properties to use according to its specific needs. Second, to help applications interact only with a portion of the network that should affect their behavior, we require them to bound the context over which they operate. This prevents maintaining lengthy paths in the network that, because of the dynamic nature of the links, are almost guaranteed to break. To achieve context maintenance, hosts within the context must react to changes that affect their cost. The new cost may push the node (or other downstream nodes) out of the context or pull them in. Because all needed information is stored within the hosts in the context, the reference host need not participate in this maintenance; instead it is a local adjustment to a local change. As a practical concern, all distance vector protocols must account for the "count-to-infinity" problem, where, upon loss of a link, two nodes both believe their route back to the reference node is through each other. Because the Network Abstractions model bounds the contexts, it can assume path lengths of reasonable sizes and adjust for this problem by maintaining the entire path information.

2.4 Practical Research Issues

Next we present the first implementation of the protocol described above. This protocol allows us to explore the range of distance metrics and cost functions applications can define. It also allows the development of an extensive software system that eases applications' interactions with their contexts in the ad hoc mobile environment. We also provide a simulation of the protocol over a simple metric (the hop count example discussed above) used to examine the feasibility of the consistency assumptions we make and to study the guarantees we can associate with the protocol in a variety of networks. Specifically, we test the limits of the network changes our protocol can handle and measure the correctness of the context building mechanisms.

3 Implementation

Our implementation is written in Java. This decision is driven by the fact that we aim to ease application development, which means placing control over the context in the hands of novice programmers. It is imperative that we provide a flexible protocol that an application programmer can tailor to his needs. Applications must be able to define individualized distance metrics and add new environmental monitors to the system to increase the flexibility of link weight definition.

The implementation allows issuance of both one-time and persistent queries and maintains contexts which have persistent queries. Our system includes built-in metrics (e.g., hop count) but also provide a general framework for defining new metrics. Our implementation uses the support of two additional packages; one for neighbor discovery and one for environmental monitoring. We first describe these two packages before detailing the protocol implementation.

3.1 Support Packages

3.1.1 Neighbor Discovery

In ad hoc networks, no wired infrastructure with dedicated routing nodes exists. Instead, all hosts serve as routers. To distribute messages, a host must maintain knowledge of its current set of neighbors, and, as movement causes this set to change, the host must be notified. A node in the Network Abstractions protocol receives knowledge of its neighbors from a discovery service. This service uses a periodic beaconing mechanism that can be parameterized with policies for neighbor addition and removal (e.g., a neighbor is only added when its beacon has been heard for two consecutive beacon periods, and a neighbor is removed when it has not been heard from for 10 seconds).

3.1.2 Environmental Monitoring

Essential to adapting to context information is the ability to sense environmental changes. The Context Toolkit [7] uses context widgets to abstract context sensing and provide context information to applications. It allows applications to gather context information from both local and remote sensors about which the application has a priori knowledge. The ad hoc network requires a more lightweight mechanism in which both local and neighboring environmental sensors are accessed in a context-sensitive manner. This sensor information is used to calculate the link weights needed in the Network Abstractions protocol.

The monitor service we provide maintains a registry of monitors available on the local host and neighboring hosts (within one hop). The former are referred to as *local monitors* and the latter as *remote monitors*. An application tailors the monitor package to

its needed capabilities. As an example, to add a location monitor, the application provides code that interacts with, for instance, a GPS device. In general, a monitor contains it's current value in a variable (e.g., the value of a GPS monitor might be represented by a variable of type `Location`) and contains methods that allow applications to access the value (through the `queryValue()` method) or react to changes in the value (through the `MonitorListener` interface). This functionality is contained in an abstract base class called `AbstractMonitor`. When a programmer extends the monitoring package to add a new monitor, he must extend the `AbstractMonitor` class. This extending class is responsible for ensuring that the class's `value` variable is kept consistent with the current state of the environment. Changes to this variable should be performed through the `setValue()` method in the base class to ensure that any listeners registered for changes to the variable are notified. The programmer should also add his defined monitors to the monitor registry at run-time.

Figure 3 shows the code that a programmer must write to extend the monitor package by showing the code for a class that extends `AbstractMonitor` to collect information from a GPS device. From the perspective of our package, the important pieces are how the extending class interacts with the base class. The details of communicating with a particular GPS device are omitted; their complexity is directly dependent on the individual device and its programming interface.

```
public class GPSMonitor extends AbstractMonitor{
  public GPSMonitor(...){
    //call the AbstractMonitor constructor
    super();
    //set up serial connection to GPS receiver
    ...
  }
  public void serialEvent(SerialPortEvent event){
    //handle periodic events from GPS receiver
    ...
    //turn GPS event into a GPSLocation object
    ...
    //set local value variable, notify listeners
    setValue(gpsLocation);
  }
}
```

Figure 3. The GPSMonitor Class

To monitor context information on neighboring hosts, the monitor registry creates instances of the class `RemoteMonitor` that connect to concrete monitors on the remote host. These `RemoteMonitors` serve as proxies to the actual monitors; when the values change on the monitor on the remote host, the `RemoteMonitor`'s value is also updated. To gain access to `RemoteMonitors`, the application provides the id of the host (which can be retrieved from the discovery package) and the name of the monitor (e.g., "GPSMonitor"). The monitor registry creates the proxy on the local host, connects it to the remote monitor, and returns a handle to the proxy to the application. The application interacts with the remote monitor in the same manner as with a local monitor (e.g., by calling the `queryValue()` method or registering listeners for changes in the value).

3.2 Protocol Implementation

We describe the implementation of the protocol in two phases. First, we discuss what an application programmer must do to use this implementation of the Network Abstractions protocol, both in terms of the classes the programmer must define and the interface to the protocol that allows sending queries. We then move on to describe the underpinnings of the protocol that are transparent to the application.

Before defining a context, an application must create the components that build a distance metric. This includes two pieces: a `Cost` that defines the components of the costs of paths and a `Metric` that provides that algorithm for computing these costs from a previous hop cost and a link weight.

The `Cost` interface is simple; it requires the extending class to implement a single method that allows two instances of the derived `Cost` to be compared. This interface is shown in Figure 4.

int compareTo(Cost cost)
— compares two instances of the cost and returns whether the passed cost is equal to, greater than, or less than the stored cost.

Figure 4. The `Cost` interface

An extending class must first define any instance variables needed to store the state of the particular cost. It must then provide a definition for the `compareTo()` method. It can provide any other methods that its corresponding `Metric` class may require, which is likely to include access methods for the instance variables. As an example, consider the `Cost` class a programmer must define if he wants to build the distance-based metric described in the previous section. The code for this `Cost` class is shown in Figure 5.

```
public class DistanceCost implements Cost{
  private double maxD;
  private int c;
  private DistanceVector v;
  public DistanceCost(double in_d, int in_c,
                      DistanceVector in_v){
    //initialize the variables
    maxD = in_d; c = in_c; v = in_v;
  }
  public int compareTo(Cost cost){
    //compare each variable
    ...
  }
  public double getD(){ return maxD; }
  public int getC(){ return c; }
  public DistanceVector getV(){ return v; }
}
```

Figure 5. The `DistanceCost` Class

The `Metric` base class is more complicated than the `Cost` because it defines how the costs are generated along paths in the context. The API for this base class is shown in Figure 6. For an extending class, the tricky

private String[] monitorNames
— this instance variable holds the names of the monitors whose values affect this metric. This information is used when a context is maintained to ensure the weight values are correct. An extending class should take care to ensure this variable is initialized.

public void setMonitorNames(String[] names)
— allows the extending class to set the names of the monitors that affect this metric.

public abstract Cost wFunction(HostID otherHost)
— the implementation of this method should use information about the local host (gathered through the monitor registry if necessary) and information about the remote host (identified by the host id) to calculate the weight on the link between the hosts.

public abstract Cost costFunction(Cost currentD, Cost weight)
— the implementation of this method should take the cost at the previous hop and the cost on the subsequent link and calculate the new cost.

Figure 6. The `Cost` interface

parts to adhering to these requirements include correctly implementing the logic of the cost function and precisely identifying the monitors whose values are important. To continue the distance-based example, Figure 7 shows the code the programmer must define to create this metric.

```
public class DistanceMetric extends Metric{
  public DistanceMetric(){
    String[] monitors = {``GPSMonitor''}
    setMonitorNames(monitors);
  }
  public Cost wFunction(HostID otherHost){
    //calculate the weight on the link
    (the DistanceVector from this host to otherHost)
    DistanceVector vec = ...
    //store it in a DistanceCost object
    DistanceCost weight = ...
    return weight;
  }
  public Cost costFunction(Cost currentD,
                           Cost weight){
    //implement the function from Figure 2(d)
    DistanceCost newCost = ...
    ...
    return newCost;
  }
}
```

Figure 7. The `DistanceMetric` Class

The programmer extending the metric class does not have to worry about how these methods are called; the Network Abstractions protocol, when invoked, will take the `Cost` and `Metric` that define a context and call the necessary methods as appropriate.

To define a context using the Network Abstractions protocol, an application programmer creates a `Cost` and `Metric` as discussed above and passes them to the Network Abstractions protocol. The basic interface the protocol presents to the application is detailed in Figure 8. The first method, `createContext()` allows the application to notify the Network Abstractions protocol of its intention to operate over a context defined by the provided `Metric` and bound (of type `Cost`). Once this context is defined, the application can use it to send and register queries. As will be discussed next, the protocol only maintains contexts that have persistent queries registered.

When an application sends a one time query over a defined context (via the `sendQuery()` method), the protocol layer uses information provided by the neighbor discovery and environmental monitoring services to determine which neighbors must receive the message, if any. If neighbors exist that are within the context's bound, the local host packages the application's data with the context information and broadcasts the entire packet to its qualifying neighbors.

Upon receiving a one-time context query, the receiving host stores the previous hop, and repeats the prop-

public NetAbsID createContext(Metric m, Cost b)
— initializes a context according to the provided metric and bound. the bound defines the maximum allowed cost that belongs to the context. this method returns a handle to the application that it can use to access the context.

public void sendQuery(NetAbsID id, Query q)
— this method sends the provided query to all members of the context identified by id.

public Ticket registerQuery(NetAbsID id, Query q)
— this method registers the provided query on all members of the context identified by id. the method returns a ticket to the application that it can use to deregister the query.

public void deregisterQuery(Ticket t)
– removes the persistent query identified by the provided ticket.

Figure 8. The Network Abstractions interface

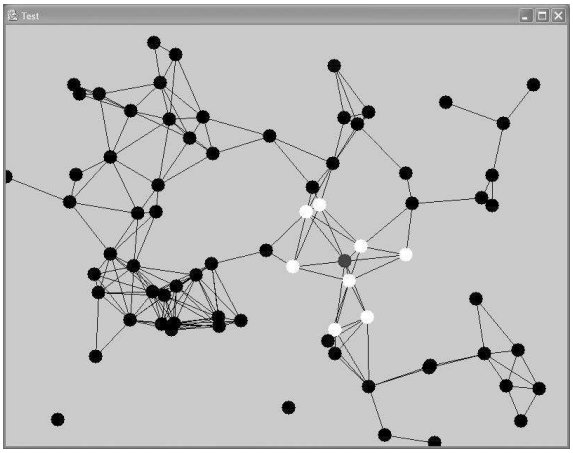


Figure 9. Screen capture of demonstration system

agation step, forwarding the packet to any of its neighbors within the bound. It also passes the packet's data portion to application level listeners registered to receive it. If this same query (identified by a sequence number) is received from another source, the new information is remembered and propagated only if the cost of the new path is less than the previous cost.

An application can also reply to a data packet. The protocol uses the stored previous hop information to route the reply back to the reference host. Because this reply is asynchronous and the context for a one-time query is not maintained, it is possible that the route no longer exists. In these cases, the reply is dropped. To provide a stronger guarantee on a reply's return, an application should use a persistent query which forces the protocol to maintain the context.

The structure of a persistent query differs slightly from a one-time query in that it must include the entire path. This information is used to overcome the count-to-infinity problem encountered as the links in the network change. The distribution of the query is the same as above, but the actions taken upon query reception vary slightly. The receiving host must remember the entire path back to the reference host. When the same query arrives on multiple paths, the host remembers every qualifying path. If the currently used path breaks, the protocol can replace it with a viable path. To keep both the current path and the list of possible paths consistent, the protocol monitors the aspects of the context that contribute to distance definition (through the monitor package); if these values change, the cost at this host or its neighbors could also change. The protocol reacts to such changes and updates its cost information locally. It also propagates these changes to affected neighbors. Therefore local changes to the metric do not affect the entire context, only from the point of change out to the bound. Before replacing a path, the protocol checks that the path is loop-free.

Replies to persistent queries propagate back towards the reference host along the paths maintained by the protocol. A reply is not guaranteed to reach the reference. Our practical experience shows, however, that, in reasonably sized networks with a fair amount of mobility, the delivery assumption is likely to hold. Section 4 provides an empirical evaluation of this assumption.

3.3 Demonstration System

Figure 9 shows a screen capture of our demonstration system. In this example, each circle depicts a single host running an instance of the protocol. The demonstration system uses the network for communication, which allows this system to display information gathered from actual mobile hosts. This figure shows a single context defined by a host (the gray host in the center of the white hosts). This context is simple; it includes all hosts within one hop. When a host moves within the context's bound, it receives a query registered on the context that causes the node to turn its displayed circle white. When the node moves out of the context, it turns itself black. The demonstration system provides simulations using a variety of mobility models, including a markov model, a random waypoint model [4], and a highway model. This system is partic-

ularly useful because it allows us to visually evaluate what kinds of contexts match what styles of mobility. This gives us some intuition into what our goals should be before we start extensive simulation or implementation of a complex application.

3.4 Example Usage

The protocol implementation described here is currently in use to support the ongoing implementation of a middleware model for ad hoc mobile computing. In this system, called EgoSpaces, application agents operate over projections (*views*) of the data available in the world. EgoSpaces addresses the specific needs of individual application agents, allowing them to define what data is to be included in a view by constraining properties of the data items, the agents that own the data, the hosts on which those agents are running, and attributes of the ad hoc network. This protocol provides the latter in a flexible manner, and EgoSpaces uses the Network Abstractions protocol to deliver all communication among agents.

4 Analysis and Experimental Results

To examine the definitions of contexts on real mobile ad hoc networks, we used the ns-2 network simulator, version 2.26. This section provides simulation results for context dissemination. These simulations are a first step in analyzing the practicality of the protocol we have implemented. Not only do they serve to show that it is beneficial to define contexts in the manner described in ad hoc networks, the measurements also provide information to application programmers about what types or sizes of contexts should be used under given mobility conditions or to achieve required guarantees. All of the simulations we describe in this section implement a context defined by the number of hops from the reference node. Because this is the simplest type of context to define using the Network Abstractions protocol, this provides a baseline against which we can compare simulations of more complex or computationally difficult definitions. Before providing the experimental results, we detail the simulation settings we used.

4.1 Simulation Settings

We generated random 100 node ad hoc networks that use the random waypoint mobility model. The simulation is restricted to a 1000x1000m^2 space. We vary the network density (measured in average number of neighbors) by varying the transmission range. We measured the average number of neighbors over our simulation runs for each transmission range we used; these averages are shown in Figure 10. While the random waypoint mobility model suffers from "density waves" as described in [18], it does not adversely affect our simulations. An average of 1.09 neighbors (i.e., $50m$ transmission range) represents an almost disconnected network, while an average of 23.89 neighbors (i.e. $250m$ transmission range) is extremely dense. While the optimal number of neighbors for a static ad hoc network was shown to be the "magic number" six [11], more recent work [18] shows that the optimal number of neighbors in mobile ad hoc networks varies with the degree of mobility and mobility model. The extreme densities in our simulations lie well above the optimum for our mobility degrees.

In our simulations, we used the MAC 802.11 standard [9] implementation built in to ns-2. Our protocol sends only broadcast packets, for which MAC 802.11 uses Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) [1]. This broadcast mechanism is not reliable, and we will measure our protocol's reliability over this broadcast scheme in our simulations. We implemented a simple "routing protocol" on top of the MAC layer that, when it receives a packet to send simply broadcasts it once but does not repeat it.

We also tested our protocol over a variety of mobility scenarios using the random waypoint mobility model with a $0s$ pause time. In the least dynamic scenarios, we use a fixed speed of $1m/s$ for each mobile node. We vary the maximum speed up to $20m/s$ while holding a fixed minimum speed of $1m/s$ to avoid the speed degradation described in [20].

4.2 Simulation Results

The results presented evaluate our protocol for three metrics in a variety of settings. The first metric measures the context's consistency, i.e., the percentage of nodes receiving a context notification given the nodes that were actually within the context when the query was issued. Using this method to evaluate a proposed context definition, we can give an application using the protocol an idea of how successful it will be in reaching the members of its contexts. Applications can use this information to tailor their context definitions to the combination of their needs and requirements. For example, an application that relies on strong guarantees (e.g., the application transfers money or measures

[1] In CSMA/CA a node ready to send senses the medium for activity and uses a back off timer to wait if the medium is busy. When the node senses a clear medium, it broadcasts the packet but waits for no acknowledgements.

Range (m)	50	75	100	125	150	175	200	225	250
Neighbors	1.09	2.47	4.21	6.38	9.18	12.30	15.51	19.47	23.89

Figure 10. Average number of neighbors for varying transmission ranges

safety criticality) will have to define contexts that have an extremely high level of consistency. At the other end of the spectrum, many applications can accept a best-effort style of interaction, and can therefore define wider contexts that provide weaker guarantees.

The second metric measures the context notification's settling time, i.e., the time that passes between the reference host's issuance of a context query and the time that every node in the context that will receive the query has received it. This is the first step in providing applications with information about how long they should wait for responses from differently sized contexts before timing out and resending a query if necessary. This metric also gives us, as protocol implementers, some information about how long a single context definition is utilizing network resources.

The third metric evaluates the protocol's efficiency through the rate of "useful broadcasts", i.e., the percentage of broadcast transmissions that reached nodes that had not yet received the context query. As we will see in the discussion of the results, this measurement provides us insight into under what conditions (e.g., high speeds, densities, or loads) the protocol might require tailoring in the dynamic ad hoc network.

The first set of results compare context definitions of varying sizes, specifically, definitions of one, two, three, and four hop contexts. We then evaluate our protocol's performance as network load increases, specifically as multiple nodes define contexts simultaneously. Unless otherwise specified, nodes move with a $20m/s$ maximum speed.

Reasonably Sized Contexts Have Good Consistency Guarantees. In comparing contexts of varying sizes, we found that as the size of the context (measured in this example in the number of hops) increases, the consistency of the context decreases slightly. Results for different context sizes are shown in Figure 11. These results show a single context definition on our 100 node network. The protocol can provide localized contexts (e.g., one or two hops) with near 100% consistency. With broader context definitions, the percentage of the context notified can drop as low as 94%. The disparity between large and small context definitions becomes most apparent with increasing network density. At large densities, the extended contexts contain almost the entire network, e.g., at a transmission range of $175m$, a four hop context contains ∼80% of

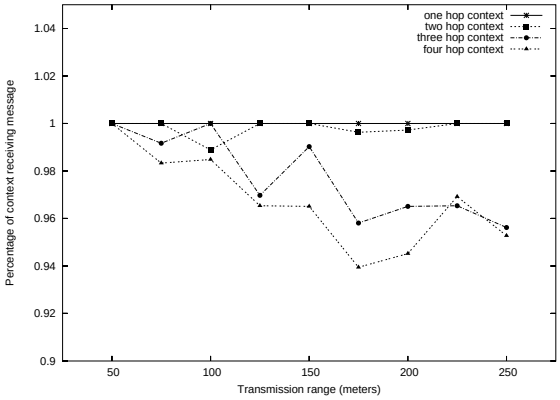


Figure 11. Percentage of context members receiving the message for contexts of varying sizes

the network's nodes. In addition, the number of neighbors is 12.3, leading to network congestion when many neighboring nodes rebroadcast. This finding lends credence to the idea that applications should define contexts which require guarantees (e.g., automobile collision detection) as more localized, while contexts that can tolerate some inconsistency (e.g., traffic information collection) can cover a larger region. In addition, small modifications to the protocol that address the fact that neighboring nodes should not rebroadcast simultaneously may positively benefit performance. We discuss this problem (called the "broadcast storm") and some possible solutions in the next section.

Context Building Settles Quickly. As the size of the defined context increases, more time is required to notify all the context members. Figure 12 shows the settling times for contexts of varying sizes defined on networks of increasing density. For a two hop context with a reasonable density (9.18 neighbors at $150m$ transmission range), the maximum time to notify a context member was $20.12ms$. The settling times for different sized networks eventually become similar as network density increases. This is due to the fact that even though the context is defined to be four hops, all nodes are within two hops of each other, effectively rendering a four hop context definition a two hop context.

Efficiency Decreases Almost Linearly with In-

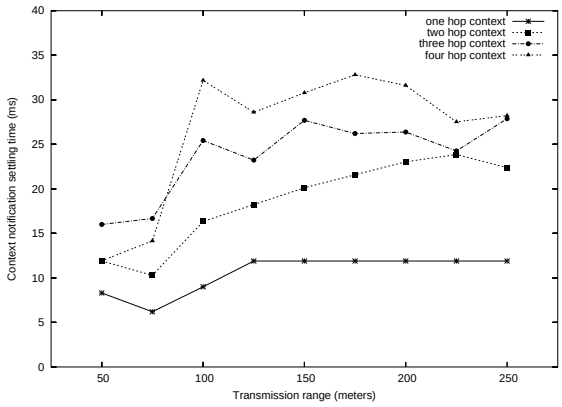


Figure 12. Settling time for contexts of varying sizes

creasing Density. Figure 13 shows the protocol's efficiency versus density for different sized contexts. First, notice that the efficiency for a one hop network is always 100% because only one broadcast (the initial one) is ever sent. For larger contexts, the efficiency is

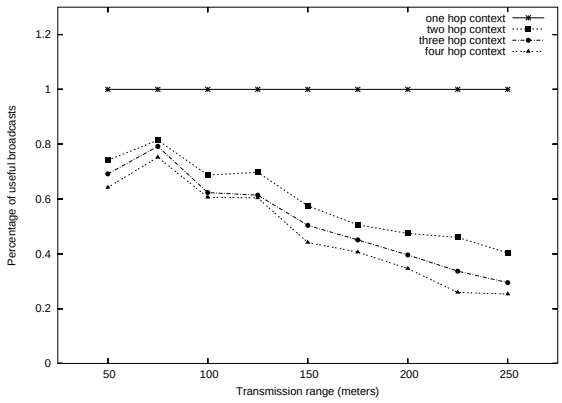


Figure 13. Percentage of broadcasts that reached new context members for contexts of varying sizes

lower and decreases with increasing density. Most of the lower efficiency and the descending nature of the curve results from the fact that rebroadcasting neighbors are likely to reach the same set of additional nodes. This becomes increasingly the case as the density of the network increases. Even at high densities, however, a good number ($> 20\%$) of the broadcasts reach additional context members. In the next section, we discuss possible solutions to increase the performance of the protocol in these cases.

Consistency Remains above 80% with Increased Network Load. The remainder of the analysis focuses on an increasing load in the network, caused by multiple simultaneous context definitions by multiple nodes in the network. We show only results for four hop contexts because they are the largest and have the worst behavior; results for smaller contexts are discussed in comparison. As Figure 14 shows, five context definitions have no significant impact on the consistency as compared to a single definition. For ten def-

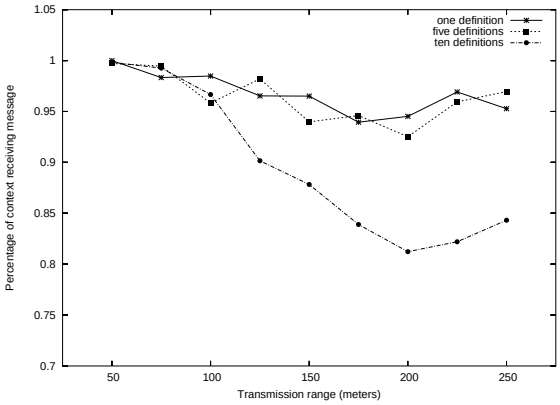


Figure 14. Percentage of context members receiving context messages for varying network loads

initions, the atomicity starts to decrease, but remains above $\sim$80% at all transmission ranges. With more registrations, especially at the larger densities, the different context messages interfere significantly with each other. Two factors contribute to this observation. The first is that the broadcast messages collide and are never delivered. The second results from the fact that MAC 802.11 uses CSMA/CA. Because the medium is busier (more neighboring nodes are broadcasting), nodes are more likely to back off and wait their turn to transmit. During this extended waiting time, the context members are moving (at a maximum speed of $20m/s$). Because the hosts are moving rapidly, context members that were in the context initially move out of it before the query can traverse the entire context. These effects decrease significantly with smaller context sizes, e.g., at a transmission rate of $175m$, ten definitions on a two hop context can be delivered with $\sim$97% consistency, and twenty can be delivered with $\sim$89.5% consistency. This type of information informs applications that, in extremely mobile, dense, or active networks, contexts that span a smaller set of nodes are likely to be more consistent with respect to delivery guarantees. Applications can use this information to

determine which types of contexts are appropriate in different environments.

Changing Speed has No Impact on Context Notification. In our analysis of this protocol, we tested scenarios with a wide variety of network speeds. We found that even the consistency of context message delivery is not greatly affected by the speed of the nodes. It is likely that, were we to analyze transmission of replies to queries, we would find that routes are somewhat less likely to hold up for scenarios with higher node speeds. Such concerns are addressed by the maintenance protocol; simulation results for this portion of the protocol lie outside the scope of this paper and are planned as future work.

5 Discussions and Future Work

Several of our results show that increased network congestion negatively affects our protocol. Specifically, Figure 11 showed that the consistency of context dissemination decreases as more neighboring hosts rebroadcast, and Figure 13 showed that the efficiency of the broadcast mechanism decreased with increasing density. This results from a commonly known problem called a "broadcast storm". [13] describes this problem in mobile ad hoc networks and quantifies the additional coverage a broadcast gains. Several alternative broadcasting mechanisms have been proposed, many of which are compared in [19]. Such methods include using probabilistic methods or knowledge about the environment or neighbor set to determine when to rebroadcast. Integrating these or similar intelligent broadcast mechanisms may increase the resulting consistency and efficiency of context notification.

Figure 14 shows that the consistency of context notification tends to fall off when network load increases. Future work includes investigating ways to handle this undesirable effect. This could include reusing information available about already constructed contexts to limit the amount of work required to construct another context for a new node. Also, one-time context distributions may be able to use information stored on nodes servicing persistent queries over maintained contexts.

The Network Abstractions protocol uses the application's specified distance metric to build the associated context. With some knowledge about the system (e.g., radio transmission range, maximum node speed, etc.) [16] shows that a node can predict a "safe distance" for a link. Incorporating a similar idea may allow us to redefine applications' contexts on the fly to essentially replace a context specification like "all nodes within two hops" with one like "all nodes guaranteed to remain within two hops for $20ms$".

6 Conclusions

This work implements and analyzes a protocol for providing contexts in mobile ad hoc networks. The protocol provides a flexible interface that gives the application explicit control over the expense of its operation while maintaining ease of programming by making the definition of sophisticated contexts simple. This protocol generalizes the notion of distance to account for any property, allowing an application to adjust its context definitions to account for its instantaneous needs or environment. Most importantly, the protocol explicitly bounds the computation of the application's context to exactly what the application needs. In general, these interactions will be localized in the neighborhood surrounding the host of interest, and therefore the host's operations do not affect distant nodes. This bounding allows an application to tailor its context definitions based on its needed guarantees The implementation presented here is currently in use by the EgoSpaces middleware system. This will provide further evaluation and feedback for protocol refinement. We also presented an initial analysis of our protocol over a variety of networks and situations, showing that it is practical in reasonable situations.

Acknowledgements

This research was supported in part by the Office of Naval Research MURI Research Contract No. N00014-02-1-0715. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the Office of Naval Research.

References

[1] G. Abowd, C. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: A mobile context-aware tour guide. *ACM Wireless Networks*, 3:421–433, 1997.

[2] S. Bae, S.-J. Lee, W. Su, and M. Gerla. The design, implementation, and performance evaluation of the On-Demand Multicast Routing Protocol in multihop wireless networks. *IEEE Network, Special Issue on Multicasting Empowering the Next Generation Internet*, 14(1):70–77, 2000.

[3] J. Broch, D. B. Johnson, and D. A. Maltz. The dynamic source routing protocol for mobile ad hoc networks. Internet Draft, March 1998. IETF Mobile Ad Hoc Networking Working Group.

[4] J. Broch, D. Maltz, D. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop

wireless ad hoc network routing protocols. In *Proc. of MobiCom*, pages 85–97, Oct. 1998.

[5] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. Experiences of developing and deploying a context-aware tourist guide: The GUIDE project. In *Proc. of MobiCom*, pages 20–31. ACM Press, 2000.

[6] C. Chiang, M. Gerla, and L. Zhang. Adaptive shared tree multicast in mobile wireless networks. In *Proc. of GLOBECOM '98*, pages 1817–1822, Nov. 1998.

[7] A. K. Dey, D. Salber, and G. D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human Computer Interaction*, 16(2–4):97–166, 2001.

[8] S. Gupta and P. Srimani. An adaptive protocol for reliable multicast in mobile multi-hop radio networks. In *IEEE Workshop on Mobile Computing Systems and Applications*, pages 111–122, 1999.

[9] IEEE Standards Department. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. IEEE standard 802.11-1999, 1999.

[10] C. Julien, G.-C. Roman, and Q. Huang. Declarative and dynamic context specification supporting mobile computing in ad hoc networks. Technical Report WUCSE-03-31, Washington University, 2003.

[11] L. Kleinrock and J. Silvester. Optimum transmission radii in packet radio networks or why six is a magic number. In *Proc. of the IEEE Nat'l. Telecommunications Conf.*, pages 4.3.1–4.3.5, 1978.

[12] E. Madruga and J. Garcia-Luna-Aceves. Scalable multicasting: The core assisted mesh protocol. *ACM/Baltzer Mobile Networks and Applications, Special Issue on Management of Mobility*, 1999.

[13] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proc. of MobiCom*, pages 151–162, 1999.

[14] C. Perkins and P. Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. In *ACM SIGCOMM '94 Conf. on Communications Architectures, Protocols and Applications*, pages 234–244, Oct. 1994.

[15] C. Perkins and E. Royer. Ad hoc on-demand distance vector routing. In *Proc. of the 2^{nd} IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, Feb. 1999.

[16] G.-C. Roman, Q. Huang, and A. Hazemi. Consistent group membership in ad hoc networks. In *Proc. of the 23^{rd} Int'l. Conf. on Software Engineering*, May 2001.

[17] G.-C. Roman, C. Julien, and Q. Huang. Network abstractions for context-aware mobile computing. In *Proc. of the 24^{th} Int'l. Conf. on Software Engineering*, pages 363–373, May 2002.

[18] E. Royer, P. Melliar-Smith, and L. Moser. An analysis of the optimum node density for ad hoc mobile networks. In *Proc. of the IEEE Conference on Communications*, 2001.

[19] B. Williams and T. Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *Proc. of MobiHoc*, pages 194–205, 2002.

[20] J. Yoon, M. Liu, and B. Noble. Random waypoint considered harmful. In *Proc. of INFOCOM*, 2003.

[21] H. Zhou and S. Singh. Content based multicast (CBM) in ad hoc networks. In *Proc. of MobiHoc*, 2000.