

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

Christine Julien and Gruia-Catalin Roman
Department of Computer Science and Engineering
Washington University
Saint Louis, MO 63130
Email: {julien, roman}@cse.wustl.edu

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 in an on-demand manner to mobile applications. We also provide an analysis of the protocol in highly dynamic ad hoc networks.

I. 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 [1], DSR [2], AODV [3]) 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 [4], [5] or mesh [6], [7] for delivering messages to registered receivers.

In both unicast and multicast, the paths 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. To function, senders and receivers in traditional routing protocols require knowledge about

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.

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 [8], [9] 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 interact to gather the information being presented to the user. The scope of this interaction is driven by the instantaneous needs of the application, 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 should be evaluated in a timely fashion to ensure the freshness of the information. Previous work resulted in the Content-Based Multicast model (CBM) [10], which focuses on disseminating information collected by sensors. In general, this model is tailored to 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. 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. 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 con-

text. Here we present the first implementation of the *Network Abstractions* model presented in [11]. In this paper, we explore the protocol in more detail, focusing on its practicality, implementation, and performance in an effort to quantify the cost of working with extended contexts in ad hoc networks.

The remainder of this paper is organized as follows. Section II provides an overview of our Network Abstractions model and protocol. Section III discusses our current implementation. Section IV provides an analysis of the model through simulation. Discussions and conclusions appear in Sections V and VI respectively.

II. NETWORK ABSTRACTIONS OVERVIEW

Ad hoc mobile networks contain many hosts and links with varying properties which define the context for any individual host in the network. The behavior of an adaptive application depends on this continuously changing context. This context definition is broader than traditional definitions that include only local information. This 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. Our approach allows the context specification to remain as general and flexible as possible while ensuring the feasibility of the protocol to dynamically compute the context. The *Network Abstractions* model provides an application on a particular host, called the reference, the ability to specify a context that spans a subset of the network.

A. Model Overview

As discussed previously, an application in an ad hoc mobile network operates optimally 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 Fig. 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 represents 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

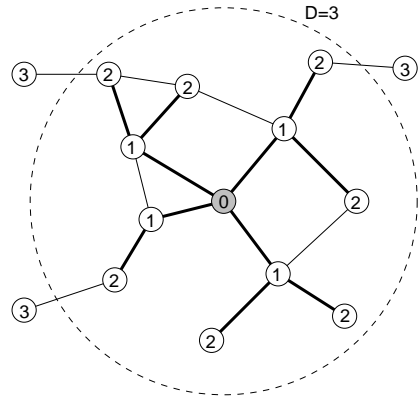


Fig. 1. A Network Abstraction defined to include all hosts within three hops of the reference (shown in gray)

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. This can be computed using information available to the two nodes connected by the link. 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 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 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. As will become evident in the upcoming examples, we must require that the cost function strictly increases with the number of hops from the reference host. 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 Fig. 1, we build a tree rooted at the reference node that includes only the lowest cost path to each node in the network.

An application exploits the availability of the cost function and its associated properties to limit the scope of the context computation by providing a bound on the maximum allowable cost. Nodes to which the cost is less than the bound are included in the context. This allows the computation to avoid touching nodes outside its context bound.

B. Example Metrics

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

1) *Network Latency*: Consider an application in which field researchers share sensor data and video feeds. The context requirements for each researcher's tasks will likely be different. The Network Abstractions model allows each researcher to tailor his context definition to his needs by defining a weight for each network link. Because we are sending video, we want a link's weight to account for the node-to-node latency:

$$w_{ij} = \frac{\text{node latency}_i}{2} + \frac{\text{node latency}_j}{2} + \text{link latency}_{ij}.$$

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 the node's 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) *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, accidents, traffic patterns, etc. As a car moves, it wants to operate over the information that will affect its immediate route, so the data should be restricted to information within a certain physical distance (e.g., within a mile).

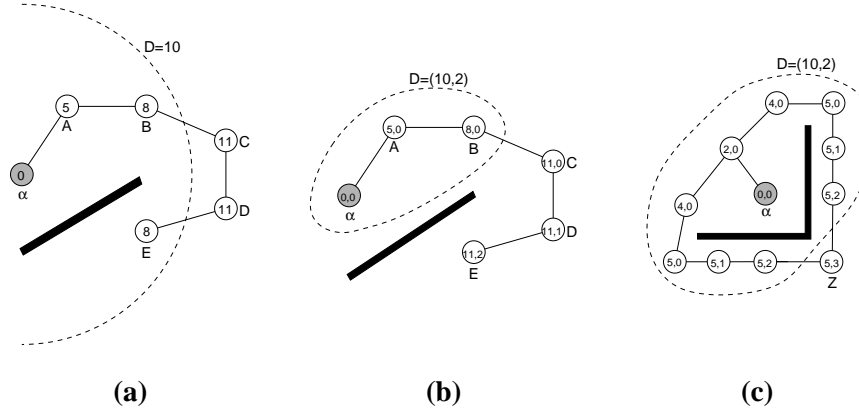
The calculated context should be based on the physical distance between the reference host and other hosts. For this example, a link's weight reflects the distance vector between two connected nodes, accounting for both the displacement and the direction of displacement between the two nodes: $w_{ij}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

Fig. 2a 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.

Fig. 2d 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.

Fig. 2b shows the same nodes as Fig. 2a using this new



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

(d)

Fig. 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

cost function. The application specified bound shown in Fig. 2b is $D = (10, 2)$ where 10 is the bound on $\text{max}D$ and 2 is the bound on C . This cost function can be correctly bounded, and no hosts that should qualify are missed. Fig. 2c 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.

C. 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 builds a tree over the network 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. More details of the protocol can be found in [11] and [12].

In general, the protocol can be divided into two components. The first deals with the dissemination of one-time queries on the context. Such queries may require replies from context members, but the context that is built need not be maintained. This lack of maintenance is beneficial when operation over the context occurs in a periodic polling fashion, because it reduces the overhead needed to maintain the context in a highly dynamic

ad hoc network. The second portion of the protocol deals with maintaining the context when the application needs continuous information. 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).

1) *Assumptions*: This protocol assumes a message passing mechanism that guarantees reliable delivery with associated acknowledgements. The protocol also assumes that when a link disappears, both hosts that were connected by the link can detect the disconnection. Finally, we assume that the underlying system maintains the weights on links in the network by responding to changes in the contextual information required by applications.

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 should receive this message.

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 all the neighbors; those not in the context disregard the message. 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.

4) *Tree Maintenance*: As discussed above, contexts over which an application issues persistent queries require maintenance. The protocol for maintaining the context builds on the one-time query protocol above. Ultimately, the entire protocol is an extension of a distance-vector protocol with modifications for managing the distance metric and bound. 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. Due to the nature of distance vector routing, this protocol suffers from 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. Under the assumption that maintained contexts will be small, this problem can be overcome by maintaining the entire routing path.

D. Practical Research Issues

In the remainder of this paper, we present an implementation of the protocol described above. This implementation will allow us to explore the range of distance metrics and cost functions an application can use and to build an extensive software system for operating over contexts in an ad hoc setting. We also provide an analysis 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 performance of 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.

III. IMPLEMENTATION

Our implementation is written entirely 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 can tailor to its needs. Thus, applications can 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. We include 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 describe these two packages briefly before detailing the protocol implementation.

A. Support Packages

1) *Neighbor Discovery*: A node in our protocol receives knowledge of its neighbors from a discovery service. This service uses a periodic beaconing mechanism and 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).

2) *Environmental Monitoring*: Our protocol relies on the availability of context information from the environment. A monitor service provides this information by maintaining a registry of monitors available on the local host and neighboring hosts. An application extends this package to include its needed capabilities. For example, to add a location monitor to the system, the application provides the code that interacts with, for example, a GPS monitor. New monitors must adhere to a standard monitor interface that allows general monitor interaction. The behavior of this package is similar to that provided by the Context Toolkit [13]. Instead of gathering information directly from hosts an arbitrary distance away, however, we focus on gathering context information only about the links that connect a node to its neighbors.

B. Network Abstractions Protocol Implementation

Before defining a context, an application must build a distance metric. This requires developing an object that adheres to a well defined metric interface and includes two methods. The first determines the weights on links

to neighbors using monitors available on the local host and its neighbors. Because this link weight definition is a Java method in the base class that is overridden by the application's subclass, it can include arbitrary code. The second method determines the cost of a path, given a previous cost and a next hop weight. Again, because this can include any code, the cost function definition can be tailored to the application's needs.

An application defines a context by providing a distance metric and a bound. Until a query is registered on the context, however, the protocol simply stores the information locally. It returns to the application a handle to the defined context.

To send a one-time query, the application passes a data packet to the protocol with a handle to a context. 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 propagation 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 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. 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

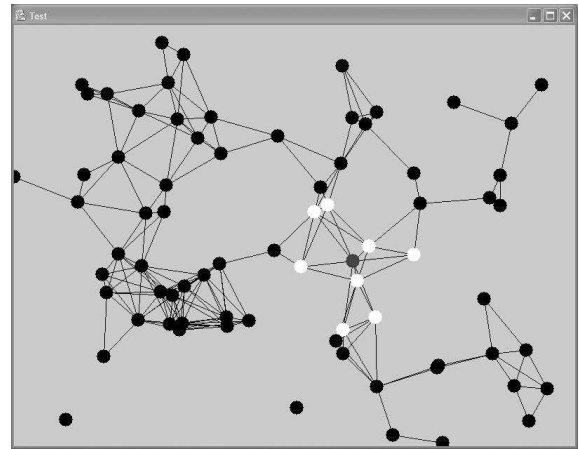


Fig. 3. Screen capture of demonstration system

the current path and the list of possible paths consistent, the protocol monitors the aspects of the context that contribute to distance definition; 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 query 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 IV provides an empirical evaluation of this assumption.

C. Demonstration System

Fig. 3 shows a screen capture of our demonstration system. In this example, each circle depicts a single host running an instance of the protocol. Even though, in this case, all of the code runs on one machine, 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 allows simulation of a variety of

mobility models, including a markov model, a random waypoint model [14], and a highway model.

D. 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.

IV. ANALYSIS AND EXPERIMENTAL RESULTS

To examine the practicality of defining contexts on real mobile ad hoc networks, we used the ns-2 network simulator, version 2.26. Before providing the experimental results, we detail the simulation settings and parameters we used.

A. Simulation Settings

We generated random 100 node ad hoc networks that use the random waypoint mobility model. The simulation is restricted to a $1000 \times 1000 m^2$ 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 Fig. 4. While the random waypoint mobility model suffers from “density waves” as described in [15], it does not adversely affect our simulations. An average of 1.09 neighbors (e.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 [16], more recent work [15] 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 [17] 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 [18].

This section provides simulation results only for context dissemination; in this paper we do not provide results for transmitting replies or context maintenance. All simulations described in this section implement a context defined by the number of hops from the reference node.

B. Simulation Results for Context Query Dissemination

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. 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. 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.

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.

Increased Size of Logical Context Decreases Consistency. In comparing contexts of varying sizes, we found that as the size increases, the consistency of the context decreases. Results for different context sizes are

¹In 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

Fig. 4. Average number of neighbors for varying transmission ranges

shown in Fig. 5. These results show a single context definition on our 100 node network. The protocol can

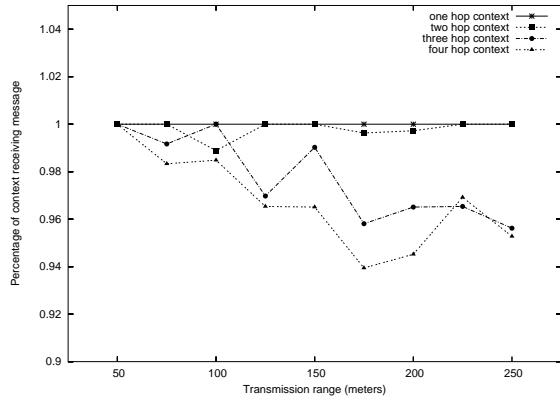


Fig. 5. Percentage of context members receiving the message for contexts of varying sizes

provide localized contexts (e.g., one or two hops) with near 100% consistency. With broader context definitions, the percentage of the context notified drops to 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 $\sim 80\%$ of 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., collision detection) as more localized, while contexts that can tolerate some inconsistency (e.g., traffic information collection) can cover a larger region.

Larger Contexts Take Longer to Settle. As the size of the defined context increases, more time is required to notify all the context members. 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 Increasing Density. Fig. 6 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

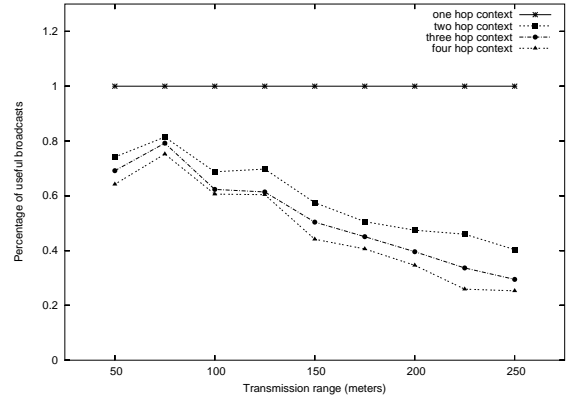


Fig. 6. Percentage of broadcasts that reached new context members for contexts of varying sizes

larger contexts, the efficiency is 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.

Increased Network Load Decreases Consistency. The remainder of the analysis focuses on an increasing load in the network, caused by multiple context definitions by multiple nodes in the network. In all cases, the multiple registrations were issued at randomly distributed times within a 100ms window. We show only results for four hop contexts; results for smaller contexts are discussed in comparison. As Fig. 7 shows, five context definitions have no significant impact on the consistency as compared to a single definition. This is due to the fact that, on average, the different contexts issue queries after other queries have had time to settle. For ten definitions, the atomicity starts to decrease, bottoming out at $\sim 80\%$ at a 200m transmission range. With more registrations, especially at the larger densities, the different context messages interfere with each other.

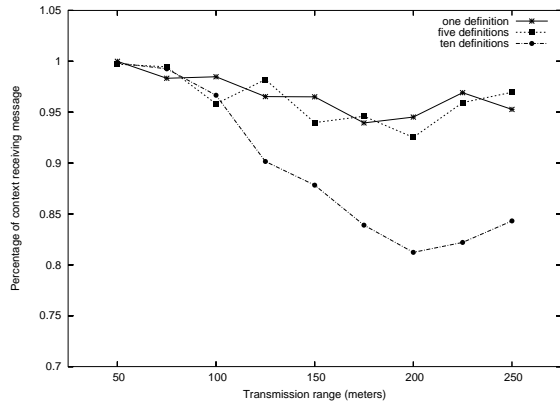


Fig. 7. Percentage of context members receiving context messages for varying network loads

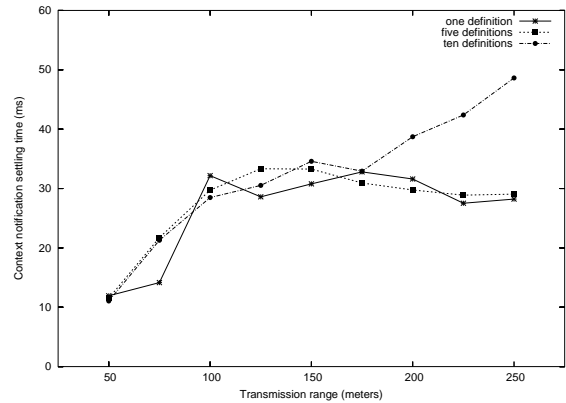


Fig. 8. Maximum time for last context recipient to receive notification for varying network loads

This has two ramifications. 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 $20m/s$). By the time the medium is available, context members that were in the context initially have moved out of it and will not be notified. 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.

Increased Network Load Increases Settling Time at High Densities. Given the previous results, it is not surprising that increasing the network load to five context definitions does not increase settling time. As shown in Fig. 8, however, increasing the network load to ten definitions increases settling times of networks with high densities. Again, when the network density is large and multiple nodes are building contexts, the dispersions of their contexts queries interfere with each other, causing the broadcasting nodes to use their back off timers. This increased back off causes a longer delay in the delivery of context messages, especially to outlying context members.

We do not present any results for efficiency with changing network load, since network load seems to have no real effect on the percentage of useful broadcasts.

Changing Speed has No Impact on Context Notification. In our analysis of this protocol over a variety of network speeds, we found that the dissemination of context messages is not greatly affected by the speed of

the nodes. This is because the queries are only being sent out, and replies are not attempted. Were we to provide results for reply transmission back to the reference host, we would see that the routes are less likely to hold up for the scenarios with higher node speeds. This concern is addressed by the maintenance protocol, but simulation results for this portion of the protocol are outside the scope of this paper.

V. DISCUSSIONS AND FUTURE WORK

Several of our results show that increased network congestion negatively affects our protocol. Specifically, Fig. 5 showed that the consistency of context dissemination decreases as more neighboring hosts rebroadcast, and Fig. 6 showed that the efficiency of the broadcast mechanism decreased with increasing density. This results from a commonly known problem called a “broadcast storm”. [19] 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 [20]. Integrating these or similar intelligent broadcast mechanisms may increase the resulting consistency and efficiency of context notification.

Fig. 7 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.

Our protocol uses the application’s specified distance metric to build the associated context. With some knowl-

edge about the system (e.g., radio transmission range, maximum node speed, etc.) [21] 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”.

VI. CONCLUSIONS

This work develops 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. The authors would also like to thank Qingfeng Huang for his work on the initial model, implementation of mobility models, and simulation advice.

REFERENCES

- [1] 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*, Oct. 1994, pp. 234–244.
- [2] 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.
- [3] C. Perkins and E. Royer, “Ad hoc on-demand distance vector routing,” in *Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, Feb. 1999, pp. 90–100.
- [4] C. Chiang, M. Gerla, and L. Zhang, “Adaptive shared tree multicast in mobile wireless networks,” in *Proc. of GLOBECOM '98*, Nov. 1998, pp. 1817–1822.
- [5] 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*, 1999, pp. 111–122.
- [6] 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*, vol. 14, no. 1, pp. 70–77, 2000.
- [7] 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.
- [8] G. Abowd, C. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton, “Cyberguide: A mobile context-aware tour guide,” *ACM Wireless Networks*, vol. 3, pp. 421–433, 1997.
- [9] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstathiou, “Experiences of developing and deploying a context-aware tourist guide: The GUIDE project,” in *Proc. of MobiCom*, 2000, pp. 20–31, ACM Press.
- [10] H. Zhou and S. Singh, “Content based multicast (CBM) in ad hoc networks,” in *Proc. of MobiHoc*, 2000.
- [11] G.-C. Roman, C. Julien, and Q. Huang, “Network abstractions for context-aware mobile computing,” in *Proc. of the 24th Int'l. Conf. on Software Engineering*, May 2002, pp. 363–373.
- [12] C. Julien, G.-C. Roman, and Q. Huang, “Declarative and dynamic context specification supporting mobile computing in ad hoc networks,” Tech. Rep. WUCSE-03-31, Washington University, 2003.
- [13] 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*, vol. 16, no. 2–4, pp. 97–166, 2001.
- [14] 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*, Oct. 1998, pp. 85–97.
- [15] 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.
- [16] 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.*, 1978, pp. 4.3.1–4.3.5.
- [17] IEEE Standards Department, “Wireless LAN medium access control (MAC) and physical layer (PHY) specifications,” IEEE standard 802.11-1999, 1999.
- [18] J. Yoon, M. Liu, and B. Noble, “Random waypoint considered harmful,” in *Proc. of INFOCOM*, 2003.
- [19] 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*, 1999, pp. 151–162.
- [20] B. Williams and T. Camp, “Comparison of broadcasting techniques for mobile ad hoc networks,” in *Proc. of MobiHoc*, 2002, pp. 194–205.
- [21] G.-C. Roman, Q. Huang, and A. Hazemi, “Consistent group membership in ad hoc networks,” in *Proc. of the 23rd Int'l. Conf. on Software Engineering*, May 2001.