Supporting Context-Aware Interaction in Dynamic Multi-Agent Systems

Christine Julien[1] and Gruia-Catalin Roman[2]

[1] Department of Electrical and Computer Engineering
The University of Texas at Austin
c.julien@mail.utexas.edu
[2] Department of Computer Science and Engineering
Washington University in Saint Louis
roman@wustl.edu

Abstract. The increasing ubiquity of mobile computing devices has made mobile ad hoc networks an everyday occurrence. Applications in these networks are commonly structured as a logical network of mobile agents that coordinate with each other to achieve their goals. In these highly dynamic multi-agent systems, the multitude of devices provides a varied and rapidly changing context in which agents must learn to operate. Successful end-user applications will not only learn to handle dynamic conditions, but will take advantage of the wide variety of available information and resources. Any environment that supports agents and their interactions must facilitate flexible communication mechanisms. Such protocols for enabling an application agents task of gathering contextual information must function in a timely and adaptive fashion. This paper presents a protocol for mediating these context-based interactions among mobile agents. We present an implementation and show how it facilitates information exchange among mobile application agents. We also provide an analysis of the tradeoffs between consistency and range of context definitions in highly dynamic ad hoc networks.

1 Introduction

In large-scale multi agent systems, a software agent must adapt its behavior to a constantly changing environment defined by a multitude of mobile computing devices supporting a variety of other application agents and services. Mobile networks form opportunistically and change rapidly in response to the movement of the hosts and agents that define the network. To communicate, applications 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.

Directly applying these routing techniques to gathering the context information needed by a particular application agent poses several drawbacks. In both

unicast and multicast routing, the paths along which messages are delivered may extend across the entire network. As the ubiquity of mobile devices increases, mobile networks may grow very large, have large network diameters, and support increasing numbers of coordinating agents. Consider a network composed of cars on a highway. Cars may be transitively connected for hundreds of miles, but it is generally not necessary or desirable for an application to communicate at great distances. Each car may support several agents, but many application agents require only local interactions, e.g., an agent may be responsible for gathering local traffic information for a particular driver. 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 agents and services with which it will want to interact, since components in the networks move at will, and agents or services that are encountered once may never be encountered again. Supporting context-aware agents in this unpredictable environment requires reevaluating what application agents need from underlying protocols and providing solutions tailored to these needs.

Emerging applications for this environment (like the traffic example above) focus on using application agents to provide context information to the user. This context can be defined by physical properties of the host or surrounding hosts and by information or services 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, application agents 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 providing a protocol to support an agent's ability to specify what context information it needs from its environment and to gather that information in a manner that adapts to environmental changes. Because the network is constantly being reshaped, an agent'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) [10], 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 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 network surrounding the host where the requesting agent resides. The protocol constructs and dynamically maintains a tree over a subnet of neighboring hosts and links whose attributes contribute to an application agent's specific definition of context. Here we present the first protocol implementing the *Network Abstractions* model [11]. 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 dynamic mobile 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. Conclusions appear in Section 5.

2 Network Abstractions Overview

Today's dynamic mobile networks contain many hosts and links with varying properties which define the context for any individual agent in the network. The behavior of an adaptive agent 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 agent desires the ability to precisely specify its context based on the properties of hosts and links in the network. For example, a network on a highway might extend for hundreds of miles, but an agent operating on behalf of a driver may be interested only in gas stations within five miles. Our approach allows the corresponding agent's 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 agent on a particular host, called the reference, the ability to specify a context that spans a subset of the network.

2.1 Model Overview

As discussed previously, an adaptive application in a 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 agent. Consider the example network shown in Fig. 1. In this network, the reference host where the agent is running is shown in gray. The links shown are available communication links. This figure represents the agent'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 agent's bound on the context (three hops), while the darkened links indicate paths in a tree that define the context.

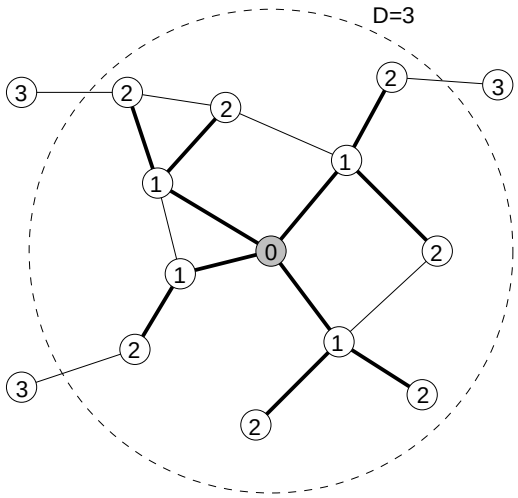


Fig. 1. A Network Abstraction defined to include all hosts within three hops of the reference (shown in gray)

By defining such a context, the agent has restricted its operation to a subnet of the 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 agent 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, an agent 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 agent-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 agent 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.

2.2 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.

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 the agents running on behalf of each researcher to tailor their context definitions to the researcher's 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:

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

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

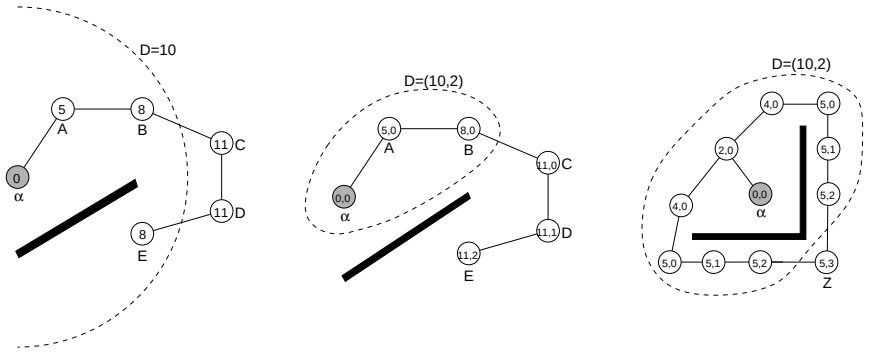
$$f_0(P_k) = f_0(P_{k-1}) + w_{k-1,k},$$

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.

Physical Distance Next we present a general-purpose metric based on physical distance. Agents running on cars traveling on a highway collect information about weather conditions, highway exits, accidents, traffic patterns, etc. As a car moves, its agent wants to operate over the information that will affect the driver's immediate route, so the data should be restricted to information within a certain physical distance (e.g., within a mile).

The agent's 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:

$$w_{ij} = \mathbf{IJ}$$



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

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

(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

Fig. 2a shows an example network where specifying distance alone causes an agent's 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 car agent's cost function should be based on a combination of the distance vector *and* a hop count. The cost function's value (ν) at a given node consists of three values:

$$\nu = (maxD, C, \mathbf{V})$$

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 cost function. The agent specified bound shown in Fig. 2b 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. 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.

2.3 Protocol Overview

An agent desires the guarantee that any message it sends will be received only by hosts within its context *and* that it is received by all hosts within its context. Our protocol builds a tree over the network based on an application agent'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 an agent's one-time queries on its 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 an agent's operation over its context occurs in a periodic polling fashion, because it reduces the overhead needed to maintain the context in a highly dynamic network. The second portion of the protocol deals with maintaining the context when the agent needs

continuous information. Due to the maintenance cost involved, ideal interactions would extend one-time queries to larger contexts (e.g., poll for traffic conditions for the next five miles), but only maintain smaller contexts (e.g., react to cars within potential collision range of my car).

Assumptions The 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. The protocol requires that all configuration changes and an agent's issuance of queries over the context are serializable with respect to each other. A configuration change is defined as the change in the value of the distance metric at a given link and the propagation of those changes through the tree structure. 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 application agents.

The Query Component The protocol is on-demand in that a tree is built only when an agent sends a data query. 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.

Tree Building Because any information required for computing an agent's context arrives in a query, hosts need not keep information about the system's global state. An agent with a data query to send bundles the context specification with the query and hands it to the protocol implementation which in turn determines which of the reference host's 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 if any of their neighbors are also in the context and, if so, 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. Each node that receives the context message for the first time also passes the application level information carried with the query to the designated application agent(s) running on the host.

Tree Maintenance As discussed above, contexts over which an agent issues persistent queries require maintenance. One example of an application that needs

such a persistent query is one in which the application agent wishes to notify the driver of the car if any other cars come within a potential collision radius. 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.

2.4 Practical Research Issues

In the remainder of this paper, we present an implementation and analysis of the protocol described above. The particular reference implementation discussed allows us to explore the range of distance metrics and cost functions application agents can use and to build an extensive software system for operating over contexts in a dynamic mobile environment. We also provide an analysis of the protocol over a simple metric (the hop count example discussed previously) 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.

3 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. We feel that by using Java to provide interfaces to application programmers, we can leverage its object-oriented abstractions to ease the programming task. It is also imperative that we provide a flexible protocol that an application developer can tailor to its needs. Thus, application agents can define individualized distance metrics and add new environmental monitors to the system to increase the flexibility of link weight definitions.

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.

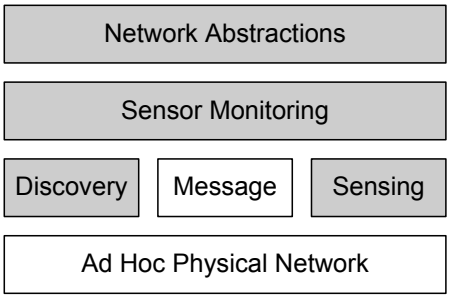


Fig. 3. Architecture of a system using Network Abstractions

3.1 Support Packages

Fig. 3 shows the overall architecture of a system utilizing the Network Abstractions protocol we will describe. The Network Abstractions protocol assumes a physical network and a message passing mechanism to exist. It also relies on two additional packages: a neighbor discovery protocol and an environmental monitoring component comprising both local sensing and neighborhood sensor monitoring.

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).

Environmental Monitoring Our protocol relies on the availability of context information from the environment. To perform this context-sensing service in mobile ad hoc networks, we use the CONSUL monitoring package [13]. As shown in Fig. 3, two components contribute to providing environmental monitoring functionality: the sensing component and the sensor monitoring component. The sensing component allows software to interface with sensing devices connected to a host. Each device has a corresponding piece of software (a *monitor*) within the CONSUL service. An application (or in this case, the Network Abstractions protocol) can interact with a monitor by polling for its value or by reacting to changes in its value. The sensor monitoring component maintains a registry of monitors available on the local hosts (*local monitors*) and on hosts found by the discovery package (*remote monitors*). Local monitors make the services available on a host accessible to applications on that host. To gain access to local monitors, the application provides the name of the monitor (e.g., "location") to the registry.

To monitor context information on remote hosts (i.e., on neighboring hosts), the registry creates local proxies that connect to and interact with monitor components on the remote devices. To access remote monitors, the application provides the ID of the remote host (which can be retrieved from the discovery package) and the name of the monitor. The behavior of this package is similar to that provided by the Context Toolkit [14]. 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 as defined by the discovery package. This allows the CONSUL package to not rely on any centralized infrastructure or even any a priori knowledge, making it highly applicable to dynamic ad hoc networks.

3.2 Network Abstractions Protocol Implementation

Before defining a context, an agent 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 agent'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.

While some application programmers enjoy the flexibility this open interface provides them, the complexity increases the development burden, especially for those programmers unfamiliar with the inner workings of the Network Abstractions protocol. To further ease the use of the protocol, we provide several build in distance metrics and cost functions. These include commonly used metrics, e.g., a cost function based on hop count and a cost function based on physical distance.

An agent defines a context by providing the aforementioned 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 agent 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 agent'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. These listeners are registered by agents or services running on the receiving host that can respond to the sending agent. 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 agent or service on a host receiving a query can reply to a data packet. The protocol uses the stored previous hop information to route the reply back to the reference host and ultimately the sending agent. 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 agent 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 in distance vector protocols. 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; if these values change, the cost at this host or its neighbors could also change. For example, to maintain a context built around physical distance, the protocol must monitor the physical location of this host and the physical locations of all neighbors also in the same context. This is accomplished through the local and remote monitors of the environmental monitoring package. The protocol reacts to these 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; instead they only affect nodes from the point of change out to the bound. Before replacing a path, the protocol checks that the new 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 4 provides an empirical evaluation of this assumption.

3.3 Demonstration System

Fig. 4 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 an agent on the reference 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, the persistent query is removed, and the pictured node

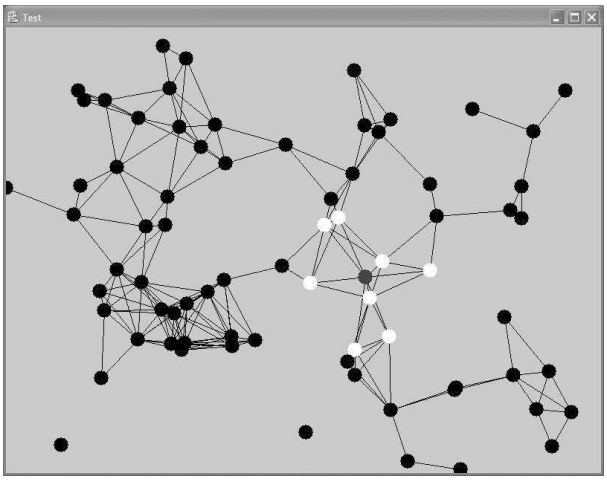


Fig. 4. Screen capture of demonstration system

turns itself black. The demonstration system allows simulation of a variety of mobility models, including a Markov model, a random waypoint model [15], and a highway model. It is useful to developers who wish to visualize the behavior of their context definitions (distance metrics and cost functions) before deploying an application in the real world.

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 [16], 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

The previous sections have overviewed the Network Abstractions protocol and its implementation. In this section, we further motivate the use of this package by developers of mobile agent systems by providing some performance measurements. Ideally, a suite of such measurements will be used by application

developers in determining which context definitions are appropriate for different needs or situations.

To examine the practicality of defining 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 and parameters we used.

4.1 Simulation Settings

We generated random 100 node ad hoc networks that use the random waypoint mobility model [15]. 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 Fig. 5. While the random waypoint mobility model suffers from "density waves" as described in [17], 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 [18], more recent work [17] 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.

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. 5. Average number of neighbors for varying transmission ranges

In our simulations, we used the MAC 802.11 standard [19] 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) [3].

[3] 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.

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 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 shown in Fig. 6. 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 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 $20.12ms$. Results

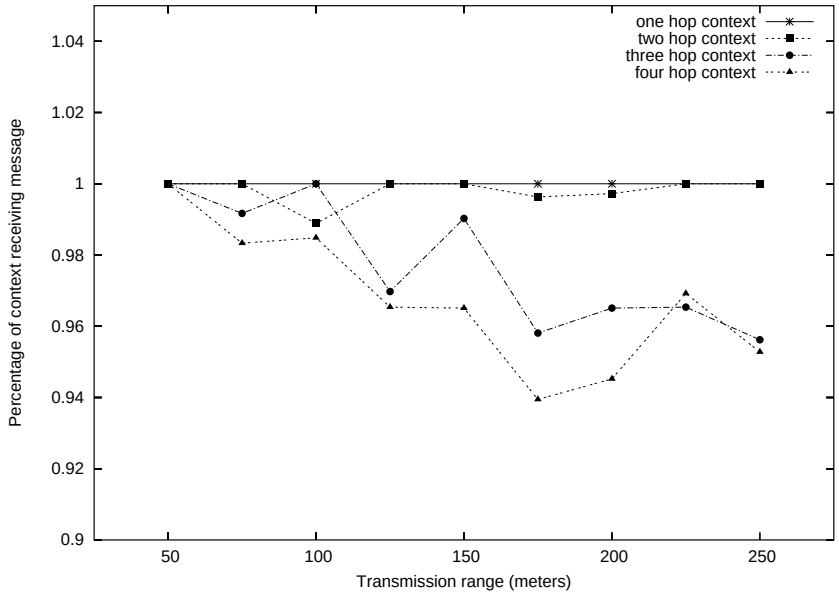


Fig. 6. Percentage of context members receiving the message for contexts of varying sizes

for this measurement are shown in Fig. 7 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. 8 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 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.

This drop in efficiency as the density increases (as well as the corresponding drop in context consistency) is caused in part by a "broadcast storm," a commonly known problem well defined even in ad hoc networks. Previous work [21] has quantified the additional coverage a broadcast gains in mobile ad hoc net-

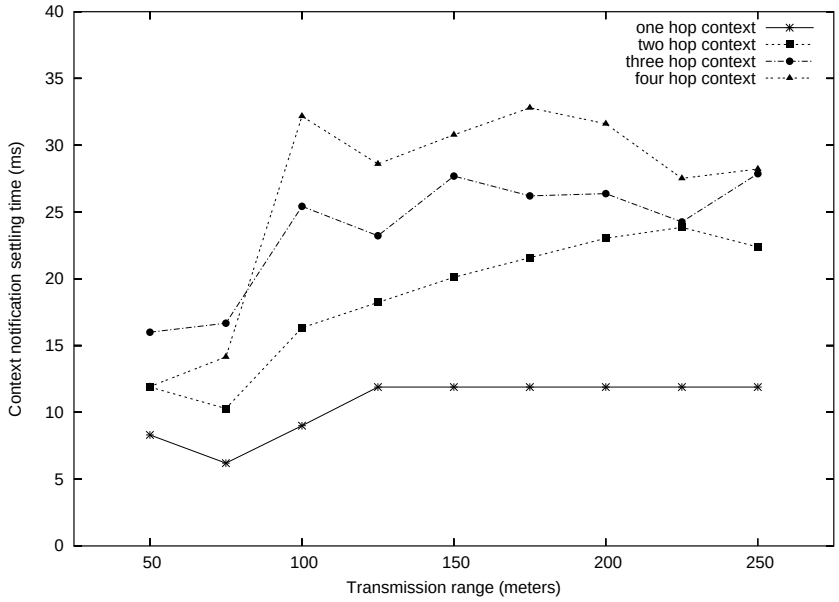


Fig. 7. Maximum time for last context recipient to receive notification for contexts of varying sizes

works. Several alternative broadcasting mechanisms have been proposed, many of which are compared in [22]. Integrating these or similar intelligent broadcast mechanisms may increase the resulting consistency and efficiency of context notification.

Increased Network Load Decreases Consistency. 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. 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. 9 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. This has two ramifications. The first is that the broadcast messages collide and are never delivered. The second results from the fact

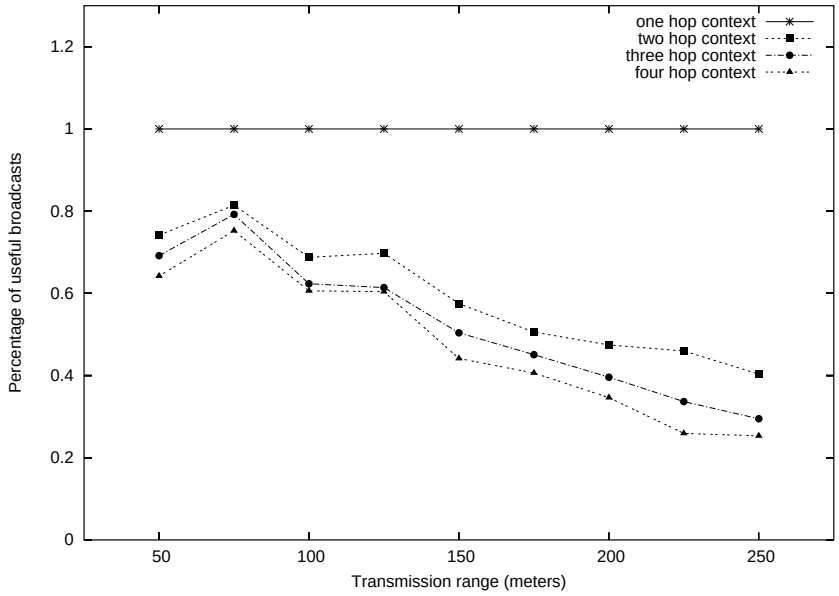


Fig. 8. Percentage of broadcasts that reached new context members for contexts of varying sizes

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.

Extensions to this protocol may be able to start to handle the negative effect that increased network load has on the atomicity metric. These extensions could include reusing information available about already constructed contexts to limit the amount of work required to construct another context for a new agent. Also, one-time context distributions may be able to use information stored on nodes servicing persistent queries over maintained contexts.

Increased Network Load Increases Settling Time at High Densities. Given the previous results, it is not surprising that increasing the network load

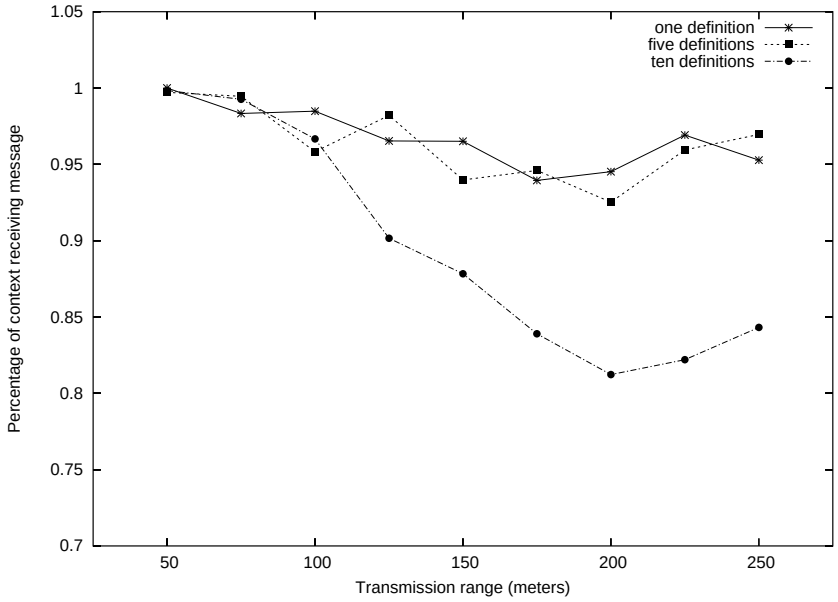


Fig. 9. Percentage of context members receiving context messages for varying network loads

to five context definitions does not increase settling time. As shown in Fig. 10, 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.

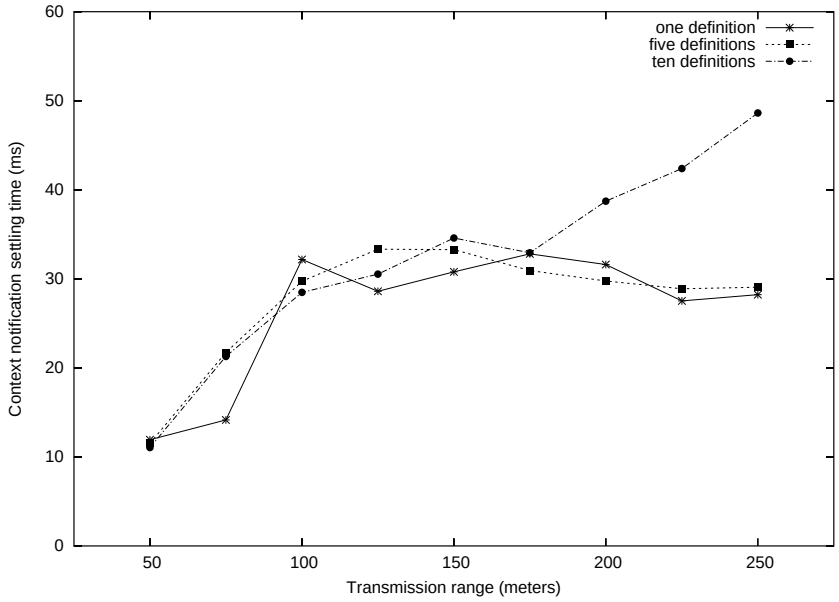


Fig. 10. Maximum time for last context recipient to receive notification for varying network loads

5 Conclusions

The ideas behind this work are rooted in the notion that communication in multi-agent systems for mobile ad hoc networks is an essential component of any environment supporting the execution of such agents. These types of systems are open, decentralized environments in which no centralized authority can control who enters into communication range or even mediate communication among agents who do manage to connect. The agents themselves are often quite autonomous, each with its own independent task and goals to meet. This paper demonstrates the feasibility of the Network Abstractions protocol to specifically support the communication needs of such agents. While the protocol was presented and has been used within the context of mobile ad hoc networks, it can extend to other genres of multi-agent systems in which the communication requirements of the agents can be expressed in some form of a strictly increasing distance metric. The dynamic nature of the protocol allows it to adapt to the openness and unpredictability of a variety of multi-agent environments. In the Network Abstractions protocol, the notion of an agent's *context* is broadened to include, in principle all available information, yet it can be conveniently limited

in scope to a neighborhood whose size and scope is determined by the specific needs of a particular application agent as it changes over time.

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 agent explicit control over the expense of its operation while maintaining ease of programming by making the definition of sophisticated contexts simple. This protocol generalized the notion of "distance" to account for any properties, allowing an application agent to adjust its context definitions to account for its instantaneous needs or environment. Most importantly, the protocol explicitly bounds the computation of the agent's context to exactly what the application needs. In general, in an ad hoc network, these interactions will be localized in the neighborhood surrounding the host of interest, and therefore the agent's operations do not affect distant nodes. This bounding allows the agent to tailor its context definitions based on its needed guarantees. The protocol has been integrated with EgoSpaces, a middleware system for mediating coordination among distributed agents in mobile ad hoc networks. This, coupled with extensions to the analysis presented in this paper will provide further evaluation and feedback for protocol refinement and extension.

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. Perkins, C., Bhagwat, P.: Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. In: ACM SIGCOMM Conference on Communications Architectures, Protocols and Applications. (1994) 234–244
2. Broch, J., Johnson, D.B., Maltz, D.A.: The dynamic source routing protocol for mobile ad hoc networks. Internet Draft (1998) IETF Mobile Ad Hoc Networking Working Group.
3. Perkins, C., Royer, E.: Ad hoc on-demand distance vector routing. In: Proceedings of the 2^{nd} IEEE Workshop on Mobile Computing Systems and Applications. (1999) 90–100
4. Chiang, C., Gerla, M., Zhang, L.: Adaptive shared tree multicast in mobile wireless networks. In: Proceedings of GLOBECOM. (1998) 1817–1822
5. Gupta, S., Srimani, P.: An adaptive protocol for reliable multicast in mobile multihop radio networks. In: IEEE Workshop on Mobile Computing Systems and Applications. (1999) 111–122
6. Bae, S., Lee, S.J., Su, W., Gerla, M.: 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** (2000) 70–77
7. Madruga, E., Garcia-Luna-Aceves, J.: Scalable multicasting: The core assisted mesh protocol. ACM/Baltzer Mobile Networks and Applications, Special Issue on Management of Mobility **6** (1999) 151–165
8. Abowd, G., Atkeson, C., Hong, J., Long, S., Kooper, R., Pinkerton, M.: Cyberguide: A mobile context-aware tour guide. ACM Wireless Networks **3** (1997) 421–433
9. Cheverst, K., Davies, N., Mitchell, K., Friday, A., Efstratiou, C.: Experiences of developing and deploying a context-aware tourist guide: The GUIDE project. In: Proceedings of the International Converence on Mobile Computing and Networking (MobiCom). (2000) 20–31
10. Zhou, H., Singh, S.: Content based multicast (CBM) in ad hoc networks. In: Proceedings of International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc). (2000) 51–60
11. Roman, G.C., Julien, C., Huang, Q.: Network abstractions for context-aware mobile computing. In: Proceedings of the 24^{th} International Conference on Software Engineering. (2002) 363–373
12. Julien, C., Roman, G.C., Huang, Q.: Network abstractions for simplifying mobile application development. Technical Report WUCSE-04-37, Washington University (2004)
13. Hackmann, G., Julien, C., Payton, J., Roman, G.C.: Supporting generalized context interactions. In: Proceedings of the 4^{th} International Workshop on Software Engineering for Middleware. (2004)
14. Dey, A.K., Salber, D., Abowd, G.D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human Computer Interaction **16** (2001) 97–166
15. Broch, J., Maltz, D., Johnson, D., Hu, Y.C., Jetcheva, J.: A performance comparison of multi-hop wireless ad hoc network routing protocols. In: Proceedings of the International Converence on Mobile Computing and Networking (MobiCom). (1998) 85–97
16. Julien, C., Roman, G.C.: Egocentric context-aware programming in ad hoc mobile environments. In: Proceedings of the 10^{th} International Symposium on the Foundations of Software Engineering. (2002) 21–30
17. Royer, E., Melliar-Smith, P., Moser, L.: An analysis of the optimum node density for ad hoc mobile networks. In: Proceedings of the IEEE Conference on Communications. (2001) 857–861
18. Kleinrock, L., Silvester, J.: Optimum transmission radii in packet radio networks or why six is a magic number. In: Proceedimgs of the IEEE National. Telecommunications Conference. (1978) 4.3.1–4.3.5
19. IEEE Standards Department: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. IEEE standard 802.11-1999 (1999)
20. Yoon, J., Liu, M., Noble, B.: Random waypoint considered harmful. In: Proceedings of INFOCOM. (2003) 1312–1321
21. Ni, S.Y., Tseng, Y.C., Chen, Y.S., Sheu, J.P.: The broadcast storm problem in a mobile ad hoc network. In: Proc. of MobiCom. (1999) 151–162
22. Williams, B., Camp, T.: Comparison of broadcasting techniques for mobile ad hoc networks. In: Proc. of MobiHoc. (2002) 194–205