SICC: Source Initiated Context Construction in Mobile Ad Hoc Networks

Christine Julien
Gruia-Catalin Roman
Qingfeng Huang

TR-UTEDGE-2005-003

SICC: Source-Initiated Context Construction in Mobile Ad Hoc Networks

Christine Julien, Gruia-Catalin Roman, and Qingfeng Huang

***Abstract*— Context-aware computing is characterized by software's ability to continuously adapt its behavior to an environment over which it has little control. This style of interaction is imperative in ad hoc mobile networks that consist of numerous mobile hosts coordinating opportunistically via transient wireless connections. In this paper, we provide a formal abstract characterization of an application's context that extends to encompass a neighborhood within the ad hoc network. We provide a context specification mechanism that allows individual applications to tailor their operating contexts to their personalized needs. We describe a context maintenance protocol that provides this context abstraction in ad hoc networks through continuous evaluation of the context. This relieves the application developer of the obligation of explicitly managing mobility and its implications on behavior. We also characterize the performance of this protocol in ad hoc networks through simulation experiments. Finally, we examine real world application examples demonstrating its use.**

I. Introduction

The ubiquity of mobile computing devices opens a user's operating environment to a rapidly changing world where the network topology, or physical connections between hosts, must be constantly recomputed. These network connections link a host to information provided by other members of the computing environment—information we refer to as a host's *context*. In ad hoc networks especially, software must adapt continuously in response to this changing context. In addition, devices are commonly constrained and must rely on other connected devices for information and computation. Mobile units therefore become part of other hosts' contexts. In general, ad hoc mobile networks contain many hosts and links, which define the context for an individual host.

The next section discusses the current trends in context-aware computing, which often limit a host's context to what it can immediately sense or limit the type of information used to define a context to specific information (e.g., location). To support more general context-aware applications, an application must be allowed to define an individualized context; such definitions may need to include varying facets of the environment. Our goals include broadening the context available to a host to include not only properties that can be sensed directly by a host, but also properties of other reachable hosts and properties of the links among them. This approach has the potential to greatly increase the amount of context information available, and therefore an application running on a host should specify exactly the context it needs. For example, an ad hoc network on a highway might extend for hundreds of miles. A driver in a particular car, however, may be interested in only gas stations within five miles along his projected route. An application should supply a definition of its desired context; subsequent operations issued by the application should be performed only over that context. Because we aim to provide both a manner for an application to specify its context and a protocol that computes and maintains the context, we need to allow the context specification to remain as general and flexible as possible while ensuring the feasibility and efficiency of dynamically computing the context.

Many application scenarios will benefit directly from the use of such *declarative context specifications* that explicitly describe an application's communication requirements. Imagine field researchers studying the behavioral patterns of a group of animals. Each researcher monitors a particular animal or animals. The researchers also use temperature and location information to supplement their notes. Perhaps not every researcher carries a thermometer, but temperature information sensed by another researcher within a certain distance will suffice. Therefore, one could define a context to extend just as far as temperature information is valid and use the information contained in the constructed subnet. Extending this particular example, each researcher might carry a camera to automatically record observations. If one researcher's subject moves behind a boulder, the researcher can no longer see it from his location, but he can use another's camera feed to observe the target. The context defined in this case might be bounded by network latency—only cameras within a certain end-to-end latency can provide a camera feed with a high enough frame rate to be useful. This particular example can easily extrapolate to more generalized surveillance applications.

We provide this network abstraction to the application developer through a simple interface for defining expressive metrics over an ad hoc network. Specifically, an application on a particular host, henceforth called the *reference host*, formally specifies a context that spans a subnet of the ad hoc network. This work starts from the premise that the development of mobile applications can be simplified by allowing developers to specify such individualized contexts. First is the question of how to facilitate a formal specification of context that is general, flexible, and amenable for use in ad hoc settings. The solution maps all nodes in the ad hoc network to points in a abstract multi-dimensional space and defines context as the set of points whose distance from the reference does not exceed

Christine Julien is with the Electrical and Computer Engineering Department at the University of Texas at Austin, 1 University Station, C5000, Austin, TX 78712, Email: c.julien@mail.utexas.edu

Gruia-Catalin Roman is with the Department of Computer Science and Engineering at Washington University in Saint Louis, Campus Box 1045, One Brookings Drive, St. Louis, MO 63130, Email: roman@wustl.edu

Qingfeng Huang is with Palo Alto Research Center (PARC), Inc., 3333 Coyote Hill Road, Palo Alto, CA 94303, Email: qhuang@parc.com

a bound that can change throughout the application's lifetime. We show that a number of useful contexts can be defined in this manner. The second issue is one of maintaining the specified context and operating on it. This paper presents the Source-Initiated Context Construction (SICC) protocol which constructs and dynamically maintains a tree over a subnet of hosts and links whose attributes contribute to a context definition, as required by an application on a particular mobile host. Context-sensitive operations are carried out through a cooperative effort involving only hosts in the given context. The final concern is of the presentation of this construct to the application developer. We build a simple programming API that not only includes built-in general-purpose metrics but also provides a usable mechanism for creating additional metrics.

In summary, the specific novel contributions of this work are threefold. First, we present the first formal characterization of an application's context with respect to a dynamic mobile ad hoc network. Second, we develop a communication protocol (SICC) and its implementation that support this notion of context for real-world applications. Finally, we provide an initial performance characterization evaluating the feasibility of deploying our protocol in mobile ad hoc networks.

This paper is organized as follows. Section II details previous work in context-aware computing and presents a new perspective. Section III provides background on existing communication support in ad hoc networks. We present our formal abstraction of the network into a context in Section IV and provide some example applications using this abstraction in Section V. Section VI describes the SICC protocol for context computation and maintenance in detail. Section VII provides initial simulation results for the protocol. Finally, Section VIII concludes.

II. Context-Aware Computing

Initial context-aware projects at Olivetti Research Lab and Xerox PARC laid the foundation for more recent context-aware software. Active Badge [35] uses infrared communication between users' badges and building sensors to forward telephone calls to roving users. PARCTab [34] also uses infrared communication between users' palm top devices and desktop computers to allow applications to adapt to the user's environment. Applications range from simply presenting information to the user about his current location to attaching a file directory to a room for use as a blackboard by users in the room. More recent work [12] in building ubiquitous computing environments uses CORBA and operates over a wired network infrastructure that supports both localization and communication. These systems require extensive infrastructures and constant maintenance. They also rely on a wired communication backbone and do not address the issues inherent to ad hoc networks, including the need to scale to large dynamic networks that span more than single buildings.

More recent context-aware applications serve as tour guides [1], [7] by presenting information about the user's current environment. Fieldwork tools [23] automatically attach context information, e.g., location and time, to notes taken by field researchers. Memory aids [28] record notes about the current context that might later be useful to the user. These applications each collect their own context information and focus on providing a specific type of context, e.g., the guide tools use only location.

Generalized software built to support context-aware application development has begun to be developed. The Context Toolkit [32] abstracts context information through the use of context widgets that form a library that developers can use when constructing context-aware applications. The Context Fabric [13] is founded on the observation that an infrastructure approach to providing context information has advantages over a toolkit approach. While these solutions offer developers much needed building blocks for constructing context-aware applications, even collecting information from distributed sets of sensors, these systems do not explicitly address the needs of applications in large scale ad hoc networks to dynamically discover and operate over a constantly changing context. The work presented in this paper allows applications to limit their operating contexts to a portion of the context data provided by low level sensors or even widget-like components.

GAIA [29] introduces *Active Spaces* as immersive computing environments for context-aware applications. This work addresses the needs of context-aware applications in small networked environments where the available resources in the space can be centrally managed by a kernel in each Active Space. This type of approach does not map well to large-scale context-aware applications in completely wireless environments. CORTEX [33] proposes an infrastructure for context-awareness in nomadic mobile environments that combine mobile entities with a wired infrastructure. This project focuses on quality of service guarantees that can be provided within a region of the network. The goals of CORTEX are in line with ours—to support large-scale mobile computing—but the target environment differs in that the concerns apparent in ad hoc networks require specialized solutions. In addition, the work presented in this paper focuses on generalized context provision, while CORTEX is tailored to concerns related to quality of service guarantees.

From this review, it becomes apparent that supporting context-aware applications in large-scale ad hoc networks requires a redefinition of the notion of context. The key components of this new context definition are:

1) The definition of context should be generalized so that applications interact with different types of context (e.g., location, bandwidth, etc.) in a similar manner.
2) Different applications require contexts tailored to their individual needs.
3) In an ad hoc network, an application's context includes information collected from a distributed network surrounding the application's host.
4) Due to the large-scale environment, context computation and operation must be decentralized.
5) Abstractions of context collection from sensors ease the development burden.

Constructing and operating over this dynamic context in large-scale ad hoc networks requires protocols for supporting the definition of these contexts. Because this information must

be gathered from the network, this protocol must coordinate devices to provide context-awareness.

III. Communication in Ad Hoc Networks

Much work in ad hoc networks has focused on routing. Because gathering context information requires communicating with a set of nodes in the ad hoc network, it will either use an existing algorithm or make use of similar interactions in defining its own specialized behavior. In this section, we review existing mobile ad hoc routing protocols and examine applying these techniques directly to context acquisition.

Ad hoc routing protocols are generally divided into two categories. Table-driven protocols, such as Destination Sequenced Distance Vector (DSDV) routing [26] and Clusterhead Gateway Switch Routing [8] maintain consistent up-to-date information for routes to all other nodes in the network [31]. This class of algorithms is based on modifications to the classical Bellman-Ford routing algorithm [6]. Maintaining routes for every other node in the network can become quite costly. The overhead can be lessened by utilizing routing protocols from the second class, source initiated on-demand routing protocols. Ad-Hoc On-Demand Distance Vector (AODV) routing [27] builds on DSDV but minimizes routing overhead by creating routes on demand. Dynamic Source Routing (DSR) [16] requires that nodes maintain routes for source nodes of which they are aware in the system. Finally, the Temporally Ordered Routing Algorithm (TORA) [22] uses link reversal to present a loop-free and adaptive protocol. It is source initiated, provides multiple routes, and has the ability to localize control messages to a small set of nodes near a topological change. Another type of routing that relates well to the work presented here is Distributed Quality of Service Routing [5] in which routes are selected based on network resources available along that path.

While this is not an exhaustive survey of the current ad hoc routing protocols, it highlights well-known and fundamental approaches. The main gap betwee these protocols and the needs of context-aware applications lies in the fact that the ad hoc routing protocols described require a known source and a known destination. Instead, context-aware programs as described in the previous section require the ability to abstractly specify a group of hosts with which to communicate.

Communication with a subset of nodes in a network is commonly accomplished using multicast. Early approaches to multicasting in ad hoc networks used the shared tree paradigm commonly seen in wired networks, adapting these protocols to account for mobility [9], [10]. More recent work has realized that maintaining a multicast tree in a highly mobile environment can drastically increase the network overhead. This has led to shared mesh approaches [3], [20] for improved reliability. Both the tree- and mesh-based protocols use a shared data structure. That is, they assume that all members in the group have or need the same view about the membership of the multicast group. However, in context-aware applications, each node may need an egocentric view regarding relevant context and in turn an egocentric view of the membership in its context group. As a result, the shared structure paradigm in conventional multicast protocols is not applicable for our context provision and context maintenance purposes.

In summary, while previous research clearly points out many issues a context provision and maintenance protocol must address (e.g., host mobility, transient connections, and changing properties of hosts and links), existing protocols do not apply well to the context construction problem. The distinctive features of our solution include:

1) property-based communication specification: instead of asking a host to explicitly name with which other hosts it wishes to communicate, the host specifies properties of the paths to the communicating parties;
2) context-aware network structure: any data structure built over the network must guarantee that the path used to communicate with a host satisfies the contextual property;
3) scalability: the context space is formed under localized communication thus avoiding a network-wide search.

IV. A Network Abstraction for Context Provision

Extending the availability of context information beyond a host's immediate scope is facilitated by an abstraction of network properties. Without this facility, a programmer must explicitly program at the socket level to find and connect to desired hosts. Additionally, he must directly access context sensors and know how to interact with each different type of sensor. By abstracting these properties, we provide a logical view of available resources and unify interactions. After specifying some constraints that include a distance definition and a maximum allowable distance, an application on the reference host would like a list of qualifying acquaintances. That is:

> *Given a host α and a positive value 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.*

To build this list, we first define a way to determine a shortest path and its cost. Costs derive from quantifiable context aspects. In any network, both hosts and links have attributes that affect communication. We combine these properties to achieve a single weight for each link. An application has the freedom to specify which properties define weights.

Once each link has a weight, an application-specified cost function can be evaluated to determine the cost of a network path. Multiple paths are likely to exist between two given nodes. Therefore, we build a tree rooted at the reference that includes only the lowest cost path to each node. Because we aim to restrict the scope of an application's context, calculating the lowest cost to every node in the network is not reasonable. To limit the context specification, we require the application to specify a bound for its cost function. Nodes to which the cost is less than the bound are included in the context.

A. The Physical Network

Because each application individually specifies which properties to use in its context specification, each application has its own interpretation of the physical network. To begin mapping the ad hoc network to an abstract space, we represent the entire network as a graph $G = (V, E)$ where mobile hosts

are mapped to V, the graph's vertices, and communication links between hosts are mapped to E, the graph's edges. In the ad hoc network, every host and link has attributes that we map to the abstract space represented by the graph G by placing values on every vertex and edge. We quantify relevant properties of a mobile host as a value ρ_i on the vertex $v_i \in V$. Formally, $\rho : V \rightarrow R$. The value of ρ_i can combine properties such as a host's battery power, location, load, service availability, etc. We quantify the properties of a network link as a value ω_{ij} on the edge $e_{ij} \in E$. Formally, $\omega : E \rightarrow \Omega$. The value of ω_{ij} may combine values representing a link's length, throughput, etc.

B. Logical View of the Network

Each application creates a logical network view based on the context that interests it. We designate an application's logical network $\overline{G} = (\overline{V}, \overline{E})$, formed from the original mapping G. We use the information about node and link properties to create a topological *distance* between each pair of connected nodes in the logical network $\overline{G}$ by creating weights on edges in $\overline{G}$. Given an edge $e_{ij} \in E$ and the two nodes it connects $v_i, v_j \in V$, the weights of the two nodes ρ_i and ρ_j are combined with the weight of the edge ω_{ij}, resulting in a single weight m_{ij} on the edge $\overline{e_{ij}} \in \overline{E}$ in the logical network $\overline{G}$. No host $\overline{v_i} \in \overline{V}$ has a weight. Formally, this projection from the physical world to the virtual one can be represented as: $\Gamma : R \times R \times \Omega \rightarrow M$, or more specifically: $m_{ij} = \Gamma(\rho_i, \rho_j, \omega_{ij})$. The value of m_{ij} is defined only if nodes v_i and v_j are connected as we assume $m_{ij} = \infty$ for missing edges.

C. The Path Cost Function

Given the logical network view, we need to assign a cost from the reference $\alpha \in \overline{V}$ to any reachable node $\beta \in \overline{V}$. An application on the reference node specifies a cost function providing instructions on calculating the cost of a given path in $\overline{G}$. A path $P = \langle \overline{v_0}, \overline{v_1}, \cdots, \overline{v_k} \rangle$ indicates the path originating at the reference host, now referred to as $\overline{v_0}$, traversing nodes $\overline{v_1}$ through $\overline{v_{k-1}}$ and terminating at $\overline{v_k}$. As a shorthand, we introduce the notation p_n to indicate the portion of the path P from $\overline{v_0}$ to $\overline{v_n}$ where $\overline{v_n}$ is one of the nodes on the path.

Given a path in $\overline{G}$, the topological cost from $\overline{v_0}$ to $\overline{v_k}$, represented by $f_{v_0}(P_k)$, is defined recursively using a path cost function *Cost*, specified by the reference host's application. The recursive evaluation to determine this value is:

$$f_{v_0}(P_k) = Cost(f_{v_0}(P_{k-1}, m_{k-1,k})$$

$$f_{v_0}(\langle \overline{v_0} \rangle) = 0$$

Fig. 1 shows this function. The figure shows that the cost of, or distance to, host $\overline{v_i}$, represented by ν_i results from the evaluation of the application-specified cost function over the weight of edge $\overline{e_{i-1,i}}$ and the cost of, or distance to, host $\overline{v_{i-1}}$.

For the field research application scenario discussed in Section I, assume the weight of each link in the network is a combination of the total link latency and the inverse of the link's bandwidth. In this case, the cost function is additive with respect to the latency, but maximizing with respect to the inverse of the bandwidth. The entire cost function and its reasoning are presented in Section V.

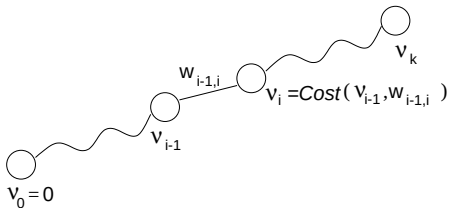


Fig. 1. The recursive cost function

D. The Minimum Cost Path

In an arbitrary graph, multiple paths may exist from α to another node β. We call the cost of the shortest of these paths $g_\alpha(\beta)$. That is,

$$g_\alpha(\beta) = \min_{over\ all\ P\ from\ \alpha\ to\ \beta} f_\alpha(P)$$

There is a shortest path tree T rooted at α. For all nodes β in this tree, the path from α to β in T has cost $g_\alpha(\beta)$. Fig. 2 shows an example tree. The numbers near each edge

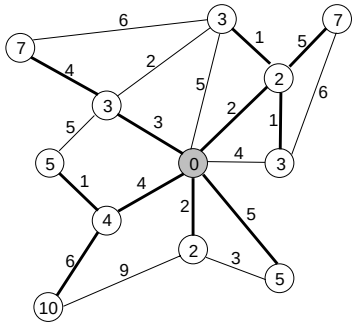


Fig. 2. The logical network and shortest path tree

in the graph represent the weight (m_{ij}) on the link. The cost function used in this example simply adds the weights of the links along the path. The links that form the shortest path tree are darkened. Though the graph shown contains multiple paths from the reference node to each other node, the tree includes only the shortest paths.

E. Ensuring Boundedness

Given a shortest path tree constructed over an ad hoc network, we define a bound on the context. Any nodes for which the cost of the shortest path is greater than the bound are not included in the set of acquaintances. In the case of a field researcher needing video information, the context might be bounded by a combination of tolerable latency and required bandwidth. Therefore, only hosts to which the latency is less than some maximum while the bandwidth satisfies some end-to-end requirement will be included in the context.

Fig. 3 shows the shortest path tree from Fig. 2. The bound D is indicated by the dashed circle. Nodes inside the dashed

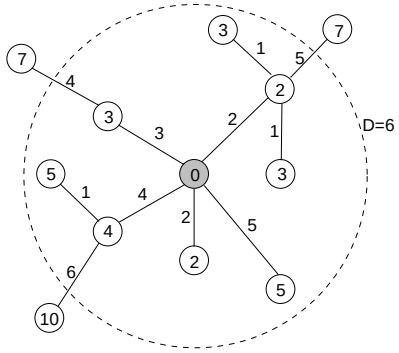


Fig. 3. The bounded shortest path tree

circle are part of α's acquaintance list Q_α, while nodes outside the dashed circle are not.

Notice that this bound is useful only if the shrotest path's cost is strictly increasing as the path extends away from the reference node. That is, if we number the nodes on a path $\langle 1, 2, \ldots, i, \ldots, n\rangle$ and designate the value of the cost to node i as ν_i, we require that $\nu_i > \nu_{i-1}$. This guarantees that a parent in the tree is always topologically closer to the root than its children. With this guarantee, the application can enforce a topological constraint over the search space by specifying the context's bound. This strictly increasing requirement prevents an infinite number of nodes on a path having the same cost, resulting in a context that cannot be bounded.

Defining the properties that contribute to a link's weight and constructing cost functions are the most important aspects of this network abstraction. In the next section we show how the metric allows the definition of a variety of simple, expressive, and flexible context specifications.

V. Sample Metrics

In this section, we explore sample metrics and relate them to specific applications. We start with a very simple metric and then show how more sophisticated application scenarios can be supported using more tailored and complex metrics.

A. Hop Count Calculation

Application. The simplest metric constructed with the abstraction presented in the previous section is one that makes context inclusion decisions based solely on the number of hops from the reference. This can be useful in many applications, and, as we will see, will also serve as a building block for more sophisticated metrics.

Metric. The weight on each link $\overline{e_{ij}}$ connecting two nodes i and j is always one, i.e., m_{ij} = 1. The cost function is additive, i.e., $f_{v_0}(P_k) = f_{v_0}(P_{k-1}) + m_{k-1,k}$. By construction, this cost function is guaranteed to increase at every hop; in fact, it increases by exactly one each step.

B. Building Floor Restriction

Application. This metric constructs a context based on the building locations. The building has a fixed infrastructure of sensors and devices providing information regarding structural integrity, frequency of sounds, movement of occupants, etc. Engineers and inspectors carry PDAs or laptops that provide additional context and assimilate context information. As an engineer moves through the building, he may wish to see structural information not for the whole building, but only for his current floor and the floors adjacent to it.

Metric. The weight on link $\overline{e_{ij}}$ connecting nodes i and j accounts for the floors of the nodes. We define $\rho_i = node\ floor\ \#$ so that the value of ρ corresponds to the floor where the node is located. We do not use the link weight, ω. To generate logical weights, we combine the floors of nodes i and j so that m_{ij} consists of the range of floors of the two nodes $m_{ij} = \{\rho_i, \rho_{i+1}, \ldots, \rho_{j-1}, \rho_j\}$. For example, if nodes on floors 2 and 4 are directly connected, the weight on the link between them will be the range $\{2, 3, 4\}$.

Using a cost function based only on this property does not guarantee the metric will increase. We incorporate the hop count metric to measure the number of network hops the path has taken without moving to a new floor (i.e., a floor that the path has not traversed in the past). The cost function's value ν consists of two values: $\nu = (r, c)$. The first value is the range of floors covered by the path. The second counts the number of hops taken in the current range of floors.

The cost function generates a cost for each node:

$$f_{v_0}(P_k) = \begin{cases} (f_{v_0}(P_{k-1}).r, f_{v_0}(P_{k-1}).c + 1) \\ \qquad \text{if } m_{k-1,k} \in f_{v_0}(P_{k-1}).r \\ (f_{v_0}(P_{k-1}).r \cup m_{k-1,k}, 0) \\ \qquad \text{otherwise} \end{cases}$$

We use $\in$ to refer to the fact that one range is entirely contained in another. The union of two ranges ($\cup$) is the range that exactly covers the two input ranges. The first case in the function corresponds to the situation when the current link does not move to a new floor. The range of floors for the path does not change and the hop count increments by one. In the second case, the current link does move to a new floor. The range of floors for the path is the union of the previous node's range with this link's range. The counter resets to 0. Note that this cost function increases at every hop because either the range expands or the hop count increments.

To bound this context, the application specifies the acceptable range of floors and a hop count. For example, the building engineer might define the bound: $(\{f-1, f, f+1\}, 10)$, where f is his current floor, and this context contains only hosts on his current floor or adjacent ones. As he moves throughout the building, f changes, and his context automatically changes to reflect this. The use of 10 as a hop count is arbitrary; the engineer's application will choose something large enough to ensure that he includes as many nodes as possible while ensuring that performance does not degrade.

C. Network Latency

Application. Next we design a metric for the application scenario introduced in Section I. Briefly, this application consists of field researchers who share sensor data and video feeds. It is likely that the context requirements for each task will be different due to differences in data requirements. For

each task, the researcher defines the context. We focus on the video transmission.

Metric. The weight on link $\overline{e_{ij}}$ connecting two nodes accounts for the node-to-node latency. We later extend the metric to include bandwidth. These are not the only network measurements that might affect video transmissions; more complicated metrics could account for additional constraints. To create this metric, we define

$$\rho_i = \frac{node\ packet\ processing\ latency_i}{2}$$

where $node\ packet\ processing\ latency_i$ is the average time between when node i receives a packet and when it propagates the packet. We use half of this number to avoid counting the node's latency twice, which suffices under the assumption that the incoming and outgoing latencies are approximately equivalent. We define $\omega_{ij} = link\ latency_{ij}$, where $link\ latency$ is the time it takes for a message to travel from i to j.

Possible mappings to the logical network abound; the link latency and node latency can each be given a different importance by weighting the ρ and ω values. For simplicity's sake, the value m_{ij} in the logical network is defined as $m_{ij} = \rho_i + \rho_j + \omega_{ij}$, and the cost function as $f_{v_0}(P_k) = f_{v_0}(P_{k-1}) + m_{k-1,k}$. This cost function increases at every hop because it is additive and each latency term must be strictly positive. A bound on this cost function is defined by a bound on the total latency.

Metric Extension. Because the usefulness of the video feed might also depend on bandwidth, we extend the previous metric to include bandwidth. The ρ values remain the same, but the ω values become pairs of values:

$$\omega_{ij} = (link\ latency_{ij}, \frac{1}{bandwidth_{ij}})$$

We treat this pair as an array; to access the latency, we use: $\omega_{ij}[0]$, and to access the bandwidth, we use: $\omega_{ij}[1]$. We use the inverse of the bandwidth because a connection with a higher bandwidth can be considered "shorter." The value m_{ij} is defined as a pair of values: $m_{ij} = ((\rho_i + \rho_j + \omega_{ij}[0]), \omega_{ij}[1])$.

The cost function computes a pair of values for each node's cost. The first value corresponds to the path's total latency, and the second value stores the minimum bandwidth encountered: $\nu = (latency, bandwidth)$. The cost function is:

$$f_{v_0}(P_k) = (f_{v_0}(P_{k-1})[0] + m_{k-1,k}[0], \\ \mathbf{max}(f_{v_0}(P_{k-1})[1], m_{k-1,k}[1]))$$

This cost function also increases at every hop. Because the latency is completely additive, the $latency$ component increases every hop. Additionally, because we take the maximum of the $bandwidth$ each hop, it is guaranteed not to decrease.

A bound on this cost function consists of a bound on the total latency and a bound on the bandwidth. When either of the cost function components increases beyond its corresponding bound, the path's cost is no longer within the context.

D. Physical Distance

Application. Finally, we present a general-purpose metric based on physical distance. Imagine a network of vehicles on a highway. Each vehicle gathers information about weather conditions, highway exits, accidents, traffic patterns, etc. As a car moves, its driver gathers information that will affect his immediate trip. This data should be restricted to information within a certain physical distance (e.g., within a mile).

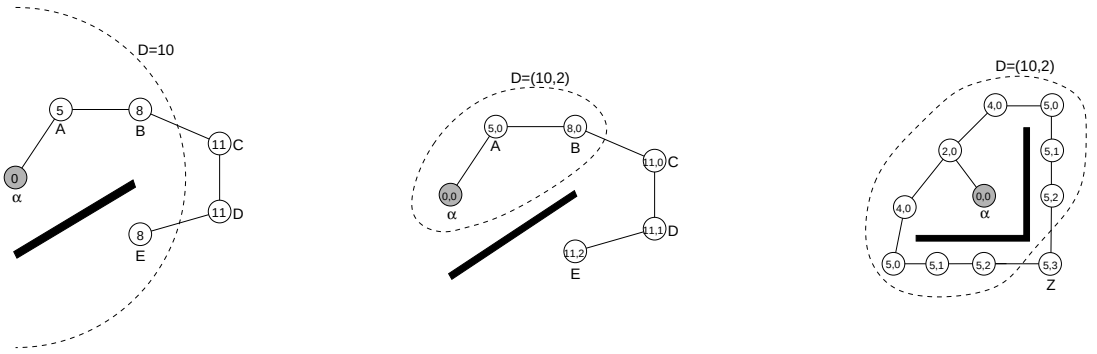
Metric. The calculated context should be based on the physical distance between hosts. The weight placed on edges in the logical network reflects the distance vector between two connected nodes and accounts for both the displacement and the direction of the displacement: $m_{ij} = \vec{\mathrm{IJ}}$.

Fig. 4a shows an example network where specifying distance alone causes the cost function to not be strictly increasing. The reference host, α, is shaded. The numbers on each node indicate the node's cost, given the reference host's cost function. The cost function simply assigns as the cost of a node the distance to the reference. Notice that nodes C and D are outside the context while E should be placed inside the context. In this case, node A cannot communicate directly with node E due to some obstruction (e.g., a wall) between them. When the cost of the path is strictly increasing, host C knows that no hosts farther on the path will qualify for context membership. In this example, this condition is not satisfied, and no limit can be placed on how long context building messages must be propagated.

To overcome this problem, we base the cost function on both the distance vector and a hop count. The cost consists of three values: $\nu = (maxD, C, \mathbf{V})$. The first value, $maxD$, stores the maximum distance of any node seen on this path. 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. Bounding this cost function requires bounding both $maxD$ and C. As will become clear with the definition of our cost function, neither the value of $maxD$ nor the value of C can ever decrease. Also, if one value remains constant for any hop, the other is guaranteed to increase, therefore this cost function is strictly increasing.

Fig. 4d shows the revised 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 $maxD$, and $maxD$ 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. Here, $maxD$ remains the same, C is set to its old value incremented by one, and the distance vector to this host is stored.

Fig. 4b shows the same nodes as Fig. 4a. In this figure the cost function from Fig. 4d assigns the path costs shown. The application specified bound is $D = (10, 2)$ where 10 is the bound on $maxD$ and 2 is the bound on C. The values shown on the nodes in the figure reflect the pair $maxD$ and C. Because the cost function includes a hop count and is based on maximum distance instead of actual distance, node C can correctly determine that no host farther on the path will satisfy the context's membership requirements. Fig. 4c 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.



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

$$f_{v_0}(P_k) = \begin{cases} (|f_{v_0}(P_{k-1}).\mathbf{V} + m_{k-1,k}|, f_{v_0}(P_{k-1}).C, f_{v_0}(P_{k-1}).\mathbf{V} + m_{k-1,k}) & \text{if } |\mathrm{f_{v_0}(P_{k-1}).\mathbf{V} + m_{k-1,k}}| > \mathrm{f_{v_0}(P_{k-1}).}maxD \\ (f_{v_0}(P_{k-1}).maxD, f_{v_0}(P_{k-1}).C + 1, f_{v_0}(P_{k-1}).\mathbf{V} + m_{k-1,k}) & \text{otherwise} \end{cases}$$

(d)

Fig. 4. (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

VI. Context Computation and Maintenance

Our protocol for computing context, Source-Initiated Context Construction (SICC) takes advantage of the fact that an application running on a reference host does not necessarily need to know which other hosts are part of its context. Instead, the application needs to be guaranteed that a message sent to its context is received only by hosts belonging to the context and that all hosts belonging to the context receive the message. SICC builds and maintains a tree over the network. By nature, this tree defines a single route from the reference node to each other node in the context. To send a message only to the members of the context, the reference node needs only to broadcast the message over the tree.

A. Assumptions

SICC relies on a few assumptions regarding the underlying system. First, it assumes that there exists a message passing mechanism that guarantees reliable delivery. Providing this type of guarantee in the highly dynamic ad hoc network can prove difficult and has been much studied. Work on building consistent group membership [14], for example, ensures stable communication given information about hosts' positions, relative velocities, and properties of the wireless network.

We also assume that when a link disappears, both hosts connected by the link detect the disconnection. SICC also requires that all configuration changes and an application'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 metric at a given link and the propagation of those changes through the tree.

Finally, we assume that the underlying system maintains the weights on the links in the network by updating weights in response to changes in node and link properties designated by the application. For each link it participates in, a host should have access to both the weight of the link and the identity of the host on the other side of the link. In our implementation, this information is made available to SICC through the CONSUL [11] sensing package.

B. Protocol Foundations

SICC takes advantage of the fact that a reference host (i.e., the host building the context) specifies the context over which it would like to operate but does not need to know the identities of the hosts in the context. Therefore, the context computation can operate in a purely distributed fashion, where responses to data queries are simply sent back along the path from whence they came. SICC is also on-demand in that a shortest path tree is built only when a reference node sends a query. Piggybacked on this message are the context specification and the information necessary for its computation.

Query, q	
$q.initiator$	initiator's id (the reference node)
$q.num$	application sequence number of q
$q.s$	sender of this copy of q
$q.sd$	distance from the reference to $q.s$
$q.d$	distance from the reference to the host at which the query is arriving
$q.D$	bound on the cost function
$q.Cost$	cost function
$q.data$	application level data associated with this query

Fig. 5. The Components of a Query

Fig. 5 shows a query's components. The sequence number allows SICC to determine whether this query is a duplicate. This prevents a host from responding to the same query multiple times. It should be noted here that a query's sender and a query's reference are not necessarily the same. The reference for a query is the host running the application for

which the context is being constructed. The sender of a query is the most recent host on the path to the current host.

We divide SICC's explanation into three sections: tree building, tree maintenance, and reply propagation. Before we describe the algorithm itself, however, we present the information each host remembers about a single context specification.

C. SICC State Information

id	this host's unique identifier
num	application sequence number, initialized to -1
d	distance from the reference node, initialized to ∞
p	this host's parent in the tree
pd	parent's distance (or cost) from reference node
D	bound on the cost function
$Cost$	cost function
C	set of connected neighbors, the weight of the link to each, and the cost of the path to the neighbor
I	a subset of C containing the connected neighbors that are in the reference's context, initially empty

Fig. 6. State Variables

Fig. 6 shows the state variables that a host β participating in α's context computation holds. This shows only the information needed for one of α's contexts; in general, a host would likely participate in multiple contexts and would therefore store separate state for each.

The set C holds the list of all connected neighbors. Each neighbor has a link to it from this host; the weight of that link is stored in C and is referred to as w_c for some $c \in C$. This set is also used to store other paths to this host. If a host receives a query from host c that would give it a cost $d_c < D$ that it does not use as its shortest path, it remembers c's cost in C. This information will prove useful in quickly finding a new shortest path to replace a defunct path.

D. Context Building

Information required for computing a context arrives in a query; the protocol maintains no global state. An application with a query to send bundles the context specification with the query and sends it to all its neighbors. When a query arrives at a host, it brings the cost function and the bound which together define the context. It also brings the cost to this host.

Any query a host receives is guaranteed to be within the context's bound because the sending node determines the destination node's cost before sending it the query. Only neighbors that fall within the bound are sent the message. Subsequent copies of the same query are disregarded unless they offer a lower cost path. As shown in the second **if** block of the QUERYARRIVES action in Fig. 7, when a shorter cost path is found, the cost of the new path, the new parent, and the new parent's cost are all stored. Also, the query is propagated to non-parent neighbors whose distance will keep them inside the context. This is done through the *PropagateQuery* function, described with SICC's other support functions in Fig. 8. For each non-parent neighbor, c, this host applies the cost function to its own distance and the weight of the link to c. If this results in a cost less than the bound, the host propagates the query to c. A host must propagate a query with a lower cost even if its application has already processed it from a previous parent because this shorter path might allow additional downstream hosts to be included. Finally, upon reception of any query, the host adds the information about the parent to the set C.

```
QUERYARRIVES(q)
  Effect:
    if q.num = num + 1 then
      save query specific information (Cost := q.Cost, D := q.D)
      clear C
      record information (d := q.d, p := q.s, pd := q.sd)
      Propagate Query(q)
      AppProcessQuery(q)
      save the sequence number (num := q.num)
    else if q.d < d then
      record information (d := q.d, p := q.s, pd := q.sd)
      PropagateQuery(q)
    end   update C (d_{q.s} := q.sd)
```

Fig. 7. SICC Context Computation

When a host receives a query that it has not seen before (i.e., the sequence number of the arriving query is larger than the stored one), the application automatically processes it regardless of whether or not it arrived on the currently stored shortest path. A host does not wait for more additional copies of a query to come *only* from its parent because it is possible that the path through the parent no longer exists. If the path does still exist and is still the shortest path, the query will eventually arrive along that path, causing the cost to be updated and the effects to be propagated to the children. Upon receiving a new query, the host processes it in the manner described above. Finally, the host sends the data portion of the query to the application for processing using the *AppProcessQuery* support function described in Fig. 8.

Earlier, we introduced an application in which a researcher associates temperature data with his notes, but he may not carry a thermometer. Others may have thermometers whose data could be used. Once the researcher defines a context to include some thermometers (e.g., a context based on physical distance or thermometer accuracy), he issues a variety of queries over his context, depending on his needs. He may use a *one-time query* if he needs a single piece of data. On the other hand, if the surveillance of the target is ongoing and the temperature data needs to be constantly correlated with notes, the researcher may need a *persistent query*. Next, we classify various types of operations and show how SICC is modified to handle long lasting queries through tree maintenance.

E. Context Maintenance

As discussed above, an application can perform two different types of operations: transient and persistent. A transient operation is a one-time query or instruction. An application issues a persistent query with an initial registration. As long as the persistent operation remains registered, the associated query propagates to hosts that enter the context and is deregistered from hosts that move out of the context. When an application wants to deregister a persistent operation from the entire context, it issues a deregistration query.

PropagateQuery(q)	for each non-parent neighbor, c, send the query to c if $Cost(d, w_c) < D$ by calling SENDQUERY to c after setting $q.d = Cost(d, w_c)$ and $q.s = id$ in the query; update I to include exactly those c to which the query was propagated
AppProcessQuery(q)	application processing of the data message part of the query
SendCleanUps	for each non-parent neighbor, c, send a clean up message to c if $Cost(d, w_c) \geq D$ by calling SENDCLEANUP to c
PropagateCleanUps	for every member of I, send a clean up message by calling SENDCLEANUP
PropagateReply(r)	send the reply to p
AppProcessReply(r)	application processing of the data message part of the reply

Fig. 8. Support Functions

The protocol presented in Fig. 7 is sufficient if the application issues only transient operations. The context needs to be recomputed only if a new query is issued. Because SICC propagates each query to all included neighbors of a host, the shortest path will be computed each time, even if the weights of the links have changed between the queries.

At times, an application needs to register persistent operations on other hosts in its context. These persistent operations should remain registered at all hosts in the context until such time that the reference host deregisters them. The reference host's context needs to be maintained whenever the topology of the network changes, even when no new queries are issued. Any topology change that affects the current context directly reflects a change in at least one link's weight. As stated, we assume that both hosts connected by the link detect the change. That is, if w_{ij} changes, then hosts v_i and v_j are both notified. Hosts whose costs grow as a result of a network topology change may have to be removed from the context, while hosts that enter the context should be notified of the query. To do this, the system reacts to changes in weights on links and recalculates the shortest paths if necessary.

```
QUERYARRIVES(q)
   ... as before

WEIGHTCHANGEARRIVES(wnew_id)
   Effect:
      if id = p then
         calculate the cost (d := Cost(pd, wnew_id))
         if wnew_id > w_p then
            calculate shortest path not through p
               (minpath := min_c Cost(d_c, w_c))
            if minpath < d then
               reset the cost (d := minpath)
               assign new parent
            end
         end
         set the query fields (q := ⟨num, id, d, D, Cost⟩)
         PropagateQuery(q)
      else if wnew_id < w_id then
         if Cost(d_id, wnew_id) < d then
            recalculate cost (d := Cost(d_id, wnew_id))
            reset the parent (p := id)
            set the query fields (q := ⟨num, id, d, D, Cost⟩)
            PropagateQuery(q)
         end
      end
      store the new weight (w_id := wnew_id)
```

Fig. 9. SICC Context Computation and Maintenance

Because both hosts connected by the link are notified of any change, both can take measures to recalculate the shortest path tree. Fig. 9 shows a new action, WEIGHTCHANGEARRIVES added to the protocol to deal with dynamic topology. This action is activated when the notification of a weight change arrives at a host. The weight changes are divided into two categories: the weight of the link to the parent has changed, and any other weight has changed.

In the first case, if the path through the parent has either lengthened, it is possible that the shortest path to this node from the reference node is through a different neighbor. The node sets its cost to be the minimum of the cost through the old parent and the shortest path through any other neighbor (as stored in C). If the length of the path through the parent has shortened, the node should still be included in the context, and the shortest path to it from the reference should still be through the same parent. In both cases, the node recalculates its distance and propagates the information to its neighbors.

If the weight change has occurred on a link to a non-parent neighbor, then the change interests this host only if it causes the path through the neighbor to be shorter than the path through the parent. Because this host is storing distance information for all of its neighbors it simply calculates what the new distance would be, compares it to the stored cost, and resets its values if they have changed. If these calculations change the cost to the node, it should package the current context values in a query and propagate that query using the *PropagateQuery* support function.

The protocol presented in Fig. 9 does not free the memory used to store information about a reference host's context. For example, as a car moves across the country, it leaves information about its specified contexts on every other car it encounters. The car may never come back, so each car that was part of one of these contexts would like to recover its memory when it is no longer part of the context. We add a clean up mechanism as shown in Fig. 10. Whenever it is possible that a change has pushed a host that was in the context out of the context, the parent notifies the child that its context information is no longer useful and should be deleted. There are two places in the algorithm where a change might push another node out of the context. The first is when a weight changes and the path through the parent becomes longer. Not only might this node be pushed out of the context, any of its descendants in the tree might also be pushed out. After calculating its new cost, the node verifies that it is still within the bound. If not, it cleans up its own storage. If this node is still within the bound, it propagates a copy of the current query to its neighbors that will remain within the bound and sends a message to the neighbors that are not within the bound instructing them to clean up this context information.

The second change occurs in the QUERYARRIVES action. When a new query arrives, it is possible that the shortest path has increased in cost, thereby pushing neighbors out of the

context. To account for this, after propagating the query to all neighbors within the bound, the host also sends a clean up message to all neighbors not within the bound.

```
QUERYARRIVES(q)
   ... as before

WEIGHTCHANGEARRIVES(wnew_id)
   ... as before

CLEANUPARRIVES(id)
   Effect:
      if id = p then
         calculate shortest path not through p
            (minpath := min_c Cost(d_c, w_c))
         if minpath < D then
            reset the cost (d := minpath)
            reset the parent
            set the query fields (q := ⟨num, id, d, D, Cost⟩)
            PropagateQuery(q)
            SendCleanUps
         else
            PropagateCleanUps
            clean up local memory
         end
      else
         update d_id in C
      end
```

Fig. 10. SICC Context Computation, Maintenance, and Clean Up

A new action, CLEANUPARRIVES has been added to the protocol shown in Fig. 10. If the clean up message comes from the parent, it is an indication that there no longer exists a path to the reference that satisfies the context specification. A new shortest path is selected using the information in C and the information propagated. If no qualifying shortest path exists, the local memory is recovered. If the clean up message comes from a node other than the parent, C needs to be updated to reflect that the cost to the source is ∞.

F. Reply Propagation

Most applications require responses from the hosts in their contexts. To guarantee the application's bound requirements, these responses must traverse the shortest cost path back to the initiator. Fig. 11 shows the components of a reply. The

Reply, r	
$r.initiator$	the initiator's id
$r.num$	the application sequence number of the query
$r.id$	the replying node's id
$r.cost$	the cost from the initiator to the replying node
$r.data$	the application level data associated with this reply

Fig. 11. The Components of a Reply

initiator's id and sequence number allow the reference to differentiate replies that correspond to different queries.

The information needed to propagate this reply back to the reference is contained within the network. As shown in Fig. 6, each node (other than the reference) maintains a variable p that stores the identity of the next hop back along the shortest path. When a host receives a reply, it checks the destination of the reply, i.e., the initiator. If this host is the destination, SICC passes the reply to the application level using the support function, *AppProcessReply* listed in Fig. 8. If this host is not the destination, SICC propagates the reply back through this host's parent in the context's tree.

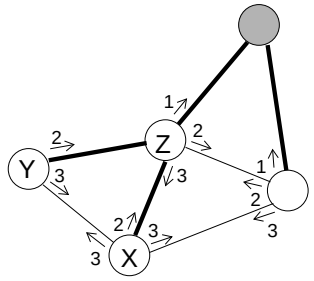


Fig. 12. Mesh reply routing example

The context's tree contains exactly the minimum cost paths to every node in the context. However, a single host in the network may be connected to the reference by multiple paths that satisfy the context's bound. Forcing the routing of replies back to the initiator only along the links present in the shortest path tree ignores using links that have the capability of performing useful work. We extend SICC based on this observation. Most of the information necessary for this mesh routing is already stored at each intermediate node in the state variable C. The necessary changes arise in the structure of the reply message itself and in the behavior of the nodes.

An application requires that every response to a context query travels a path whose total cost is less than the bound. This path can be the shortest to the node, but, when there are multiple paths connecting the reference node to a responding node, the reply can travel any qualifying path. We accomplish this on a mesh by starting each reply message with a bag of tokens. Because the reply has not yet traveled on any link and therefore has not yet incurred any cost, the initial number of tokens in the bag is equal to the context query's bound. As the reply travels toward the initiator, tokens are removed from the message based on the cost of the links traveled.

Fig. 12 shows an example network with a mesh for routing reply messages. The shaded reference host has defined a context to include all nodes within three hops. The shortest path tree constructed for this specification is shown with darker links. The other links can be used for routing reply messages if their costs qualify. In this figure, the numbers on the arrows along the links refer to the shortest possible path from that link back to the reference host. Consider the host labeled X sending a reply. The host packages the reply with a bag of three tokens (because three is the bound). At this point, X can send this message to any of its neighbors because all of the paths are qualifying. Let's say X chooses host Y. X first updates the bag of tokens by subtracting one (the cost of every link in our example is one) and then sends the reply to Y. Y has only one choice of path to send the reply along because the message is not sent back to any previous node on its path. This prevents reply messages from cycling unnecessarily in the network. Y therefore updates the bag of tokens by subtracting one and sends the reply to Z. When the message reaches Z,

the bag contains only a single token. This forces Z to consume the last token and route the reply along the direct link to the reference host. Fig. 13 shows the modified reply.

Reply, r	
$r.initiator$	the initiator's id
$r.num$	the application sequence number of the query
$r.tokens$	the number of tokens remaining for this reply to use, i.e., initially equal to the context query's bound
$r.path$	the hosts that this reply has passed through so far
$r.id$	the relying node's id
$r.cost$	the cost from the initiator to the replying node

Fig. 13. The Components of a Reply

Within the REPLYARRIVES(r) action, shown in Fig. 14, instead of sending the reply back along only the shortest cost path, the node chooses a host from C through which the cost back to the initiator is less than the tokens carried by the reply. Assuming a node does not always choose the same path for replies, this method will increase the performance of the reply propagation by spreading the network traffic to previously unused links. The action uses a support function *SendReply*(r, c) which sends the reply r to the connected neighbor identified by c.

```
QUERYARRIVES(q)
    ... as before

WEIGHTCHANGEARRIVES(wnew_id)
    ... as before

CLEANUPARRIVES(id)
    ... as before

REPLYARRIVES(r)
    Effect:
        if id = r.initiator then
            AppProcessReply(r)
        else
            Choose a host to send the reply through
                (c := c'.(c' ∈ C ∧ c'.cost + c'.weight < r.tokens
                        ∧c' ∉ r.path))[1]
            Update the reply
                (r.tokens := r.tokens − c.weight, r.path.append(c))
            SendReply(r, c)
        end
```

Fig. 14. Reply Propagation Over a Mesh

G. Demonstration System

Fig. 15 shows a screen capture of our demonstration system. Each circle depicts a single host running an instance of the protocol. The system uses the network for communication, which allows us to display information gathered from actual mobile hosts. The figure shows a single context defined by a host (the gray host in the center of the white hosts). The context is simple; it includes all hosts within one hop. When a host moves within the context's bound, it receives a query registered on the context that causes the node to turn its displayed circle white. When the node moves out of the context, it turns itself black. The demonstration system provides simulations using a variety of mobility models, including a markov model, a random waypoint model [4], and a highway model. This system is particularly useful because it allows us to visually match types of context definitions to styles of mobility.

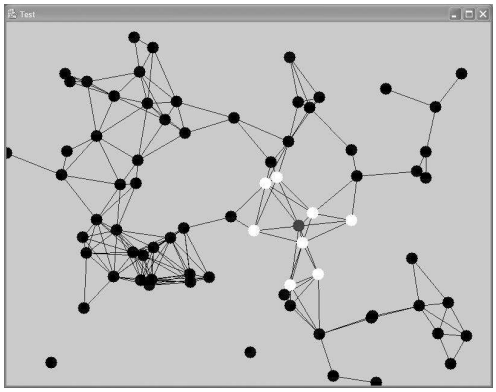


Fig. 15. Screen capture of demonstration system

Our implementation of SICC is currently used to support the EgoSpaces middleware [17], [18], which provides programming constructs for context-aware data management in dynamic mobile ad hoc networks. SICC also serves as the foundation for a customizable query service [24], [25] that performs tailored in-network processing of queries according to policies supplied by the application programmer.

VII. ANALYSIS AND EXPERIMENTAL RESULTS

In this section, we further motivate SICC's applicability by providing initial performance measurements. A suite of such measurements will be essential to application developers in determining which context definitions are appropriate for different needs or situations.

We used the ns-2 network simulator to provide results for context dissemination as a first step in analyzing the practicality of our protocol. Not only do they serve to show that it is beneficial to define contexts in the manner described, 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. The simulations we describe use a context defined by the number of hops from the reference node (as described in Section V-A). This provides a baseline against which we can compare simulations of more complex or computationally difficult definitions.

1) Simulation Settings: We generated 100 node ad hoc networks that use the random waypoint model. While this model suffers from "density waves" as described in [30], it does not adversely affect our simulations. The simulation is restricted to a 1000x1000m^2 space. We vary the network

[1]The nondeterministic selection of a host from the set C uses the nondeterministic assignment statement [2]. A statement $x := x'.Q$ assigns to x a value x' nondeterministically selected from among the values satisfying the predicate Q. As long as our assumptions hold, the statement succeeds because SICC guarantees that there is always at least one path back to the reference.

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. 16. Average number of neighbors for varying transmission ranges

density (measured in average number of neighbors) by varying the transmission range. The average number of neighbors in our simulations for each transmission range are shown in Fig. 16. An average of 1.09 neighbors (i.e., $50m$ transmission range) represents an almost disconnected network, while an average of 23.89 neighbors (i.e., $250m$ transmission range) is extremely dense. While the optimal number of neighbors for a static ad hoc network was shown to be the "magic number" six [19], more recent work [30] 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 [15] in ns-2. Our protocol sends only broadcast packets, for which MAC 802.11 uses Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). This broadcast mechanism is not reliable, and we will measure our protocol's reliability over this broadcast scheme in our simulations.

We also tested our protocol over a variety of mobility scenarios using the random waypoint model with a zero second 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 [37].

2) Simulation Results: The results presented evaluate our protocol for three metrics in a variety of settings. The first metric measures the context's consistency, i.e., the percentage of nodes receiving a context notification given the nodes that were actually within the context when the query was issued. Using this method to evaluate a proposed context definition, we can give an application using the protocol an idea of how successful it will be in reaching the members of its contexts. For example, an application that relies on strong guarantees (e.g., the application transfers money or measures safety criticality) will have to define contexts with high levels of consistency. At the other end of the spectrum, many applications can accept a best-effort style of interaction, and can define wider contexts with weaker guarantees.

The second metric measures the context's settling time, i.e., the time between the reference host's issuance of a query and the time that every node in the context that will receive the query has received it. This is the first step in providing applications with information about how long they should wait for responses from differently sized contexts.

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

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

Reasonably Sized Contexts Have Good Consistency Guarantees. In comparing contexts of varying sizes, we found that as the size of the context increases, the consistency of the context decreases slightly. Results for different context sizes are shown in Fig. 17. These results show a single context definition on our 100 node network. As for all of the results presented throughout this section, the x-axis shows the transmission range of the nodes in our simulated networks. This quantity is a measure of network density and increases from left to right. The protocol can provide

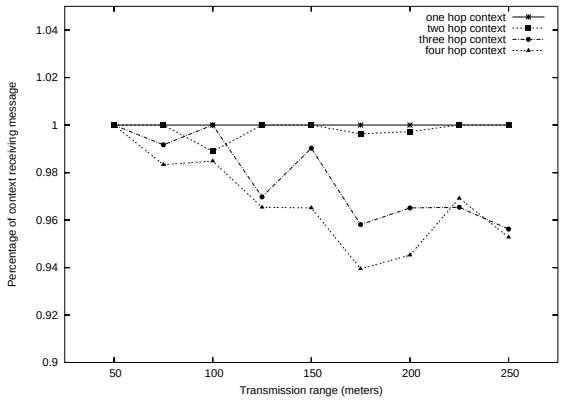


Fig. 17. Percentage of context members receiving the message for contexts of varying sizes

localized contexts (e.g., one or two hops) with near 100% consistency. With broader context definitions, the percentage of the context notified can drop as low as 94%. The disparity between large and small context definitions becomes most apparent with increasing network density. At large densities, the extended contexts contain almost the entire network, e.g., at a transmission range of $175m$, a four hop context contains $\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 as more localized, while contexts that can tolerate some inconsistency can cover a larger region. In addition, small modifications to the protocol that address the fact that neighboring nodes should not rebroadcast simultaneously may positively benefit performance.

Context Building Settles Quickly. Fig. 18 shows the settling times for contexts of varying sizes defined on networks of increasing density. For a two hop context with a reasonable

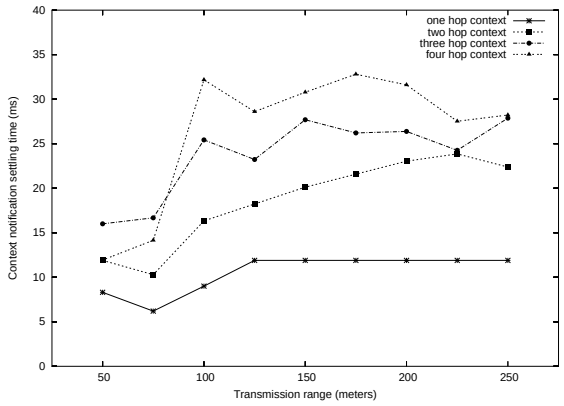


Fig. 18. Settling time for contexts of varying sizes

density, the maximum time to notify a context member was around $20ms$. 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 a two hop context.

Efficiency Decreases Almost Linearly with Increasing Density. Fig. 19 shows the protocol's efficiency versus density. 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

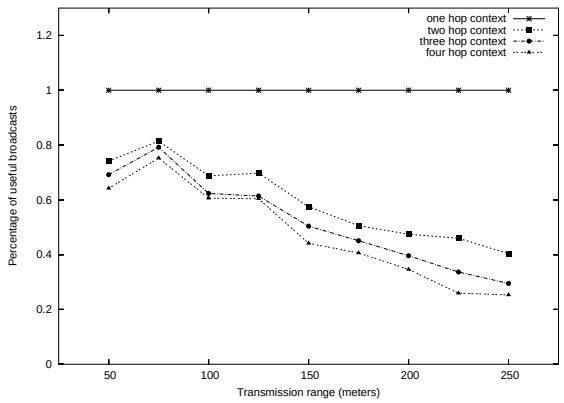


Fig. 19. Percentage of broadcasts that reached new context members for contexts of varying sizes

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. In the next section, we discuss possible solutions to increase the performance of the protocol in these cases.

Consistency Remains above 80% with Increased Network Load. The remainder of the analysis focuses on an increasing load in the network, caused by multiple simultaneous context definitions. We show only results for four hop contexts because they are the largest and have the worst behavior. As Fig. 20 shows, five context definitions have no significant impact on the consistency as compared to a single definition. For ten definitions, the consistency starts

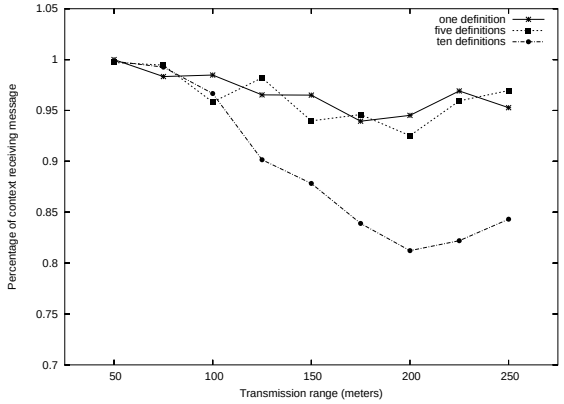


Fig. 20. Percentage of context members receiving context messages for varying network loads

to decrease, but remains above ∼80% at all transmission ranges. With more registrations, especially at larger densities, the different context messages interfere significantly with each other. Two factors contribute to this observation. The first is that 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, nodes are more likely to back off and wait to transmit. During this extended waiting time, the context members are moving, and context members that were in the context initially move out of it before receiving the query. 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 ∼97% consistency, and twenty can be delivered with ∼89.5% consistency. This type of information advises applications that, in extremely mobile, dense, or active networks, contexts that span a smaller set of nodes are likely to be more consistent.

Increased Network Load Increases Settling Time at High Densities. As shown in Fig. 21, 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 context queries interfere, causing the broadcasting nodes to back off. This increased back off causes a longer delay in the delivery of context messages.

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 Little Impact on Context Notification. In our analysis of this protocol, we tested scenarios with a wide variety of node speeds. We found that even the

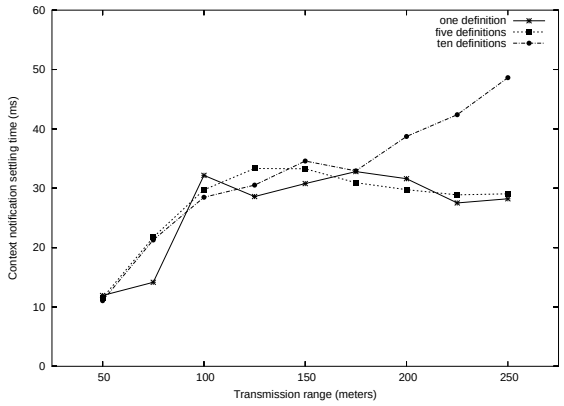


Fig. 21. Maximum time for last context recipient to receive notification for varying network loads

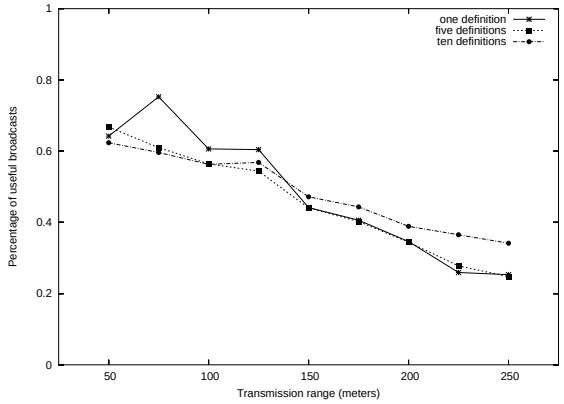


Fig. 22. Percentage of broadcasts that reached new context members for varying network loads

consistency of context message delivery is not greatly affected by the speed. It is likely that, were we to analyze transmission of replies to queries, we would find that routes are less likely to hold up for scenarios with higher node speeds. Such concerns are addressed by the maintenance protocol, and we expect further simulation of the protocol to be beneficial in further understanding the impact of node speed.

A. Discussion

To ensure application-level data consistency, we make assumptions about the atomicity of network topology changes and their propagation through the network for rebuilding the context. Future work will explore ways to relax these assumptions by weakening the required guarantees on context maintenance and context operations. For example, with some knowledge about the system (e.g., radio transmission range, maximum node speed, etc.), a node can predict a "safe distance" for a link [14]. This 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 $20ms$".

Several of our results show that increased network congestion negatively affects our protocol. This results from a commonly known problem called a "broadcast storm" [21]. Several alternative broadcasting mechanisms have been proposed [36] and include using probabilistic methods or knowledge about the environment or neighbor set to determine when to rebroadcast. Integrating these or similar intelligent broadcast mechanisms may increase the resulting consistency and efficiency of context notification. Fig. 20 showed 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.

VIII. Conclusions

This work is rooted in the fundamental observation that existing techniques and protocols supporting communication in mobile ad hoc networks today are not sufficient to provide the capabilities that real-world context-aware applications require. To that end, we precisely define the notion of context-awareness and build the underlying communication constructs necessary to support it. Specifically, we provide a formal characterization (Section IV) of an application's context in a mobile ad hoc network and describe how applications operate over their defined context. We presented several generalizable real world applications (Section V) and explicitly show how they take advantage of the abstraction to build and operate over dynamic contexts. We present a protocol (Section VI) that implements this abstraction in a distributed fashion, and provide initial simulation results (Section VII) demonstrating the feasibility and practicality of context-aware communication in mobile ad hoc networks. To our knowledge, this work is the first targeted to mobile ad hoc networks that focuses on first evaluating the communication requirements of context-aware applications and providing a generalized protocol that directly supports these requirements. The protocol presented here is already in use supporting a context-aware middleware and further extensions and incorporations of it into additional systems are underway.

References

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

[2] R. J. R. Back and K. Sere. Stepwise refinement of parallel algorithms. *Science of Computer Programming*, 13(2–3):133–180, 1990.

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

[4] J. Broch, D. Maltz, D. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proc. of MobiCom*, pages 85–97, 1998.

[5] S. Chen and K. Nahrstedt. Distributed quality-of-service routing in ad-hoc networks. *IEEE Journal on Selected Areas in Communications*, 17(8):2580–2592, 1999.

[6] C. Cheng, R. Riley, and S. Kumar. A loop-free extended Bellman-Ford routing protocol without bouncing effect. In *Proc. of SIGCOMM*, pages 224–236, 1989.

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

[8] C. Chiang and M. Gerla. Routing and multicast in multihop, mobile wireless networks. In *Proc. of the Int'l. Conf. on Universal Personal Communications*, pages 546–551, 1997.

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

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

[11] G. Hackmann, C. Julien, J. Payton, and G.-C. Roman. Supporting generalized context interactions. In *Proc. of the 4^{th} Int'l. Workshop on Software Engineering and Middleware*, 2004.

[12] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The anatomy of a context-aware application. *Mobile Networks*, 8(2/3):187–197, 2002.

[13] J. Hong and J. Landay. An infrastructure approach to context-aware computing. *Human Computer Interaction*, 16, 2001.

[14] Q. Huang, C. Julien, and G.-C. Roman. Relying on safe distance to achieve strong partitionable group membership in ad hoc networks. *IEEE Transactions on Mobile Computing*, 3(2):192–205, 2004.

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

[16] D. Johnson and D. Maltz. Dynamic Source Routing in ad hoc wireless networks. In *Mobile Computing*, volume 353. 1996.

[17] C. Julien and G.-C. Roman. Egocentric context-aware programming in ad hoc mobile environments. In *Proc. of the 10^{th} Int'l. Symp. on the Foundations of Software Engineering*, pages 21–30, November 2002.

[18] C. Julien and G.-C. Roman. Active coordination in ad hoc networks. In *Proc. of the 6^{th} Int'l. Conf. on Coordination Models and Languages*, pages 199–215, February 2005.

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

[20] 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.

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

[22] V. Park. and M. S. Corson. Temporally-ordered routing algorithm (TORA) version 1: functional specification. Internet Draft, August 1998. IETF Mobile Ad Hoc Networking Working Group.

[23] J. Pascoe. Adding generic contextual capabilities to wearable computers. In *Proc. of the 2^{nd} Int'l. Symp. on Wearable Computers*, pages 92–99, 1998.

[24] J. Payton, C. Julien, and G.-C. Roman. Context-sensitive data structures supporting software development in ad hoc networks. In *Proc. of the 3^{rd} Int'l. Workshop on Software Engineering for Large Scale Multi-Agent Systems*, pages 42–48, 2004.

[25] J. Payton, C. Simon, and G.-C. Roman. A query-centered perspective on context-awareness in mobile ad hoc networks. Technical Report WUCSE-05-8, Washington University in Saint Louis, Department of Computer Science and Engineering, 2005.

[26] C. Perkins and P. Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. In *Proc. of SIGCOMM*, pages 234–244, 1994.

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

[28] B. Rhodes. The wearable remembrance agent: A system for augmented memory. In *Proc. of the 1^{st} Int'l. Symp. on Wearable Computers*, pages 123–128, 1997.

[29] M. Roman, C. Hess, R. Cerqueira, A. Ranganat, R. Campbell, and K. Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.

[30] 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.

[31] E. Royer and C.-K. Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications*, pages 46–55, April 1999.

[32] D. Salber, A. Dey, and G. Abowd. The Context Toolkit: Aiding the development of context-enabled applications. In *Proc. of CHI*, pages 434–441, 1999.

[33] P. Verissimo, V. Cahill, A. Casimiro, K. C. A. Friday, and J. Kaiser. CORTEX: Towards supporting autonomous and cooperating sentient entities. In *Proc. of European Wireless*, 2002.

[34] R. Want et al. An overview of the PARCTab ubiquitous computing environment. *IEEE Personal Communications*, 2(6):28–33, 1995.

[35] R. Want, A. Hopper, V. Falco, and J. Gibbons. The Active Badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, 1992.

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

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