Grapevine: Efficient Situational Awareness in Pervasive Computing Environments

Evan Grim, Chien-Liang Fok, and Christine Julien

The University of Texas-Austin, The Center for Advanced Research in Software Engineering, Austin, TX, USA, {evangrim, liangfok, c.julien}@utexas.edu

TR-ARiSE-2011-011

ARiSE
Advanced Research in Software Engineering

Grapevine: Efficient Situational Awareness in Pervasive Computing Environments

Evan Grim, Chien-Liang Fok, and Christine Julien
The Center for Advanced Research in Software Engineering
{evangrim, liangfok, c.julien}@mail.utexas.edu

***Abstract*—Many pervasive computing applications demand expressive *situational awareness*, which entails an entity in the pervasive computing environment learning detailed information about its immediate and surrounding context. Much work over the past decade focused on how to acquire and represent context information. However, this work is largely *egocentric*, focusing on individual entities in the pervasive computing environment sensing their own context. Distributed acquisition of surrounding context information is much more challenging, largely because of the expense of communication among these resource-constrained devices. In this paper, we present Grapevine, a framework for efficiently sharing context information in a localized region of a pervasive computing network, using that information to dynamically form groups defined by their shared situations, and assessing the aggregate context of that group. We provide an implementation of Grapevine and benchmark its performance in a live pervasive computing network deployment.**

***Keywords*- context-awareness, network context, Bloom filters, context coordination**

I. Introduction

In pervasive computing, an entity's ability to acquire expressive *situational awareness* requires myriad sensing and communication activities. An entity must assess its local environment, including information about its own capabilities, status, location, and environmental conditions. It must also coordinate with other entities to assess shared local conditions including the state of the network, availability of data and resources, and social network connections. From a user's perspective, the availability of this detailed *context* information is essential for applications to integrate themselves into the environment, adapting their behavior to suit the current situation and take advantage of currently available resources.

There are many instances in which knowledge about the shared situation of a group is invaluable, particularly in emerging social scenarios. Consider an opportunistic network of mobile devices in a public park where collective context information identifies a group of people interested in a pick-up game of football and having similar skill levels that could support *ad hoc* team formation. A device on an automobile may generate an individualized group containing nearby automobiles that can collide with it. Knowledge about connection qualities in a neighborhood of a dynamic mobile network can support selecting the best routing protocol for a region of the network [19]. While these groups are all very different in structure and purpose, all are defined by their context, and the group itself exhibits an aggregate context that can in turn affect its component entities (and perhaps the definition of the group itself). For instance, a group of football players may exhibit a context of its aggregate skill level or degree of fatigue; a group of nearby vehicles may generate an overall context of a dangerous condition; and a group of devices in a dynamic mobile network may exhibit an aggregate context of the minimum communication quality between any pair of nodes in the group.

Much previous work has focused on how to sense, aggregate, and share context; we review the state of the art in Section II. Our motivation differs in that we explicitly investigate the interplay between *context* and *groups*. Specifically, we tackle the practicalities surrounding two interrelated challenges in dynamic pervasive computing environments: (1) how to expressively identify *groups* of entities based on predicates over their contexts and (2) how to define, assess, and share *context* of individual entities and groups of entities. These challenges are intertwined because the groups may be defined by members' contexts, which must be continuously assessed in dynamic environments. To make the problem tractable, we untangle these two challenges into three concrete objectives. First, we define a mechanism and architecture for efficiently and effectively sharing context in a local region of a network. We then formalize the notion of a *group* and extend our architecture to compute, monitor and maintain group membership information. Finally, we demonstrate how our mechanism and architecture seamlessly support the assessment and sharing of the context of a group of entities.

Previously, we defined a succinct representation of context and showed that using such a representation has the theoretical potential to significantly reduce communication overhead associated with sharing context in a local neighborhood [18]. We revisit this approach in Section III. We then realize this model in Section IV by designing and implementing Grapevine, an architecture for context summarization and dissemination. We benchmark our approach in both simulation and a testbed, showing both the communication performance gains and the potential use of Grapevine for shared situational awareness in real networks. Our previous work posited on the use of this approach for forming groups based on their shared situation; in this paper, we demonstrate the use of Grapevine to achieve this and evaluate its ability to form groups based on their shared context information *and* to assess the shared situation of that group in a real pervasive computing network; these evaluations appear in Section VI.

II. Related Work

Sensing and understanding the environment is a necessity in pervasive computing, and many solutions acquire, aggregate, and assess pervasive contexts [16], [17], [31], including approaches that assess network context [2], [8]. Sensing context is often expensive; joint sensing approaches reduce the costs of determining context in a distributed fashion, for example, by relying on eavesdropping in broadcast environments [2], [25], using application-driven sensing [35], or reporting only *changes* [20]. Our work is enabled by these approaches; the next logical step is to equip devices to share context and to compute shared context properties that extend beyond existing approaches' egocentric scopes.

Existing work on context aggregation automates acquiring high level context from low level data and applying context to adapt application behavior. Context aggregated from a personal area network has been used to automatically add semantic tags to documents on a personal device [21]. The Solar system defines context services for aggregating streaming context data in peer-to-peer pervasive computing networks [7]. Similarly, a programming framework based on sentient objects simplifies context fusion and interpretation [3]. Researchers have also looked at breaking complex aggregation tasks into numerous simpler tasks that can be distributed in a pervasive computing network to reduce centralization [30]. In these cases, the nature of context and the means of aggregating and fusing it are fixed and known *a priori*, and the focus is on how *multimodal* sensing can compensate for decreased quality of low-level sensing. Recent work has explicitly codified *quality of context* and directs the sensing process to trade sensing quality for cost [10], [29]. In-network data aggregation in sensor networks similarly reduces the overhead of shipping redundant information to a single sink [12], [22], [23]

To compute groups and their context, approaches are generally limited to statically defined groups or groups based on co-location [13]. The Team Analysis and Adaptation Framework (TAAF) [11] observes the behavior of a distributed team and adapts supporting services and team coordination. TAAF assumes the team is already assembled, and sensing the team's context is centralized. Context-Aware Ephemeral Groups (CAEG) [33] explore a more abstract definition of groups based on social connections and are used to maintain persistent state among the users, though CAEG focuses on interface design instead of efficient context representation.

Work in sensor networks has defined groups based on more expressive properties. Logical Neighborhoods [24] define connected regions that satisfy predicates over static and dynamic values. Abstract Regions [36] define (orthogonal) regions of the network based on physical or network properties. Query Domains [28] uses a *proximity function* to define a group that responds to an application-specified query. These approaches are functionally limited as they target very resource-poor sensor networks, and they only focus on identifying the group and not on a generic mechanism to determine (and share) the group's context. Hood [37] enables a group of nodes to share all of their context attributes, but group definitions are constrained to be exactly those nodes within one hop.

These existing approaches either determine dynamic group membership or assess the shared state of a statically-defined group. Our work bridges the two; in assembling a dynamic group, we assess their shared state (i.e., their *group context*), to generate a shared view of that context for the group members; we also use the group's context to impact its membership. We address these goals in a dynamic and distributed manner conforming to the resource constraints of pervasive computing networks, minimizing communication overhead associated with both group formation and context dissemination.

III. Summarizing Context and Defining Groups

Succinctly representing both individual and group context is crucial because we intend to share context using throughput limited wireless links and between devices with limited amounts of energy. Sharing detailed context can add significant overhead, and the ability to communicate such information has remained too expensive for practical deployments. We have defined a space efficient context representation that can be shared with limited overhead [18]; in this work, we piggyback this context on application packets already being transmitted.

Summary Data Structures. Our context summary is based on a derivative of a Bloom filter [4], which succinctly represents set membership using a bit array, m, and k hash functions. To add an element to a Bloom filter, we use the k hash functions to get k positions in m and set each position to 1. To test whether an element e is in the set, we check whether the positions associated with e's k hash values are 1. If any position is *not* 1, e is not in the set. Otherwise, e is in the set *with high probability*. False positives occur if insertions of other elements happen to set all k positions associated with e's hash values. Bloom filters trade size for false positive rate; a false positive rate of 1% requires 9.6 bits per element.

A *Bloomier filter* [6] associates a value with each element. The intuitive construction consists of cascades of Bloom filters, one for each bit of a value. Consider the case where each value is either 0 or 1; the Bloomier filter consists of only one cascade. Within the Bloomier filter, a Bloom filter A_0 contains all of the keys that map to 0; A_1 contains all of the keys that map to 1. If the value associated with e is 0, it is inserted in A_0; if the value is 1, it is inserted in A_1. A query for the value of e', first checks whether e' is in A_0 and in A_1. There are four possible results. (1) If e' is in neither A_0 or A_1, e' has not been associated with a value in the Bloomier filter. (2) If e' is in A_0 but not A_1, e' was inserted in the Bloomier filter with high likelihood, and when it was inserted, it was associated the value 0. It is possible (albeit unlikely) that e' was not inserted, but it was not inserted with value 1. (3) Similarly, if e' is in A_1 but not in A_0, the query returns 1. (4) If e' is in both A_0 and A_1, one of these is a false positive. To handle this case, we add another pair of Bloom filters to resolve false positives in the first pair. B_0 contains keys that map to 0 and generated false positives in A_1. B_1 contains keys that map to 1 and generated false positives in

A_0. The problem is the same as before but with a smaller key set whose size depends on the false positive rates of A_0 and A_1. The Bloomier filter continues to add levels until the key set becomes small enough to store in a map.

To handle longer values, a Bloomier filter uses a cascade for each bit, i.e., when the range of values is $\{0,1\}^r$, it creates r of the above Bloom filter cascades. Fig. 1 depicts a Bloomier filter, showing the first two levels of each cascade. The Bloomier filter has space complexity of $O(rn)$, where n is the number of elements and r is the number of bits needed to represent an element's value. This is in comparison to $O(rn \log N)$ for enumerating the value of every element in the set (where N is the number of possible types) [6] or the $O(n(r+l))$ complexity of enumerating pairs of types' labels and values (where l is the length of a type's label). The Bloomier filter has a false positive rate $\epsilon \propto 2^{-r}$. Although this intuitive construction is theoretically optimal, we employ the algebraic techniques from [6] to optimize our implementation; different constructions make varying tradeoffs in space and time complexity [5], [26].

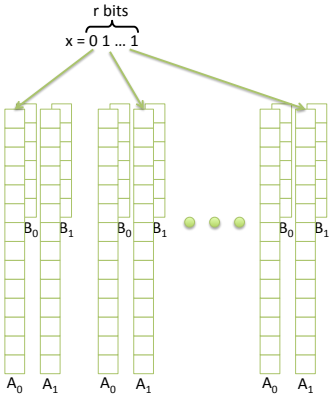

Fig. 1. Structure of a Bloomier Filter

Our context summary is a straightforward Bloomier filter. We assume every entity has a unique identifier, $node_i$. We also assume that the universe of context types $\mathcal{C}$ is well-known and shared *a priori*, i.e., all entities share a *language* of context, which may be represented, for example, as a context ontology [27], [32], [34]. Let $|\mathcal{C}| = N$. An entity senses a (small) subset of the possible context types. Let $\mathcal{C}_{node_i} \subseteq \mathcal{C}$ be the types that $node_i$ senses. $|\mathcal{C}_{node_i}| = n$; $n \ll N$. Context sensing is a function $context_{node_i} : \mathcal{C}_{node_i} \rightarrow \{0,1\}^r$ where r is the maximum number of bits needed to represent any type in $\mathcal{C}$. Each context item $c \in \mathcal{C}_{node_i}$ has a value $context_{node_i}(c)$ in the range $[0\,..\,2^r]$. For any $\overline{c} \notin \mathcal{C}_{node_i}$, $context_{node_i}(\overline{c}) = \perp$. That is, if $node_i$ does not sense $\overline{c}$, its value is null.

When our context summary is queried with c', if $c' \in \mathcal{C}_{node_i}$, the summary returns $context_{node_i}(c')$. If $\overline{c'} \notin \mathcal{C}_{node_i}$, the summary returns $\perp$ with high probability. Every summary contains the key cs_id mapped to the value $node\_id$; when queried with cs_id a summary will, without fail, return the id of the entity whose context is summarized.

Given a set of context summaries, an aggregate captures the *context of the group*, or the group members' shared situation, by summarizing the values included in the individual summaries. Assuming complete knowledge of all of the context summaries of a group G, we define the group's context:

$$\textsc{GroupContext}_G = \{(l, v_{agg}) : v_{agg} = f_{agg,l}(\{\mathrm{CS}_{n_i}.l \ : \ n_i \in G\}),\ \forall l \in \bigcup_{n_i \in G} \mathrm{CS}_{n_i}\}$$

where CS_{n_i} is the context summary of entity n_i, l is a context label, $\mathrm{CS}_{n_i}.l$ is the value stored for label l in CS_{n_i}, and $f_{agg,l}$ is an aggregation function associated with the context type l. Aggregation functions can be standard functions like average, maximum, or minimum, or they may be defined as part of the context ontology. Different types of context have different forms of aggregation, e.g., to aggregate location values, $f_{agg,loc}$ may construct a bounding box. A group context can be represented by a context summary, where cs_id is a unique group identifier instead of the $node\_id$. To ensure consistency, group context summaries always contain a list of the $node_id$s of the nodes who are members of the group (i.e., of the nodes whose context is represented by the aggregate). Further details of constructing group context summaries are omitted for brevity; details can be found in [18].

Defining Groups. Previously, we identified four ways to determine which entities constitute a group; these four types capture many of the coordination needs of pervasive computing applications. A group, G's, membership is defined by a function f_G that constrains individual members' contexts'.

In *Labeled Groups*, all group members share a label *a priori* (e.g., a group of students enrolled in the same course). Formally, an entity n's labeled group G is the set of entities such that $\forall n' \in G : f_G(n')$, where $f_G(n')$ requires the label to be in the context summary of n'.

Asymmetric Groups are defined by a single entity representative of its perspective (e.g., all nearby vehicles with the potential to collide with my vehicle in the next 10 seconds). For asymmetric groups, f_G evaluates the pairwise constraint. Formally, the membership of n's asymmetric group is a set G_n such that: $\forall n' \in G_n : f_G(n, n')$.

In *Symmetric Groups*, all members have the same *view* of the group, defined by constraints that reference the context (e.g., a group in which communication between any pair of members will take less than 100ms). For symmetric groups, f_G is a shared (global) function that evaluates pairs of contexts. Formally, a symmetric group is a set G such that: $\forall n, n' \in G : f_G(n, n')$. The difference between symmetric and asymmetric groups is the perspective, represented by the quantification of n; a symmetric group is a shared (ideally, consistent) view; an asymmetric group is the perspective of a single entity.

In *Context-Defined Groups*, a group together must satisfy some requirement (e.g., a set of mutually reachable devices that provide a set of needed application capabilities). A context-defined group G is constrained by a function f_G evaluated over a set of contexts: G is valid if and only if $f_G(G)$. Aset of entities constitutes a context-defined group if, as a group, they jointly satisfy some requirement. There is no requirement that G is minimal, i.e., that removing any member of G will cause the group to no longer be valid.

A labeled group is a special case of a symmetric group; we separate the labeled group because of its common occurrence and simplicity in specification and implementation. Individual entities compute group membership and their contexts in a distributed fashion. These computed group contexts are summarized using the same constructs described above, and

they are shared in the network in the same way as individual context summaries. The only exception is the asymmetric group summary, which is defined from the perspective of a single entity. The context of such a group is meaningful only to this entity so they are not shared.

We next describe Grapevine's architecture, implementation, and use to collect and share context in distributed pervasive computing environments and to use context summaries to form expressive groups. We demonstrate these contributions through a set of evaluations in a live pervasive computing network and benchmark the ability of our approach to practically summarize highly detailed context information.

IV. Grapevine: An Architecture for Context Sharing

We reduced the size of entities' and groups' contexts to reduce the overhead of sharing them. Grapevine's architecture, shown in Fig. 2, piggybacks these summaries on existing packets via the *Context Handler* and the *Context Shim*.

The *Context Handler* maintains and processes all context information generated by its host or received from the network. It computes the Bloomier Filter context summaries on demand, and provides the application updated local and remote information. It also uses received summaries to compute groups. The application notifies the *Context Handler* of relevant group definitions, and the *Context Handler* dynamically computes group membership based on the context data it receives.

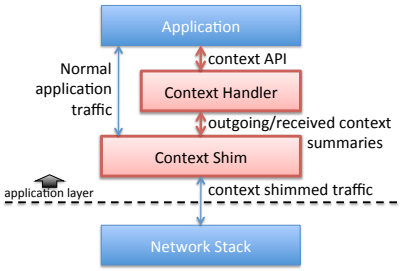

Fig. 2. Grapevine Architecture

The *Context Shim* is an *interceptor* that appends context information to every outgoing packet; we could also attach context summaries to only some subset of packets. Upon receiving a packet from the application, the *Context Shim* retrieves an up-to-date Bloomier filter context summary from the *Context Handler*, attaches it to the packet, and passes the packet down the network stack. On the receiving end, the *Context Shim* does the reverse, removing context summaries from incoming packets, sending context information to the *Context Handler*, and passing the remainder to the application.

An important design concern in Grapevine is a parameter τ, which is the number of network hops context information may propagate. Every entity in the network piggybacks its own context on its outgoing packets. It can also piggyback context received from other entities. τ designates how widely context information is disseminated: if $\tau = 1$ then an entity's context is shared only with its directly connected neighbors, i.e., nodes do not forward received context information at all. If $\tau > 1$, context is shared more widely. In our current implementation, τ is a fixed setting; dynamically adjusting τ based on application demands and network conditions is an area of future work. The obvious tradeoff τ addresses is the cost of disseminating more context (in terms of the amount of data piggybacked on packets) versus the amount of shared information. However, because nodes also share group context summaries, neighbors can learn about each other in aggregate instead of individually; this is a significant added benefit of our joint approach to sharing context and forming groups. Many (but not all) group contexts can be learned this way. Consider a group that requires all members to be within a given physical distance of each other. A *context* of this group may represent the bounding box of the members' physical locations; a new node can quickly look at its own context and this group context and tell whether it can be added to the group.

Consider the example in Fig. 3, where the goal is to discover the (labeled) group defined by the shaded nodes. When $\tau = 1$ (i.e., every node shares context only with its directly connected neighbors) and the network provides enough packet traffic to disseminate context, the group of shaded nodes in Fig. 3(a) can be discovered by all of its members by incrementally growing the membership represented in the shared group summaries. While t will not get a context summary for w directly, it can learn about the presence of w in the shared labeled group because u shares a context summary of the labeled group and the group context summary contains the identities of all its members. The group of six members in Fig. 3(b) cannot be discovered when $\tau = 1$; instead x and y will form a group of two, while s, t, u, and v will form a group of four. When $\tau > 1$, both networks will eventually have only a single large group because, in Fig. 3(b), w forwards the context summaries for u and x, allowing them to discover each other (and their groups).

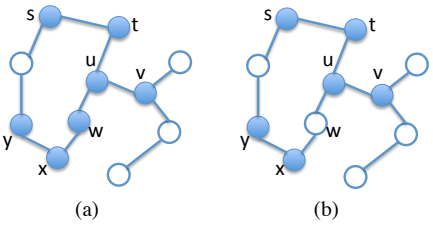

Fig. 3. Impact of τ on Context Dissemination

More generally, since we do not distribute summaries beyond τ, an entity may not have network-wide information about group membership. We refer to the *partition* of a group that contains entity n as $G[n]$, which is what we expect n to be able to discover (e.g., when $\tau = 1$ in Fig. 3(b), $G[\mathsf{x}] = G[\mathsf{y}] = \{\mathsf{x}, \mathsf{y}\}$). We define $(\mathcal{K}^\tau)^+$ to be the transitive closure of connectivity under τ and add a restriction to our group definitions that all group members must be related to n under $(\mathcal{K}^\tau)^+$. For example, the refined formal definition of an entity's partition of a labeled group $G[n]$ is the set such that: $\forall n' \in G[n] : (n, n') \in (\mathcal{K}^\tau)^+ \wedge f_G(n')$. The importance is subtle and stems from the fact that sharing group context summaries enables groups to be built incrementally by combining a (partial) group summary and individual context summaries [18]. Coupled with our succinct representation of context, this is a significant benefit of Grapevine: *an entity can construct a group and compute its context without directly collecting the context of each individual group member.*

V. Grapevine: The Implementation

We implemented Grapevine in Java, in the application-layer, which has the advantages of ease of implementation, ease of use and maintenance, and disjointness from the complexities of the network stack in the operating system. On the other hand, we can only attach context information on outgoing *application* packets. Integrating Grapevine with the operating system's network layer would decrease portability, but it would give us finer grained control over how and when context is disseminated; understanding these detailed tradeoffs is an important piece of future work. Fig. 4 shows the information flow within Grapevine's *Context Handler* and *Context Shim*.

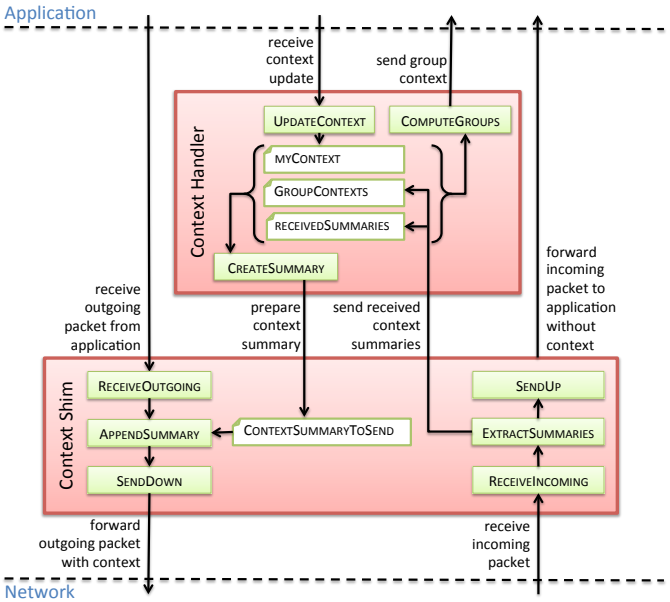

Fig. 4. Information Flow in the Context Shim and Context Handler

A. Context Handler

All context information flows through the *Context Handler*, whose interfaces to the application and the *Context Shim* are shown in Fig. 5. We do not introduce new metrics for context nor any new approaches for assessing individuals' contexts. We assume these tasks are handled by an application-level thread that shares collected context with the *Context Handler* using `updateLocalContext(ContextMap context)`; the *Context Handler* stores this information as MyContext, a simple map of context labels to values.

The *Context Handler* stores received (Bloomier filter) summaries of other entities' contexts as ReceivedSummaries. The API allows the application to access the summaries received from other connected entities all at once via `getReceivedSummaries()`, one summary at a time using the entity's identifier via `get(ID id)`, or one field at a time via `get(ID id, String label)` or to register via the observer pattern to be notified of new context information. The application also sets the value of τ, which determines which received summaries are included in outgoing packets.

The final interactions between the application and the *Context Handler* surround groups. The application declares groups,

```
public class ContextHandler {
  // upper interface (application)
  public void updateLocalContext(ContextMap context){...}

  public ContextSummary get(ID id){...}
  public Value get(ID id, String label){...}
  public List<ContextMap> getReceivedSummaries(){...}
  public void
    addReceiveSummariesObserver(Observer observer){...}

  public void setTau(int newTau){...}

  public addGroupDefinition(GroupDefinition group){...}
  public addGroupChangeObserver(Observer observer){...}

  // lower interface (context shim)
  public void
    handleIncomingSummaries(Collection<BFContextSummary>
        summaries){...}
  public ArrayList<BFContextSummary>
    getSummariesToSend(){...}
}
```

Fig. 5. *Context Handler* API

and the *Context Handler* monitors stored context information and assesses group membership and group contexts. Group summaries are stored in the *Context Handler* as simple maps of labels to values (GroupContexts in Fig. 4). To define groups, applications define a function in a subclass of the `GroupDefinition` class that is evaluated over one or more summaries. The format for this function depends on the type of group; we provide more details below. Applications can also register to be notified of changes to groups or their contexts.

The *Context Handler* has two simple interactions with the *Context Shim*. First, when the *Context Shim* requests up-to-date context information via `getSummariesToSend()`, the *Context Handler* creates a Bloomier filter summary that represents this entity's context (constructed from MyContext) and the contexts of this entity's groups (constructed from GroupContexts). The *Context Handler* also appends any summaries in ReceivedSummaries whose hop count has not yet reached τ. This summary information is returned to the *Context Shim*. Second, upon receiving incoming context information the *Context Shim* uses `handleIncomingSummaries` to alert the *Context Handler* that updated context information is available.

B. Context Shim

The *Context Shim* piggybacks context on outgoing packets and extracts it from incoming packets. When a packet is about to be sent, the *Context Shim* retrieves up-to-date context from the *Context Handler* and adds it to the packet. It reverses the process for received packets. We constructed a drop-in replacement for Java's datagram socket that automatically injects and extracts context information to and from datagram packets that pass through. As previously described, when the shim can be more explicitly integrated with the network stack in the operating system, this interception of packets will be migrated into the operating system's networking components.

C. Computing Groups

In the *Context Handler*, ComputeGroups monitors the reception of new context information and updates the mem-

bership and contexts of the groups defined by the application. The specific mechanisms depend on the type of group; the application provides a subclass of `GroupDefinition` that defines the requirements of the group (as a `ComputeGroup` method). This method is a codified realization of the formal group constraint functions f_G, defined in Section III.

A `LabeledGroup`'s `ComputeGroup` method requires that the label associated with G appears in all members' context summaries[1]; any entity whose context summary has the label is considered a group member. When an application provides a definition of a labeled group, the *Context Handler* inserts the label into the local entity's context (i.e., MYCONTEXT).

An `AsymmetricGroup`'s `ComputeGroup` method evaluates this entity's context (MYCONTEXT) and a received summary. If the two satisfy the specified constraint, then the entity represented by the summary is part of the group. Asymmetric group contexts are *not* appended to outgoing packets.

Like asymmetric groups, the `SymmetricGroup`'s `ComputeGroup` method is evaluated over pairs of context summaries, but all pairs in the group must satisfy the constraint (not just pairs that include MYCONTEXT). Symmetry refers to the fact that the function is shared and symmetric, and if entity a is in b's symmetric group, then entity b is in a's symmetric group with the same definition.

The `ContextDefinedGroup`'s `ComputeGroup` method is evaluated over sets of summaries. When a new context summary is received, we generate all permutations of RECEIVEDSUMMARIES, looking for the largest subset that results in a valid group when combined with the local entity's context. In incrementally building context-defined groups, we combine received context-defined group summaries with MYCONTEXT and RECEIVEDSUMMARIES. Different heuristics exist for choosing which valid group is "best," including application-defined metrics for the quality of a group. Evaluating the merits of these alternatives is an area for future research.

Within the definition of a group, the application also specifies what constitutes the group's summary information, both in terms of the context's fields and how the values for those fields aggregate the constituent individual context summaries.

VI. EVALUATION

In this section we evaluate Grapevine to explore how its ability to reduce the overhead of sharing context information can improve the quality of context and group knowledge in real distributed pervasive computing networks.

Our summary technique theoretically generates summaries of size $O(rn)$ bits, where n is the number of context elements and r is the maximum size (in bits) of a context element. The obvious alternative, which is used in existing context-aware systems [16], is to enumerate a pair $\langle$context label, context value$\rangle$ for each context value sensed. Enumerating the pairs has theoretical size complexity of $O(n(r+l))$, where l is the length (in bits) of a context label. One goal of our evaluation is to determine how these approaches compare *practically* and the impact that the reduction of context information has on communication in both real and simulated networks.

We are also interested in how quickly context and group information can be disseminated. Context information often measures a dynamic situation, meaning it can quickly become stale. Further, in assembling groups based on context, the context information should be able to spread quickly enough to discover groups and assess their shared situation before network dynamics cause changes to the groups. As previously described, τ has a significant potential impact on the spread of context, and therefore on nodes' abilities to compute groups in a distributed manner. At the same time, increasing τ also increases the amount of data transmitted, which has a real and significant impact in actual network deployments.

We are also interested in how sharing context information impacts existing application traffic. By increasing the amount of data sent across the wireless links, Grapevine may prevent application traffic from being delivered; we measure this impact in real pervasive computing networks.

We used descriptive Java `String` objects as context labels (e.g., "current system time" for a timestamp or "gps longitude" for the device's longitude) and Java `Integer` objects for context values (i.e., $r = 32$). Grapevine's architecture is independent of these specific choices. For application traffic, we use periodic "beacons" sent by an application on each node; clearly the rate of this periodic beacon influences the rate at which our nodes learn about shared situations.

Our evaluations are divided into two categories. First, we describe benchmarking experiments for our implementation in simulation. Then we use a real network deployment to evaluate the performance of our architecture on real hardware, with real network links, and amidst real application traffic.

A. Simulation Results

We created a simple simulated network of 100 nodes in an equidistant 10 by 10 grid. A node could communicate only with its nearest neighbor nodes in each cardinal direction. We did not carefully simulate real-world network effects in our simulation's network stack; this evaluation focuses instead on the relative sizes and ideal propagations of the different context dissemination options. Our second set of evaluations incorporates a real network.

First, we demonstrate the savings (in terms of number of bytes appended to outgoing packets) of our approach in comparison with the labeled context approach used in existing systems. We vary the number of context attributes a

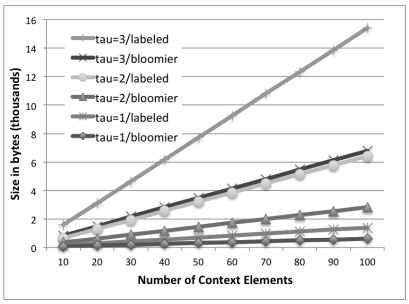

Fig. 6. Size of context information piggybacked for labeled and bloomier context representations

[1]The details of representing an entity's labeled groups within its summary is omitted here for brevity; see [18] for details.

node senses (i.e., n) from 0 to 100, and we vary the number of hops over which this context is forwarded (i.e., τ) from 1 to 3. As Fig. 6 shows, Bloomier-based context distribution has significant space savings—an average of 46% in our results.

This reduction in size is especially important in real networks when this context is appended to real packets. Any additional data transmitted can interfere with application-level traffic to the point of preventing the application from functioning, which has traditionally prevented sharing of context in pervasive computing environments.

We also evaluated Grapevine's ability to use shared context to form groups. Again, since we do not carefully simulate wireless link behavior, this evaluation provides a theoretical upper limit on our ability to form the groups described in this paper. We compared the groups that Grapevine discovered to those that *should* have been discovered (given our omniscient view as the network controllers), and plotted how the group awareness of the collective nodes progressed over time. We again simulated 100 nodes, this time assigning a third of them to a labeled group. As Fig. 7 shows, group membership awareness reached 50% after only three packet beacon intervals. This means that on average each node knew half the group members (network wide) after receiving just three packets from each of its direct neighbors. Group awareness increases to 80% after nine packet intervals and full group awareness is achieved for every node in the network after 23 intervals[2].

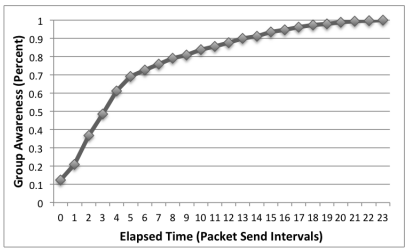

Fig. 7. Propagation of labeled group membership

B. Evaluation in the Pharos Testbed

Simulation is useful in benchmarking Grapevine, especially in comparison to the other context representation options. However it does not provide an in-depth sense of how Grapevine will perform in real networks. Indeed, an exploration of both the negative impacts on application-level communication and positive impacts on applications' perspectives of their environments is necessary. This is especially important with Grapevine, which, by adding context information to packets, can cause packet fragmentation or otherwise interfere with application-level traffic. Therefore, we performed experiments in a real pervasive computing network deployment in two pieces: first we use a static scenario to quantify the impact of our context summaries on application traffic. We then we use our approach to support a real application, multi-robot patrol, demonstrating the potential gains of using Grapevine. Our deployment environment is a testbed for mobile and pervasive computing systems; we integrate Grapevine with the testbed both for evaluating it and in support of other applications on the testbed that need context and group information.

We use the Pharos testbed [14], which consists of highly modular Proteus mobile robots. For our evaluation, we used the node configuration shown in Fig. 8. It consists of a modified Traxxas Stampede mobile chassis and a plane containing computational elements, a Garmin eTrex GPS receiver, a CMPS03 digital compass, a CMUCam2 vision sensor, and an IR range finder. The node's main computational components are a general-purpose x86 computer and a microcontroller. The x86 is a VIA EPIA Nano-ITX motherboard that contains a 32-bit VIA C7 CPU running at 1GHz, 1GB of DDR2 RAM, a 16GB compact flash drive, and a CM9-GP IEEE 802.11g WiFi mini-PCI module based on the Atheros AR5213A chipset. It runs Ubuntu Linux 11.04 server and Player 3.02 [15]. Custom Player drivers provide high-level programming abstractions for node movement and sensing; it is through these interfaces that we access much of our instantaneous context information, either through the GPS and compass sensors, or other special purpose sensors like ultrasonic or infrared range finders, accelerometers, and gyros, or sensors of the ambient environment, e.g., via TelosB, mica2, or SunSpot devices. The WiFi interface is configured to form a wireless ad hoc network among the Proteus nodes in an experiment.

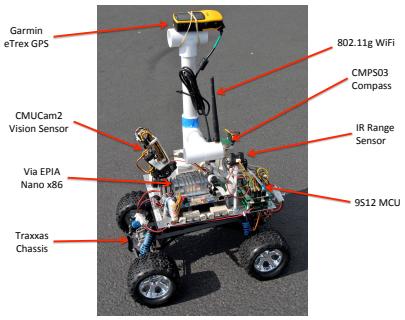

Fig. 8. The Proteus Node

We first report experiments with a static topology. This allows us to better examine the impact of real network links on context dissemination given different context sizes and the impact of the context summaries on other traffic. It also allows us to more carefully manipulate the network topology, ensuring that the width of the network is always more than one hop. We arranged ten Proteus nodes in a two by five grid as shown in Figure 9. The distances result in a multi-hop network in which context must be shared through intermediate nodes.

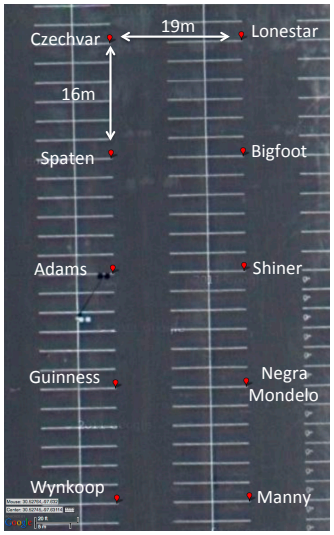

Fig. 9. The static network's layout.

In these experiments, Grapevine shimmed context summaries onto UDP application broadcast packets sent by each robot every 300ms. We first measured the practical impact of

[2]Altering τ for this experiment had negligible impact since nodes forward aggregated membership information to each of their neighbors

adding context information to these packets in terms of the number of lost packets when we appended varying amounts of context. To ensure that context information is widely disseminated, we set τ to three hops.

We measured the percentage of packets dropped, normalized to when no context information is sent, for both Bloomier Filter context summaries and labeled context summaries. As Fig. 10 shows, increases in packet size naturally lead to reduced network reliability. However, in comparison to sending labeled context information, the current state of the art, our more succinct context distribution mechanism increases packet reception by 25% on average.

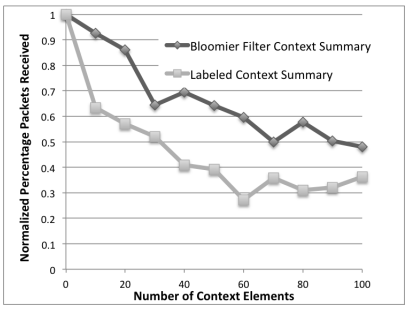

Fig. 10. Normalized percentage of dropped packets for both Bloomier Filter context summaries and labeled context summaries

Pushing the implementation even further, Fig. 11 shows that when extreme amounts of context are added to application packets, there is a significant negative impact on the existing application traffic. In these cases, we added very large amounts of context to each outgoing packet (up to 3200 pieces of context information), and forwarded all context information across three network hops. This amount of context information is so large that it causes increased packet fragmentation, resulting in higher likelihood of unsuccessful deliveries. Section VII discusses some potential enhancements to handle these extreme situations.

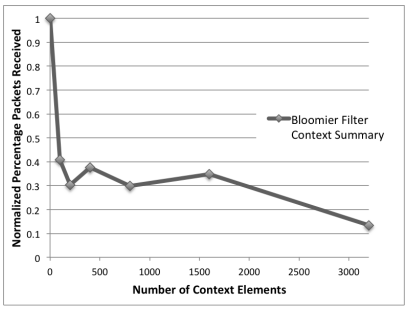

Fig. 11. Normalized percentage of dropped packets for very large context summaries

Finally, we evaluate Grapevine with a real world application previously implemented on the Pharos Testbed: multi-robot patrol [1]. In this application, a team of mobile robots must patrol a route specified by a sequence of waypoints. To ensure each waypoint is fairly visited with minimal variation in idle time (the time between robot visits) the robots must coordinate to achieve equal spacing along the route [9]. This requires each robot to be aware of its teammates' contexts, specifically their positions and movements. Collectively, the team forms a group context that captures whether they are patrolling in an ideal manner in terms of minimizing waypoint idle times.

We modified the existing multi-robot patrol application to use Grapevine and allow it to support context-sensitive coordination and creation of patrol groups. In the original application, sharing relevant context information was accomplished by software specifically written for this particular application. We simplified this implementation by replacing this custom-built software with Grapevine.

To evaluate Grapevine's impact on the multi-robot patrol application, we tested it in a patrol route in our laboratory, as shown in Fig. 12. One lap of the patrol route is 17.5m. The route is demarcated by a white line that Proteus robots can follow using a CMUCam2 vision sensor. Waypoints are marked using overhead markers that the robots can detect using short-range IR range sensors. The markers also assist in robot localization since they are placed at equal distances along the route. Along with an odometer sensor, this enables the robots to localize themselves with sub-centimeter accuracy.

We configured four robots to patrol the route five times at 0.5m/s. Our first test, with no robot coordination, is a baseline in which each robot moves independently without communicating with its teammates. Our second test evaluates the original loosely coordinated mechanism [1], in which the robots periodically broadcast their location and synchronize their movements at every waypoint; coordination is "loose" because robots ignore teammates that are not within wireless range. Finally, we evaluated the application using Grapevine to dissementate context; here the application no longer needs to explicitly transmit and receive context information. Instead it simply tells Grapevine which context elements and groups to track and registers as an observer for context information.

Fig. 12. The multi-robot patrol testbed.

Table I shows the results[3], which demonstrate that Grapevine can replace the original custom-built context sharing mechanism in the multi-robot patrol application without adversely affecting the application's operation. Grapevine maintained the waypoint idle times of the loosely coordinated approach and decreased the amount of time robots spent waiting for context. Furthermore, using Grapevine has several additional benefits. First, the complexity of the application is decreased by offloading the work to an external library tailored to context handling. Second, the application gains Grapevine's inexpensive multi-hop dissemination capabilities, which allow it to operate in networks that are not fully or reliably connected. Lastly, one interesting discovery was that, when using Grapevine, the multi-robot patrol application became more resilient to robots getting physically stuck. In the original implementation, if a robot

	Idle Time	Wait Time
Uncoordinated	10.4 $\pm$ 0.6	0.00 $\pm$ 0.0
Loosely	13.7 $\pm$ 0.4	1.59 $\pm$ 0.2
Bloomier	13.7 $\pm$ 0.5	1.38 $\pm$ 0.1

TABLE I
WAYPOINT IDLE AND ROBOT WAIT TIMES.

[3]See http://bit.ly/grapevine_mrp for details

got stuck, other nodes would also get stuck waiting for this robot. Using Grapevine, the neighboring robots eventually assume that this stuck robot left the group and continue. This is because Grapevine does not repeatedly provide context information to the application if it does not change. The lack of context information updates from the stuck robot enables the neighboring nodes to avoid deadlock.

VII. Conclusions and Future Work

Pervasive computing applications demand greater expressiveness in gathering and sharing local and distributed context. We presented Grapevine, a practical framework with strong theoretical underpinnings that supports sharing the context of individuals and groups. Grapevine provides a succinct representation of context, efficiently shares context in the immediate network, and uses that shared information to create a view of a situation shared among groups of entities. While we have demonstrated the feasibility of our approach in real pervasive computing networks, we have also opened many interesting future directions. Sending large amounts of context, even when efficiently summarized, can tax resource-constrained networks. We see several avenues for further improving our space efficiency using traditional compression and additional Bloomier related encoding. Furthermore, intelligently adapting the piggybacking strategy to context priorities or the current communication environment, and avoiding packet fragmentation can improve Grapevine's performance. In addition, as we apply Grapevine to additional applications we find that context information is often the bulk of the information a pervasive computing application communicates. This hints that refinements that handle context prioritization and network tuning automatically will greatly increase its value and may eventually supplant the need for many other communications altogether. Lastly, we believe Grapevine's built-in support for groups is an important step forward and will focus future research on addressing the scalability of such computations through heuristic group algorithms with reduced costs.

Ultimately, the vision of pervasive computing demands coordination and collaboration among entities in shared physical spaces. This work is foundational in providing practical support for defining and assessing this shared situational awareness and facilitates building such applications with significantly reduced effort and improved performance.

References

[1] N. Agmon, S. Kraus, and G. A. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *ICRA*, 2008.

[2] P. Basu, N. Khan, and T. Little. A mobility based metric for clustering in mobile ad hoc networks. In *ICDCS Wkshps.*, pages 413–418, 2001.

[3] G. Biegel and V. Cahill. A framework for developing mobile, context-aware applications. In *PerCom*, 2004.

[4] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. *Comm. of the ACM*, 13(7), 1970.

[5] D. Charles and K. Chellapilla. Bloomier filters: A second look. In *ESA*, 2008.

[6] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal. The bloomier filter: An efficient data structure for static support lookup tables. In *SIAM*, 2004.

[7] G. Chen, M. Li, and D. Kotz. Data-centric middleware for context-aware pervasive computing. *Pervasive and Mobile Comp.*, 4(2), 2008.

[8] L. Cheng and I. Marsic. Piecewise network awareness service for wireless/mobile pervasive computing. *Mobile Netw. and App.*, 7(4), Aug. 2004.

[9] Y. Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *IAT*, 2004.

[10] A. Deshpande, C. Guestrin, S. Madden, J. Hellersetin, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[11] C. Dorn, H.-L. Truong, and S. Dustdar. Measuring and analyzing emerging properties for autonomic collaboration service adaptation. In *ATC*, 2008.

[12] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi. In-network aggregation techniques for wireless sensor networks: A survey. *IEEE Wireless Comm.*, 14(2), 2007.

[13] A. Ferscha, C. Holzmann, and S. Oppl. Context awareness for group interaction support. In *MobiWac*, 2004.

[14] C. Fok, A. Petz, D. Stovall, N. Paine, C. Julien, and S. Vishwanath. Pharos: A testbed for mobile cyber-physical systems. Technical Report TR-ARiSE-2011-001, Univ. of Texas at Austin, 2011.

[15] B. P. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *ICAR*, 2003.

[16] G. Hackmann, C. Julien, J. Payton, and G.-C. Roman. Supporting generalized context interactions. *Software Eng. and Middleware*, 2005.

[17] J. Hong and J. A. Landay. An infrastructure approach to context-aware computing. *Human Computer Interaction*, 16(2), Dec. 2001.

[18] C. Julien. The context of coordinating groups in dynamic mobile environments. In *Coordination*, 2011.

[19] T. Jun and C. Julien. Automated routing protocol selection in mobile ad hoc networks. In *SAC*, 2007.

[20] S. Kang, J. Lee, H. Jang, H. Lee, Y. Lee, S. Park, T. Park, and J. Song. SeeMon: Scalable and energy-efficient context monitoring framework for sensor-rich mobile environments. In *MobiSys*, 2008.

[21] A. Karypidis and S. Lalis. Automated context aggregation and file annotation for PAN-based computing. *Personal and Ubiquitous Comp.*, 11, 2007.

[22] L. Krishnamachari, D. Estrin, and S. Wicker. The impact of data aggregation in wireless sensor networks. In *ICDCS Wkshps.*, 2002.

[23] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny AGgregation service for ad-hoc sensor networks. In *OSDI*, 2002.

[24] L. Mottola and G. P. Picco. Programming wireless sensor networks with logical neighborhoods. In *InterSense*, 2006.

[25] T. Petz, T. Jun, N. Roy, C.-L. Fok, and C. Julien. Passive network-awareness for dynamic resource-constrained networks. In *DAIS*, 2011.

[26] E. Porat. An optimal bloom filter replacement based on matrix solving. In *CSR*, 2009.

[27] D. Preuveneers, J. V. den Bergh, D. Wagelaar, A. Georges, P. Rigole, R. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, and K. D. Bosschere. Towards an extensible context ontology for ambient intelligence. In *EUSAI*, 2004.

[28] V. Rajamani, S. Kabadayi, and C. Julien. An interrelational grouping abstraction for heterogeneous sensors. *ACM Trans. on Sensor Netw.*, 5(3), 2009.

[29] N. Roy, A. Misra, C. Julien, S. Das, and J. Biswas. An energy-efficient quality-adaptive framework for multi-modal sensor context recognition. In *PerCom*, 2011.

[30] H. Ryu, I. Park, S. J. Hyun, and D. Lee. A task decomposition scheme for context aggregation in personal smart space. *Software Tech. for Embedded and Ubiquitous Sys.*, 2007.

[31] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *CHI*, 1999.

[32] T. Strang, C. Linnhoff-Popien, and K. Frank. CoOL: A context ontology language to enable contextual interoperability. In *DAIS*, 2003.

[33] B. Wang, J. Bodily, and S. Gupta. Supporting persistent social groups in ubiquitous computing environments using context-aware ephemeral group service. In *PerCom*, 2004.

[34] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung. Ontology based context modeling and reasoning using OWL. In *PerCom Wkshps.*, 2004.

[35] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh. A framework of energy efficient mobile sensing for automatic user state recognition. In *MobiSys*, 2009.

[36] M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In *NSDI*, 2004.

[37] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: A neighborhood abstraction for sensor networks. In *MobiSys*, 2004.