The Context of Coordinating Groups in Dynamic Mobile Networks

Christine Julien

The University of Texas at Austin
Austin, Texas, USA
c.julien@mail.utexas.edu

Abstract. Context-awareness in dynamic and unpredictable environments is a well-studied problem, and many approaches handle sensing, understanding, and acting upon context information. Entities in these environments are not in isolation, and oftentimes the manner in which entities coordinate depends on some (implicit) notion of their *shared* context. In this paper, we are motivated by the need to explicitly construct notions of the *context of a group* that can support better coordination within the group. First we identify an efficient representation of context (both of an individual and of a group) that can be shared across wireless connections without incurring a significant communication overhead. Second we provide precise semantics for different types of groups, each with compelling use cases in these dynamic computing environments. Finally, we define and demonstrate protocols for efficiently computing groups and their context in a distributed manner.

1 Introduction

In this paper, we motivate the need to share context information in a local neighborhood that is a subset of a larger dynamic mobile or pervasive computing environment. Existing context-aware approaches tend to be exclusively ego-centric, focusing on how to sense the context of a single entity and how to use that sensed context to create better behavior of that single entity. We posit that the context of a *group* of entities within a dynamic network can be just as important if not more important to the overall behavior of the system.

There are many situations in which knowledge about the context of a group is invaluable, not the least of which are emerging social scenarios. Consider a dynamic and opportunistic network of mobile devices in a public space like a park; a group context that identifies a group of people interested in a pick-up game of football and having a similar skill level could support ad hoc formation of groups in physical space. A device on an automobile on a highway may want to generate an individualized group containing nearby automobiles that have a potential to collide with it. Knowledge about network protocols used in a neighborhood of a dynamic mobile computing network can support regional protocol selection [15]. While these groups are all fundamentally very different in structure and purpose, in all of these situations, the group is defined by the context of the situation, and the group itself exhibits an aggregated context that can in turn affect the behavior of its component entities (and perhaps the definition of

the group itself). These notions of groups, context, and their intersection are all fundamental pieces of coordinated mobile and pervasive computing applications. Consider again the selection of a network protocol based on network context. Any single node observing just its own context may choose a protocol that ends up incurring more overhead or delay given the network context beyond the node itself; the ability to efficiently acquire information about a wider context of a group of network nodes allows a more informed, *coordinated*, decision.

A well recognized challenge in realizing such a notion of group context is in how to efficiently share context without overburdening an already constrained communication environment (where wireless links create serious bandwidth and energy limitations). While recent approaches have recognized the need for efficient context-awareness, they have largely focused on the acquisition of an ego-centric view of context and not on the view of the context of a group or on sharing information among entities to generate a distributed shared view of the context of that group. We tackle both of these challenges in this paper.

Consider university students' pervasive computing devices. An individual student's context (collected and maintained by his device(s)) can include his courses, his participation in activities, and his individual context (e.g., he is in the library studying, he is part of a particular group project, he is distracted, he is hungry). Existing work has created mechanisms to clearly understand, represent, and use individual context [11, 23]. What is equally interesting is the context of a *group* of students. Such group context may be symmetric (shared and coordinated among the group members) or asymmetric (egocentric and individualized to a particular entity); applications demand both forms to enable entities within groups to understand, support, and adapt coordination behaviors. As an example of a symmetric group context, given a group of students enrolled in the same course, a context measure that represents the students' aggregate understanding of the course material can provide feedback to the instructor; there is only a single view of the group's context, regardless of the perspective of the entity looking at the context. As an example of an asymmetric context, a particular student studying in the library may be interested in which the students at nearby tables have a copy of a the textbook. The latter is defined with respect to the student looking for the textbook; students in different locations in the library will have different views for the same group context definition. In the asymmetric case, the group membership is determined by the relationship between the defining entity's context and other entities' contexts; in the symmetric group, membership is defined by the aggregate relationship among all of the entities' contexts.

Abstracting the pervasive computing entities, their physical environment, and the networks that connect them into measures of *context* and *group context* eases the development of application logic, allowing one to focus on how such context measures can be used to aid entities' activities. We provide an expressive definition of a group and its context, delegate the construction of the group and the computation of its context to a middleware, and provide easy interface from the application logic to the group context infrastructure. In this paper, we demonstrate the feasibility of providing a variety of definitions of groups

and their contexts. Given the resource constrained devices and networks that comprise emerging environments, it is essential that computation and sharing of groups and their contexts is highly space and communication efficient. Our architecture enables future work in expressive coordination among these groups that can easily rely on the group's context for enhancing the coordination activities.

This paper makes three fundamental contributions. In Section 2, we design a space-efficient context summary that can be communicated and shared in multihop wireless networks, evaluate its space efficiency, and give a simple framework for communicating these summaries efficiently. In Sections 3 and 4 we define *groups* in these coordinating environments and create precise formulations of the *context* of groups. We create space- and communication-efficient protocols for distributed determination of *group context*; we demonstrate and evaluate these approaches in Section 5. We argue that supporting expressive coordination among networked entities with complex social, spatial, and temporal relationships requires a formal understanding of groups and their shared context.

2 A Space Efficient Context Summary

Sharing context can add significant overhead; communicating detailed context, which is necessary for determining arbitrary groups and their contexts, has remained too expensive for practical implementations. In this section, we describe an efficient representation of context that can be shared with limited overhead.

Summary Data Structures. Our context summary is based on a derivative of a Bloom filter [2], which succinctly represents set membership using a bit array m and k hash functions. To add an element to a Bloom filter, we use 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 inserting other elements happens 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 a cascade of Bloom filters on each bit of the values. Consider the case where each value is either 0 or 1. Within a Bloomier filter, a Bloom filter A_0 contains all of the keys that map to 0; B_0 contains all of the keys that map to 1. If the value associated with element e is 0, it is inserted in A_0; if the value is 1, it is inserted in B_0. When one queries the Bloomier filter for the value of e', the query checks whether e' is in A_0 and in B_0. There are four possible results. (1) If e' is in neither A_0 or B_0, e' has not been associated with a value in the Bloomier filter. (2) If e' is in A_0 but not B_0, e' was inserted in the Bloomier filter with high likelihood, and when it was inserted, it was associated the value 0. It is possible that e' was not inserted at all (the unlikely false positive), but it was not inserted with value 1. (3) Similarly, if e' is in B_0 but not in A_0, the query returns 1. (4) If e' is in both A_0 and B_0, one of these is a false positive. To handle this fourth case, another pair of Bloom filters attempts to resolve false

positives in the first pair. A_1 contains keys that map to 0 and generated false positives in B_0. 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 B_0. The Bloomier filter continues to add levels of filters until the key set becomes small enough to store in a map.

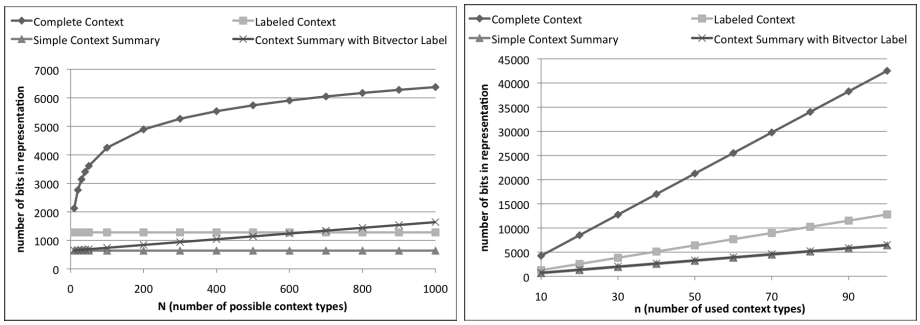
To handle longer values, a Bloomier filter uses a cascaded filter for each bit, i.e., when the range of values is $\{0,1\}^r$, it creates r Bloom filter cascades. This has space complexity of $O(rn)$, where n is the number of elements stored and r is the number of bits needed to represent an element's value. This is in comparison to the $O(rn \log N)$ space complexity of enumerating the value of every element in the set (where N is the number of possible context elements). The Bloomier filter has a false positive rate $\epsilon \propto 2^{-r}$. We use this construction, which achieves fast computation with slightly higher than optimal space use; different constructions make varying tradeoffs in space and time complexity [5, 21].

Defining Context Summaries. Our context summary must be space efficient while retaining semantic fidelity. 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* among all entities in the network. Let $|\mathcal{C}| = N$. A given 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. $|\mathcal{C}_{node_i}| = n$; $n \ll N$. A general statement of context sensing is as 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 $\overline{c}$, the sensed value is null.

We aim to create a summary that, when queried with a context type, returns the relevant state of the entity. If $c' \in \mathcal{C}_{node_i}$, then the summary should return $context_{node_i}(c')$. If $\overline{c'} \notin \mathcal{C}_{node_i}$, the summary should return $\perp$ with high probability. False positives are allowed (but undesirable) for context types that were not sensed by $node_i$. Every summary contains the key *cs_id* mapped to the value *node_id*; when a summary is queried with *cs_id* it will, without fail, return the unique id of the entity whose context is summarized. Our context summary is a straightforward Bloomier filter. This summary achieves a space complexity of $O(rn)$ bits. The time required to traverse the cascaded data structures and retrieve a value is $O(\log \log n)$. With probability $O(2^{-r})$, when the summary is queried with $\overline{c'} \notin \mathcal{C}_{node_i}$, it returns a false positive, i.e., a value that is garbage.

To determine whether a summary contains an attribute's value, we have to poll all N possible attributes. If N is large (and it usually is), this is excessive. The summary can also return false positives (albeit with low probability) that can negatively impact applications that use context. We refine our summary by adding a bit vector bv of length N, each element of which is a flag for a $c \in \mathcal{C}$. If the value for c is included in the summary, $bv[c]$ is 1; otherwise it is 0. This adds overhead, particularly if N is huge, but it removes all false positives. This summary requires $O(N + rn)$ bits; it retains the $O(\log \log n)$ lookup time.

We adopt some simple notational conveniences to refer to context summaries and their components. Given a context summary CS and a context attribute



(a) Varying N ($n = 10$, $l = 64$, $r = 64$) (b) Varying n ($N = 100$, $l = 64$, $r = 64$)

Fig. 1. Number of bits required to represent context using the four different approaches. (l is the length of a context label; r is the the max context size (both in bits.)

l and its value v that has been inserted, CS.$l = v$; if an attribute referred to by l has not had a value inserted, CS.$l = \perp$ with high probability. We use the notation CS.$l \leftarrow v$ to insert the value v associated with the key l into CS, e.g. CS.$cs\_id \leftarrow node\_id$. CS.bv refers to the bit vector of the extended context summary; CS.$bv[l]$ is 1 if attribute l is stored in CS and 0 otherwise.

Fig. 1 compares four approaches to context representation analytically. *Simple Context Summary* is our simple Bloomier filter representation. *Context Summary with Bitvector Label* is our approach that extends the simple context summary with a bit vector of length N. *Complete Context* enumerates the value of every element in the set (which requires $O(nr \log N)$ space [6]). *Labeled Context* enumerates only the values of context attributes in $\mathcal{C}_{node_i}$, each labeled with its key (which requires a $O(n(r + l))$ space, where l is the length of a label).

The summary approaches significantly reduce the size of the context representation, especially as N grows (Fig. 1(a)). However, when we hold N constant, but increase n (Fig. 1(b)), sending labels becomes expensive. Fig. 1 assumes context labels that are 64 bits in length (8 characters); longer labels with more semantic information increase this gap. From this analysis, we can conclude that our context summary achieves the highest reduction in size of context representation. In addition, when the number of possible context types remains relatively small, it is reasonable to add a bit vector summary of the context summary.

Communicating Context Summaries. We have reduced the size of the context representation to reduce communication overhead of sharing context in pervasive computing environments. We construct a framework that transparently piggybacks context summaries on outgoing wireless transmissions. Fig. 2 shows our framework's architecture. An application sends context to the *Context Handler* and retrieves context about other entities and groups. The *Context Shim* attaches context summaries to outgoing packets and examines incoming packets for received contexts. The right of Fig. 2 shows the internals of *Context Shim*, which stores MYCONTEXT, this entity's context information. This architecture allows neighboring nodes to exchange context; we also spread summaries beyond one-hop neighbors by inserting an attribute into each summary, *hops*, initialized

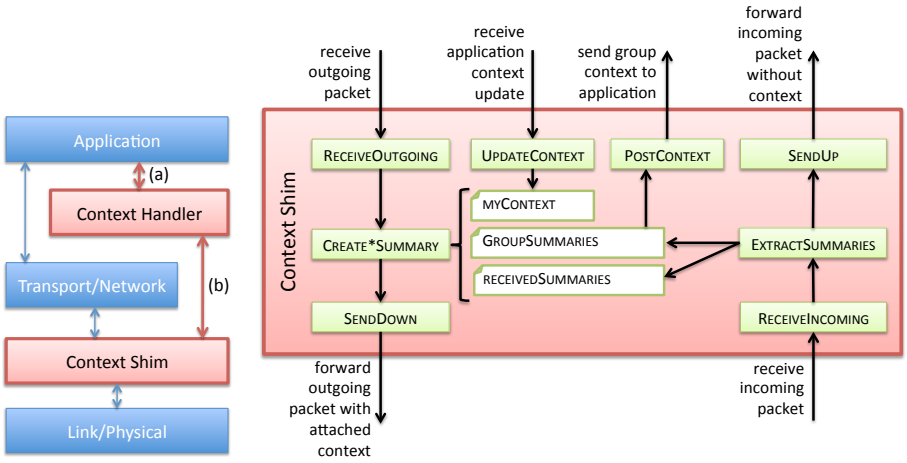


Fig. 2. Architecture for Context Sharing

to 0. Every time the shim receives an incoming summary, it increments this hop count before storing it in RECEIVEDSUMMARIES. As long as the summary's hop count has not surpassed a provided threshold, τ, the receiving node appends the summary to its outgoing packets (in addition to its own summary).

Fig. 3 gives the behavior of the CREATECONTEXTSUMMARY and EXTRACTSUMMARIES from Fig. 2. The $\oplus$ appends the summary to the packet without altering the packet in any other way. This shows creation of the extended context summary; the simple context summary is the same without line 4.

```
CreateContextSummary(pkt)
 1  for (l, v) ∈ myContext
 2       do CS.cs_id ← node_id
 3          CS.l ← v
 4          CS.bv[l] = 1
 5          CS.hops = 0
 6  pkt' = pkt ⊕ CS
 7  for CS ∈ReceivedSummaries
 8       do if CS.hops < τ
 9             then pkt' = pkt' ⊕ CS
10  return pkt'
```

```
ExtractSummaries(pkt)
1  pkt' = pkt
2  for CS appended to pkt'
3       do CS.hops ← (CS.hops + 1)
4          insert (CS.cs_id, CS)
              in ReceivedSummaries
5          pkt' = pkt' ⊖ CS
6  return pkt'
```

Fig. 3. Pseudocode for CREATECONTEXTSUMMARY and EXTRACTSUMMARIES

3 Supporting Groups and Their Context

A *group* is a set of nodes (identified by *node_ids* or context summaries). In this section, we define the *context* of such groups using individual context summaries.

The Context of a Group. Given a group of entities and their context summaries, we must devise a space-efficient representation of the aggregate context of the group. Given a set of context summaries, we define an aggregate of those summaries that captures the *context of the group* 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 context of g as:

$$\textsc{GroupContext}_{g\_id} = \{(l, v_{agg}) : v_{agg} = f_{agg,l}(\{\text{CS}_{n_i}.l : n_i \in g\}), \forall l \in \bigcup_{n_i \in g} \text{CS}_{n_i}\}$$

where $f_{agg,l}$ is an aggregation function associated with the context type l. Different types of context have different forms of aggregation. For example, to aggregate multiple location values, $f_{agg,loc}$ may construct a bounding box. Aggregation functions can be standard functions like average, maximum, or minimum, or they may be defined by the context ontology. We assume the aggregation function for each context label is defined and shared *a priori*. A group context is a set of pairs of labels and values and can be represented by a context summary. In this case, the context attribute *cs_id* is a unique *g_id* instead of the *node_id*. We can construct a group context summary iteratively as shown in Fig. 4.

A group summary tracks the nodes it summarizes to ensure that it does not aggregate information for a node twice. This information is added in line 4 of AggSummaries (which uses f_{ids} to make a list of included *node_ids*). Line 3 in CreateGroupAgg checks before incorporating CS into GS that GS does not already include the node. Context summaries and group summaries are interchangeable; it is possible that a CS ∈ receivedSummaries is a group summary; the check on the second half of line 3 in CreateGroupAgg ensures that we never include information about the same node more than once in an aggregate summary.

```
CreateGroupAgg(g_id)
1  Create empty group summary GS
2  for CS ∈ receivedSummaries
3       do if g_id ∈ CS.G ∧ ¬CS.cs_id ∈ GS.agg_nodes
                ∧CS.agg_nodes ∩ GS.agg_nodes = ∅
4            then GS = AggSummaries(GS, CS)
5  return GS

AggSummaries(CS1, CS2)
1  create empty aggregate summary CS_agg
2  for l ∈ (CS1.labels ∪ CS2.labels)
3       do if l = cs_id
4            then CS_agg.agg_nodes ← f_ids(CS1.l, CS2.l)
5          else if l ∈ CS1.labels and l ∈ CS2.labels
6            then CS_agg.l ← f_agg,l(CS1.l, CS2, l)
7          else if l ∉ CS1.labels and l ∈ CS2.labels
8            then CS_agg.l ← CS2.l
9          else CS_agg.l ← CS1.l
10 return CS_agg
```

Fig. 4. Pseudocode to create group context summaries

Capturing Connectivity. Our group definitions can be expressed from a global perspective that assumes knowledge of all entities and their contexts. This is unreasonable in a distributed, dynamic environment since it does not account for connectivity, which is necessary for entities to share context and determine groups in a distributed manner. We define connectivity as a binary relation, $\mathcal{K}$, that captures the ability of two entities to communicate via a single network hop. We assume $\mathcal{K}$ is symmetric. We define $\mathcal{K}^*$ to contain pairs of mutually reachable entities (i.e., $(n, n') \in \mathcal{K}^*$ if there exists a (potentially multihop) path between n and n'). Of course, actually carrying out communication across a multihop path may or may not succeed due to dynamics, noise, dropped packets, and subtleties of communication protocols; $\mathcal{K}^*$ represents the *ideal* connectivity. We forward

summaries only in a limited region defined by a constraint on the number of hops, τ. While $\mathcal{K}^*$ captures all paths, $\mathcal{K}^\tau$ defines pairs of entities within τ of each other; these entities should mutually know each other's context. Obviously, $\mathcal{K}^\tau \subseteq \mathcal{K}^*$. The summaries stored in RECEIVEDSUMMARIES of entity n should be a subset of the summaries of the entities to which n is related under $\mathcal{K}^\tau$.

4 Defining Groups: A Distributed Emergent Approach

We construct formal definitions of four types of groups that capture coordination needs of entities in dynamic environments. This set is not meant to be complete, but it is expressive. We refine our context handling protocols to create, maintain, and share groups. CS_n is the summary for entity with $node\_id = n$. Every group has a unique id g_id that is also unique from all node ids.

Labeled Groups. The simplest type of group is one in which members are determined *a priori*. Such groups are effectively *labeled*; the group's g_id is a shared unique id (whether public or secreted in some way). As a convenient way for designating that entity n is in group g_id, we use the entity's context summary; for every group of which it is a member, an entity inserts a pair (Group$i$, g_id) in its summary. Labels of the form Group$i$ are reserved identifying groups; the value of i ranges from 1 to the number of groups of which the entity is a member, and g_id is the (unique) group id for one of these groups. If an entity is a member of three groups with ids g_id_1, g_id_2, and g_id_3, its summary may contain the following three mappings: (Group1, g_id_1), (Group2, g_id_2), and (Group3, g_id_3), though there is no association between the i in Group$i$ and the group ids. We refer to entity n's groups as $\mathrm{CS}_n.G$, where G is a set of group ids. As an application example, consider devices carried by students enrolled in a course. These students may each belong to a group identified by the course name; a student's context summary includes a group id for each enrolled course.

Given the context summaries of all entities, the membership of a labeled group is a set g_{g_id} such that $\forall n' \in g_{g_{id}} : g\_id \in \mathrm{CS}_{n'}.G$. For our students, g_{id} contains all of the students enrolled in a course. To refine a labeled group to account for connectivity, we consider only context summaries from entities that are related to n under $\mathcal{K}^*$. The membership of the *partition* of $g_{g_{id}}$ of which n is a member (i.e., $g_{g_{id}}[n]$) is a set of entities such that $\forall n' \in g_{g_{id}}[n] : (n, n') \in \mathcal{K}^* \wedge g\_id \in \mathrm{CS}_{n'}.G$. Consider students studying in the library; $g_{g\_id}[n]$ for a student n includes all students reachable via an ad hoc network of their devices.

Since we do not distribute summaries beyond τ, we may not have information about the entire partition $g[n]$. The group membership that a given entity n may know about is defined relative to $\mathcal{K}^\tau$; to capture this, we simply replace $\mathcal{K}^*$ in the above with $\mathcal{K}^\tau$. This use of the connectivity relation is pretty strong; it requires that all members of the (partition of the) group must be related to n under $\mathcal{K}^\tau$. In a weaker form, each member must be related under the transitive closure of $\mathcal{K}^\tau$, i.e., $(\mathcal{K}^\tau)^+$: $\forall n' \in g_{g_{id}}[n] : (n, n') \in (\mathcal{K}^\tau)^+ \wedge g\_id \in \mathrm{CS}_{n'}.G$.

We replace CREATECONTEXTSUMMARY in Fig. 3 with CREATEDLGCONTEXTSUMMARY in Fig. 5, which inserts the group id (g_id) for this entity's

```
CreateLGContextSummary(pkt)
 1  for (l, v) ∈ myContext
 2        do CS.l ← v
 3           CS.bv[l] = 1
 4  i = 1
 5  for g_id ∈LabeledGroups
 6        do CS.("Group"+i) ← g_id
 7           CS.bv[("Group" + i)] = 1
 8           i = i + 1
 9  pkt' = pkt ⊕ CS
10  for CS ∈ReceivedSummaries
11        do if CS.hops < τ
12              then pkt' = pkt' ⊕ CS
13  return pkt'
```

```
LGExtractSummaries(pkt)
 1  pkt' = pkt
 2  for CS appended to pkt'
 3        do CS.hops ← (CS.hops + 1)
 4           insert (CS.cs_id, CS)
                in ReceivedSummaries
 5           pkt' = pkt' ⊖ CS
 6  i = 1
 7  g_id =myContext.("Group"+i)
 8  while g_id ≠⊥
 9        do CS_agg =
                CreateGroupAgg(g_id)
10           CS_agg.Group1 ← g_id
11           insert (g_id, CS_agg)
                in receivedSummaries
12           post group context update
                to application if changed
13           i = i + 1
14           g_id = CS.("Group"+i)
15  return pkt'
```

Fig. 5. Pseudocode for CreateLGContextSummary and LGExtractSummaries

groups. LGExtractSummaries processes received summaries and computes group membership by determining the group membership(s) of the entities from which received summaries came, updating the aggregate information for those groups, and informing the application of the changed group context. A node only creates group summaries for groups of which it is a member; the group's aggregate context is based on any context summary received from any other entity with the same group id. This implements the weak form of labeled groups; the groups will include entities related by $(\mathcal{K}^\tau)^+$. To create the strong form, we simply remove line 11 of LGExtractSummaries. When line 11 is included, the computed group summary is inserted in receivedSummaries and will be appended to sent packets. This enables the computation of groups that extend across $(\mathcal{K}^\tau)^+$. When LGExtractSummaries operates over the summaries in receivedSummaries to compute the aggregate context (lines 9-14), it does not matter if the summary used is an individual context summary or a group summary since AggSummaries is iterative. The check on line 3 of CreateGroupAgg in Fig. 4 prevents us from aggregating two summaries that include information about the same entity. The result is a heuristic; it may be possible to create a group summary that incorporates information about a larger number of entities, but this is not possible from the summary data we distribute. To maximize the number of entities represented in an aggregate, we can optimize the order in which we incorporate them; we omit these algorithms for brevity.

The context of the group of students labeled by course number could represent a variety of factors. As one example, the students' context summaries may also carry a field labeled "understanding of concept A"; this value may be filled in automatically by some assessment applications on the device; the context of a group can then be the group's average understanding of the material.

Asymmetric Groups. Entities can also define ego-centric notions of context, or *asymmetric groups*, known only to the entity at its center. As an example, an application on a car may keep the driver aware of other nearby cars that have the potential for collision. Members of an asymmetric group need not know they are part of the group; if two entities use the same asymmetric group definition, they likely end up with completely different groups. This is the case in our example; two different cars clearly have different perspectives of other cars with a potential for collision. Information about asymmetric groups is not shared (i.e., asymmetric group summaries are not placed in RECEIVEDSUMMARIES). Asymmetric groups can be defined given a function f_n provided by entity n that constrains the relationship between the entity's context and the context of any group member. Specifically, the membership of an asymmetric group is a set g_{f_n} such that $\forall n' \in g_{f_n} : (n, n') \in \mathcal{K}^\tau \wedge f_n(\mathrm{CS}_n, \mathrm{CS}_{n'})$. In our collision-awareness application, f_n may compute the "time to collision" given my velocity and another nearby car's velocity; if this time to collision is over a threshold, f_n returns true.

We use f_n to "tag" the summaries stored in RECEIVEDSUMMARIES that should be part of the asymmetric group and compute the group context for those tagged summaries. When sending summaries from RECEIVEDSUMMARIES, the tags need to be stripped from the summaries. We omit this pseudocode for brevity. The context computed by our car's f_{agg} could be the number of cars within the dangerous zone or their average speed. A more complex f_{agg} could compute the bounding box of dangerous cars to show on a heads-up display.

Symmetric Groups. Entities can share a symmetric definition of a group that constrains the pairwise relationship between any entities in the group. Consider a set of devices forming a mobile ad hoc network that wants to determine the best network protocols given current network conditions. Such a network may wish to form a group of mutually reachable devices whose context determines a set of protocols. A function f_{g_id} shared among entities *a priori* can define membership in the group based on pairwise comparisons of members' contexts. The set $g_{f_{g\_id}}[n]$ of members of n's partition of this group is one such that $\forall n, n' \in g_{f_{g\_id}}[n] : (n, n') \in \mathcal{K}^\tau \wedge f_{g\_id}(\mathrm{CS}_n, \mathrm{CS}_{n'})$. Symmetry refers to the fact that $n' \in g_{f_{g\_id}}[n] \Leftrightarrow n \in g_{f_{g\_id}}[n']$; this is ensured by the symmetry of $\mathcal{K}$ and the shared f_{g_id}. The above provides a strong requirement on connectivity; in the weaker form, each group member must be related to n under $(\mathcal{K}^\tau)^+$. With respect to our mobile ad hoc network example, f_{g_id} may require all of the nodes in the group to be mutually reachable within a specified number of hops or time.

If a receiving entity determines that a received summary is part of one of its symmetric groups, it tags the summary with the group id before inserting it in RECEIVEDSUMMARIES. Like labeled groups (and unlike asymmetric groups), these summaries are inserted in RECEIVEDSUMMARIES and are appended to outgoing packets to enable computation of groups across $(\mathcal{K}^\tau)^+$. Our protocol checks $f_{g\_id}(\textsc{myContext}, \mathrm{CS})$ for any received summary CS, calculates the aggregate context, and updates both RECEIVEDSUMMARIES and the application.

In our application scenario, metrics such as relative mobilities of the nodes, neighbor densities, and communication error rates influence the selection of the

best protocol [15]. Given information in the group members' context summaries (e.g., connectivity, velocity, position, error rates), these aggregate measures can be easily calculated and handed to a process that selects the best protocol.

Context-Defined Groups. Context-defined groups require that a group together satisfy some requirement. Consider a network with changing capabilities and an application that wants to leverage network resources to perform a task. A group may be a set of devices, which, in the aggregate, is capable of completing that task. Similarly, imagine using a local network of smart phones to form teams for a pick-up game of football, where each team should have a capable player at each position. As with symmetric groups, such a group is defined by a function over context summaries; in this case, the requirement specifies a property that the group as a whole must uphold. Given f_{g_id} that defines the group constraint(s), $g_{g\_id}[n]$ is valid if and only if $f_{g\_id}(g_{g\_id}[n])$. The strong version of a context-defined group requires all members of $g_{g\_id}[n]$ to be related to n via $\mathcal{K}^{\tau}$; the weaker version requires each member to be related to n via $(\mathcal{K}^{\tau})^{+}$. Multiple groups defined with the same f_{g_id} may overlap (i.e., a single node could define multiple groups using the same context function).

```
CDGroupExtractSummaries(pkt)
 1  pkt' = pkt
 2  for CS appended to pkt'
 3        do CS.hops ← (CS.hops + 1)
 4           insert (CS.cs_id, CS) in receivedSummaries
 5           pkt' = pkt' ⊖ CS
 6  for g_id ∈ ContextDefinedGroups
 7        do for each permutation P of
                    receivedSummaries sorted by size
 8              do if f_g_id(P)
 9                    then CS_agg =AggSummaries(P)
10                         insert (g_id, CS_agg) in
                             receivedSummaries
11                         post group context update
                             to application if changed
12                         skip remaining permutations
13  return pkt'
```

Fig. 6. Pseudocode for CDGroupExtractSummaries

Summaries for these groups are placed in receivedSummaries and appended to packets. The more complex piece of determining context-defined groups is the application of f to the receivedSummaries (both individual and group summaries), shown in Fig. 6. There are several ways one can apply f_{g_id} for a context-defined group. We generate all permutations of receivedSummaries and look at them from largest to smallest, where the size is the number of nodes for which it contains summary information. We take the largest permutation that satisfies f_{g_id} (if one exists), compute its aggregate context, and return it to the application as the context of the group. Different heuristics exist for choosing which of the groups that satisfy f_{g_id} is best, including application-defined metrics for the quality of a group. Evaluating the relative merits of these alternatives is out of the scope of this paper but is an area of future research.

In our examples, the context could capture the quality of the aggregate. For devices providing resource capabilities, the context may be the quality of service

the aggregate of entities can provide for the task. For the ad hoc sport team formation, the context may be a measure of the overall quality of a team.

5 Implementation, Demonstration, and Evaluation

We implemented our approaches in C++ using the architecture in Fig. 2. We incorporated this prototype into the OMNeT++ discrete event simulator with the INET framework and used this prototype to define contexts of individual entities, share them, create groups, and define and share the context of those groups[1]. We provide a few demonstrative results, first for the performance of the context summary mechanisms and then for some groups and their context.

Evaluation Settings. We used an available UDP implementation to generate data packets (on top of which context summaries could be piggybacked) and the provided AODV routing implementation to route data packets. While we experimented with a variety of settings, we report results we achieved when using 50 nodes moving according to the random waypoint mobility model with varying speeds (from 0 to 20 m/s). Each node generated UDP traffic at a rate of 10 packets/s and was assigned a different set of destinations (to allow for the AODV protocol to form and reuse existing routes); when a node generated a UDP packet, it selected a destination randomly from this list. Unless specified otherwise, the charts below use a τ of three hops, a context label and value lengths of 64 bits, n of 10, and N of 1000. We show 95% confidence intervals.

Sharing Context Summaries. We implemented the four approaches for context summaries described in Section 2. We executed each on our sample networks; the results given a varying n (the number of contexts used in a context summary) and N (the number of context labels known to the application) are shown in Fig. 7. It is immediately obvious that the shapes of these curves match those in Fig. 1, which indicates that our experience with the context summary structures matches our analytical expectations. These results boost our confidence that our simple Bloomier filter based context summary and the Bloomier filter based context summary with the associated bit vector provide good communication efficiency for sharing context information in a distributed network.

Calculating and Sharing Groups. Fig. 8 shows a labeled group created using context summaries with bit vectors. We varied the fraction of the 50 nodes that were labeled with the group and their speeds. Fig. 8(a) compares the correctness of calculating the groups for a scenario when 15 nodes were labeled as group members. The bottom line in the figure compares the nodes our approach determined to be within the group to 15. The second line compares this to the number of nodes that were connected (i.e., $\mathcal{K}^*$). The highest line compares the number of group members discovered to the number that were expected to be within range (i.e, $\mathcal{K}^\tau$, based on an estimated radio range). The latter is the fair comparison; given the opportunistic distribution of context information, we were able to identify $\sim 50\%$ of the actual group members. Errors in group calculation can be due to communication failures, noise, and stale knowledge (which is

[1] The code is available at `http://www.ece.utexas.edu/~julien/GroupContext.html`

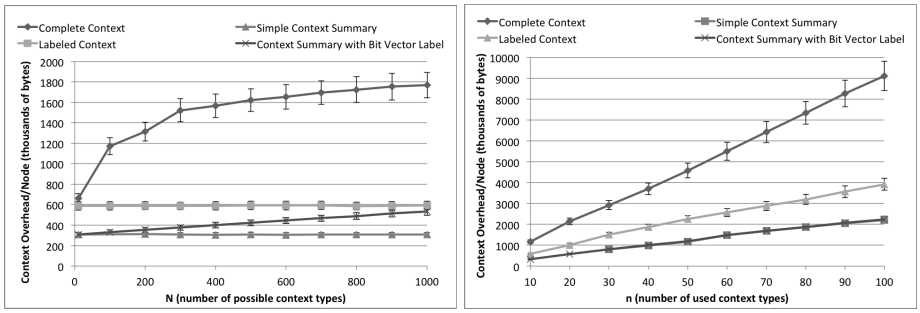
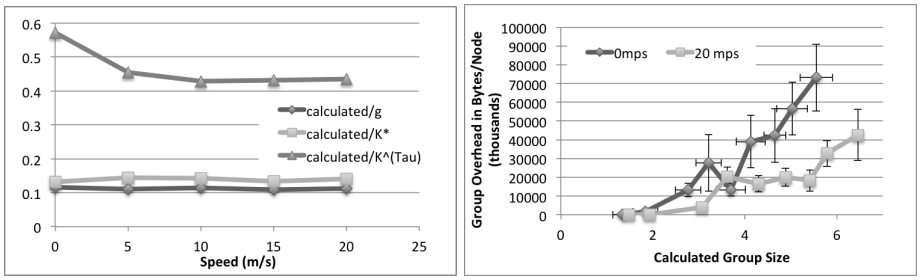


Fig. 7. Context Summary Size in Simulation



(a) Correctness of group vs. g, K^*, and K^τ (b) Additional overhead of group context

Fig. 8. Calculating and Sharing Groups (Actual Group Size = 15)

impacted by the data rate since summaries are piggybacked on data packets). Further study of the impact of data rates and update intervals for context information may help to better understand the incompleteness of group calculation.

Fig. 8(b) shows the overhead for sharing group contexts. This overhead is higher when nodes are stationary since they are more likely to succeed in calculating the group, thereby generating a summary to share. Future work will investigate heuristics to help reduce this overhead. As one example, in our prototype, we forward both group context summaries *and* the individual context summaries that the group summary is based on. Removing some of this redundant information can lower the cost of context communication, albeit at the expense of having detailed individual information spread further in the network.

6 Related Work

Because of their space efficiency, Bloom filters are widely used in networking [3], for example to succinctly represent coding symbols for efficient erasure coded communication [4] or support uniform distribution of stored data [16]. Bloom filters have been used to gossip cache entries and reduce overlaps in collections in a P2P network [1] and to summarize the shared contents of a group [18]. Similarly, Bloom filters have been used in wireless sensor networks to dynamically create clusters and aggregate data [13] or to efficiently route queries [14]. Dynamic Bloom filters [10] improve upon space efficiency by shrinking their

size when possible, specifically for the purpose of representing data shared in a network. These approaches all focus on set membership representations, which cannot represent context values.

The spectral Bloom filter [7] extends the Bloom filter to multisets, allowing one to estimate frequencies, which is applicable to managing per-flow traffic in network routers [17]. Using attenuated Bloom filters, each node stores discounted Bloom filters representing the contents of its neighbors, which it uses to route queries to locations that are likely to store results [22]. This is similar to using context summaries to form asymmetric groups, however we devise a generalized framework instead of tailoring the summary to a given application.

Other approaches have recognized that groups and their context are important. In [9] groups are formed statically or based on co-location, while our groups can be defined in many ways. The approach in [9] also does not focus on efficient representation and sharing of context and is therefore not transferable to resource constrained infrastructureless environments.

The Team Analysis and Adaptation Framework (TAAF) [8] observes the behavior of a distributed collaborative team and introduces constructs to adapt supporting services and team coordination. TAAF assumes the team is already assembled, and the generation of the context of the team is centralized. Context-Aware Ephemeral Groups (CAEG) [24] explore a more abstract definition of groups based solely on social connections that are used to guide users to likely relevant resources and to maintain persistent state among the users in a ubiquitous computing space, though the focus in on function and interface instead of on efficiency of context representation. We argue that the latter is essential in ad hoc environments, which are severely resource constrained. Our work is a complement to CAEG in supporting the necessary efficient exchange of individual and group context information and new ways to support expressive group creation and management (outside of CAEG's social group).

7 Conclusions and Future Work

In this paper, we have tackled the multipart problem of expressively summarizing the context of individual entities in a dynamic environment so that they can be shared efficiently across wireless links. We use these individual context summaries to define and compute groups on-the-fly, dependent on the context and to the compute the aggregate context of the group. The work in this paper is a first step in being able to compute and share such expressive aggregate context information without supporting infrastructure and opens many interesting new research questions. Directly related to the work in this paper are ideas for incorporating additional mechanisms to shrink the summaries further [12, 19, 20]. In computing the groups, we assumed we used a slightly larger structure (the context summary with a bit vector) that removed all false positives; if the size of this structure is undesirable, it becomes interesting to ask what the impact is of false positives on group formation. These questions and others lay the ground-

work for future use of shared distributed context state to support expressive coordination in dynamic environments.

References

1. Bender, M., Michel, S., Triantafillou, P., Weikum, G., Zimmer, C.: Improving collection selection with overlap awareness in p2p search engines. In: SIGIR (2005)
2. Bloom, B.: Space/time tradeoffs in hash coding with allowable errors. Comm. of the ACM 13(7) (1970)
3. Broder, A., Mitzenmacher, M.: Network applications of bloom filters: A survey. Internet Mathematics 1(4) (2004)
4. Byers, J., Considine, J., Mitzenmacher, M., Rost, S.: Informed content delivery across adaptive overlay networks. IEEE/ACM Trans. on Netw. 12(5) (2004)
5. Charles, D., Chellapilla, K.: Bloomier filters: A second look. In: ESA (2008)
6. Chazelle, B., Kilian, J., Rubinfeld, R., Tal, A.: The bloomier filter: An efficient data structure for static support lookup tables. In: SIAM (2004)
7. Cohen, S., Matias, Y.: Spectral bloom filters. In: SIGMOD (2003)
8. Dorn, C., Truong, H.L., Dustdar, S.: Measuring and analyzing emerging properties for autonomic collaboration service adaptation. In: ATC (2008)
9. Ferscha, A., Holzmann, C., Oppl, S.: Context awareness for group interaction support. In: MobiWac (2004)
10. Guo, D., Wu, J., Chen, H., Luo, X.: Theory and network applications of dynamic bloom filters. In: INFOCOM (2006)
11. Hackmann, G., Julien, C., Payton, J., Roman, G.C.: Supporting generalized context interactions. In: SEM (2005)
12. Hagerup, T., Tholey, T.: Efficient minimal perfect hashing in nearly minimal space. In: STACS (2001)
13. Hebden, P., Pearce, A.: Bloom filters for data aggregation and discovery: A hierarchical clustering approach. In: ISSNIP (2005)
14. Jardak, C., Riihijarvi, J., Mahonen, P.: Analyzing the optimal use of bloom filters in wireless sensor networks storing replicas. In: WCNC (2009)
15. Jun, T., Julien, C.: Automated routing protocol selection in mobile ad hoc networks. In: SAC (2007)
16. Kostic, D., Rodriguez, A., Albrecht, J., Vahdat, A.: Bullet: High bandwidth data dissemination using an overlay mesh. In: SOSP (2003)
17. Kumar, A., Xu, J., Wang, J.: Space-code bloom filter for efficient per-flow traffic measurement. IEEE J. on Selected Areas in Comm. 24(12) (2006)
18. Ledlie, J., Taylor, J., Serben, L., Seltzer, M.: Self-organization in peer-to-peer systems. In: SIGOPS European Wkshp. (2002)
19. Mitzenmacher, M.: Compressed bloom filters. IEEE Trans. on Netw. 10(5) (2002)
20. Pagh, A., Pagh, R., Rao, S.: An optimal bloom filter replacement. In: SODA (2005)
21. Porat, E.: An optimal bloom filter replacement based on matrix solving. In: CSR (2009)
22. Rhea, S., Kubiatowicz, J.: Probabilistic location and routing. In: INFOCOM (2002)
23. Salber, D., Dey, A.K., Abowd, G.D.: The context toolkit: Aiding the development of context-enabled applications. In: CHI (1999)
24. Wang, B., Bodily, J., Gupta, S.: Supporting persistent social groups in ubiquitous computing environments using context-aware ephemeral group service. In: PerCom (2004)