Efficient Decentralized Context Sharing via Smart Aggregation

Sungmin Cho and Christine Julien
Center for Advanced Research in Software Engineering
The University of Texas at Austin
Email: smcho@utexas.edu, c.julien@utexas.edu

***Abstract*—Sensing applications often require participants to share *context* information about the physical social, or network environment in which they operate. Building shared views of context requires exchanging sensed information, often via peer-to-peer links. In-network aggregation enables efficient distributed data collection, but the goal has been almost exclusively collect a single aggregate value at a single sink node. In contrast, we design and implement a simple protocol for exchanging context information in aggregate in a peer-to-peer fashion, where *every* node needs to acquire a shared view of the aggregate context. In our protocol, when a node receives new context information from a neighboring node, it aggregates the new information into its local view of the shared state of the world which it then subsequently shares with its neighbors. We demonstrate (both theoretically and empirically) the situations in which participants' raw context information is fully or partially recoverable by other participants from an aggregate and quantify the tradeoff in communication overhead for the quality of shared context knowledge. Compared with non-aggregation communication for sharing context values among 100 nodes in a simulated network, we show an overhead savings of at least 78.0%, and an overhead savings of 66.0% with 99.8% average accuracy in a 54 node emulated network driven by real world data.**

I. Introduction

With the increasing popularity of pervasive computing, it is common for networked devices to be connected each other to share locally sensed information. Each node is assumed to be equipped with sensing, computing, storage, and communication capabilities. Sensed or computed information from environments can be processed to create local views of regional context measures, and these local views can be transferred via peer-to-peer network links to other nearby nodes. These exchanges are application dependent, but in many cases the individual nodes work together to create shared views of some context measure; for example each node may disseminate its locally sensed temperature to construct an average temperature over some area. Compared to each node's local context, which represents an egocentric view, aggregate context views that are shared among participants can play a meaningful role in decision making and actuation, as each node can compare its local value(s) to aggregate ones. For example, actuator nodes may used aggregate temperatures to determine how to adjust a building's heating and cooling systems. We propose an efficient protocol for decentralized information sharing via smart aggregation among nodes in a pervasive system.

In many applications a node may function as both a source of individual context information and a consumer of some shared (aggregate) view of that context. For example, shared global contexts can enable evaluating global invariants to identify abnormal environmental conditions or to enforce constraint variations in quality [15]. A building's Heating, Ventilation, and Air-conditioning Controller (HVAC) system may require a uniform pressure distribution in the building to prevent wind; in other cases, e.g., when a volatile organic compound is detected, the HVAC system may need to create a pressure differential to protect air quality [16]. These conditions can be detected by creating spatial aggregates of air pressures in different areas of the building and comparing these aggregates against local measures. As another example, sensor nodes deployed in building structures can monitor stress and strain generate warnings when the difference between the locally measured strain and an aggregate view of the overall strain is above the threshold [5]. In a wild fire detection system deployed in a forest [20], sensing and reporting in response to heightened temperatures may lead to false alarms. Comparing a locally sensed value against a shared spatial aggregate before triggering an alarm can help better identify true deviations from normal temperature changes [14].

In-network aggregation has been widely used to reduce the cost of sharing context information [4]. Multiple distinct pieces of context information are collected into a single representative measure so that the total amount of data exchanged is reduced to benefit both communication bandwidth and power consumption. Existing approaches largely target collecting an aggregate value at a single node (usually a base station); that is, the focus has traditionally been on a centralized many-to-one aggregation, and not the distributed many-to-many aggregation that the examples above motivate. There are limited approaches to many-to-many aggregation, which we review in the next section. In addition to enabling new applications, this style of many-to-many aggregation has another advantage over many-to-one aggregation in terms of reliability; each node receives aggregated information form multiple sources to compensate for missing information from lost packets or noise caused by communication or sensing errors.

In Section IV, we present a protocol for decentralized context sharing that is capable of both smart aggregation and disaggregation. Our approach shares aggregate contexts among nodes; each node only redistributes context that provides unique information to its neighbors. To achieve this goal, each node disaggregates received aggregate contexts to maximize the unique values it knows and can combine into a new, information-rich aggregate. This scheme enables efficient dissemination of aggregated contexts so that all participating nodes can recover the individual or aggregated contexts to build a shared view with lower communication burden.

We demonstrate our approach over two types of networks:

tree networks guaranteed to not contain cycles, and mesh networks that do have cycles. In trees, we can recover average values without error, as there is no duplication of any individual nodes' sensed values. In mesh networks, a node's received aggregates may contain data that overlaps the nodes' existing knowledge. In these cases, the node cannot simply merge the two pieces of information to create a larger aggregate. These challenges motivate our design for more sophisticated algorithms for disaggregating information to recover maximal aggregates without duplicating individual nodes; information. Heuristic approaches to this disaggregation and reaggregation can naturally introduce error in the final aggregate calculation, and one of our primary goals is to reduce that error.

In Section V, we analytically examine our protocol's communication overhead (the total amount of context information exchanged), the speed (how quickly the nodes' view of the shared context reaches a steady state), the identification rate (the degree with which each individual node was able to recover the correct aggregate values or individual values for the other participants), and the accuracy (the difference between calculated aggregate and and the correct aggregate). Our analysis shows that our approaches reduce the communication overhead by up to 86.63% while retaining the same speed, identification rates, and accuracies as compared to broadcasting nodes' individual contexts throughout the entire network. In simulations using sample data from 54 deployed sensor nodes, we achieved a 66.0% reduction in communication overhead with 99.84% accuracy and 94.9% identification rate. We report these simulation results in Section VI.

II. Related work

In this paper, we propose a many-to-many dissemination protocol that achieves intelligent aggregation and disaggregation. We focus on decentralized communication in which all nodes function as a data source and a data sink.

Logical neighborhoods [12], Hood [9] and Abstract Regions [18] support many-to-many communication in sensor networks using various neighborhood abstractions. These abstractions allow sensor nodes to share state with other nearby nodes and to identify a set of nodes in the neighborhood based on network cohesion or logical properties. Our research resembles these approaches in attempting to enable many-to-many communication by providing programming interfaces for communicating with neighbor nodes, but our approach targets a higher-level of abstraction, using the context information that nodes share in their neighborhoods to drive the interactions and reduce communication overhead.

This *context* is any information that is used to characterize the situation of an entity [2]. Traditional approaches to context are largely egocentric. Our own work on Grapevine [3], [8] attempts to rethink this approach by enabling *shared* local views of various context metrics. The work in this paper builds on the Grapevine approach to focus explicitly on aggregation and how knowledge about the contents of aggregates can positively influence the nature of communication to make the communication more accurate and result in lower overhead.

Autonomous monitoring of physical processes, such as collecting and aggregating samples from sensors, are often deficient in one or more aspects of network communication or computation (e.g., aggregation) [17]. The inherent capabilities of WSANs make them ideal for detecting physical phenomena [11]. Using WSANs, applications can monitor the behavior of physical systems to determine whether given behaviors are *safe* or of high *quality* by writing global invariants that capture these properties [15]. Our approach targets these WSANs, and shares many goals with these distributed invariant monitoring systems. However, we focus on efficiently sharing information using aggregation and dissemination of contexts among all participating nodes; further, we also focus on expressive disaggregation, allowing the recovery of some fine-grained context information of individual nodes, even when only aggregate information is shared.

Aggregation has been a popular research topic in WSANs for enabling higher throughput with lower energy consumption. We do not exhaustively review these approaches here; [4] provides a detailed review. Tree-based (or hierarchical) structures are based on routing algorithms with a tree rooted at a sink where the aggregation is targeted. Directed diffusion [6] is a reactive data-centric protocol whose routing scheme is directed by a sink (or multiple sinks). Our approach is similar in the use of aggregation, but we focus on generating this aggregate view at *every* node in the network instead of simply at the designated sink(s).

III. Preliminaries

Before we address the challenges described above, we first define some terms we will use throughout the paper and the notations we will use to explain our algorithms.

Single value: any kind of data: sensed and processed physical, environmental or situational data from a node.

Aggregate value: aggregation of values from multiple nodes.

Context: a tuple of four elements: a single value, cohort, hop-count, sample time.

Cohort: (of an aggregated value) the set of nodes that contribute the single values that comprise an aggregate value.

Member: (of a cohort) a node that belongs to a cohort.

Prime cohort: (defined relative to another cohort or cohorts) a cohort in which none of its members is a member of any of a set of other cohorts

Prime context: an aggregate context whose cohort is prime.

Subcontext: a context whose cohort is a subset of another context's cohort.

Notations. Throughout the remainder of this paper, we will use in-text graphics to demonstrate how our algorithms and protocols function. In those graphics, we use the following conventions. We represent a single value as a circled number. We use the node id as a stand-in for the actual (sensed) value a node contributes to the aggregate. For example, ⑤ represents the value sensed by node with id 5. We represent a cohort (and the cohort's aggregated value) with a circled set of node ids. For example, $\boxed{1, 2, 3, 4, 5}$ represents the aggregate value that results from combining the sensed values from nodes 1, 2, 3, 4, and 5. We represent a context value (whether it is a single context value or an aggregated context value) using lower case letters and a set of context values using capital letters.

Assumptions. We assume context dynamics change more slowly than the timescale of aggregation, i.e., we assume that the speeds of computation and communication are much faster than the change of the context values we sample. We assume that the nodes' mobility dynamics are slower than the timescale of aggregation, that is, we assume a single "snapshot" of the network over which we attempt to generate an aggregate value.

Cohort and context operations. Finally, in the latter sections of this paper, we take for granted the ability to perform some relatively standard operations over contexts and cohorts. We use $+$ to represent the merging (i.e., aggregation) of two contexts, e.g., $c = a + b$ indicates aggregating contexts a and b, to create a new aggregated context c. The cohort of context c is the union of the cohorts of a and b. The nature of this merge is defined by the nature of the aggregation itself; some aggregate measures (e.g., min/max) are *duplicate insensitive* and will not lose information if the cohorts of a and b are not prime relative to each other. On the other hand, some aggregate measures (e.g., mean) are *duplicate sensitive*; averaging two averages that are not prime relative to each other will impact the ultimate accuracy of the computation. We use $-$ to indicate a disaggregation of aggregate context, i.e., $c = c_1 - c_2$ creates a context c by removing the cohort of c_2 from the cohort of c_1. It is required that $c_2 \subset c_1$. For example, $\boxed{1,2,3,4}$ - $\boxed{1,2}$ = $\boxed{3,4}$. With enough available cohorts, it is often possible to regenerate a single context from a set of aggregates, allowing nodes to completely recover other nodes' individual contexts. The $|\cdot|$ operator returns the size of a context's cohort; e.g., for a context c with cohort $\boxed{1,2,3}$, $|c| = 3$.

IV. Approach

We aim to (1) disseminate aggregate information in a wirelessly connected environment that allows each node in the network to achieve a shared view of some aggregate measure of the global context and (2) enable individual nodes to recover the individual context measures for other nodes in the network by exchanging only aggregate information. In our previous work, we showed that nodes must carefully control which aggregate information is disseminated in order to *protect* their own private information [19]. In this work, we turn that on its head and look at how the diversity present in opportunistically created aggregates can enable very fine-grained data recovery.

Aggregation can support assessing rules written about some shared spatiotemporal state or for measuring the distance of some local value from the aggregate. Aggregation is also useful from a networking perspective in that it can drastically reduce the amount of raw data shared on resource-constrained wireless links. On the other hand, disaggregation, or the recovery of individual values from a set of aggregate values, can be just as useful in many applications, for example, in identifying the locations of anomalous sensed conditions.

We only aggregate *prime contexts*, i.e., contexts that do not contain overlapping cohorts. Imagine that, at some point during execution, node with id 1 has received two aggregate contexts from its neighbors: $\boxed{2,3,4}$ and $\boxed{4,5}$. Considering these and the node's own individual context, $\boxed{1}$, the node can generate two distinct views of the aggregate context: $\boxed{1,2,3,4}$ or $\boxed{1,4,5}$. Naïvely, $\boxed{1,2,3,4}$ is "better" simply because its cohort is larger. This may not be universally true, and occasionally opting to share aggregate contexts with smaller cohorts may provide a measure of information diversity in the aggregate computation [19]. In disaggregation, on the other hand, multiple aggregates enable the identification of smaller subcontexts, potentially enabling the resolution of individual context values, even when those values are shared only within aggregate contexts. Consider a case where a node has, at some point received aggregated contexts $\boxed{1,3}$, $\boxed{1,2}$, and $\boxed{1,2,3}$. From $\boxed{1,2,3}$ - $\boxed{1,2}$, the node can easily recover the exact value of the single context $\boxed{3}$. In turn, from $\boxed{1,3}$ - $\boxed{3}$, the node can recover $\boxed{1}$; $\boxed{1,2}$ - $\boxed{1}$ gives $\boxed{2}$.

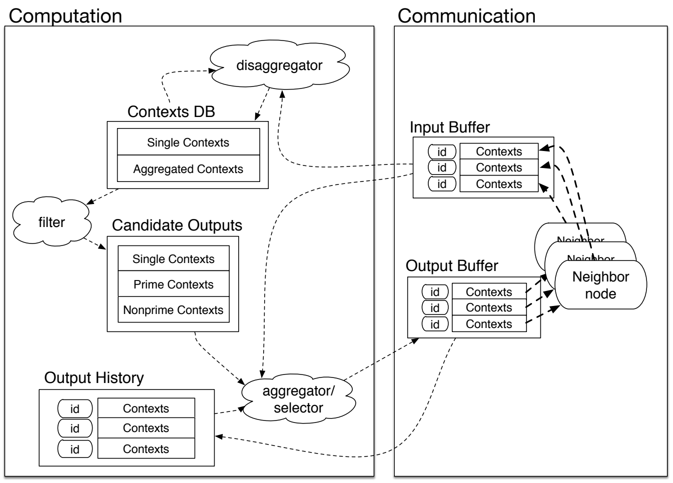


Fig. 1: Data Structures

Fig. 1 shows an architectural view of our approach. Each node places received context information in an *Input Buffer*. Periodically, the *Input Buffer* is processed, generating a new local view of the aggregate context and possibly a set of output context information that the node will share with its neighbors. When the node begins processing received context, it is sent to the DISAGGREGATOR to retrieve the cohorts of smallest possible size (ideally single contexts); the node stores the recovered contexts in the *Contexts DB*. A FILTER is applied to the *ContextDB* to generate a set of *Candidate Outputs*, potentially based on application-provided constraints. An AGGREGATOR and a SELECTOR then choose a set of contexts to share with neighboring nodes. For each neighbor, a node chooses the aggregate context that provides the particular neighbor the most additional information based on the known history of context shared with that neighbor. This is computed by comparing the available aggregates from the *Contexts Database* to the previously shared contexts stored in the *Contexts History*; the latter maps a neighbor node's id to the contexts that this node has previously shared with it. By consulting this mapping before sending new aggregates, this node can determine the degree of information that is added by sharing the newly constructed aggregate.

Because we assume that context information changes more slowly than the network's ability to communicate it and compute a shared global aggregate, we assume that each node eventually reaches a *steady state* with respect to its knowledge about the aggregate value before computing another aggregate. Each node periodically samples some contextual value, which it then attempts to share and aggregate alongside the values sampled by the other nodes in the network. A node has reached a steady state when no additional context exchanges will improve its knowledge about the aggregate; practically, we say a node has reached steady state when either a specified amount of time t has passed without the node's view of the shared state changing or the last k received pieces of context have not changed the node's view of the aggregate.

Algorithm 1 shows a macroscopic view of a single node's execution; this codifies the process depicted in Fig. 1. As long as the node has not achieved a steady state view of the target aggregate context measure, the node continues processing in three steps: (1) the node transmits available context information (lines 2-4); (2) the node collects context from neighboring nodes (lines 5-8); (3) the node processes received context and updates the local state of the aggregate (lines 9-12).

Algorithm 1: Overall algorithm

```
1  while $\neg$steadystate do
2      for $(n, c) \in$ OutputBuffer do
3          $send(n, c)$
4          $OutputHistory.insert(n, c)$
5      while $\neg$timeout and $\neg isFilled(InputBuffer)$ do
6          if $receive(n, c)$ then
7              $c.\eta \leftarrow c.\eta + 1$
8              $InputBuffer.insert(c)$
9      DISAGGREGATE$(ContextDB, InputBuffer)$
10     FILTER$(ContextDB, applicationFilterConstraints)$
11     $aggContext \leftarrow$ AGGREGATE$(CandidateOutputs)$
12     SELECT$(OutputHistory, aggContext)$
```

Initially, the *Output Buffer* contains only the node's single context, i.e., its own sampled value, and the first time through the outer loop in **Algorithm 1**, the node simply sends its own sampled value to each neighbor. In lines 5-8, the node collects context shared by its neighbors. This process is controlled by a pair of parameters: a timeout value and the size of the *Input Buffer*. When the node receives a piece of context, it increments the context packet's hop count (η) and then inserts the context into the *InputBuffer*. Once the node either collects enough context information from neighboring nodes to fill the *Input Buffer* or it has waited *timeout* time to receive context information, the node proceeds to the computation phase (lines 9-12). The computation phase strings together four sub-algorithms: DISAGGREGATE, FILTER, AGGREGATE, and SELECT, which in turn generate the set of (aggregate) contexts to send to the neighbors in the next iteration of the outer loop.

Algorithm 2: DISAGGREGATE

```
1  $C \leftarrow ContextDB \cup InputBuffer$
2  while $\exists c, c' \in C$ such that $members(c) \subset members(c')$ do
3      $C \leftarrow sortByIncreasingSize(C)$
4      $c \leftarrow C.first$
5      while $\neg(\exists c' \in C$ such that $members(c) \subset members(c'))$
       do
6          $c \leftarrow C.next$
7      $super \leftarrow \{c' : members(c) \subset members(c')\}$
8      $C \leftarrow (C \cup split(c, super)) - super$
9  $singles \leftarrow \{c \in C : isSingle(c)\}$
10 $aggregates \leftarrow \{c \in C : \neg isSingle(c)\}$
11 $ContextDB \leftarrow (singles, aggregates)$
```

When new context information is processed, the first step is to attempt to disaggregate the known context information into the smallest cohorts possible. Ideally the node eventually receives enough information to completely recover the individual context values for every other node in the network. **Algorithm 2** starts by using the context with the smallest cohort (c), finds all possible cohorts that are larger than c, and attempts to use c to SPLIT the larger cohorts into smaller ones. The DISAGGREGATE algorithm continues this process until it has examined all possible combinations of c and c' such that *members*$(c) \subset$ *members*(c'). It relies on the SPLIT algorithm in **Algorithm 3**. At the end, DISAGGREGATE separates the single contexts from the aggregate ones and updates the contexts stored in the *ContextDB*. Many optimizations can speed up the execution of DISAGGREGATE; in the **Algorithm 2** listing, we omit these to focus purely on the function. Our implementation described later takes copious advantage of these optimizations.

Algorithm 3: SPLIT

```
   Input: A context $c$, and a set of contexts $C$
   Output: A set of contexts results
1  $results = \{\}$
2  for $c'$ in $C$ do
3      $results \leftarrow results \cup \{(c' - c)\}$
4  return results
```

While in the default case, the goal of sharing aggregate information may be simply to generate the largest amount of representative information at each node, applications can influence which specific contexts are shared between nodes by applying a filter. For example, an application may impose a condition that selects single contexts to share based on their content or a special tag (e.g., "alarm"). Applications can also limit the size of a node's *Output Buffer* to reduce communication overhead. As a simple example, one might restrict individual context information to not propagate further than a specified hop limit, while aggregate information can propagate indefinitely. **Algorithm 4** shows the implementation of such a filter. This filter relies on a check *isPrime* (lines 4 and 5), which takes a context c and a set of context C. The *isPrime* check returns *true* if and only if either (1) c is a single context or (2) for all $c' \in C$, *members*$(c) \cap$ *members*$(c') = \emptyset$.

Algorithm 4: Hop Count Based FILTER

```
   Input: hop limit max
1  $singles \leftarrow ContextDB.SingleContexts$
2  $aggregates \leftarrow ContextDB.AggregateContexts$
3  $selectedSingles \leftarrow \{c \in singles : c.\eta < max\}$
4  $primes \leftarrow \{c \in aggregates : isPrime(c, aggregates)\}$
5  $nonPrimes \leftarrow \{c \in aggregates : \neg isPrime(c, aggregates)\}$
6  $CandidateOutputs \leftarrow (selectedSingles, primes, nonPrimes)$
```

Given the filtered set of contexts, the aggregate algorithm finds the maximum aggregate context this node can create from its aggregate context information. When the filtered aggregates contain only single contexts and prime contexts, the algorithm is simple: we just combine the singles and prime contexts. When non-prime contexts are available, we may want to select from them to increase the available information. **Algorithm 5** shows the aggregate algorithm.

Computing the maximum cover (line 4 in **Algorithm 5**) is NP hard [13]; using a brute force search takes hours when the number of input non-prime contexts exceeds 25. Therefore, we introduce heuristic approaches to bypass this complexity. We execute two different heuristics, compare the results of the two, and accept the result with higher coverage. The first heuristic takes a greedy approach, selecting the largest cohort member from the given set of non-prime context inputs. It then removes all overlapping contexts from the set and repeats the greedy choice. The second heuristic selects the non-prime context inputs that result in the largest disjoint set. Specifically, this heuristic scores each non-prime context by computing the difference in the size of the cohorts that it overlaps and the size of the cohorts that it does not overlap. Consider a set of three non-prime contexts: c_1: $\boxed{1,2,3}$, c_2: $\boxed{4,5,6}$, and c_3: $\boxed{1,2,3,4}$. The first heuristic will choose c_3 and then will be unable to choose either c_1 or c_2 and will simply return. On the other hand, the second heuristic will compute a score for c_1 as $|c_2|-|c_3| = -1$ since the c_2 is disjoint with respect to c_1 but c_3 is not. Similarly, the heuristic will compute a score of -1 for c_2 and -6 for c_3. As a result, the heuristic selects contexts c_1 and c_2. Neither heuristic always results in the optimal solution, hence we execute both and select the best option.

Algorithm 5: AGGREGATE

```
Output: Aggregate Context aggContext
singles ← CandidateOutputs.singleContexts
primes ← CandidateOutputs.primeContexts
nonPrimes ← CandidateOutputs.nonPrimeContexts
selectedNonPrimes ← maximumCover(nonPrimes)
aggregateContext ← singles ∪ primes ∪ selectedNonPrimes
return aggregatedContext
```

The final stage in **Algorithm** 1 is to select the specific subcontext of the aggregate to send to each neighbor. Once the contexts to send are selected, the *Output Buffer* stores the context to send to each neighbor, which is picked up on the subsequent iteration of the outer loop in **Algorithm 1**. SELECT, shown in **Algorithm 6**, compares the aggregate context computed in the previous step to the *OutputHistory* for each node to determine whether the computed aggregate offers new information to the neighbor. SELECT compares the cohort of the newly computed aggregate context against the cohort of the aggregate contexts previously sent to the neighbor. If the new aggregate adds information, it is sent.

Algorithm 6: SELECT

```
Input: Aggregate Context aggContext
for $n$ in neighbors do
    history ← OutputHistory.get($n$)
    receivedContexts ← InputBuffer.get($n$)
    knownSingles ← getSingles(history ∪ receivedContexts)
    newSingles ←
            CandidateOutputs.SingleContexts − knownSingles
    knownAggs ← getAggregates(history ∪ receivedContexts)
    novelCohortMembers ←
            cohort(aggContext) − cohort(knownAggregates)
    if |novelCohortMembers| > 0 then
        newAggregates ← aggContext
    else
        newAggregates ← {}
    OutputBuffer.put($n$, newSingles ∪ newAggregates)
```

V. ANALYSIS

We perform both an analysis and a set of simulations over networks modeled on real-world scenarios. We start with the analysis, targeted at two styles of networks: trees that are guaranteed to not contain cycles and mesh networks where cycles are possible. We compare the use of the algorithms described in the previous case to a context sharing approach in which nodes exchange only single context information. We assume perfect networks in which packets are never lost or delayed, and we assume that the nodes operate in a synchronized manner (i.e., we assume that all of the nodes completes an iteration of the **Algorithm 1** before any continue to the next iteration); we weaken these assumptions when reporting simulation results in the next section. We quantify the following metrics:

communication overhead: the total amount of context information exchanged. We also compute the reduction in communication overhead as $100 \times (s - a)/s$, where a and s are the communication overheads for the aggregate and single approaches, respectively.

speed: the average number of iterations of the loop in **Algorithm 1** required to reach steady state.

identification rate: the average ability of an individual node to account for other individual values in its computed aggregate. As an example, in a network of ten nodes, when node 1 recovers the context values ②, ③, and $\boxed{4,5,6}$, the identification rate for node 1 is 60% since node 1's maximum aggregate is $\boxed{1,2,3,4,5,6}$.

accuracy: the difference between calculated aggregate value and the correct aggregate value. It is calculated as $100 \times (a - e)/a$, where a is the correct aggregate value and e is the calculated aggregate value.

Fig. 2 shows an example of a tree network. For our analysis, we consider a value of $\tau = 1$, where τ is the max hop count applied in the FILTER algorithm shown in **Algorithm 4**. When $\tau = 1$, single context values are shared only with neighbors; any other recovery of a node's individual context is done through disaggregation. The ovals with dashed borders shows the nodes that receive single contexts for node 3 and node 7.

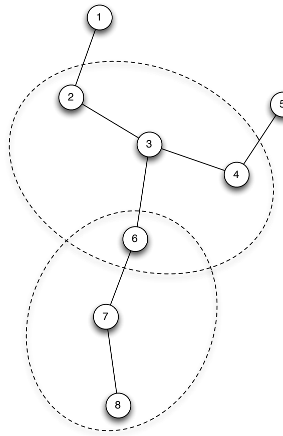


Fig. 2: Tree network

Table I shows the contexts stored, sent, and received by nodes 3 and 7. Each entry contains a row showing the nodes' *ContextDB*, its *Input Buffer* at the beginning of the iteration, and its *Output Buffer* at the end of the iteration. Numbers circled in black are newly processed context information in the given iteration, while white circles indicate information known in the previous iteration. For example, in the final iteration shown, node 7 knows the individual context values for nodes 3, 6, 7, and 8; it knows the aggregate value combining nodes 2 and 4; and it has just learned the aggregate value for nodes 1 and 5. The small square labels indicate the node from which the indicated information was received or to which the information will be sent. For example, **2** $\boxed{1,2,3}$ indicates that aggregate $\boxed{1,2,3}$ should be sent to node 2.

By working the packet transmissions by hand as shown in Table I, we achieve a communication overhead reduction of 25%: with 100% accuracy and 100% identification rate in this specific network. We provide detailed analytical results below, but we first examine our algorithms' behavior. We first prove that single contexts can be recovered from aggregated contexts in a network with nodes connected in series.

Lemma 1: Consider a line network in which $\tau = 1$ (i.e., nodes communicate single context information only to their

		Node 3	Node 7
i1	DB	12❸45678	123456❼8
	In	$\boxed{2}$② $\boxed{4}$④ $\boxed{6}$⑥	$\boxed{6}$⑥ $\boxed{8}$⑧
	Out	■2,4,6 ③	■6,8 ⑦
i2	DB	1❷③❹5❻78	12345❻⑦❽
	In	$\boxed{2}$ (1, 2, 3) $\boxed{4}$ (3, 4, 5) $\boxed{6}$ (3, 6, 7)	$\boxed{6}$ (3, 6, 7)
	Out	■2,4,6 (2, 3, 4, 6)	■6,8 (6, 7, 8)
i3	DB	❶②③④❺⑥❼8	12❸45⑥⑦⑧
	In	$\boxed{6}$ (2, 3, 4, 6, 7, 8)	$\boxed{6}$ (2, 3, 4, 6, 7, 8)
	Out	■2,4,6 (1, 2, 3, 4, 5, 6, 7)	■8 (3, 6, 7, 8)
i4	DB	①②③④⑤⑥⑦❽	1 ●(2, 4) ③5⑥⑦⑧
	In		$\boxed{6}$ (1, 2, 3, 4, 5, 6, 7, 8)
	Out	■2,4 (1, 2, 3, 4, 5, 6, 7, 8)	■8 (2, 3, 4, 6, 7, 8)
i5	DB	①②③④⑤⑥⑦⑧	●(1, 5) (2, 4) ③⑥⑦⑧
	In		
	Out		■8 (1, 2, 3, 4, 5, 6, 7, 8)

TABLE I: Recovered contexts in tree network

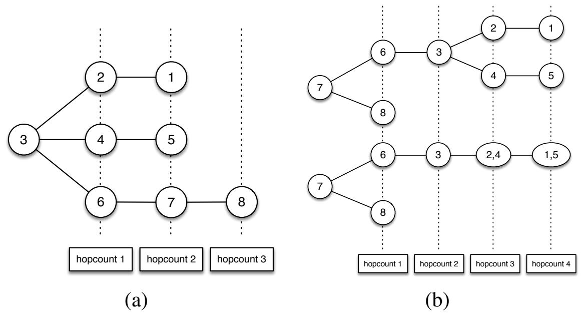


Fig. 3: Topological re-arrangements. (a) From the perspective of node 3. (b) From the perspective of node 7.

two immediate neighbors). When all other context sharing is done via aggregate information, a terminal node (i.e., one located at one of the two ends of the line) can successfully disaggregate the single contexts of all nodes located at k hop distance after k iterations of **Algorithm 1**.

Proof: Let node a be a terminal node, and let $a+k$ be a node located at k hops from node a. At iteration 1, the single context from node $a+1$ is received. In iteration 2, a receives the aggregated value from node $a+1$, which is an aggregate of the values of nodes a, $a+1$, and $a+2$. Because a has the single values of a and $a+1$, it can recover the individual value of $a+2$ via disaggregation. Likewise, at iteration step k, a can compute the single values from nodes a up to $a+k$. ■

We can extend this to arbitrary trees by considering a tree to be made up of multiple line networks. For example consider node 3 as a terminal network of three such line networks, as shown in Fig. 3(a). Contexts from nodes 2, 4, and 6 are recovered individually in the first iteration. Single contexts from nodes 1, 5, and 7 are similarly recovered in the next iteration, and so on. Fig. 3(b) shows the network from the perspective of node 7. Notice that contexts from nodes 2 and 4 are in a sub-branch of node 3, and they are not recovered as single contexts but as an aggregated context. The right most column of Table I shows the recovered values from node 7. After the second iteration, single contexts from nodes 6 and 8 are recovered. A single context 3 is recovered at the subsequent

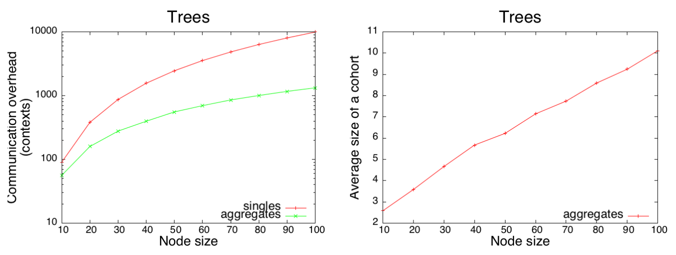


Fig. 4: (a) Comm. overhead. (b) Average number of cohorts

iteration, followed by the aggregate contexts (2, 4) and (1, 5).

In a tree network, the recovered aggregates are always prime. Because all of the contexts are prime, we always achieve a 100% identification rate and 100% accuracy. Further, tree networks achieve a significant reduction in communication overhead, since this overhead depends on the sum of the maximum hop counts for each context delivery path. Specifically, we can prove that with a tree network with n elements, the communication overhead is bounded by $n \times (n-1)$.

Lemma 2: In a tree network of n nodes, the communication overhead required to achieve 100% identification rate and accuracy in all of the nodes is bounded by $n \times (n-1)$.

Proof: (By induction.) Base case: when $n = 2$, the communication overhead is 1×2: there is one edge and two nodes' (single) context information is communicated. Induction step: assume that, when $n = k$, the lemma holds, i.e., that for k nodes, the communication burden is bounded by $k(k-1) = k^2 - k$. When we add one more node ($n = k+1$), the added communication burden from the newly added node to the existing node is bounded by the number of edges, which is k in a tree of $k+1$ nodes. The added communication burden to the newly added node is also k The total communication burden is therefore bounded by $k^2 - k + 2k = (k+1)(k+1-1)$. Thus the lemma holds for $n = k+1$. ■

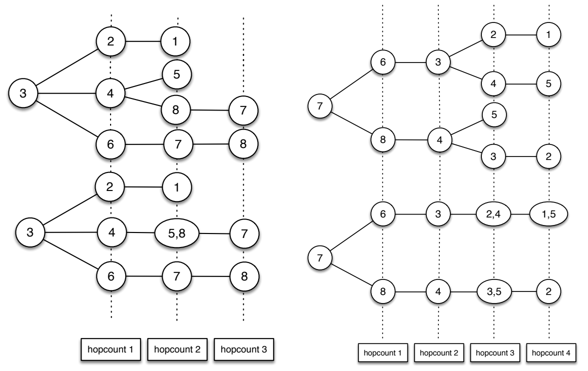
Using this lemma, and from the communication overhead as the sum of maximum hop count from each path, we can compute the communication overhead reduction rate (relative to simply communicating single contexts) when $\tau = 1$ as:

$$Gain = 1 - \frac{\sum_{i=1}^{n} h(i)}{n \times (n-1)}$$

where $h(i)$ is the sum of the lengths of all disjoint branches when the tree is rooted at node i. For example, considering node 3 in Fig. 3(a), $h(3) = 2+2+3 = 7$, while $h(7) = 4+1 = 5$. Computing this for all n nodes in our tree in Fig. 2, we can compute a gain of $100 \times (1 - 42/56) = 25\%$, which matches our computation reported earlier.

The example tree network we have used is just that: an example. We next provide a set of numerical analyses that examine these and similar properties for tree networks more generally. We randomly generated trees ranging in size from 10 to 100 nodes. For each network size, we created 100 different random topologies with varying depths and widths. We executed our algorithms and computed the reduction in communication overhead, the identification rate, the speed, and the average number of cohorts. Fig. 4 shows the results for three of these metrics against increasing network size.

Fig. 4(a) shows, on a logarithmic scale, the growth of the singles only communication overhead and the aggregate context communication. The reduction rate increases with



(a) (b)

Fig. 5: Topological re-arrangements. (a) From the perspective of node 3. (b) From the perspective of node 7.

		Node 3	Node 7
i1	DB	12❸45678	123456❼8
	In	$\boxed{\scriptstyle 2}$② $\boxed{\scriptstyle 4}$④ $\boxed{\scriptstyle 6}$⑥	$\boxed{\scriptstyle 6}$⑥ $\boxed{\scriptstyle 8}$⑧
	Out	$\colorbox{black}{\color{white}\scriptstyle 2,4,6}$③	$\colorbox{black}{\color{white}\scriptstyle 6,8}$⑦
i2	DB	1❷③❹5⑥78	12345❻⑦❽
	In	$\boxed{\scriptstyle 2}\boxed{1,2,3}$ $\boxed{\scriptstyle 4}\boxed{3,4,5}$ $\boxed{\scriptstyle 6}\boxed{3,6,7}$	$\boxed{\scriptstyle 6}\boxed{3,6,7}$ $\boxed{\scriptstyle 8}\boxed{4,7,8}$
	Out	$\colorbox{black}{\color{white}\scriptstyle 2,4,6}\boxed{2,3,4,6}$	$\colorbox{black}{\color{white}\scriptstyle 6,8}\boxed{6,7,8}$
i3	DB	❶②③④$\colorbox{black}{\color{white}5,8}$⑥❼	12❸❹5⑥⑦⑧
	In	$\boxed{\scriptstyle 4}\boxed{2,3,4,5,6,7,8}$ $\boxed{\scriptstyle 6}\boxed{2,3,4,6,7,8}$	$\boxed{\scriptstyle 6}\boxed{2,3,4,6,7,8}$ $\boxed{\scriptstyle 8}\boxed{3,4,5,6,7,8}$
	Out	$\colorbox{black}{\color{white}\scriptstyle 2,4,6}\boxed{1,2,3,4,5,6,7,8}$	$\colorbox{black}{\color{white}\scriptstyle 6,8}\boxed{3,4,6,7,8}$
i4	DB	①②③④⑤⑥⑦❽	1❷③④❺⑥⑦⑧
	In	$\boxed{\scriptstyle 6}\boxed{1,2,3,4,5,6,7,8}$	$\boxed{\scriptstyle 6}\boxed{1,2,3,4,5,6,7,8}$ $\boxed{\scriptstyle 8}\boxed{2,3,4,5,6,7,8}$
	Out		$\colorbox{black}{\color{white}\scriptstyle 6,8}\boxed{2,3,4,5,6,7,8}$
i5	DB	①②③④⑤⑥⑦⑧	❶②③④⑤⑥⑦⑧
	In		
	Out		$\colorbox{black}{\color{white}\scriptstyle 8}\boxed{1,2,3,4,5,6,7,8}$

TABLE II: Recovered contexts in mesh network

increasing network size, from a 37.77% reduction, on average, to an 86.63% reduction, on average. We do not plot the identification rate, as it is 100% for all networks for both single and aggregate scenarios. The speed for both single and aggregated scenarios are the same: between 5.01 to 11.73 iterations of **Algorithm 1**. Finally, Fig. 4(b) shows the average size of a cohort as the network size increases. As expected, both the number of cohorts (not plotted) and the average size of a cohort grow linearly with the network size.

Mesh networks contain cycles; consider for example the same tree network as in Fig. 2 with one additional edge, the edge between nodes 4 and 8. Fig. 5 shows rearrangements of this network from the perspectives of nodes 3 and 7. The cycles in the mesh network cause redundant distribution of context information because there are now potentially multiple paths connecting any two given nodes. For example, in our example mesh network, node 3 receives context information from node 8 in aggregate form via paths through node 4 and node 6. Table II shows the iterations of context receptions when **Algorithm 1** executes on our example network.

In many cases, these extra connections also produce non-prime contexts, which was not the case in tree networks. Consider the segment of a mesh network shown in Fig. 6. Via nodes 2 and 4, respectively, node 1 will receive the aggregate contexts $\boxed{1,2,3,5}$ and $\boxed{1,4,5,6}$. The single contexts ② and ④ that node 1 has received previously enable it to split these contexts into smaller, non-prime contexts $\boxed{3,5}$ and $\boxed{5,6}$. When the maximum cover algorithm is executed during the AGGREGATE algorithm, one of the two non-primes is selected as they have the same member size.

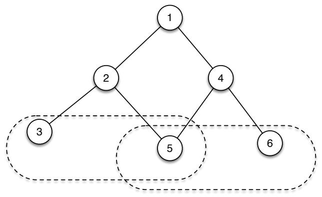


Fig. 6: Non-prime contexts generated in mesh network

The DISAGGREGATE algorithm's use of *sortByIncreasingSize* causes deterministic consideration of the contexts that have the same size. For example, in a deterministic implementation, when we have two non-prime contexts $\boxed{1,2,3}$ and $\boxed{3,4,5}$ that we use to split $\boxed{1,2,3,4,5,6,7}$, we always use subcontext $\boxed{1,2,3}$ before $\boxed{3,4,5}$ to get $\boxed{4,5,6,7}$. This deterministic strategy often leads to early detection of steady state, as nodes cannot find new information to send even when they have the contexts with the information that neighbor nodes do not have. This is often the case in a dense mesh network or a mesh network with a large number of nodes. We can modify our previous algorithms to remove this determinism. More specifically, we use information about the contexts already sent (i.e., the *Output History*) to inform the order of context consideration in DISAGGREGATE. In addition, we can modify the SELECT algorithm to, instead of sending the largest possible aggregate to the neighbor node (as in lines 8-9 in **Algorithm 6**) send one of the non-prime contexts that contains information that has not been sent previously.

Starting with the tree networks that had an average of 1.94 neighbors per node, we generated in the previous subsection, we connected 5% and 20% of the nodes in the trees randomly to generate both *light* mesh networks that had an average of 2.05 neighbors per node and *dense* mesh networks with an average of 2.34 neighbors per nodes.

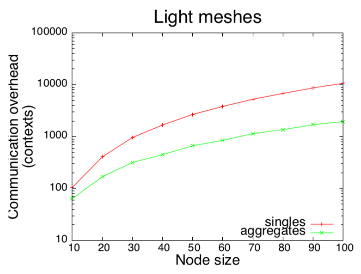


Fig. 7: Light mesh networks communication overhead

Fig. 7 shows the communication overhead reduction for light mesh networks where the number of nodes in the network ranges from 10 to 100. The reduction rate shows a similar pattern as in the tree networks, with a 39.0% reduction in networks with 10 nodes and a 81.71% reduction in 100 node networks. Fig. 8(a) shows that the identification rate is between 77.58% (100 nodes) and 98.97% (10 nodes). Out of the 77.58% identification, we get 4.8% identification rate from single contexts. Finally, Fig. 8(b) shows the average speed with which the protocols reach the steady state; we also computed the average speed decrease as $100 \times (s_a - s_s)/s_s$ when s_a is the speed for aggregated communication, and s_s is the

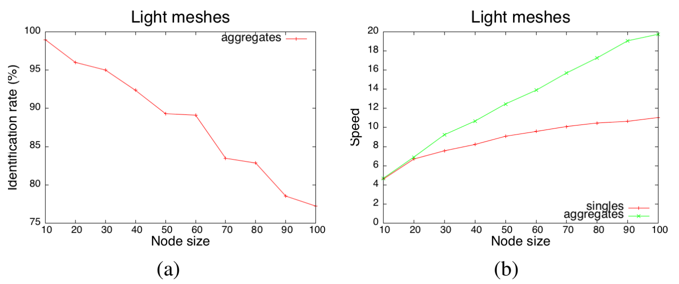


Fig. 8: Analytical results for light mesh networks. (a) Identification rate. (b) Speed

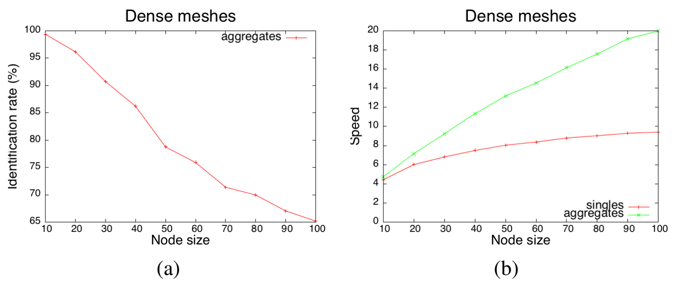


Fig. 10: Analytical results for dense mesh networks. (a) Identification rate. (b) Speed

speed for singles only communication. The calculated speed decrease is between 1.7% and 78.8% for the aggregate context communication relative to exchanging only single contexts. The larger gap with node size increase is caused by the sending of non-prime contexts.

Fig. 9 shows the communication overhead for the dense mesh networks, while the identification rate and speed are shown in Fig. 10. The identification rate and speed are poorer than in the light networks; this is due to the even more creation of the non-prime contexts from the non-disjoint communication paths.

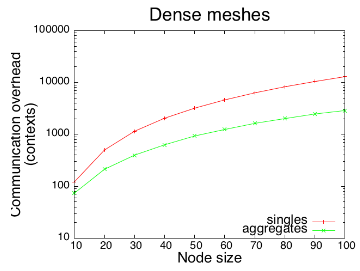


Fig. 9: Light mesh networks communication overhead

VI. Simulation

We next describe our simulations, which were driven by real world data and network topologies. We used the ONE network simulator [10]. We used two sets of data; one is sampled data from a wireless sensor network in the Intel Berkeley Research lab [1]. This data set contains physical sensor information such as temperature, humidity, light, and voltage from 54 sensor nodes deployed in an office building. We extracted periodic temperature data from all of the 54 sensors to drive our simulation. We used the sensors' location data to create two network topologies: one in which nodes were assigned a 6m communication radius (termed i54l, for "Intel 54 light mesh"), and another in which the nodes were assigned a 10m communication radius (termed i54d, for "Intel 54 dense mesh"). We also created an artificial tree network from i54l by randomly removing the cycles (termed i54t for "Intel 54 tree"). From the first mesh network, we removed the cycles to have a tree network: i54t (intel 54 nodes tree). The three resulting network topologies are shown in Fig. 11.

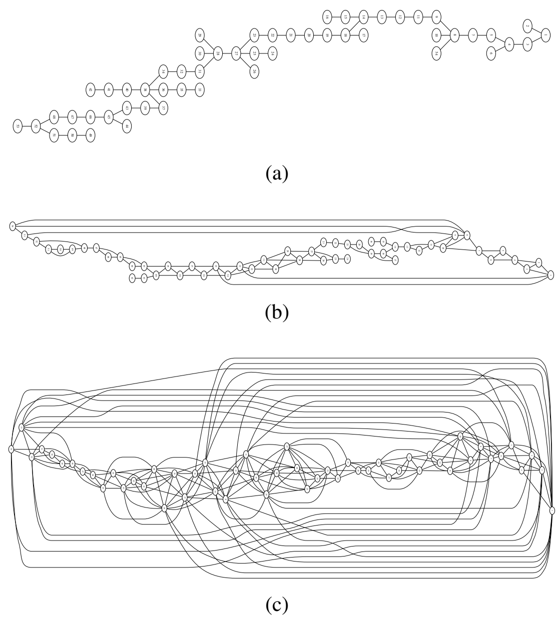


Fig. 11: Intel testbed networks. (a) i54t. (b) i54l. (c) i54d.

As a second set of data, we used a tool to create spatially correlated data [7][1]. We created spatially correlated data for randomly generated networks of 49 and 100 nodes, with densities ranging from 2 to 4.5 neighbors per node. We refer to these simulated networks as p49t, p49l, p49d, p100l, p100t: a tree, light mesh, and dense mesh of 49 nodes each, and a light mesh and dense mesh of 100 nodes each.

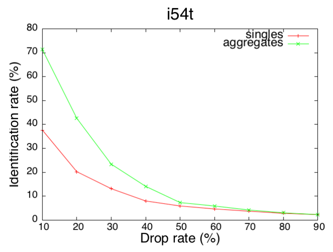
Table III shows the complete results. For size reduction and speed decrease, the numbers are relative to disseminating only single context values.

TABLE III: Simulation results

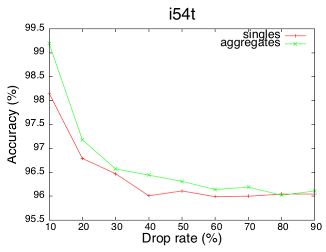
Name	Size reduction	Id rate	Avg accuracy	Speed decrease
i54t	45.1%	100%	100%	0%
i54l	66.0%	94.9%	99.84%	18.6%
i54d	87.2%	90.7%	99.66%	25.5%
p49t	54.71%	100%	100%	0%
p49l	52.64%	92.79%	99.83%	42.52%
p49d	59.57%	84.83%	99.61%	32.78%
p100l	61.98%	82.78%	99.70%	55.23%
p100d	76.94%	72.90%	99.64%	69.76%

The results correspond to our analysis in the previous section. This is expected, because these initial simulations used "perfect" network links. We also simulated under increasingly adverse network conditions, which we quantify as the likelihood of a packet being dropped. Fig. 12 shows the impact of these network conditions in the identification rate and accuracy in tree networks. Our smart aggregation and disaggregation was more resilient to these faults than communicating single context information alone.

[1]To control the spatial correlations, we used the authors' suggested $\alpha = 0.5$ and $\beta = 0.001$ to ensure a very high degree of spatial correlation since our motivating examples are likely to exhibit such high spatial correlations.

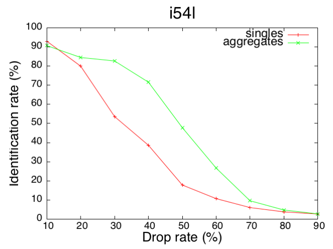


(a)

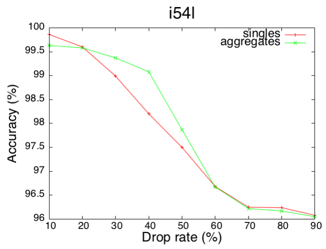


(b)

Fig. 12: Impact of dropped packets in tree networks. (a) Identification rate. (b) Accuracy.



(a)



(b)

Fig. 13: Impact of dropped packets in mesh networks. (a) Identification rate. (b) Accuracy.

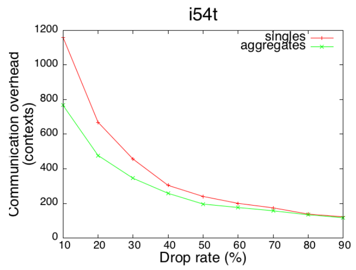


Fig. 14: Comm. overhead

Fig. 13 shows the same metrics in the (light) mesh case. We omit the remaining plots (for dense mesh and for the spatially correlated networks) for brevity; the results show the same trends. Especially in the case of mesh networks, the gains are due to the fact that there is some redundancy in the delivery of aggregates that end up being non-prime relative to each other. This redundancy provides inherent fault-tolerance to our smart aggregation and disaggregation protocols. In addition, in all cases, we achieved these gains in identification rate and accuracy while also achieving modest improvements in communication overhead, as shown, for one exemplar (the tree network) in Fig. 14.

VII. Conclusion

Decentralized context sharing applications open new opportunities by sharing individual information through many-to-many communication. The shared information can help build a shared global view so that each node can make autonomous decisions and initiate necessary actions. However, sharing individual contexts among all the participants can increase the communication burden dramatically. In this paper, we showed that smart approaches to aggregation and disaggregation can garner expressive representations of both aggregate and individual contexts, at a reduced overhead in comparison to simply sharing single context information directly. We benchmarked the detailed differences between the performance of these smart aggregation and disaggregation schemes on both tree networks such as those that arise because of sophisticated underlying routing structures and on mesh networks that do not enjoy such support. We also showed that our smart aggregation approaches are more resilient to the common network faults that occur in these highly distributed and difficult to maintain and control networks.

Acknowledgments

This work was funded, in part, by the National Science Foundation (NSF), Grant #CNS-1218232. The views and conclusions herein are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

References

[1] Intel Lab Data. http://db.csail.mit.edu/labdata/labdata.html. Accessed: 2014-04-30.

[2] G. Abowd, A. Dey, P. Brown, N. Davies, M. Smith, and P. Steggles. Towards a Better Understanding of Context and Context-Awareness. In *Handheld and Ubiquitous Computing*, volume 1707 of *Lecture Notes in Computer Science*. 1999.

[3] E. Grim, C.-L. Fok, and C. Julien. Grapevine: Efficient Situational Awareness in Pervasive Computing Environments. In *Proc. of PerCom WiP*, 2012.

[4] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi. In-network Aggregation Techniques for Wireless Sensor Networks: a Survey. *IEEE Wireless Communications*, 14(2), 2007.

[5] K. Hong, J. Lee, S. W. Choi, Y. Kim, and H. S. Park. A Strain-Based Load Identification Model for Beams in Building Structures. *Sensors*, 13(8), 2013.

[6] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed Diffusion for Wireless Sensor Networking. *IEEE/ACM Trans. on Networking*, 11(1), 2003.

[7] A. Jindal and K. Psounis. Modeling Spatially Correlated Data in Sensor Networks. *ACM Trans. on Sensor Networks*, 2(4), 2006.

[8] C. Julien, A. Petz, and E. Grim. Rethinking Context for Pervasive Computing: Adaptive Shared Perspectives. In *Proc. of ISPAN*, 2012.

[9] K. Whitehouse, C. Sharp, and E. Brewer, and D. Culler. Hood: A Neighborhood Abstraction for Sensor Networks. In *Proc. of MobiSys*, 2004.

[10] A. Keränen, J. Ott, and T. Kärkkäinen. The ONE Simulator for DTN Protocol Evaluation. In *Proc. of SIMUTools*, 2009.

[11] P. Levis, D. Gay, and D. Culler. Active Sensor Networks. In *Proc. of NSDI*, 2005.

[12] L. Mottola and G. Picco. Logical Neighborhoods: A Programming Abstraction for Wireless Sensor Networks. In *Distributed Computing in Sensor Systems*, volume 4026 of *Lecture Notes in Computer Science*. 2006.

[13] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An Analysis of Approximations for Maximizing Submodular Set Functions-I. *Mathematical Programming*, 14(1), 1978.

[14] C. Ramachandran, S. Misra, and M. Obaidat. A probabilistic zonal approach for swarm-inspired wildfire detection using sensor networks. *Int'l. J. of Communication Systems*, 21(10):1047–1073, 2008.

[15] Ş. Gună, L. Mottola, and G.-P. Picco. DICE: Monitoring Global Invariants with Wireless Sensor Networks. *(To appear) ACM Trans. on Sensor Networks*, 2014.

[16] D. Shea, C. Lund, and B. Green. HVAC influence on vapor intrusion in commercial and industrial buildings. In *Proc. of the Air and Waste Management Associations Vapor Intrusion Conference*, 2010.

[17] J. Stankovic, I. Lee, A. Mok, and R. Rajkumar. Opportunities and Obligations for Physical Computing Systems. *Computer*, 38(11), 2005.

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

[19] M. Xing and C. Julien. Trust-Based, Privacy-Preserving Context Aggregation and Sharing in Mobile Ubiquitous Computing. In *Proc. of Mobiquitous*, 2013.

[20] I. Yoon, D. K. Noh, D. Lee, R. Teguh, T. Honma, and H. Shin. Reliable Wildfire Monitoring with Sparsely Deployed Wireless Sensor Networks. In *Proc. of AINA*, 2012.