Sparse Data Aggregation in Sensor Networks

Jie Gao
Computer Science Department
SUNY Stony Brook
Stony Brook, NY 11794
jgao@cs.sunysb.edu

Leonidas Guibas
Nikola Milosavljevic
Computer Science Department
Stanford University
Stanford, CA 94305
guibas@cs.stanford.edu
nikolam@cs.stanford.edu

John Hershberger
Mentor Graphics
8005 S.W. Boeckman Road
Wilsonville, OR 97070
john_hershberger@mentor.com

Abstract

We study the problem of aggregating data from a *sparse* set of nodes in a wireless sensor network. This is a common situation when a sensor network is deployed to detect relatively rare events. In such situations, each node that should participate in the aggregation knows this fact based on its own sensor readings, but there is no global knowledge in the network of where all these interesting nodes are located. Instead of blindly querying all nodes in the network, we show how the interesting nodes can autonomously discover each other in a distributed fashion and form an *ad hoc* aggregation structure that can be used to compute cumulants, moments, or other statistical summaries. Key to our approach is the capability for two nodes that wish to communicate at roughly the same time to discover each other at a cost that is proportional to their network distance. We show how to build nearly optimal aggregation structures that can further deal with network volatility and compensate for the loss or duplication of data by exploiting probabilistic techniques.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network protocols—*Routing protocols*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

General Terms

Algorithms, Design, Theory

Keywords

Aggregation, Sensor Networks

1. INTRODUCTION AND RATIONALE

Information aggregation is a common operation in sensor networks. Traditionally, information sampled at the sensor nodes needs to be conveyed to a central base station for further processing, analysis, and visualization by the network users. Information aggregation in this context can refer to the computation of statistical means and moments, as well as other cumulative quantities that summarize the data obtained by the network. Such accumulation is important for data analysis and for obtaining a deeper understanding of the signal landscapes observed by the network. In certain contexts, when atomic sensor readings might reveal inappropriate information about a location or an individual, aggregation is also essential for alleviating privacy concerns.

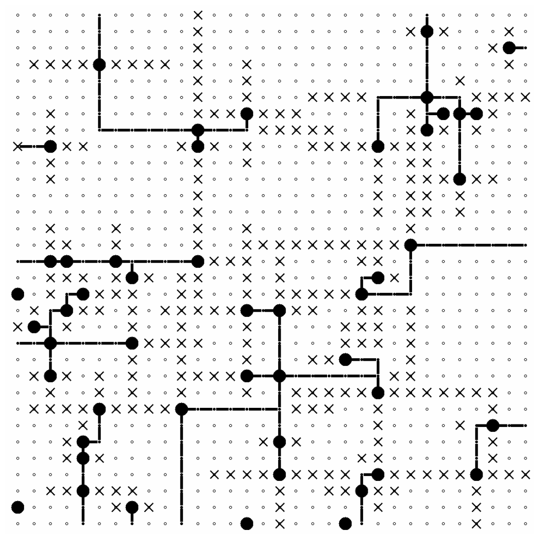

Figure 1. An example of sparse aggregation trees in a $1,024$-node (32×32) grid network. The heavier solid lines represent the tree edges. Nodes marked by crosses are not in the tree, but have been involved in the tree formation phase (by relaying probe and recall messages).

An important topic addressed by the wireless sensor networks community over the last several years has been *in-network aggregation*. The notion is that cumulants, moments, or summaries can be computed directly in the network by sensor nodes endowed with computational power, thus avoiding the expensive transmission of all sensor data to a (possibly distant) base station. It is well known that, for nearly all node technologies, data transmission over wireless links costs hundreds to thousands of times more energy than performing local computation on the same data. In the setting of untethered, energy-constrained sensor nodes monitoring a physical environment, it thus becomes quite significant, for energy conser-

IPSN'07, April 25-27, 2007, Cambridge, Massachusetts, USA.

vation and network lifetime considerations, to be able to trade local computation for communication as much as possible. There has been a lot of interesting prior work on the in-network aggregation problem, some of which is briefly surveyed in Section 2 below.

Most of the extant work assumes that the goal of the aggregation is to accumulate data from *all* the network nodes. This is certainly the case in scientific applications, where scientists are aiming to obtain as detailed information as they can about the observed physical phenomenon. In fact, physical scientists are often unhappy about in-network aggregation, as they want to have access to the raw sensor data, whenever possible. To some extent these concerns can be alleviated by providing tools for *filtered aggregation*, where only a select subset of nodes participate in the aggregation, according to the values of some of their sensors, their geographic location, etc. Even in these settings, however, all the nodes see all the queries—they simply decide whether to participate in the aggregation or not.

In this paper we are interested in exploring a new part of the aggregation problem space, by focusing on situations where only a relatively small subset of the nodes have useful data to report, or want to participate in an aggregation. This is appropriate, for example, in settings where nodes perform some local processing first, to map raw sensor data into event detections, and where events of interest are relatively rare. Many applications of sensor networks fit this model, as sensor networks are often deployed to monitor and protect the health or safety of a system/space/environment, and anomalous events are, by their nature, infrequent. For example, if we have a sensor network monitoring a city, natural catastrophes such as flooding, the spread of disease, earthquake damage, etc., may affect a relatively small portion of the city at any one time. The causative factors of these disasters may depend on combinations of external forces (e.g., rain, ground movement) and local conditions (e.g., building construction, health habits), making their locations hard to predict. What is common in such situations is that the sensor nodes observing the event of interest know that it has occurred, but there is no centralized knowledge in the network about where all these event detections are.

In such situations existing aggregation methods are quite suboptimal. One can poll and query the network for these events, but such queries will also have to be routed to the many nodes that have nothing to report, wasting large amounts of energy. Clearly, for rare events, frequent polling is inefficient. A purely event-driven approach, where each node with a detection independently transmits a record of the event to a base station, may also not be optimal. In the settings we have in mind multiple detections may occur within a relatively short time period and it can be advantageous to aggregate in-network for all the reasons given earlier, as well as for avoiding the creation of hot-spots and single points of failure around the base station. In other settings, where for example rescue workers are embedded and operating in the same space as the network, there may be no single fixed information sink. Instead the detections should be reported to the nearest network users, with minimal latency. Similar challenges arise in settings where we wish to track intermittently observable targets moving through a sensor network. Traditionally, leader nodes aggregate information and perform hand offs as the target moves. But when the detecting nodes are all disconnected, this no longer works. How should these nodes locally discover each other and pool their data to estimate the target tracks?

Our goal in this paper is to aggregate data in-network from a small set of nodes that is possibly disconnected, but all of which have data to aggregate within a relatively short period. Each node knows locally whether it wants to be aggregated or not, but no global information about the location of this set is available in the network. We call this the *sparse aggregation* problem. We focus on the problem of counting the number of nodes in the sparse detection set, or of summing a set of values, one from each of these nodes, as the canonical aggregation problems. Many other variations are possible. For brevity, we refer from now on to the nodes in the aggregation set as the *hot* nodes.

We present a class of solutions to the sparse aggregation problem with several attractive properties:

- Our protocols are entirely distributed and need no prior information about whether hot nodes exist or where they are located; nevertheless, they can exploit such information, if available.
- We bring the hot nodes into a connected aggregation structure, such as a tree, whose total cost is within a small factor of what would be possible if we had a full network map and perfect knowledge of where the hot nodes are located.
- We exploit probabilistic techniques in data aggregation, to make our protocols robust to network volatility and the incorporation of duplicate data, at the expense of a small probability of modest errors.

An interesting aspect of our work is the way that the data processing and information aggregation can be interleaved with the routing, hot node discovery, and tree-formation aspects of our protocols—to the benefit of both. This paper provides a first theoretical analysis and simulation validation of this sparse aggregation framework.

2. PREVIOUS WORK

We are not aware of previous work that specifically addresses the sparse aggregation problem. However, there is a vast amount of extant work on general in-network aggregation that we cannot hope to fully survey here. Thus we focus on what is most relevant to our approach.

Early work on in-network aggregation, e.g., TinyDB [11], uses a tree-like structure to disseminate queries and collect results back to the sink. Queries are pushed down the network to construct a spanning tree. Aggregate values are routed up the tree with partial data aggregated, compressed and pruned at internal tree nodes to reduce communication cost. Various algorithmic techniques have been proposed to allow efficient aggregation without increasing the message size [3, 4, 17].

A serious problem with tree structures is that they are fragile under transmission failures, which are common in sensor networks. With in-network aggregation, packet loss can be fatal—as it may lead to the loss of the values from an entire subtree. To improve routing robustness, multi-path routing schemes can be adopted. But this may lead to duplicate packets arriving at the base station and subsequently cause double counting. To deal with this problem, methods such as synopsis diffusion [3, 12, 13] were proposed to decouple aggregation from message routing by designing aggregation schemes that are insensitive to message duplication and arrival orders, a property named *order and duplicate insensitive (ODI)*. Many of these techniques intuitively implement an aggregation operation by MAX/MIN or boolean operations on bit arrays, which are by nature insensitive to duplicate data. The aggregation methods we use in this paper exploit similar tools.

Non-tree based methods, such as sweeps, have also been proposed [18].

3. NETWORK SETUP AND AN OVERVIEW

Our approach to the sparse aggregation problem is based on and inspired by a number of tools that we briefly mention here, then develop further in later sections.

Suppose that sensor nodes are uniformly deployed inside a regular region with sufficient density for connectivity and coverage. The boundary of the field is known. Furthermore, nodes on the boundary have connections to an external high-speed network. Based on their own sensor measurements, certain sensors in the field decide that they are 'hot' (i.e., they believe they have detected something of interest). Our goal is to count the number of hot nodes in the network, or more generally to compute efficiently certain aggregate statistics about the set of hot nodes. These statistics may involve the number of nodes, the locations of the nodes (if known), or other secondary physical measurements that convey more information about the phenomenon being observed. We assume that the set of hot nodes is relatively small compared to the whole network. There is no assumption that the hot nodes are 'clustered' in any way, though this may turn out to be the case in practice, as physical phenomena exhibit spatial coherence. We would like to collect these statistics at periodic intervals.

To accomplish this goal without explicitly interrogating all nodes, we make a few assumptions that we believe to be reasonable. Since we want the data collection to be initiated by the hot nodes themselves, we need to assume a common temporal framework for the nodes, with a shared clock that lets a node know when to initiate data collection. We do not require perfect (and unrealistic) clock synchronization among the nodes—instead we assume that clock drift between a pair of nodes is proportional to the distance between those nodes in the network. This can be accomplished by a low-overhead time synchronization protocol, simply by guaranteeing that each node has a bounded drift with respect to each of its 1-hop neighbors. In the case of target tracking, nodes detect events and participate in the aggregation protocol when a target passes by. As long as the target keeps moving with a minimum speed, detection time differences at two nodes will also be bounded by some constant times the relative distance of the nodes or the target path length.

Distance-sensitive neighbor discovery. Key to our approach is a *probe protocol* that can be initiated independently by a node and involves emitting a few packets out of the node that follow certain paths in the network while leaving a trail behind, until the probes are terminated. The key property of the probe protocol is that if two nodes, say u and v, at network distance d (with no knowledge of each other) initiate probes at the same time, then a packet from one of the nodes will encounter a trail left by a packet from the other node after $O(d)$ steps. We call this crucial property *distance sensitivity*. At that encounter event

- if the probe packets carry data from the two nodes, the data can be aggregated, and
- a path between the two nodes has been established whose length is $O(d)$ as well.

The same holds if the two nodes initiate probes within a time-lag of $O(d)$ of each other. Furthermore, at this point it is possible to initiate another *recall protocol* whose function is to terminate the probe packets of one of the two nodes, say u. This can be done so that when all the packets of u have been stopped by the recall protocol, the total work performed by both the probe and recall protocols on both nodes u and v is still $O(d)$.

In the simplest scenario, when we use the probe and recall protocols, we will think of u and v as 'competing' in a tournament. When an encounter event occurs, the winner of the tournament is decided based on the data carried or left behind by the probe packets. The losing node then has his probes terminated by invoking the recall protocol, while the probes of the winner continue to propagate. Probe packets that are not recalled will eventually reach the boundary of the network, where they stop. Their data is aggregated in an appropriate base station, using the high-speed network available for the boundary nodes.

We will use the above communication pattern in several variations in what follows. Effectively, our intersecting probes provide a lightweight rendez-vous mechanism, allowing nodes within distance d to discover each other's existence at a cost of $O(d)$, and not the $\Omega(d^2)$ that a traditional broadcast in a planar network requires.

Tree formation. To form an aggregation tree, we let each node choose a random number from some distribution, named the *weight* of this node. The weights of the hot nodes determine how the aggregation tree is formed. The weights can also be exploited in the specific aggregation to be performed, as will be explained later.

At the right time, according to their local clocks, hot nodes initiate probes carrying their weights. When two probes encounter each other, the packet with lower weight wins and initiates a recall protocol for the packets of the node with the larger weight. Probe messages that are not recalled will eventually reach the network boundary. Thus a spanning forest of the hot nodes all connected to the network boundary is formed which can be used for subsequent data aggregation.

What is the total cost of the tree formation? Note that each node u will have its probes recalled by a node v of smaller weight, to whom it loses in some encounter event during the tournament[1]. This happens at a total cost proportional to the distance d from u to v. Thus each losing node spends effort that is proportional to its distance to the node that defeats it. We show in Section 5 that this simple property is enough to guarantee that the overall effort spent in the aggregation for nodes other than the one of minimum weight is $O(T \log n)$, where T is the cost of a minimum spanning tree connecting the hot nodes.

Our distributed construction of the aggregation tree was motivated by the nearest neighbor trees by Khan, Kumar, and Pandurangan [8], which are built by giving desired nodes (the hot nodes in our context) certain priorities and then connecting, in a distributed manner, each node to its nearest node of higher priority. We note that in the aggregation tree (or forest) formed by our protocol, a node is not necessarily connected to its nearest neighbor of higher priority. But our tree still has a small total weight. The overall cost of the algorithm is $O(T \log n + K)$, where K is the time needed for the probes of the minimum weight node to reach the sensor field boundary.

Aggregation. After the aggregation tree or forest is formed, data aggregation can be conducted by having the data flow up the tree and aggregated in the standard way. An interesting aspect of our protocol is that we can actually integrate data aggregation with the tree formation process so that they both happen simultaneously. Recall that in the tree formation procedure we use random priorities to determine which probe message should proceed. These random priorities can be used to implement data aggregation, by using probabilistic counting techniques. Essentially the formation of the tree achieves the computation of the minimum of some random numbers chosen by the hot nodes, with which one can derive an approximation to the final aggregation value. Note that since

[1]We assign the nodes on the boundary weight $-\infty$ so that probes that reach the boundary (and thus deliver their data to the sink) are recalled.

we are aggregating by computing mins, the framework is robust to packet duplication, multiple encounter events between packets from the same two nodes, etc.

4. FORMATION OF A SPARSE AGGREGATION TREE

This section generalizes and fleshes out the details of the methods in Section 3. Note that an unknown sink (or sinks) seeking the results of the aggregation can easily be integrated into this tree discovery/formation scheme. Thus the the aggregated data can be delivered to any desired location or locations in the network.

4.1 Double Rulings

To achieve the distance-sensitivity property for neighbor discovery, in our probing protocol each participating node initiates probe packets and sends them along *double ruling* trails. A double ruling scheme defines a set of trails through every node in the network with the following properties: (i) the maximum length of the trail through each node is $O(D)$, where D is the network diameter; (ii) the trails through two nodes p and q are guaranteed to intersect in the network; (iii) there is a path connecting p and q through the trail intersection with total length $O(d)$, where d is the network distance between p, q. Double ruling schemes were initially designed for information discovery in a sensor network where a sink desires to obtain specific data from a data source while neither of the source or sink knows the location of each other. Different from a pure 'pull' or pure 'push' approach (both of which involve expensive flooding operations), a double ruling scheme adopts a symmetric information discovery principle with both the source and sink actively searching for each other, as first suggested in rumor-routing [1]. The notion of trails has appeared in a number of other papers as well, including in trajectory forwarding [14], in the asymptotics of query strategies [16], and the combs and needles work [10]. A number of double ruling schemes that guarantee distance sensitivity have been devised in the literature [15]. For example, with geographical location information, a simple double ruling scheme defines the trail for each node p as a cross, i.e., horizontal and vertical paths that go through p. See Figure 2 for an example. The length of a path connecting p, q through either of the two intersections is the ℓ_1 distance between p, q, which is at most a factor of $\sqrt{2}$ more than the Euclidean distance between p and q.

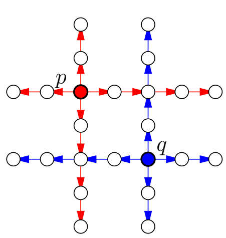

Figure 2. The double ruling trail through each node consists of a cross. The two crosses from nodes p and q intersect.

This rectilinear double ruling scheme can be implemented by geographical greedy forwarding. A node sending a probe packet to the north will select the next hop as the one among all 1-hop neighbors that has the maximum y-coordinate. The other three directions are treated similarly. Under uniform deployment of sensor nodes with sufficient density, the greedy algorithm will create paths that resemble a cross passing through the initial node.

We call a node receiving probe packets from both p and q a *junction node*. A junction node can be a node on both double ruling trails from p and q. Or, if two double ruling trails do not intersect at sensor nodes, then we have two crossing links, ab and cd. Assuming sufficient sensor density, because of the broadcast nature of a wireless network, there will a node in the vicinity of the crossing that receives and stores probes from both p and q, and this node is a junction. Junction nodes may not be unique, however. What is crucial is that there is always a path through some junction node connecting p and q with total length $O(d)$, where d is the distance between p, q.

To improve the system robustness to link failures and packet loss, we can augment this basic double ruling scheme by using multi-path routing. That is, a node holding important data (say, in the sense discussed at the end of Section 6) can send its probe packets along a wider band, rather than a single path. This can be controlled by specifying the width W of the band. A probe packet from source p heading north is broadcast to the nodes in its 1-hop neighborhood; all the nodes with x-coordinate within $[p_x - W/2, p_x + W/2]$ and y-coordinate higher than the transmitting node will re-transmit the probe. W can be chosen according to the importance of the data. More about the coupling of data and routing will be discussed in Section 6.

In the sequel we use this rectilinear cross scheme to describe the probe and recall protocols. For our sparse aggregation problem, any double ruling scheme satisfying the three properties above is adequate and such schemes can be devised for quite general field morphologies and sensor distributions [15].

4.2 Probe and Recall Protocols

The sparse aggregation protocol starts at a particular time. We assume the nodes have bounded clock drift. The clock difference between two nodes p, q is bounded by $\alpha||pq||$, where α is a constant, and $||pq||$ is the network distance between p, q.

The hot nodes with data to report participate in the tree formation protocol, composed of two sub-protocols, the *probe protocol* and the *recall protocol*, which together establish an aggregation tree connecting all the hot nodes. The goal of the probe protocol is to discover other hot nodes in the network and establish communication paths between them. The goal of the recall protocol is to prune excess packets traveling in the network, after hot nodes discover each other, so as to reduce the communication overhead.

To maintain a consistent tree structure in a distributed environment, we invoke a ranking system on the hot nodes. In general, each hot node uses a pseudo-random hash function on its node ID to generate a unique priority number. When two nodes discover each other by the probe protocol, the node with higher priority 'wins' and continues its probe packet. The node with lower priority 'loses' and initiates the recall protocol to terminate its probe packets. We assume that the probe packets travel in the network with 'unit' speed. The recall packets travel with speed $M > 1$ in order to catch up to and terminate the probe packets. M depends on the bound on clock drift and is typically a small constant.

At time zero, a hot node sends out probe packets in the four directions north, south, west and east. A probe packet from a node p contains the source of the probe, p, and the direction the probe travels. A node sends the probe packet to all the nodes in its 1-hop neighborhood and those in the direction of the probe re-transmit. The number of nodes that retransmit can be adjusted for improved robustness, as explained earlier. A similar trade-off was recently analyzed in [19].

Each node keeps a *status table* recording the information it has received from each hot node. This includes: a tuple (SOURCE, MSGTYPE, PARENT, JUNCTION), where SOURCE is the ID of the hot node, MSGTYPE is PROBE for a probe packet and RECALLED for a recall packet, PARENT and JUNCTION are the parent and junction node in the aggregation tree, to be defined later.

When a node w receives a probe packet from p, it first checks whether it has already received any packet from p; if so, this packet is ignored. Otherwise w records the probe packet. In addition, if w has received messages from other hot nodes, w checks if it is a junction node and if any probes need to be recalled. If w has received a message from a node q with priority higher than p, then the probe message from p will be recalled. Node q is a candidate to be the parent of p in the aggregation tree. If p's priority is higher than all the other hot nodes whose status entries have MSGTYPE = PROBE at node w, then those probes will be recalled. They are assigned p as a potential parent. The probe packet for p will be re-transmitted.

The recall protocol is initiated when a hot node p 'loses' to another hot node q at a junction node w. Node q is a possible parent of p. The recall packet contains the IDs of p (the child), q (the parent), and w (the junction node). A recall packet travels at speed M, faster than the speed of the probe packets. A recall packet has no direction associated with it but uses flood-fill to travel through the trails of the probe packets. When a node x receives (or initiates) a recall message, it first checks the status for node p in its local table. If x has received a message from p before and the current status is PROBE, then x will change the status of p to RECALLED. The recall message will be broadcast to all the 1-hop neighbors of x. If x has no message from p, or the status is RECALLED, x does not send the recall message further.

When node p itself receives a recall packet for p, it sets its parent to be the parent-candidate q in the packet and remembers the junction node w where the trails of p and q met. Thus the parent of p in the aggregation tree is determined by the first recall packet that reaches p.

The boundary of the sensor region is incorporated into the protocol by initializing every boundary node as a junction whose probing node (the base station b) has the highest priority. When a probe packet for hot node p reaches a boundary node w, p loses to b, and w initiates the recall protocol for p with potential parent b and junction node w. Finding boundaries in networks has been addressed by a number of authors [5, 6, 9, 20].

The status tables in the nodes record the trails of probe and recall packets, and also provide a way to suppress redundant retransmissions. Due to the broadcast nature of wireless communication, a packet may be received by multiple nodes. In the probe and recall protocol, multiple nodes may realize they are junction nodes and thus initiate the recall procedure. However, once a node has received a recall packet and updated its status, it will not respond to any further recall packets other nodes may have generated.

4.3 Forming an Aggregation Tree

As described in the preceding subsection, the aggregation tree is a *logical* structure recorded in the hot nodes themselves: each node p knows its parent q in the tree. The parent may either be another hot node or the boundary of the sensor region. Thus the 'tree' is actually a forest of trees, with the number of trees in the forest dependent on the distance from the region of hot nodes to the boundary. Nevertheless, we will continue to use the word 'tree' to describe the structure—it is still an aggregation tree, but the children of the base station are joined to it by high-performance links.

Our simulations, described in Section 7, use the simple parent-child representation of the aggregation tree. The earlier figure 1 shows an example of the tree we obtained. One can see that only a small fraction of the nodes are involved in the tree formation. Although our simulations use the simple tree representation, there are several variants on the tree structure that pay greater attention to the physical embedding of the tree in the sensor network, with varying impacts on communication cost and reliability.

Using the logical representation of the aggregation tree, aggregation happens at the hot nodes themselves. The children of a node p send their results to p, p aggregates them, and then p sends the resulting summary on to its parent. There are several choices for how each node p sends data to its parent q.

- Send a packet from p along p's trail to the junction node w, then along q's trail to q. This has the advantage that the probe and recall protocol has already demonstrated the existence of this path—no path discovery is required.
- Use some network-specific point-to-point routing mechanism, such as geographic routing [7], to send a packet from p to q, bypassing the junction w entirely. This reduces the load on the trail nodes, but forces the packet to discover its own path.
- If p's results are known to be particularly important and their successful delivery is crucial, the preceding two possibilities can be combined, or either can be complemented by transmitting in a band around the basic path, with the width of the band dependent on the importance of the message, as described in Section 4.1.

Routing along the trails of the logical aggregation tree may cause some inefficiency in the routing. Packets may be sent in one direction along p's trails from its children, only to have the summary packet from p traverse the same trail in the opposite direction on the way to its parent. We can avoid this redundancy (possibly saving a constant factor in communication costs) by exploiting the embedding of the aggregation tree. We add a third phase to the probe and recall protocol, in which a node p notifies all the nodes in its trails of its parent q and junction node w. This allows all the nodes in p's trails to direct the trail edges toward w. Node w becomes a proxy for p in the aggregation tree. Instead of sending data to p for aggregation, all p's children (represented by their proxy nodes on p's trail) send data along p's trail to w, where aggregation occurs before w sends a summary packet toward the junction representing p's parent. Node p also sends its own individual data to w for aggregation. Two variants of this approach are possible:

- Each proxy for a child of p sends its data along p's trail independently to w. A constant fraction of p's trail may be traversed for each child.
- The child proxies along p's trails may be used as intermediate aggregation points, combining data from the child with data flowing along p's trail from nodes farther from w along the trail. This reduces the number of messages that must be sent—each trail segment is traversed just once—but makes the aggregation tree more vulnerable to deadlock because of single-node failures.

A final modification enhances robustness by replacing the aggregation tree with a DAG. To do this, we modify the recall protocol so that all recall messages for a node p propagate back to p. This increases the amount of communication in the recall phase of the protocol, but has the advantage that p is informed of all its candidate parents, not just the one whose recall packet reaches it first. Node p can choose multiple parents among the candidates, perhaps up to four, one for each cardinal direction. When p performs its aggregation (here we assume the simple tree structure, with aggregation performed at hot nodes rather than proxies), it sends the resulting data to *all* its parents, thereby increasing the likelihood that the data will reach the base station. Of course this requires that we use an ODI (order and duplicate insensitive) scheme.

5. QUALITY OF THE SPARSE AGGREGATION TREE

In this section we discuss the performance of the protocols in the previous section and characterize the aggregation tree that they build. The asymptotic bounds we present apply to all the tree variants discussed in Section 4.3, with the caveat that multiplying the width of important communication paths to increase reliability has a corresponding multiplicative cost in communication resources.

As noted in Section 4.2, the clock skew between any two nodes u and v is bounded by $\alpha||uv||$, where $||uv||$ is the network distance between u and v. The speed of probe packets is 1 and the speed of recall packets is $M > 1$. We denote the set of hot nodes by S.

For any hot node p, let $trail(p)$ be the set of nodes in p's trails. Because of the uniform distribution of sensor nodes, $|trail(p)|$ is proportional to the distance between p and the farthest node in $trail(p)$. Furthermore, $|trail(p)|$ is proportional to the total number of packets in the probe and recall protocol attributable to p.

When a probe meets the trail of another node's probe, at least one of the two will be recalled, if neither has already been recalled. This observation leads to the following lemma (proof omitted):

Lemma 5.1. *Let u and v be two hot nodes. Then*

$$\min(|trail(u)|, |trail(v)|) \leq C \cdot ||uv||$$

for some constant C.

Let E be the set of nearest-neighbor edges: for each hot node u, E contains the edge from u to its nearest hot node, denoted $nn(u)$.

Lemma 5.2. *E contains a partial matching[2] of size $\Theta(|S|)$.*

It is well-known that the minimum spanning tree of S, denoted $MST(S)$, contains all the edges of E. (If $MST(S)$ does not contain $e = (v, nn(v))$ for some node $v \in S$, adding e to $MST(S)$ creates a cycle, and the tree weight can be decreased by retaining e and removing the other cycle edge incident to v.) The weight of $MST(S)$, i.e., its total length, is $|MST(S)|$. The length of an individual edge $e = (p, q)$ is $len(e) = ||pq||$.

Theorem 5.3. *Let K be the maximum, over all $v \in S$, of the shortest axis-aligned distance from v to the boundary of the sensor region. Then*

$$\sum_{v \in S} |trail(v)| \; = \; O(K + |MST(S)| \cdot \log |S|).$$

PROOF. For any subset $X \subseteq S$, define $f(X) = \sum_{v \in X} |trail(v)|$. We want to estimate $f(S)$.

For a given $X \subseteq S$, let E be the set of nearest-neighbor edges defined with respect to X, and let E_M be a maximal-cardinality matching in E. Because every edge of E_M belongs to $MST(X)$,

$$\sum_{e \in E_M} len(e) \leq |MST(X)|.$$

Also note that $|MST(X)| \leq |TSP(X)| \leq |TSP(S)| \leq 2|MST(S)|$, where $TSP(\cdot)$ is the minimum-length traveling salesman tour of a point set.

By Lemma 5.1, every edge $e \in E_M$ is incident to a node v such that $|trail(v)| \leq C \cdot len(e)$. Let the set of these nodes be X_M. We have

$$f(X_M) = \sum_{v \in X_M} |trail(v)| \leq C \cdot |MST(X)|.$$

[2] A partial matching is a set of edges such that each node is an endpoint of at most one edge.

Create a new set $X' = X \setminus X_M$; by Lemma 5.2, $|X'| \leq \beta |X|$, for some constant $\beta < 1$. Assembling the pieces, we have

$$\begin{aligned} f(X) = f(X_M) + f(X') &\leq C \cdot |MST(X)| + f(X') \\ &\leq 2C \cdot |MST(S)| + f(X'). \end{aligned}$$

The base case for $|X| = 1$ is $f(X) = O(K)$. By induction on $|X|$, we have $f(X) \leq 2C \cdot |MST(S)| \cdot \log_{1/\alpha} |X| + O(K)$. □

Because the aggregation tree built in Section 4.3 uses a subset of the nodes in all the trails, the theorem has the following immediate consequence:

Theorem 5.4. *The aggregation tree constructed by our protocol has total length $O(|MST(S)| \cdot \log |S| + K)$, where K is as defined in Theorem 5.3.*

The maximum degree of the aggregation tree is also important for its performance. Nodes with high degree must perform much more aggregation work than the average (which is just constant). In the following theorem we bound the maximum degree of our aggregation tree. The proof is omitted from this version of the paper.

Theorem 5.5. *Let A be the aspect ratio of S, i.e., the ratio between the maximum and minimum distances between elements of S. If the clock skew parameter α is less than 1, then the maximum degree of any node in the aggregation tree of Section 4.3 is $O(\log A)$.*

6. AGGREGATION

Many possible aggregation strategies are possible, within the framework discussed above. If the network has solid connectivity and point-to-point routing already in place, then a logical aggregation tree is all that is needed and aggregation can happen in the traditional way, after tree formation. If the network is volatile, however, then a DAG-based approach with multi-path connectivity is preferable, together with an ODI aggregation scheme. Note also that our tree discovery protocol naturally leads to a multiply-connected structure, as for each pair of nodes the multiple probe paths establish multiple routes between the nodes.

Of special interest is the fact that our protocols allow fine interleaving between tree formation and aggregation. Specifically, the random priority needed by the tree formation process can be used to implement a probabilistic aggregation scheme. In this way aggregation is already completed, as soon as the tree is formed.

Specifically, we can make use of certain properties of the classical exponential distribution. A random variable x chosen from an exponential distribution with parameter λ, $\lambda > 0$, has a probability density function $f(x; \lambda)$ given by

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x}, & \text{if} \quad x \geq 0 \\ 0, & \text{if} \quad x < 0 \end{cases}$$

The cumulative distribution function $F(x; \lambda)$ is given by

$$F(x;\lambda) = \begin{cases} 1 - e^{-\lambda x}, & \text{if} \quad x \geq 0 \\ 0, & \text{if} \quad x < 0 \end{cases}$$

The distribution has mean $1/\lambda$ and variance $1/\lambda^2$. The following two properties of the exponential distribution are crucial for us.

Theorem 6.1. *If x_1 and x_2 are independent random variables with parameters λ_1 and λ_1 respectively, then*

$$Prob(x_1 < x_2) = \frac{\lambda_1}{\lambda_1 + \lambda_2}.$$

Theorem 6.2. *If $x_1, x_2, \ldots, x_n$ are independent exponential random variables, where x_i has parameter λ_i, then*

$$\min(x_1, x_2, \ldots, x_n)$$

is an exponential random variable with parameter $\sum_{i=1}^{n} \lambda_i$.

A consequence of these properties is that if $x_1, x_2, \ldots, x_n$ are independent exponential random variables chosen from exponential distributions with parameters $\lambda_1, \lambda_2, \ldots, \lambda_n$ respectively, then their minimum is a random variable with mean $1/\sum_{i=1}^{n} \lambda_i$. This effectively allows us to replace a data summation by a data minimization operation which, as mentioned, is insensitive to data duplication and thus is advantageous in a volatile network setting.

This use, to our knowledge, was first pointed out by E. Cohen [2], in the context of solving counting problems on large graphs. In our adaptation of the Cohen framework, each network node i holding data value f_i independently selects a random number drawn from an exponential distribution with density function $f_i e^{-f_i x}$. If μ is the minimum number chosen by any hot node, then $1/\mu$ is an unbiased estimator of the sum of $F = f_1 + f_2 + \cdots + f_n$. Summing values at the hot nodes is a surprisingly powerful aggregation primitive. If node locations are known, this primitive can be used to compute arbitrary moments of the hot node position distribution, conveying a picture of where the exceptional events are being detected. Alternatively, summation can be used to compute moments of the values sensed by the nodes, covariances between node locations and values, etc.

For the primitive of summing values, we can use the random number chosen from the exponential distribution described above as the weights in the tree formation protocol. Recall that during tree formation, a number of packets may reach the field boundary, from where these weights can be transmitted to the base station and the minimum weight can be retained. No packet can stop the packets of the hot node whose weight is globally minimal—and therefore these packets will reach the boundary and the base station. Thus as the tree is formed, the base station can immediately derive an approximation to the aggregated value. For applications in which a quick and dirty approximation suffices, our job is done at the moment when the tree is formed. For better accuracy, we can repeat this experiment a number of times. Cohen shows that, for a given parameter ε, if we repeat this experiment $O(\frac{1}{\varepsilon^2} \log n)$ times, where n is the number of nodes involved, average the minimum weights obtained in each experiment, and then take the inverse of that, this quantity lies, with high probability, within relative error ε of F. With $O(\frac{1}{\varepsilon^2})$ experiments, the *expected* relative error is at most ε. We report on some simulations validating these bounds in Section 7.

In summary, by integrating data aggregation with node discovery and tree formation, the simple distributed probe protocol performs two important tasks at once: it connects the hot nodes into an aggregation tree that is nearly optimal in terms of communication cost, while at the same time performing the min-based aggregation that can translate into an estimate of the final aggregation.

The robustness of aggregation deserves further exploration. Note that this type of probabilistic counting has built-in robustness to data loss, as well as duplicate insensitivity. Some of the simulations reported below quantify how many exponential variates are necessary to obtain a given degree of accuracy. For example, if the smallest variate in the counting problem is lost, but the second smallest makes it, our final count will be off by only 1 in expectation—while in regular aggregation a single lost subtree count can cause unbounded error. If other variates are lost, they have no impact on the count. The counting problem also allows this self-determined yet globally consistent notion of data importance, which when coupled with multi-path routing can guarantee further robustness. Although this does not directly extend to the completely general case of summation of hot node values, in most scenarios prior knowledge of the range of node values, past measurements, etc., can be used to develop a notion of the significance of data locally, without requiring additional extensive communication.

7. SIMULATIONS AND EXPERIMENTS

In this section we evaluate by means of simulation the performance of our sparse aggregation scheme in a number of settings.

7.1 Communication Costs

The main metric we consider is the *communication cost* of a single aggregation task. If aggregation is initiated by a base station, then we include the costs of both the query phase and the data collection phase. In our protocol there is no query phase. Our aggregation cost is defined to be the cost of forming an aggregation *tree*, as the basic logical structure described in subsection 4.2, and the cost of sending partial aggregates along child-parent paths if data aggregation/delivery has not been accomplished (if data aggregation is combined with tree formation as in Section 6 this cost is 0).

We assume that the cost of sending a message is proportional to the Euclidean distance between the sender and the receiver, a justified assumption in the case of dense deployment. Although not completely realistic, we believe that this assumption does not influence the overall outcomes of our comparisons, but only the absolute costs, which we are less interested in.

We compare our algorithm to two general aggregation paradigms, commonly known as 'push'- and 'pull'-style aggregation, rather than any specific method proposed in the literature. Almost all existing methods that we are aware of (see Section 2) belong to one of these categories. In Figures 3 and 4, the vertical axis is the ratio of the cost of our scheme vs. the cost of the corresponding 'push' or 'pull' scheme—thus the smaller the better.

'Pull'-based method: The base station queries *all* nodes, and only the hot nodes reply. The query cost is essentially the cost of flooding the entire network. We consider two variations for the data collection cost. In the first data collection, the hot nodes reply by sending the data upstream along the spanning tree constructed in the flooding phase. Data can be aggregated at internal nodes of the tree. We also consider the case when each hot node replies independently along its shortest path to the base station, disregarding any message trails it may encounter along the way. This method invokes higher communication cost, but requires no coordination and scheduling of partial aggregates, which makes it much easier to implement. Since the hot nodes are scattered at unpredictable locations, scheduling for in-network aggregation of irregularly spaced participants is non-trivial. Our two performance metrics are the sum of the fixed query cost and that of the two reply costs, as described above. Note that we discount the costs of both reply methods significantly by disregarding the non-trivial amount of work required to discover the shortest paths and the cost of coordination and scheduling for in-network aggregation, respectively. Another serious overhead that we did not account for in the 'pull'-method is that the user/sink needs to know when to 'pull'. As interesting events happen irregularly and unexpectedly, a 'pull' approach in practice may waste a lot of communication by querying when the data is not yet be available. Our comparison is unrealistically favorable to this approach, as we simply ignore the cost of the unfruitful polling queries that a practical 'pull'-method would definitely include.

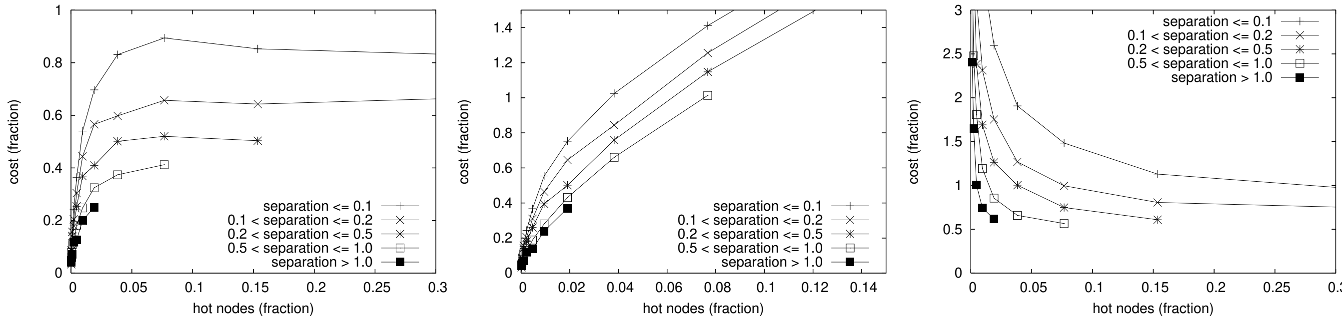

Figure 3. (i), (ii) Communication cost of 'pull'-aggregation in the 100×100 grid network. The query dissemination cost is the cost of the flooding on all nodes. Hot nodes reply using shortest paths to the base station (the boundary in this case). (i) Without in-network aggregation; (ii) With in-network aggregation; (iii) Communication cost of 'push'-aggregation. Hot nodes reply using shortest paths to the boundary.

'Push'-based method: The hot nodes themselves report to the sink via shortest paths without being explicitly queried. There is no query cost. Again we heavily underestimate the cost in this method by disregarding the cost of maintaining the shortest paths to the sinks (which requires some form of flooding).

We consider the regular grid topology and the simulation of the real-world deployment built by the Extreme Scaling (ExScal) project at the Ohio State University, as deployed in Florida in December 2004. We examine how the performance of our aggregation scheme is influenced by the number and density of hot nodes, as well as the delay with which various nodes launch their probes. The delay may come from imperfect synchronization or may be inherent in the application, as in the case of intermittent tracking.

7.1.1 Simulated Grid Network

We tested the algorithm on a network with regular grid topology, consisting of $10,000$ nodes (100×100 grid). The whole boundary acts as a base station, i.e., the boundary nodes can communicate quickly and reliably using an external network.

'Pull' and 'push' aggregation: Figure 3 shows how our protocol compares to the 'pull'- and 'push'-based methods described above, depending on the number of hot nodes and their spatial proximity. Different curves correspond to varying spatial distributions of hot nodes, specifically the extent to which they are 'clustered' in space. In the experiment, hot nodes are generated by first picking a random square of side length equal to the prescribed 'cluster size,' then selecting a prescribed number of nodes within the cluster uniformly at random. This models the emergence of some physical events that influence the sensors in its proximity. During the experiments we discovered that the distances of hot nodes to the boundary are important parameters for the cost metric, so we grouped the data points by the *separation* parameter: the ratio of the diameter of the set of hot nodes in the ℓ_∞ norm and the shortest distance between the hot nodes and the boundary. The larger the separation, the further away the hot nodes from the boundary.

In the 'pull' scenario without in-network aggregation, when compared to the simpler shortest-path-based method (Figure 3 (i)), our algorithm performs better for almost all parameter settings, which is expected, as it avoids flooding the network to announce the query to all nodes. The benefit of our method when hot nodes are colocated is quite significant. In the pull model with in-network aggregation (Figure 3 (ii)), our advantage is not as significant. Again we remark that scheduling in-network aggregation between sparse nodes is not trivial. The major challenge is that, an internal node has no knowledge of whether or not there are hot nodes in its subtree. Thus data propagating upstream has no idea whether it should wait at internal nodes to be aggregated with data from other hot nodes. In our implementation we actually 'cheat' in favor of the pull method by assuming that this is known. Even in this very pessimistic comparison, our sparse aggregation scheme is still better when the hot nodes are sparse and well-clustered. For the same reason, in the remaining experiments we only compare with the pull model with hot nodes sending their data separately with no in-network aggregation.

In the 'push' scenario (Figure 3 (iii)), our tree is better by 20 to 30 percent unless the hot nodes are too few and their separation is small. With only a few very scattered nodes, the sum of their individual shortest paths is indeed nearly optimal. So the 'peaks' come from the overhead of our protocol. In other words, most probes get propagated all the way to the boundary (because there is not much opportunity for early aggregation), and our algorithm gets penalized for trying four directions and traversing the same trails twice.

Tree quality: We measured the quality of the aggregation tree built by our algorithm. Equivalently, this is the cost of collecting the data from all hot nodes, once an aggregation tree has been built. For instance, this could be the per-iteration cost of *repeated aggregation* from a fixed set of sources in a short time interval during which the tree is sufficiently stable.

In light of our theoretical result (Theorem 5.4), we normalize the cost by that of the MST of the hot nodes and the base station. The results are shown in Figure 4 (i). One can see that, at least for networks of this size, the MST approximation factor does appear to grow nearly logarithmically with the number of hot nodes. However, the constant factor is very good (much better than the explicit constants of the theorem would suggest): our aggregation tree was never more than twice the length of the MST.

Tracking: We now consider the application of sparse data aggregation to the problem of tracking a target that is only intermittently detected. In this experiment we assume for simplicity that the target moves along a straight-line path with constant speed. We vary the target speed (relative to the message speed), and the density of detections along the track. For each fixed setting of the two parameters, we generate 50 line segments with randomly chosen endpoints as the trajectories of the target. For a given track, we distribute the target detections uniformly along the track and trigger each observing node's probes when the target passes by this node.

We examine how the speed of the tracked object influences the cost of the sparse aggregation tree. The probes of consecutive hot nodes along the object's path are now launched with clock skew inversely proportional to the speed of the object (i.e., the faster the target moves, the smaller the clock skew of the probe packets).

As before, we compare our cost to the cost of the 'pull' aggregation initiated by the boundary nodes, where the observers reply

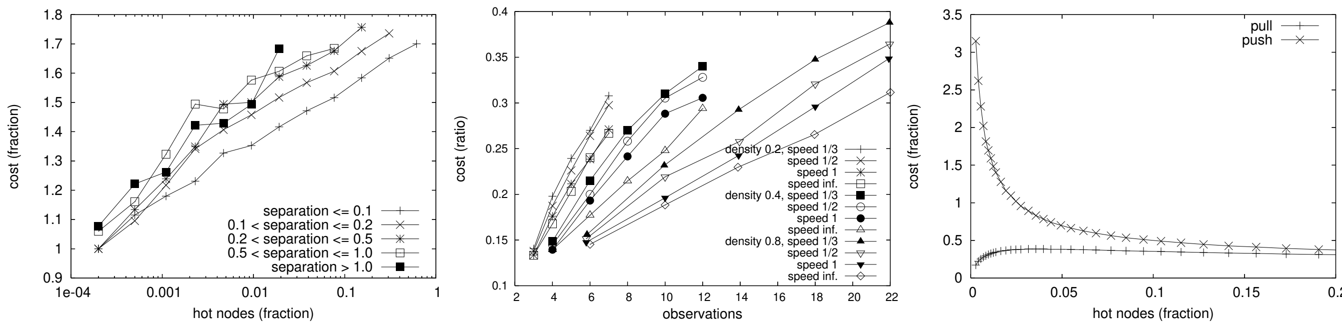

Figure 4. (i) The cost of our aggregation tree (*excluding* the tree formation cost) for a 100×100 grid network, vs. the cost of the MST of the hot nodes and the base station. The behavior agrees with Theorem 5.4. (ii) Tracking an intermittently observable target. 'Density' is in number of observers per one grid cell of track length. 'Speed' is the ratio of the speeds of the target and probe message propagation. (iii) Performance on the ExScal deployment.

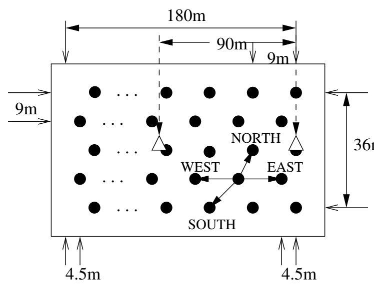

Figure 6. A typical section of the Thick Horizontal part of the ExScal deployment. There are 100 motes (shown as disks) and 4 stargates (only two shown as triangles).

independently using shortest paths. The results are shown in Figure 4 (ii). The simulation results show the following observation. We are at least 50% better in terms of total communication cost. When the clock skew of the probe packets is smaller, the probe packets are recalled earlier. This results in lower total communication cost.

7.1.2 Simulated ExScal Deployment

Our second set of simulations used the ExScal deployment model, in particular the deployment setup that the ExScal documents refer to as *Thick Horizontal*. It is a two-tier deployment, with Tier 1 consisting of XSM motes, and Tier 2 of more powerful stargate nodes. Some stargates are designated as gateways between the tiers. The deployment consists of 7 *sections* (see Figure 6), each containing about 100 motes, 2 gateway stargates and 2 stargates that communicate strictly in Tier 2. The sections are laid out in sequence along a straight line (see Figure 5).

The network is organized hierarchically to facilitate routing and other applications. Each stargate is responsible for its section. For the sparse aggregation problem, these gateway stargates provide natural aggregation points for hot nodes within one section. For this deployment we let the leftmost stargate (in Figure 5) be the base station. In tree formation, we choose the four neighbors for double ruling as shown in Figure 6.

The results are in Figure 4 (iii). In this simulation we only used the metric where the hot nodes reply to the base station using shortest paths (in this case, this is the shortest path to the nearest stargate plus a path up the hierarchy). The important features of the resulting plots consistently agree with those of the grid network (Figures 3 (i) and 3 (iii)). When the hot nodes are extremely sparse (less than 3%), the 'push' scheme is the most efficient, otherwise our scheme works better. The 'pull' scheme is consistently the worst.

7.1.3 Summary

Our simulation results lead to the following conclusions regarding the comparative communication costs of the sparse aggregation framework and the 'push' and 'pull' aggregation paradigms:

- The sparse aggregation scheme is consistently more communication efficient compared to the 'pull'-based approach without in-network aggregation, for both the grid and ExScal topologies. This remains true even when compared to 'pull'-based aggregation (favored by our assumptions), as long as the percentage of hot nodes is less than 8% or so. The sparse aggregation scheme is also superior to the 'push'-based approach, if more than 3% of nodes are hot. Thus sparse aggregation fills in an interesting gap between scenarios where 'pull' and 'push' methods are respectively preferred.
- The efficiency of our sparse aggregation scheme becomes more evident when hot nodes are clustered and far from the network boundary.
- Modest clock drift does not affect the aggregation cost significantly.

7.2 Quality of Probabilistic Aggregates

Recall from Section 6 that the theoretical bound for the number of experiments with exponential variates required for a relative error of ε in expectation and with high (inverse polynomial in the number of hot nodes) probability is $O(\varepsilon^{-2})$ and $O(\varepsilon^{-2} \log n)$, respectively, where n is the number of hot nodes. We ran an experiment to test how the algorithm performs in practice.

We focused on the simplest problem of counting, and chose several ground truth (exact count) values ranging from 10 to $10,000$. Notice that the theoretical convergence speed does not depend on the actual count, but only on desired accuracy (relative error) and confidence (probability of success).

For each exact count we recorded the number of experiments needed to achieve relative error of 10%, 5% and 1%. Figure 7 shows the failure probability (one minus the confidence) vs. the number of experiments. Estimates of this probability are obtained from $1,000$ independent experiments[3]. The results form three dense 'clusters' of curves. A cluster corresponds to a target accuracy, and a curve within a cluster to a ground truth value. Thus we confirm that the exact count has little influence on the convergence. The remaining three curves are the theoretical counterparts. They are simply plots of the function (for complete derivation see [2])

[3]These experiments are not to be confused with the experiments for estimating the counts. The latter are run an *a priori* unbounded number of times, i.e., until the desired accuracy is reached.

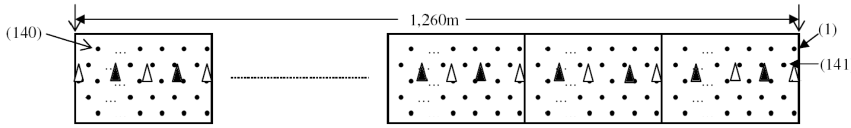

Figure 5. The complete layout of the Thick Horizontal part of the ExScal deployment. There are 729 motes and 29 stargates (notice that the leftmost section contains an extra stargate). Stargates shown as black triangles communicate only in Tier 2.

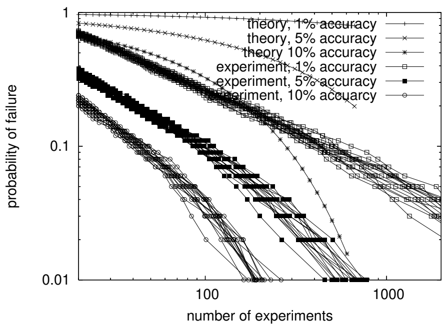

Figure 7. Convergence speed of probabilistic count estimates based on exponential variates.

$k \to 1 - e^{-\frac{k}{1+\varepsilon}} \sum_{i=0}^{k-1} \frac{k^i}{i!(1+\varepsilon)^i} + e^{-\frac{k}{1-\varepsilon}} \sum_{i=0}^{k-1} \frac{k^i}{i!(1-\varepsilon)^i}$, where $\varepsilon \in \{0.01, 0.05, 0.10\}$. The actual estimates are consistently better than theoretical predictions. An estimate from around 50 experiments is good enough for many applications—it is within 10% of the exact count with probability 0.9. The theoretical requirement for the same accuracy and confidence is about 10 times as high.

8. CONCLUSION

We have shown how to aggregate data from a sparse set of relevant nodes in a wireless sensor network. We believe that the probe and recall protocols proposed here can be useful in other contexts, e.g. for stopping a flood once its task has been accomplished, etc.

Acknowledgement: The authors wish to thank Anish Arora and Santosh Kumar of Ohio State for sharing with us the topology of their ExScal deployment, as well as the anonymous referees for their valuable comments. Jie Gao was supported by NSF CNS-0643687 and funding from Stony Brook's Center of Excellence in Wireless and Information Technology. Nikola Milosavljevic and Leonidas Guibas were supported by NSF grants CNS-0435111, CNS-0626151, ARO grant W911NF-06-1-0275, the Max Planck Center for Visual Computing and Communications, and the DoD Multidisciplinary University Research Initiative (MURI) program administered by ONR under Grant N00014-00-1-0637.

9. REFERENCES

[1] D. Braginsky and D. Estrin. Rumor routing algorithm for sensor networks. In *Proc. 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 22–31, 2002.

[2] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997.

[3] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, pages 449–460, 2004.

[4] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 25–36, 2005.

[5] S. P. Fekete, M. Kaufmann, A. Kröller, and K. Lehmann. A new approach for boundary recognition in geometric sensor networks. In *17th Canaidan Conference on Computational Geometry*, pages 82–85, 2005.

[6] S. Funke. Topological hole detection in wireless sensor networks and its applications. In *DIALM-POMC '05: Proceedings of the 2005 Joint Workshop on Foundations of Mobile Computing*, pages 44–53, 2005.

[7] B. Karp and H. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 243–254, 2000.

[8] M. Khan, V. Kumar, and G. Pandurangan. Distributed local algorithms for constructing approximate minimum spanning trees with applications to wireless sensor networks. Dept. Comput. Sci., Purdue University, West Lafayette, Indiana, 2006.

[9] A. Kröller, S. P. Fekete, D. Pfisterer, and S. Fischer. Deterministic boundary recognition and topology extraction for large sensor networks. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1000–1009, 2006.

[10] X. Liu, Q. Huang, and Y. Zhang. Combs, needles, haystacks: balancing push and pull for discovery in large-scale sensor networks. In *SenSys '04: Proceedings of the 2nd International Conference on Embedded networked sensor systems*, pages 122–133, New York, NY, USA, 2004. ACM Press.

[11] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny AGgregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 2002.

[12] A. Manjhi, S. Nath, and P. B. Gibbons. Tributaries and deltas: Efficient and robust aggregation in sensor network streams. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 287–298, 2005.

[13] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 250–262, 2004.

[14] D. Niculescu and B. Nath. Trajectory-based forwarding and its applications. In *Proc. 9th Annual International Conference on Mobile Computing and Networking (MobiCom 2003)*, pages 260–272. ACM Press, 2003.

[15] R. Sarkar, X. Zhu, and J. Gao. Double rulings for information brokerage in sensor networks. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 286–297, September 2006.

[16] S. Shakkottai. Asymptotics of query strategies over a sensor network. In *Proc. of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (InfoCom)*, pages 557–565, March 2004.

[17] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: New aggregation techniques for sensor networks. In *SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 239–249, 2004.

[18] P. Skraba, Q. Fang, A. Nguyen, and L. Guibas. Sweeps over wireless sensor networks. In *IPSN '06: Proceedings of the 5th international conference on Information processing in sensor networks*, pages 143–151, 2006.

[19] F. Stann, J. Heidemann, R. Shroff, and M. Z. Murtaza. RBP: robust broadcast propagation in wireless networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 85–98, 2006.

[20] Y. Wang, J. Gao, and J. S. B. Mitchell. Boundary recognition in sensor networks by topological methods. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 122–133, September 2006.