On Decentralized In-Network Aggregation in Real-World Scenarios with Crowd Mobility

Michal Gregorczyk, Tomasz Pazurkiewicz, and Konrad Iwanicki
University of Warsaw
Warsaw, Poland
{mg277528, tp277655}@students.mimuw.edu.pl   iwanicki@mimuw.edu.pl

***Abstract*—Recently proposed applications for monitoring the behavior of real-world crowds with wireless sensor nodes rely on decentralized in-network aggregation. Although some of the aggregation algorithms for wireless sensor networks seem appealing for such applications, we are not aware of any deployments of these algorithms in real-world scenarios with crowd mobility. As a step toward filling this gap, we thus discuss our experiences with decentralized in-network aggregation from a few such deployments involving up to 177 nodes. We compare two main classes of algorithms for basic aggregates. We show that algorithms based on probabilistic, order- and duplicate-insensitive sketches outperform algorithms based on gradual variance reduction. To this end, however, they have to be adapted considerably to minimize the traffic, latency, and errors of the aggregation process, and to account for some real-world issues. In short, while the algorithms do have a potential for the envisioned crowd-monitoring applications, deploying them is not trivial.**

I. Introduction

Monitoring and understanding the behavior of real-world crowds has started receiving increasing attention, especially in the context of urban planning and ensuring safety during mass events. For extracting a stream of data about a crowd of people, many envisioned crowd-monitoring applications rely on wireless sensor nodes worn by the people as smart gadgets, active tickets, or even micro-devices embedded in apparel [1], [2], [3]. To cope with the resulting data deluge, such applications often propose to employ decentralized in-network aggregation that leverages local inter-node communication. To this end, they assume that—using its low-power radio—a node carried by a person can exchange local data directly with nearby nodes of other people. In effect, some global phenomena can be analyzed locally: in the ad hoc network formed by the nodes. This reduces the volume of data transferred to a central site, where a more advanced analysis can take place.

While the aggregation functions required for analyzing certain phenomena may be quite involved [3], [4], the ability to to compute in the network even the basic aggregates, namely AVG, COUNT, MAX, MIN, and SUM, can already bring communication costs down. For example, in-network counting of nodes whose data satisfy a predicate allows for locally detecting whether a crowd phenomenon corresponding to the predicate is indeed taking place in a network region. Similarly, basic aggregates can be used to compute in the network the average velocity of a group of nodes, to identify likely overcrowded areas, or to estimate the total pressure exerted on a pedestrian overpass. It is not difficult to extend this list.

Due to its potential to minimize the overall communication, in-network aggregation has received considerable research attention from the wireless embedded sensing community. As we discuss in the next section, numerous aggregation algorithms have been proposed for wireless resource-constrained nodes. While many of them target static networks, some may be suitable also for mobile scenarios involving crowds.

However, despite its importance for crowd monitoring, decentralized in-network aggregation has not been studied extensively in actual crowd-monitoring deployments. In particular, we are not aware of any such real-world deployment that would utilize the proposed decentralized in-network algorithms for computing even the basic aggregate functions, such as a population count. Neither are we aware of an evaluation of the algorithms in actual wireless sensor networks where nodes are subject to (massive) human-speed mobility. Without such empirical experiences, it is hard to predict what performance one can expect from an algorithm and what tradeoffs can be made. As a consequence, it is also difficult to judge the feasibility of the envisioned crowd-monitoring applications.

In this paper, we make a step toward filling this gap by discussing our experiences from deploying decentralized in-network aggregation algorithms in scenarios with crowd mobility. Our work combines results of a few real-world deployments involving up to 177 actual wireless nodes with results of low-level simulations. The deployments demonstrate the performance of our ideas in practice, whereas the simulations aim to reproduce the encountered phenomena in a controlled environment, as well as to study scalability. We start by comparing two main classes of decentralized algorithms for the basic aggregates: based on gradual variance reduction [5], [6], [7] and based on probabilistic, order- and duplicate-insensitive sketches [8], [9], [10], [11], [12]. We show that the second class is more suitable for crowd-monitoring applications, but requires extensive adaptation to be practical. In particular, we present several techniques that are crucial to minimizing the traffic, latency, and errors of the aggregation process, and that range from reorganizing communication to customizing sketches. We demonstrate the effectiveness and scalability of these techniques. Finally, we also share some lessons learned about organizing real-world experiments involving crowds.

The rest of the paper is structured as follows. Section II surveys related work. Section III introduces the encountered problems and our solutions. Section IV presents the performance of the solutions in real-world deployments. Finally, Section V concludes. The appendix, in turn, studies scalability.

II. Related Work and Preliminaries

Due to resource constraints of sensor nodes, in-network aggregation has received considerable research attention from the embedded sensing community [13]. A common approach is to employ an overlay network, such as a tree [14], [15] or a cluster hierarchy [16], [17], [18], to drive the aggregation process. Although in static networks this approach produces accurate aggregate values at minimal traffic and latency, it fails when the inter-node connectivity becomes dynamic, which is the case in crowd-monitoring scenarios. In such scenarios, probabilistic algorithms may be more appropriate. These algorithms sacrifice the traffic, latency, and accuracy of the aggregation process for robustness to network dynamics. They can be divided into two main classes: (1) based on gradual variance reduction and (2) based on order- and duplicate-insensitive sketches.

A. Gradual Variance Reduction

Algorithms from the first class [5], [6], [7] combine push-pull gossiping with binary functions reducing the argument difference. For example, to compute a global average, each node periodically selects another random node, exchanges its local value with the node, and the two nodes set their values to the average of the exchanged ones. Eventually, the local value of every node approximates the global average. Other basic aggregation functions can be computed similarly [5]. In particular, to count the total number of nodes, one node starts the averaging with its local value equal to one, whereas the other nodes start with zeroes; the estimate of the total node count is the inverse of the final global average.

To achieve an optimal latency, whenever initiating a value exchange, a node should select the other node uniformly at random from the entire population [5]. This, however, entails multi-hop routing, which, in addition to being expensive, tends to be highly unreliable in mobile networks of wireless sensor nodes. While geographic gossip [6] may alleviate the costs, it still relies on routing. Therefore, for the inherently dynamic crowd-monitoring scenarios, avoiding routing entirely, by having each node gossip just with the nodes within its radio range (i.e., one-hop neighbors) [7], may be an attractive option.

B. Probabilistic Sketches

The second class of algorithms relies on small-state probabilistic data structures for order- and duplicate-insensitive aggregation [8], [9], [10], [11], [12]: so-called sketches (or synopses). An example is probabilistic counting [19], in which a sketch is a short bitmask (e.g., 16 bits). To estimate the node count using such sketches, each node independently initializes its own sketch by setting a single bit and zeroing the remaining ones. The bit to set is selected geometrically at random, that is, the probability of selecting the i-th least significant bit is $2^{-(i+1)}$, where $0 \leq i < 15$. The nodes then repeatedly exchange their sketches and merge them with a bitwise OR operation. When a node has OR-ed its sketch with the sketch of every other node (possibly indirectly), it can approximate the total node count with $1.2928 \cdot 2^{pos_0}$, where pos_0 is the position of the first zero in the sketch and 1.2928 is a statistical constant. It is proved that a geometric average over a number of such approximations converges to the actual total node count [19].

Several other sketches for various aggregation functions exist [19], [20], [21], [22], [23]. They differ in size and accuracy of the approximations. Likewise, there are several algorithms that use these sketches to perform aggregation in wireless sensor networks [8], [9], [11], [12], [24]. Some of the algorithms employ overlay networks for exchanging and merging the node sketches [8], [9], which, as mentioned previously, is problematic in mobile scenarios. Others, however, adopt a simple push-only gossiping scheme, in which each node periodically broadcasts its sketch to its one-hop neighbors and repeatedly merges the sketches received from the neighbors with its local sketch [9], [11]. As such, these algorithms seem more attractive for crowd-monitoring applications.

C. Applications to Scenarios with Crowd Mobility

Nevertheless, while the theory behind probabilistic decentralized algorithms for in-network computation of basic aggregates is well established, to the best of our knowledge, no prior work has studied the applicability of the algorithms to crowd-monitoring scenarios. Although the need for in-crowd computation is widely recognized [2], [3], [25], we are not aware of any real-world crowd-monitoring deployment that would utilize the proposed decentralized in-network aggregation algorithms. What is more, we are not aware of any experiments with the algorithms in networks with human-speed mobility: the algorithms were mainly proved analytically, rarely being accompanied by simulations in static networks [8], [9] or numerical results [12]. This paper is thus a step toward filling the gap between the theory of decentralized in-network aggregation and its applicability in crowd-mobility scenarios.

III. Problems and Solutions

The inspiration for our work was a deployment in which we collected real-world proximity-based interactions between 50+ people attending a social event. The collection system involved wireless sensor nodes carried by the participants that periodically exchanged short-range radio beacons, from which proximity was inferred. In preparation for future research activities on crowds, the system incorporated an additional algorithm based on gradual variance reduction [7] that aimed to supply us with a real-time count of the participants. Despite pre-deployment testing, the algorithm failed. Post-deployment analyses of the logs indicated that the problem of in-network aggregation deserved more attention than we had initially anticipated. Through subsequent simulations and small-scale experiments, we managed to identify the causes of the problems and propose solutions, which we discuss next.

A. Methodology

To analyze our findings, in this section we employ mainly low-level simulations, as they enable isolating individual problems; real-world experiments evaluating the solutions are in turn discussed in the next section. The simulations were done in OMNeT++ [26] with the MiXiM extensions for wireless sensor networks [27]. It is a signal-level simulation engine for mobile networks with realistic low-power communication models. For consistency, we configured the radio parameters of the simulated nodes to match those of our sensor nodes.

To concisely illustrate in this paper all the major problems, we distilled the aforementioned earlier simulations and small-scale experiments into a single mobility scenario. The scenario involves three groups, G1, G2, and G3, of 333, 222, and 444 nodes, respectively, that merge into larger groups and split while moving at human-scale velocities (i.e., at up to 6 km/h). As an aggregation function, the scenario uses the count of the number of nodes in a connected component. We selected COUNT for illustrative purposes: when the groups merge or split, the count accumulates or divides, respectively. In contrast, MIN and MAX are trivially order- and duplicate-insensitive, and thus, do not require sketches. SUM, in turn, can be implemented with sketches for COUNT, and likewise, AVG can be implemented with sketches for COUNT and SUM.

Figure 1 presents the COUNT values the nodes should ideally have at different stages of the scenario. Initially, groups G1 and G2 form a larger group, and hence, the aggregate value at all their members is 555, whereas the aggregate value for the members of the isolated G3 is 444. Subsequently, G2 splits from G1 (split 1) and moves toward G3, so that G2 and G3 merge into a group of 666 nodes (merge 1). G3 later splits from G2 (split 2) and merges with G1 (merge 2), forming a group of 777 nodes. Finally, G2 joins the group (merge 3), which results in a single group of 999 nodes.

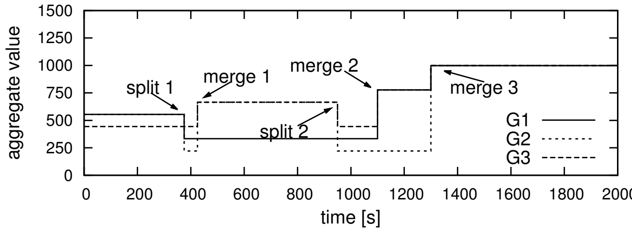


Fig. 1. The ideal evolution of the COUNT value in the simulated scenario.

B. Technique Comparison

We start by comparing the two classes of decentralized in-network aggregation algorithms: based on gradual variance reduction (GVAR) and based on order- and duplicate-insensitive sketches (ODIS). To make the comparisons systematic, for both classes, we assume local gossiping: every Δt time units, each node transmits a message with its local aggregate. In GVAR, an oracle provides the node with a random neighbor to which the node sends its current aggregate value and from which it receives a similar value. In ODIS, the node broadcasts its sketch to all neighbors. At any time, the node can receive similar messages from its neighbors. Since both the algorithms are in principle infinite, we assume that they are able to compute the final aggregates within an *epoch* lasting ΔT. Based on earlier experiments, we set Δt=100 ms, ΔT=150 s.

Moreover, in GVAR, to enable counting without fixing the node that starts with its value, c, equal to 1, rather than only c, each node aggregates a pair, (c,m). Initially, it sets c to 1 and m to its own unique identifier (ID). Whenever it receives a pair (c',m') from a neighbor, it sets the new local values as follows: if $m = m'$, then $(c_{new},m_{new}) \leftarrow (\frac{c+c'}{2},m)$; if $m > m'$, then $(c_{new},m_{new}) \leftarrow (\frac{c+0}{2},m)$; otherwise, $(c_{new},m_{new}) \leftarrow (\frac{0+c'}{2},m')$. In other words, each node participates in the counting that was started by the node with the largest ID known to the node: initially, this is the node's own ID; eventually, it is the maximal ID in the node's connected network component.

In ODIS, in turn, to use a similar amount of state as in GVAR, each node maintains four 16-bit probabilistic counting sketches instead of one. The final aggregate for an epoch is a geometric average of the estimates from all the four sketches.

Figure 2 shows the actual aggregate values obtained with GVAR in a representative run of the previous scenario. The aggregates hardly resemble those from Fig. 1. In some epochs, they are greatly overestimated and do not converge (e.g., 450 s, 1200 s, 1450 s). Sometimes they do converge within the accuracy of floating-point numbers, but their values are either overestimated or underestimated (e.g., 150 s or 1950 s).

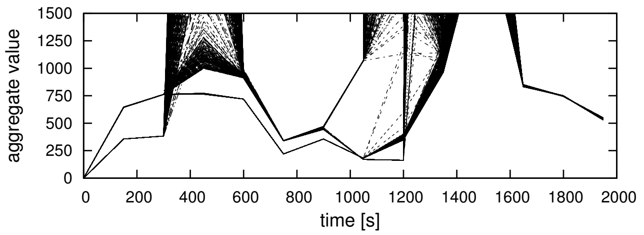


Fig. 2. The evolution of the COUNT value in the simulation for GVAR.

We have identified this behavior to be not an artifact of our implementation, but an inherent feature of the GVAR technique. More specifically, GVAR assumes that a value exchange between two nodes is atomic. If just the node with the higher initial value updates its value, the final global count will be overestimated. Conversely, if just the node with the lower initial value sets the new value, the global count will be underestimated. Guaranteeing atomicity of the exchange thus requires that the two nodes be able to consent—irrespective of any potential communication failures (i.e., message losses)—whether to update their values or not. Such a problem of two-node consensus in the presence of a failing link is known as the Two Generals Paradox [28], and it has been proved impossible to solve [29]. As a consequence, GVAR is bound to fail in the presence of unreliable communication.

Although in many networks communication is largely reliable, and thus, the atomicity problem does not have a great impact on GVAR, in scenarios with crowd mobility this is no longer true. Low-power wireless communication, especially in dense networks, is itself unreliable. The mobility of nodes further aggravates the effect, particularly when two nodes split in the middle of a value exchange not to meet again. In effect, many errors are introduced into the final aggregates. Moreover, the long convergence time of averaging with local gossiping makes reacting to massive network changes problematic, which is especially visible during the group merges in Fig. 2. All in all, GVAR is poorly suited to crowd-monitoring applications.

Figure 3, in contrast, presents the aggregate values obtained in an ODIS run. They are closer to the ideal ones from Fig. 1 than the aggregates for GVAR. Moreover, for individual nodes in a single connected network component, they are equal. This is because the order- and duplicate-insensitivity of sketches makes them highly robust to network dynamics. Suppose that a sketch broadcast by a node is not received by some of the node's neighbors. It can be received when the node rebroadcasts it in the next communication round. Furthermore, its one bits are present in the sketches of the neighbors that did receive it, and hence, when they broadcast their sketches, the neighbors missing those bits will likely receive and OR them. As a result, not only do the aggregates of individual nodes converge, but also the convergence process may be faster.

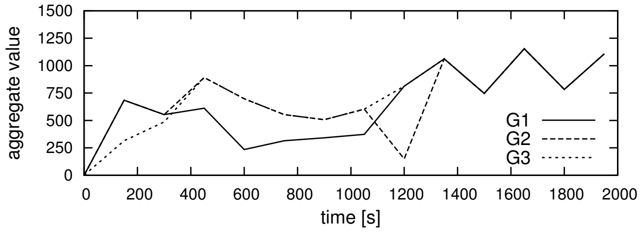


Fig. 3. The evolution of the COUNT value in the simulation for ODIS.

However, a straightforward implementation of an ODIS algorithm, such as the one considered hitherto, also exhibits problems. First, the estimated aggregates oscillate around the actual ones. The difference between two subsequent estimates can be significant [19], as can also be observed in Fig. 3 for the period between 1350 s and 1950 s. This behavior stems from the probabilistic nature of the sketches. Moreover, the employed 150-s epochs impair the consistency of the aggregates. For example, in the epoch ending at 450 s, the combined G2 and G3 estimate nearly 1000 nodes instead of 666, because the epoch covers both split 1 and merge 1, and thus, G2 and G3 also count the nodes from G1. Finally, the employed communication scheme involves a lot of redundant messages: each node broadcasts every 100 ms, which inflates the traffic. All in all, while ODIS seems more suitable for crowd-monitoring applications than GVAR, deploying it in practice requires minimizing the errors, traffic, and latency of the aggregation process. We address these issues next.

C. Organizing Communication

When it comes to communication, it is important not only to minimize traffic and latency, but also to adapt to network dynamics. The mobility patterns of crowds can lead to rapid and massive density and population changes, for instance, when many people converge on an area.

To cope with these problems, we observe that, unlike GVAR, ODIS can be made largely asynchronous. Each node can be allowed to advance arbitrarily in the computation of an aggregate, as long as all nodes finish within an epoch. For the same reason, it can broadcast its sketches at arbitrary moments. As a result, efficient communication schemes can be devised.

With this observation, we have decided to adapt the Trickle algorithm for code propagation [30] to perform ODIS-based aggregation. Originally, Trickle relies on repeatedly broadcasting the same data and combines two self-managed mechanisms: a broadcast timer and a suppression counter. The timer schedules broadcasts such that both: data can be disseminated fast and the algorithm can self-adapt to various network densities. To this end, when new data appear at a node, the node sets its timer to a short interval, T_{min} (on the order of milliseconds), to quickly push the data to other nodes. A broadcast is done randomly within this interval to alleviate transmission collisions with other nodes. After the interval finishes, subsequent intervals double up to T_{max} (on the order of minutes). In effect, even if the node's broadcast is lost in the initial intervals, it will likely succeed in later ones. The suppression counter, in turn, limits redundant traffic. More specifically, in every interval, each node counts broadcasts received with its own version of data. If the number of such broadcasts in an interval exceeds a threshold, c_{min} (typically 2 or 3), the node suppresses its own broadcast in the interval.

Our use-case differs from code propagation in a few aspects. First, there are many data sources: each node reinitializes its own sketch in every epoch. Second, rather than the same, subsequent broadcasts should contain data that advance the computation. Finally, data need not be broadcast indefinitely. On the contrary, the computation should finish within an epoch.

Therefore, although we employ Trickle's mechanisms, the new goals necessitate dedicated policies governing the use of these mechanisms. More specifically, while, like originally, subsequent timer intervals double, a node resets its timer to T_{min} when (1) it advances its epoch, (2) receives an inferior sketch from a neighbor (e.g., one with fewer bits set), or (3) its local sketch changes significantly as a result of combining it with a sketch received from a neighbor. This aims to boost the aggregation process when necessary while minimizing communication when the sketches become stable. Moreover, like originally, the node broadcasts its sketch at random moments within subsequent intervals and counts broadcasts received with the same sketch. However, the counter is not zeroed in every interval, but just upon epoch start and local sketch change. In effect, after receiving c_{min} broadcasts with the final sketch, a node stops aggregate computation until the next epoch. This aims to limit redundant traffic.

Figure 4 shows how the new communication scheme, with T_{min} = 64 ms, T_{max} = 2 s, c_{min} = 3, and an epoch of 2 s improves the aggregation process. In short, the aggregates are computed fast and with few messages per node. In particular, an epoch of just 2 s and only 6.01 messages per node per epoch on average are sufficient for the aggregates to converge globally in the 999-node network from the figure. Such self-managed, low-traffic, low-latency computation facilitates keeping the aggregates up-to-date with the changes in the network. More importantly, the scheme scales well with both node population and density, as we demonstrate in the appendix.

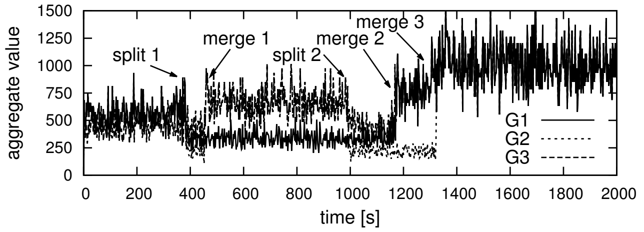


Fig. 4. The evolution of the COUNT value in the simulation for ODIS with our new communication scheme based on customized Trickle.

D. Improving Accuracy

Having an efficient communication scheme, we focus on alleviating aggregation errors. To this end, we consider two approaches, which can be combined: (1) increasing the amount of information and (2) employing more efficient sketches.

The first approach boils down to smoothing an aggregate by computing it from several sketches rather than just from one. A straightforward method is to replicate sketches and average their estimates. In particular, in the previous simulations, we used four sketches per aggregate, but it makes sense to use at least as many as can fit in a message (50 bytes for our nodes). This is because in wireless sensor networks the transmission overhead can be significant, and thus, it is reasonable to maximize message payloads. In addition, making use of the fact that our communication scheme enables computing aggregates fast, one can average aggregates from several consecutive epochs. Alternatively, exploiting the fact that the scheme also generates little traffic, one can run many instances of the same query in parallel, which we refer to as subqueries.

Although smoothing does offer benefits, its applicability is limited. In particular, beyond a few tens, adding more instances of probabilistic counting sketches gives virtually no error reduction [19]. Moreover, the additional instances increase the amount of state maintained by each node per query. For this reason, more efficient sketches are also worth consideration.

We have analyzed several sketches, also experimenting with some of them. While for many, accuracy improvements are marginal, we identified two families that differ more significantly. The first encompasses sketches that operate without additional knowledge, such as the probabilistic counting [19] or the popular hyper-log-log [22]. In contrast, sketches from the second family operate well only if configured with a parameter whose value depends on the result of aggregation. An example is linear counting [21], which uses selected bits of a virtual bitmask, whose size is proportional to the number of counted data items. Given the same number of bits, linear counting is more accurate than probabilistic counting and hyper-log-log. Yet, despite a large tolerance, the size of its virtual bitmask must depend on the number of counted items.

In the dynamic crowd-mobility scenarios, it is often not easy to predict even roughly, before a deployment, what an aggregate value should be, and thus, how to configure a parametrized sketch. On the other hand, if configured properly, such as sketch is more accurate than one without the additional knowledge. Therefore, to enable parametrized sketches in crowd-monitoring applications, we combine them with parameterless ones into a pipeline. A parameterless sketch obtains the first approximation of the result when it is not yet known. The approximation is used to configure a parametrized sketch, such that in the next epoch, it can produce a more accurate result. The two sketches operate continuously in parallel, such that their results from a given epoch are fed to the configuration of the parametrized sketch in the next epoch.

This approach is a generalization of a scheme proposed independently by Cichon et al. [31]. Moreover, in contrast to that scheme, it effectively handles the following issues stemming from mobility. First, two nodes may use different parameters for their sketches, for instance, because they were previously isolated. We resolve such conflicts deterministically, for example, by adopting the greater parameter. Second, while parametrized sketches leave a considerable slack for parameter errors, they are vulnerable to massive network changes. In particular, linear counting, configured based on an epoch, significantly overestimates the result if large node groups merge in the next epoch. To cope with this issue, we adopt the following heuristic: if in an epoch the divergence of the result from the previous epoch is larger for a parametrized sketch than for a parameterless one, we fall back to the result from the parameterless sketch. This minimizes aggregate oscillations at the cost of potentially slower reaction to changes.

Figure 5 shows how the presented solutions improve the aggregates in the previous scenario. The configuration involves sketch pipelining with 20 bytes for probabilistic counting and 20 bytes for linear counting (plus 10 bytes for metadata), and smoothing with a running window over 6 epochs (12 s). The remaining parameters are in turn configured as previously.

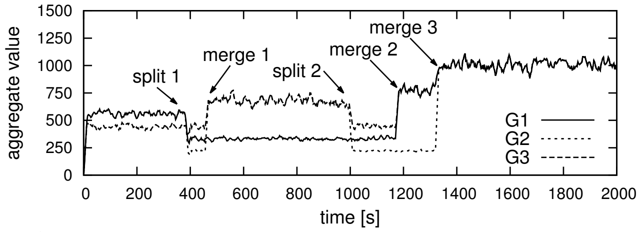


Fig. 5. The evolution of the COUNT value in the simulation for ODIS with our communication scheme and error-reducing solutions.

In addition to still being computed fast and with little traffic (14.9 packets per node per epoch on average), the aggregates resemble the ideal ones rather closely (cf. Fig. 1). Due to sketch pipelining alone (not plotted), their errors are smaller, and smoothing helps further. What is more, our solutions can be easily tuned to individual applications by exploiting their traffic-accuracy-latency tradeoffs. For instance, with multiple subqueries, the traffic grows, but the errors decrease. Likewise, smoothing reduces errors, but increases the latency of reacting to changes. Finally, altering the settings of Trickle (e.g., T_{min}) allows for balancing traffic and latency. In short, the solutions do have a potential to make ODIS applicable in the real world.

E. Final Remarks

Overall, the discussion indicates that adapting ODIS to crowd-mobility scenarios is not trivial, but the effects are promising. Due to space constraints, we could present only the most important issues and results, and had to omit others, such as studies of the traffic-accuracy-latency tradeoffs of our solutions, their behavior for different mobility patterns, aggregation functions, and configurations, or low-level techniques improving communication [32]. For the same reason, we moved scalability studies of the solutions to the appendix.

IV. Real-World Deployments

The presented solutions gradually evolved into an entire aggregation service that we implemented in TinyOS 2.1, a popular operating system for wireless sensor nodes. The service can be instantiated multiple times in an application. Each instance is an independent sketch in that its state and the associated initialization, merging, and evaluation operators can be customized with the solutions from the previous section to best exploit the traffic-accuracy-latency tradeoffs. The initialization operator can in addition contain a predicate that allows a node to locally assess whether the node's data item(s) should contribute to an aggregate. All in all, not only can multiple aggregates be computed simultaneously in a network, but also, by combining several sketches with different predicates, even complex, SQL-like aggregation queries using functions beyond AVG, COUNT, MAX, MIN, and SUM, can be expressed.

A. Deployment Organization

The TinyOS implementation has enabled evaluating our solutions on many real-world platforms: from existing sensing platforms to new smart-gadget platforms. In particular, for the presented deployments, we chose eZ430 Chronos smart watches (see Fig. 6), to which we had earlier ported TinyOS itself. The watches are comfortable to wear, feature a convenient I/O interface in the form of an LCD and buttons, and have an adjustable radio range (set to approximately 5 m in our experiments). In addition, their resources are far more constrained than for instance those of smartphones. As such, they can arguably be a realistic approximation of a platform on which the envisioned crowd-monitoring application may run.

eZ430 Chronos smart watch	
RAM	4KB
Flash	32KB
MCU frequency	16MHz
Radio throughput	max. 250kbps
Radio range (in our experiments)	∼5 m

Fig. 6. Our main deployment platform: eZ430 Chronos.

However, experimenting with the watches was challenging. Programming and preparing tens of watches for a typical experiment required a considerable amount of time. So did building passive sniffer networks that allowed us to monitor and log inter-node communication. Finally, to provide "ground truth" for the experiments, we often provisioned an independent, reliable verification infrastructure, such as cameras.

Equally important were the social challenges associated with the deployments. A meaningful experiment requires a sufficient number of participants, which is especially important when studying crowds. To attract tens of participants, we thus typically provided additional incentives. What is also unique about experiments with crowds is that the experimental scenarios, on the one hand, should not be boring for the participants, and on the other, should allow for assessing the performance of the deployed algorithms in specific situations. Preparing such scenarios for our algorithms was a challenge in itself. So was ensuring that the deployed system worked without failures, thereby not ruining our reputation for subsequent experiments. To that end, we employed multi-level pre-deployment testing, notably using our novel unit-testing framework for TinyOS.

Finally, during the deployments, we had to deal with unanticipated problems. A few watches would typically fail to start, in most cases, due to a short circuit resulting from their invalid reassembly after battery replacement. A few supposedly new batteries had their capacity way under the specified minimum, which resulted in premature deaths of the corresponding watches. A couple of watches were lost, which forced us to recover their state from the sniffer logs. Other examples of similar issues are plentiful. In summary, conducting our real-world deployments with crowd mobility was a challenging and laborious process.

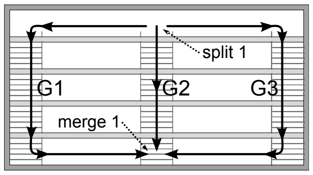
B. System Setup

Nevertheless, we have managed to conduct a few such successful deployments, each with one or more crowd behavior scenarios. Due to space constraints in this paper, we selected four scenarios that in our view are most illustrative for the presented solutions. For the same reasons, although our aggregation service supports even complex, SQL-like queries beyond the basic AVG, COUNT, MAX, MIN, and SUM, which we did make use of occasionally, here we focus only on the previous COUNT query for connected components, which was used virtually in all our deployments. The other basic functions did not exhibit unanticipated performance differences.

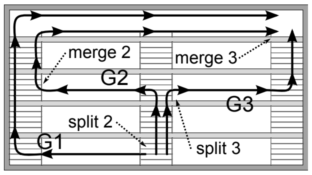
The instance of the aggregation service corresponding to the COUNT query, either, was running continuously on all the watches or was started for one-shot evaluation upon request from one of a few special nodes through which we controlled some of the experiments. We used the previous default configuration for the service. More specifically, it employed sketch pipelining with 20B of state for probabilistic counting and 20B for linear counting, and no smoothing. The minimal inter-beacon interval of the Trickle timer, T_{min}, was 64 ms, while the suppression threshold, c_{min}, was 2 messages. The duration of an epoch was set conservatively to 2.5 s, so that the aggregates could converge even in a 200-node network with other communication possibly taking place in parallel.

C. Scenario 1: Group Merge and Split

The first selected scenario was a part of a deployment conducted specifically to evaluate our aggregation solutions. It involved 54 watches worn by people divided into three groups, G1, G2, and G3, of 20, 19, and 15, respectively, and took place in a five-story building with three staircases. Its goal was to ascertain that the solutions can properly handle group merges and partitions, to which end the groups moved as depicted in Fig. 7(a). The COUNT query was continuously producing aggregates throughout the entire experiment.



(a) Scenario 1



(b) Scenario 2

Fig. 7. The group mobility schemes in Scenarios 1 and 2.

Figure 8 presents the evolution of the count values for individual group members, which the participants also observed in real time on the LCDs of their watches. The initial count growth corresponds to activating the watches at the top floor, as this was the first experiment of the deployment. The process was performed by a few assistants who used a special button sequence, intended to prevent unauthorized watch (de)activations. As it took time, Fig. 8 shows activating only the last 15 watches. Afterward, each group headed to a different staircase, which split the network into 3 smaller ones (split 1), and then down and toward the main lobby at the ground floor, until all groups merged again (merge 1).

Several observations can be made from Fig. 8. First, the count value recorded locally by individual group members was the same; yet, due to the unreliability of low-power wireless communication, this need not have been the case. Second, split 1 was first observed by G1 as an initial decrease in COUNT to 40, whereas G2 and G3 observed the partition one epoch later. The reason was that—just before the split—a member

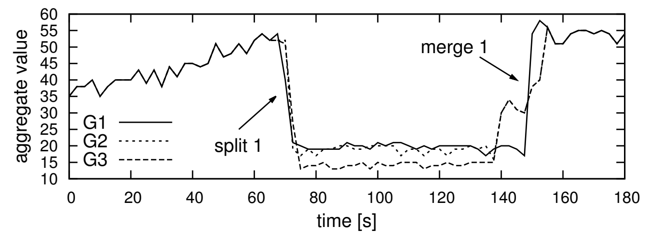


Fig. 8. The evolution of the COUNT value in Scenario 1 [Fig. 7(a)].

of G2 and G3 received a complete aggregate from a member of G1, but members of G1 received only partial aggregates (i.e., not OR-ed with all other aggregates) from the members of G2 and G3. Such inconsistencies can happen also due to the asymmetry of low-power wireless communication, especially in highly unstable periods, such as partitions and merges. Short epochs and smoothing can help to alleviate them. Third, the groups moved at different speeds: G3 merged with G2 first, and a while after was also joined by G1. Fourth, again, due to the asymmetry of wireless links, the final merge was first recorded by G1, and only after two epochs by G2 and G3.

Overall, Fig. 8 shows that apart from the brief inconsistencies resulting from the inherent properties of wireless low-power communication, our solutions exhibited small aggregate errors. Likewise (not plotted), they performed well in terms of traffic and latency: aggregate values were produced in under a second and with $\sim$5 messages per node per epoch on average.

D. Scenario 2: Synchronous Group Mobility

The second selected scenario, explained in Fig. 7(b), was among a few others exercised in the same deployment and with the same group composition, but aiming to assess increasingly more involved mobility patterns. Figure 9 presents the aggregate values for the individual members. Like previously, the sequence of splits and merges reflected in the aggregate values in the figure matches the experimental scenario. The groups split (split 2), such that G2 and G3 walked together for a while, and then also split (split 3). Subsequently, G1 and G2 merged (merge 2), to finally also merge with G3 (merge 3).

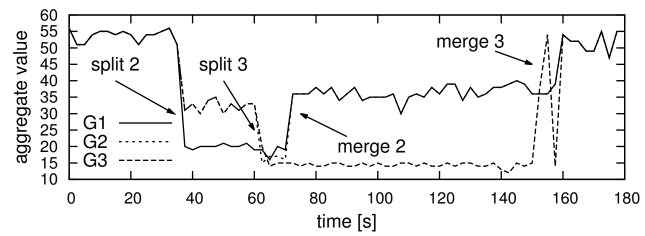


Fig. 9. The evolution of the COUNT value in Scenario 2 [Fig. 7(b)].

Again, all members of the same group observed the same count values on the LCDs of their watches. Like in Scenario 1, due to communication asymmetry, there was also a small delay in detecting split 3 by G3 compared to G2. A new interesting phenomenon is in turn the spike in merge 3. It is an artifact of a wireless link that was not only asymmetric, but also close to the signal quality threshold below which no reception is possible. More specifically, when G3 was in front of a door behind which G2 and G1 were waiting, one of its nodes heard a message from G2 over such a poor link. No message was heard in the following epoch, though, and hence, the spike. Like previously, however, the solutions performed well overall.

E. Scenario 3: Temporary Group Bridging

The third scenario comes from a different deployment in which the COUNT query was used just as a real-time feedback for monitoring and controlling the experiment. Yet, its results do provide some insight into the performance of our solutions, and hence, were selected for presentation here.

The scenario involved four groups, G1–G4, of 15, 17, 22, and 18 nodes. G3 and G4 were rather static and isolated from each other. G1 and G2, in turn, moved in the same direction but at different velocities, thereby merging and splitting with G3, G4, and each other, as sketched in Fig. 10. The aggregates observed by the members of G1 and G2 are shown in Fig. 11.

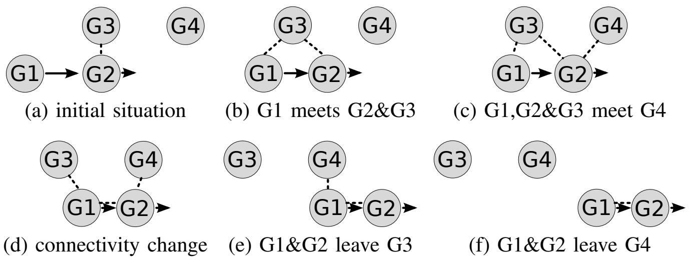


Fig. 10. The group mobility scheme in Scenario 3.

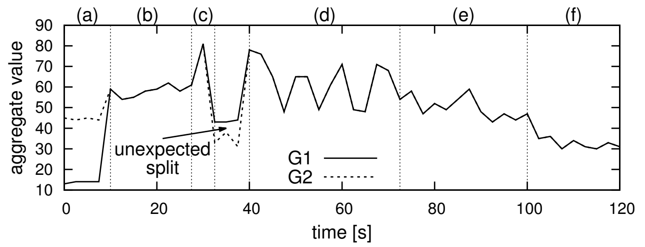


Fig. 11. The evolution of the COUNT value in Scenario 3 [Fig. 10].

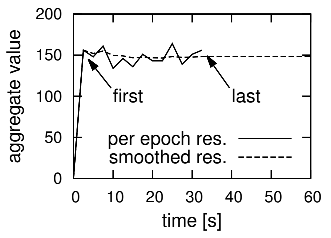
Like previously, the aggregates from Fig. 11 mostly match the scenario. Initially, G2 and G3 were close enough to form a single group, to which later G1 joined, as it was moving faster than G2. As a result of their continuous movement, G1 and G2 came close also to G4, so that for a while all four groups formed a single connected network. Subsequently, all nodes of G1 and G2 left the radio range of the nodes of G3, thereby splitting the network. Finally, they also left the range of G4.

Two phenomena in Fig. 11 are worth mentioning, though. First, during the transition between situation (c) and (d), G1 and G2 were still far apart, so the links between them were virtually nonexistent. In effect, for three epochs, the network split into a group comprising G1 and G3 and another comprising G2 and G4. Second, the oscillations of the aggregates in situation (d) (all groups connected) seem relatively large. After extensive analyses, we concluded that the aggregates were correct, given the pseudo-random numbers produced by the nodes' local generators. They might possibly have been lower if we replaced the TinyOS default linear congruential generator with a better one, such as Mersenne Twister. They would have certainly been lower if we employed smoothing by averaging. Nevertheless, we have selected this scenario deliberately, to visualize that aggregate oscillations are an inherent feature of probabilistic, sketch-based aggregation, and thus, any application on top has to be prepared to handle them.

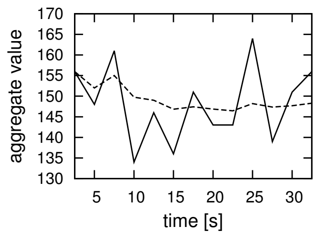
F. Scenario 4: Uncontrolled Mobility

The last selected scenario took place in a long foyer and involved 177 nodes. The COUNT query was again used as a support service, providing a real-time estimate of the participating nodes, but this time on demand, after a command from one of our additional control nodes. To alleviate the oscillations and obtain more accurate aggregate values, we configured the query with aggregate smoothing over 12 epochs (30 s). This was motivated by the fact that the mobility patterns during the deployment were less rapid than during the previous ones: people were mostly talking, rarely moving. The inter-node connectivity remained dynamic nonetheless, in particular, due to gesticulation and local movements of the participants.

Figure 12 depicts a 12-epoch COUNT query for the nodes forming the largest connected network component. Like previously, the raw aggregates produced in subsequent epochs did oscillate, but smoothing made the result progressively more stable. In particular, the actual number of nodes in the largest connected component was 148, and this was also the final value obtained after smoothing. This illustrates that while decentralized in-network aggregation algorithms are in practice far from perfect, by exploiting the traffic-accuracy-latency tradeoffs in our solutions and the domain knowledge on individual applications, one can tune their performance.



(a) big picture



(b) zoom-in on the computation

Fig. 12. The smoothing of the COUNT value in Scenario 4.

V. Conclusions

All in all, applying existing decentralized in-network aggregation algorithms for wireless sensor networks in real-world crowd-monitoring scenarios is easier said than done. Algorithms based on gradual variance reduction are bound to fail, considering their assumptions. Algorithms based on probabilistic order- and duplicate-insensitive sketches, in turn, can produce reasonably accurate aggregate values with little traffic and low latencies, but have to be adapted considerably. Even then, however, they may suffer from real-world phenomena, such as asymmetric and extremely weak links, which are inherent to low-power wireless communication, and heavy network dynamics, which occur under crowd mobility.

Crowd-monitoring applications thus have to be able to cope with such problems and exploit traffic-accuracy-latency tradeoffs. Oversimplifying, they have to be prepared that the stream of values they obtain from an aggregation service will look more like, for instance, in Fig. 9 than in Fig. 1. This may require revisiting the assumptions of some of such applications.

Finally, conducting just a medium-scale deployment studying crowds is a challenging and laborious process. Conducting a successful one is even more difficult. We thus hope that our real-world results will stimulate theoretical research on novel decentralized in-network aggregation algorithms dedicated for the envisioned crowd-monitoring applications.

Acknowledgment

The authors would like to thank K. Baranowska, M. Grabowski, P. Horban, G. Kolakowski, M. Marschall, R. Rawicki, J. Rekorajski, W. Sirko, M. Szczepaniak, M. Ziombski, and all other people that helped one way or the other in conducting the experiments. This research was supported by the (Polish) National Science Center within the SONATA program under grant no. DEC-2012/05/D/ST6/03582.

References

[1] R. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 32–39, Nov. 2011.

[2] H. Roitman, I. Cantador, and M. Fernández, Eds., *Workshop on Multi-modal Crowd Sensing (CrowdSens 2012)*, 2012.

[3] C. Martella, M. Steen, A. Halteren, C. Conrado, and J. Li, "Crowd textures as proximity graphs," *IEEE Communications Magazine*, vol. 52, no. 1, pp. 114–121, Jan. 2014.

[4] D. Roggen, M. Wirz, G. Tröster, and D. Helbing, "Recognition of crowd behavior from mobile sensors with pattern analysis and graph clustering methods," *Networks and Heterogeneous Media*, vol. 6, no. 3, pp. 521–544, Sep. 2011.

[5] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Trans. Comput. Syst.*, vol. 23, no. 3, pp. 219–252, Aug. 2005.

[6] A. G. Dimakis, A. D. Sarwate, and M. J. Wainwright, "Geographic gossip: Efficient aggregation for sensor networks," in *Proc. ACM/IEEE IPSN 06*, Nashville, TN, USA, 2006.

[7] L. Xiao, S. Boyd, and S. Lall, "A scheme for robust distributed sensor fusion based on average consensus," in *Proc. ACM/IEEE IPSN '05*, Los Angeles, CA, USA, 2005.

[8] J. Considine, F. Li, G. Kollios, and J. Byers, "Approximate aggregation techniques for sensor databases," in *Proc. IEEE ICDE '04*, Boston, MA, USA, 2004.

[9] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson, "Synopsis diffusion for robust aggregation in sensor networks," in *Proc. ACM SenSys 04*, Baltimore, MD, USA, 2004.

[10] D. Mosk-Aoyama and D. Shah, "Fast distributed algorithms for computing separable functions," *IEEE Trans. Inf. Theor.*, vol. 54, no. 7, pp. 2997–3007, Jul. 2008.

[11] C. Baquero, P. S. Almeida, R. Menezes, and P. Jesus, "Extrema propagation: Fast distributed estimation of sums and network sizes," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 4, pp. 668–675, Apr. 2012.

[12] H. Terelius, D. Varagnolo, and K. Johansson, "Distributed size estimation of dynamic anonymous networks," in *Proc. IEEE CDC 12*, Maui, HI, USA, 2012.

[13] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: A survey," *IEEE Wireless Commun.*, vol. 14, no. 2, pp. 70–87, Apr. 2007.

[14] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A scalable and robust communication paradigm for sensor networks," in *Proc. ACM MobiCom '00*, Boston, MA, USA, 2000.

[15] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," in *Proc. USENIX OSDI '02*, Boston, MA, USA, 2002.

[16] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *Trans. Wireless. Comm.*, vol. 1, no. 4, pp. 660–670, Oct. 2002.

[17] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann, "An evaluation of multi-resolution storage for sensor networks," in *Proc. ACM SenSys '03*, Los Angeles, CA, USA, 2003.

[18] K. Iwanicki and M. van Steen, "Using area hierarchy for multi-resolution storage and search in large wireless sensor networks," in *IEEE ICC 2009*, Dresden, Germany, 2009.

[19] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *J. Comput. Syst. Sci.*, vol. 31, no. 2, pp. 182–209, Sep. 1985.

[20] P. B. Gibbons and Y. Matias, "Synopsis data structures for massive data sets," in *External Memory Algorithms*, ser. DIMACS: Series in Discrete Math. and Theor. Comput. Sci., J. M. Abello, Ed. AMS, 1999, vol. 50, pp. 39–70.

[21] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 208–229, Jun. 1990.

[22] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," in *Proc. AofA 07*, Nice, France, 2007.

[23] A. Metwally, D. Agrawal, and A. E. Abbadi, "Why go logarithmic if we can go linear?: Towards effective distinct counting of search traffic," in *Proc. EDBT '08*, Nantes, France, 2008.

[24] D. Varagnolo, G. Pillonetto, and L. Schenato, "Distributed statistical estimation of the number of nodes in sensor networks," in *Proc. IEEE CDC 10*, Atlanta, GA, USA, 2010.
[25] D. G. Murray, E. Yoneki, J. Crowcroft, and S. Hand, "The case for crowd computing," in *Proc. ACM MobiHeld '10*, New Delhi, India, 2010.
[26] OMNeT++ Homepage. http://www.omnetpp.org.
[27] MiXiM Homepage. http://mixim.sourceforge.net.
[28] J. Gray, "Notes on data base operating systems," in *Operating Systems, An Advanced Course*. London, UK, UK: Springer-Verlag, 1978, pp. 393–481.
[29] E. A. Akkoyunlu, K. Ekanadham, and R. V. Huber, "Some constraints and tradeoffs in the design of network communications," in *Proc. ACM SOSP '75*, Austin, Texas, USA, 1975.
[30] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proc. USENIX NSDI '04*, San Francisco, CA, USA, 2004.
[31] J. Cichon, J. Lemiesz, W. Szpankowski, and M. Zawada, "Two-phase cardinality estimation protocols for sensor networks with provable precision," in *IEEE WCNC 2012*, Paris, France, 2012.
[32] T. Pazurkiewicz, M. Gregorczyk, and K. Iwanicki, "NarrowCast: A new link-layer primitive for gossip-based sensornet protocols," in *Proc. EWSN 2014*. Oxford, UK: Springer-Verlag LNCS 8354, 2014.

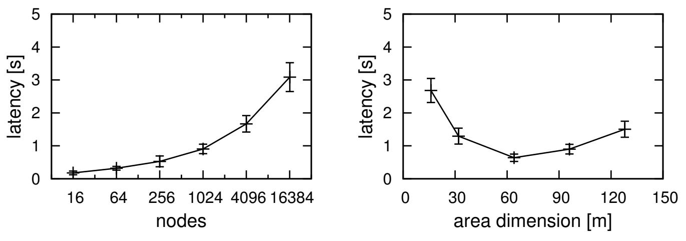
Appendix

In this appendix we study through simulations the scalability of our solutions with respect to node population and density. In contrast to the previous ones, in all simulations presented in the appendix the nodes were static. There are three main reasons for this decision. First, mobility can improve the pace of information dissemination, and hence, our solutions, so we wanted to avoid this effect. Second, in a mobile network the density and diameter change dynamically, whereas we aimed at a fairly controlled environment. Third, mobile networks require much more computations than static ones, and we wanted to simulate as large networks as possible.

To study how the solutions scale with network size, we conducted simulations for node populations increasing exponentially from 16 to 16,384. The latter one was the largest population we were able to simulate. To avoid completely homogeneous topologies, the nodes were randomly distributed in an square area. The sides of the area were $3 \times \sqrt{N}$ m long, where N was the population count. The node radio range was in turn ~5 m. As a result, a node had up to eight neighbors. Likewise, to study how the solutions scale with network density, we set the node population to 1024 and varied the sides of the area from 16 m to 128 m.

For consistency, we used the previous COUNT function. It was configured with sketch pipelining: 10 bytes for probabilistic counting and 30 bytes for linear counting. For illustrative purposes, no additional smoothing was used. The communication scheme, in turn, had the following configuration: T_{min} = 64 ms, T_{max} = 5 s, and c_{min} = 2 messages. An epoch was 5 s to handle the largest networks. Each point in the plots represents an average and standard deviation over 20 epochs.

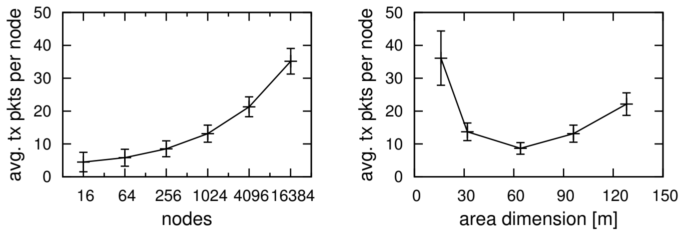
Figure 13 shows the scalability of aggregation latency: the time for all nodes to converge to the final aggregate. As expected, the latency increases with the network size (a). This is not only due to an increase in the network diameter, but also because more items are counted, and hence, more information has to be transferred. Nevertheless, the increase is graceful: our solutions are able to count 16,384 nodes in under 4 s. When it comes to density (b), there is in turn an optimal one. With a lower density (a larger area), the diameter grows, and hence, the latency increases. With a higher density (a smaller area), there is more congestion, which again affects the latency. Note, however, that our solutions recover from such congestion, and the resulting latency increase is not dramatic.



(a) wrt. network size (b) wrt. network density

Fig. 13. Scalability of aggregation latency.

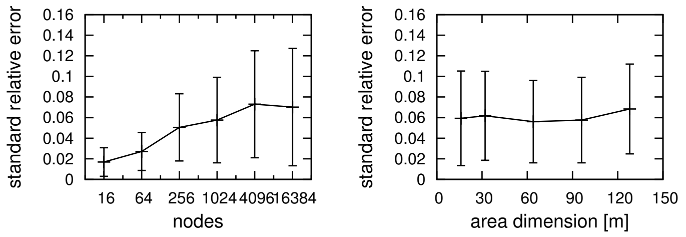
Figure 14 demonstrates that the aggregation traffic scales similarly to latency. This behavior can be explained with precisely the same arguments as for the previous figure. In addition, the high standard deviation in the densest networks (b) also confirms that such networks indeed exhibit congestion: due to transmission collisions, some sketches are transmitted more times than necessary. Overall, however, the solutions operate even under congestion and, in general, their traffic scales well with the size and density of the network.



(a) wrt. network size (b) wrt. network density

Fig. 14. Scalability of aggregation traffic.

Finally, Fig. 15 presents the scaling behavior of standard relative approximation error, that is, the ratio of an aggregate error to the actual aggregate value. The larger the number of nodes (a), the higher the ratio in linear counting of the virtual bitmask length to the number of bits from the bitmask that are present in the sketch, and hence, the larger the error. Network density (b), in contrast, hardly affects the error: even under congestion our solutions correctly compute the final results. Importantly, in all cases the error remains relatively low.



(a) wrt. network size (b) wrt. network density

Fig. 15. Scalability of aggregation error.

To sum up, the results suggest that our solutions indeed scale well. Moreover, they are just a small sample of the simulations that we have conducted to assess the solutions in various settings, including different mobility patterns, aggregation functions, and configurations. Although all those results are out of the scope of this paper, they indicate that by exploiting traffic-accuracy-latency tradeoffs, one can further tune the presented solutions to individual queries.