A PiggyBacking Approach to Reduce Overhead in Sensor Network Gossiping [*]

Ercan Ucan, Nathanael Thompson and Indranil Gupta
Department of Computer Science, University of Illinois Urbana-Champaign
201 N. Goodwin Ave.
Urbana, IL 61801-2302
eucan@nvidia.com[†] ,{nathomps,indy}@cs.uiuc.edu

ABSTRACT

Many wireless sensor network protocols are employing gossip-based message dissemination, where nodes probabilistically forward messages, to reduce message overhead. We are concerned with emerging systems in stationary sensor networks that are multiple-source with each message targeted at every recipient, such as query and code propagation. Default gossip-based approaches tend to treat each stream of messages from different senders independently of the others, overloading each node with message overhead summed from all streams. We apply intelligent scheduling strategies for gossip forwarding, effectively piggybacking streams atop one another, to address this significant message overhead. Our problem formulation introduces a new concept called the "semblance graph" used to schedule gossiping based on streams' gossip periods. Two new heuristic algorithms are proposed to solve the semblance graph problem. The performance of these two heuristics is on average within 3.5% of the optimal solution. Simulations show that the piggybacking strategy reduces the message, bandwidth, and energy overhead while still maintaining the original scalability, reliability and latency of the canonical gossip.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Network Protocols

General Terms

Algorithms, Performance

Keywords

Gossiping, PiggyBacking

*This work was supported in part by NSF CAREER grant CNS-0448246 and in part by NSF ITR grant CMS-0427089.
†Ercan Ucan is currently with NVIDIA, but was involved with this paper while at UIUC.

1. INTRODUCTION

Gossip based systems are becoming more and more popular in wireless sensor networks. Variations of gossiping have been used for basic broadcasting [6, 8, 17], code propagation [10, 16], enhancing sensor node sleep schedules [13], reliability mechanisms [4, 15] and even topology discovery [12]. The basic dissemination mechanism could either follow site percolation [6] or bond percolation [13] in which a sensor node periodically forwards the latest message with some probability typically less than one. Gossiping is an attractive approach for disseminating messages, such as network queries or code updates, in sensor networks because it offers good scalability, reliability and latency. Latency improves under gossiping because gossip's probabilistic forwarding avoids many collisions. Gossip systems' decentralized nature means there is no single point of failure or backbone infrastructure that can fail and therefore there is higher reliability. The scalability of gossip protocols in wireless sensor networks comes from the reduction in the total number of times a message is forwarded to spread throughout the network since nodes withhold messages if overhearing another node forward the same message. The per-node overhead for gossiping a message is at most a single radio broadcast to the surrounding sensor nodes.

While good at distributing messages from a single source to the whole network in a timely and reliable manner, still there is a high message overhead when multiple sources publish queries or code updates into the network. In reality, sensor network protocols often disseminate *multiple streams* (strings of messages from a single source) in the underlying gossip communications layer. Yet, most existing gossip mechanisms handle each stream independently, with each node typically generating separate gossip messages for each stream in the system. All of this causes the canonical sensor network site and bond percolation approaches mentioned above to create a per-node overhead that is the sum from all the streams. The overhead includes the extra bytes for every packet header and the energy costs to send more messages.

Because each stream is delivered to every sensor the packet overhead and total message count could be eliminated if messages from different streams were combined into a single message. Existing aggregation techniques [3, 5, 7, 9, 11] fail to help in this situation because they are designed to aggregate *data* and not *messages.* This paper adopts a general and effective approach to address this overhead problem– that of piggybacking gossip messages from one stream atop messages from another stream. In short, each node periodically sends out gossip messages that contain constituent messages from several streams. The problem of deciding which stream messages get included in which gossip message is then the *problem of gossip scheduling.* The goal in gossip scheduling is to reduce the message overhead while achieving the same latency, scalability, and reliability as canonical site/bond percolation with independent streams.

There are two constraints to be considered in the gossip scheduling problem. The first constraint comes from the fact that a collated (piggybacked) gossip message cannot exceed the maximum message payload size of the network. Therefore the number of constituent streams inside a single piggybacked message is limited. This is specified as a parameter k in our problem formulation. The second constraint comes from the fact that we would like to, in spite of piggybacking, maintain the scalability, reliability, and latency of the underlying gossip scheme, *for each individual stream.* While piggybacking reuses the same mechanisms of the canonical gossip the scalability and reliability of piggybacked gossip will be unchanged. However, messages must be delayed to be available for piggybacking, potentially affecting the latency of the streams. The above properties depend on the the stream's gossip period, which we assume is specified a priori for each stream. There are several periodic tasks in sensor networks from which to derive the period. The most obvious is the gossip period itself which is fixed according to the gossip protocol and application settings. Another solution would come from existing power saving mechanisms like IEEE 802.11 PSM which is also periodic.

Given a set of streams, each specified with a gossip period, our goal then is to come up with a static schedule of piggybacking multiple streams, subject to the above two constraints. The schedule is periodically recurring, and specifies at what times collated gossip messages are sent out, and which streams each collated message contains. Static scheduling is simpler and less of a computation burden on the motes. While scheduling it is more advantageous to piggyback two streams with *similar* gossip periods, than two streams with dissimilar ones. This is because the *utilization* of a collated gossip message will be higher in the former.

We formalize the two constraints via a new concept called the "semblance graph". The semblance graph is a logical complete graph, with each node in the semblance graph representing a distinct stream. An edge between two nodes carries a weight indicating the represented streams' "relatedness", as determined by their specified gossip periods. The relatedness metric lies in the real interval $[0, 1]$. The scheduling problem is to partition the nodes of the semblance graph into subgroups, each subgroup having at most k nodes, such that the cost of edges going *across* subgroup boundaries is as low as possible.

To solve the semblance graph problem, we propose two new heuristic algorithms based on existing Minimum Spanning Tree algorithms. Our heuristic algorithms have low running time compared to a brute force examination of the graph and achieve an average degradation of only 3.5% in accuracy. We then integrate our semblance graph based scheduling into a canonical gossip protocol for sensor networks, and through simulation show a reduction in overall message count by as much as 82% while achieving resiliency and latency close to the original gossip.

Contributions.

The contributions of this paper are as follows: We (i) identify the gossip stream scheduling problem and present a graph based definition of it called the "semblance graph"; (ii) propose two new heuristics algorithms to solve the "semblance graph" problem; (iii) present a sensor network implementation and simulation employing piggybacked probabilistic gossiping system and evaluate it's performance against existing probabilistic gossiping.

2. RELATED WORK

Variations of gossiping have been used for many different applications in wireless sensor networks. Primarily used for network wide broadcast, gossiping greatly reduces the number of messages sent in the network and also improves latency and reliability. Gossiping was first used for ad hoc routing by configuring the probability with which nodes forward messages [6]. Later protocols also forward messages based on network topology [8] or according to a configured delay/power trade-off [17]. A similar probability based technique also incorporates the sensor sleep scheduling protocol to improve energy consumption [13]. By combining local retransmission and gossip mechanisms reliable broadcast has been achieved in rapidly changing network environments [15]. Gossip based failure detectors have also been developed for sensor networks [4]. A specific class of data dissemination for which gossiping has been successfully applied is code propagation. The Trickle protocol uses gossiping to implement self-regulating code propagation [10] and the Gappa system uses gossiping mechanisms to reprogram entire sensor networks from a remote control center [16]. Finally gossiping methods have been parameterized for topology discovery in home sensor networks [12]. As these projects evidence, gossiping is an efficient and increasingly popular method for data dissemination in sensor networks. Although gossip improves message overhead compared to flooding, when multiple sources use the gossip overlay there is still an overhead equal to the sum of the multiple streams.

Other approaches have used application meta-data to reduce the number of messages sent in the network. One such gossip-based adaptive protocol uses high-level data descriptors to name data and negotiates using the meta-data to eliminate redundant transmissions [11]. A second family of adaptive protocols, called Sensor Protocols for Information via Negotiation, also efficiently disseminates information among sensors in a wireless sensor network using meta-data [7]. These approaches effectively improve the energy efficiency of the sensor network but are dependent upon application specific meta-data. Our system operates without specific knowledge of the applications.

Another related approach to energy saving is data aggregation (data fusion). For example, geographic gossip utilizes a simple re-sampling method to compute accurate data averages using fewer radio transmissions [3] than other existing protocols. Logical constructions such as likelihood maps [5] and data cubes [9] have also been used to direct the aggregation of data in sensor networks. These approaches are also data specific, meaning two streams with different data types cannot be fused. On the other hand, our piggybacking approach combines messages irregardless of the content.

3. SCHEDULING OF MULTI-STREAM GOSSIP SYSTEMS

To gossip in sensor networks nodes forward messages with

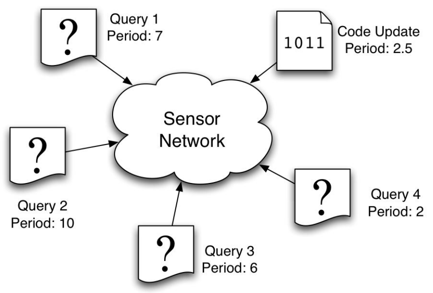


Figure 1: A sample message distribution system using a Gossip overlay with 5 different publishers.

```
GOSSIP
1: $probability \leftarrow p$
2: loop
3:   for each new message $m$ in stream do
4:     $r \leftarrow$ a random number between 0.0 and 1.0
5:     if $r < probability$ then
6:       broadcast message $m$
7:   sleep $gossip\_period$
```

Figure 2: Pseudo code for site percolation-style gossip of a single stream for sensor networks.

some probability p. Gossiping with a probability $p = 1$ is equivalent to flooding. Figure 2 shows the pseudo code for a canonical probabilistic gossiping protocol in sensor networks. Each node gossips in rounds determined by the *gossip period* forwarding latest messages with some probability p each round. The values of p and the gossip period are chosen to balance the latency of message propagation and message overhead in the network.

Emerging sensor networks must support multiple publishers injecting streams into the network. Each of these streams might have a different gossip period based on the requirements of the stream. As each of these streams spread through the network every sensor node must handle the different streams independently including timer generation, state maintenance and gossiping of the messages. Figure 1 illustrates a possible scenario in which four queries and one code update are being propagated through the sensor network. The number of messages generated in such a system is the sum from each stream causing a high energy cost from transmitting messages and responses. If the size of the message payload is small then the efficiency of the network is further reduced by overhead from packet headers.

Our solution to this overhead problem is to reduce the number of gossip streams in the system by collating related streams into single gossip streams, i.e. "piggybacking" related streams. To piggyback multiple streams any available messages from the separate constituent streams are joined into one single message. The collated message is then gossiped using the same gossiping mechanisms but with a separate gossip period. Figure 4 shows the pseudo code for piggybacked gossip. Nodes use a static schedule to determine from which streams to piggyback messages and the gossip period with which to gossip the piggybacked messages. The process of building this schedule is the process of *gossip scheduling*. The aim of gossip scheduling is to reduce the total number of messages sent in the system by

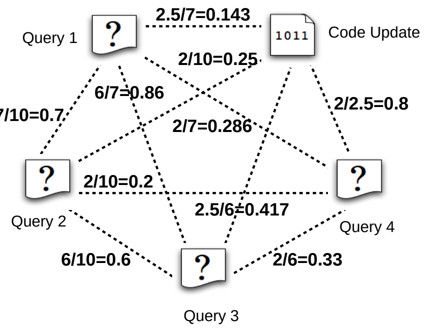


Figure 3: The semblance graph for the overlay in Figure 1. Edges labeled using equation 1.

```
PIGGYBACK-GOSSIP
1: $probability \leftarrow p$
2: loop
3:   for each constituent stream $s$ do
4:     for each new message $m$ in $s$ do
5:       $r \leftarrow$ a random number between 0.0 and 1.0
6:       if $r < probability$ then
7:         add $m$ to piggback message $p$
8:   broadcast piggyback message $p$
9:   sleep $gossip\_period$
```

Figure 4: Pseudo code for piggyback gossip based on the site percolation gossip in Figure 2.

combining gossip streams while at the same time preserving the properties of the independent streams. Our scheduling approach focuses on minimizing the delay incurred through piggybacking making it well suited for queries and other latency-constrained message streams. There are two primary constraints when gossip scheduling.

(1) There is a limit to the number of streams which can be collated into a single gossip stream based on the maximum message payload size in the sensor network system. Because each constituent stream contributes a maximum of one message to any piggyback message, the system message payload size in effect determines the maximum number of streams a piggyback can contain. We use k to express this limit. k is enforced system wide and is the same at every node in the sensor network.

(2) After piggybacking streams together the original reliability, scalability and latency of the individual streams should be maintained. Piggybacking effects the short term gossip period for some streams as messages are delayed to meet the schedule. The gossip schedule must ensure that the *average* period of each stream remains unchanged.

The solution to gossip scheduling depends on the period of the individual streams. For example, two streams with the same period will be able to piggyback every message. Two streams with very different periods will be able to piggyback very few messages. We define stream relatedness to determine which streams when piggybacked together would have a high rate of combining messages. For any two streams i and j with gossip periods t_i and t_j respectively the relatedness $R(i, j)$ can be expressed as:

$$R(i, j) = \frac{\min(t_i, t_j)}{\max(t_i, t_j)} \tag{1}$$

The relatedness between two streams gives the fraction of outgoing messages from the combined stream which would contain messages from both constituent streams. Finding the best gossip schedule is a matter of piggybacking the streams with the highest relatedness using equation 1. The gossip period for the piggyback stream is the lowest period from all of the constituent streams.

To enable efficient stream scheduling a logical graph is maintained which describes potential piggybacks between every stream. We call the graph the *semblance graph* because it describes the relatedness, or semblance, between two streams. Each gossip stream is represented in the graph by a single vertex. Edges in the graph describe the relatedness between the two streams represented by the edge endpoints. The graph is fully connected since every stream can be combined with every other. Selecting the best schedule is equivalent to partitioning the semblance graph into connected subgroups such that the sum of the edges contained in some subgroup is maximized while the number of vertices in any subgroup is less than k. To express the relatedness metric for weighting edges we use equation 1 because it is the best in expressing the relation between the two end-streams, irrespective and independent of the other streams' gossip periods. Regardless of the metric used the underlying clustering problem remains the same.

4. THE SEMBLANCE GRAPH PROBLEM

In this section we offer a formal definition of the semblance graph problem. We define the semblance graph problem as an operation on a fully connected graph. Informally, the Semblance Graph Problem can be stated as the problem of dividing a graph into disjoint subsets where the sum of weights of crossing edges between the subsets is minimum and each set has at most k vertices.

The formal definition of Semblance Graph Problem follows:

The Semblance Graph Problem
Input: Graph $G = (V, E)$, n distinguished nodes $s_1 \dots s_n$, where $1 \leq n \leq |V|$ and each nodes corresponds to each stream in the network, positive integer k, positive integer W.
Property: There is a partition $V = A_1 \cup \dots \cup A_m$ with $A_i \cap A_j = \emptyset$, $i = 1, \dots, m \wedge j = 1, \dots, m \wedge i \neq j$, and $|A_i| \leq k$, and $\Sigma\{\{u, v\} \in E : u \in A_i, v \in A_j\}$ is minimum where $i = 1, \dots, m \wedge j = 1, \dots, m \wedge i \neq j$.

A brute force solution to the problem has a prohibitively high cost to compare all possible subgroups. Because the semblance graph is fully connected either all of the partitions or all but one of the partitions will have size equal to k, depending on the number of vertices in the graph. For simplicity we assume that the number of edges in the graph, e, is perfectly divisible by k. Because the ordering within each subgroup does not matter the number of possible partitions can be expressed as $[C(e, k) \times C(e - k, k) \times C(e - 2k, k) \times .. \times C(k, k)]/(\frac{e}{k})!$ The product of combinations is divided by the factorial of (e/k) to factor out different orderings of the same group. The running time of the brute force approach then is $O(\frac{e!/(k!^{\frac{e}{k}})}{(\frac{e}{k})!})$.

5. ALGORITHMS

In this section we present two new efficient heuristic algorithms to solve the Semblance Graph problem. Our algorithms are greedy algorithms based on two existing Minimum Spanning Tree (MST) algorithms. Our problem is not the same as the MST problem but our heuristics follow the edge selection techniques used in those algorithms. We do note that the semblance graph problem is similar to graph bissection but rather tan creating equally size partition the semblance graph problem is focused on maximizing the sum of edge weights. Therefore we focus our attention on MST-like heuristics. Due to space constraints we do not provide a full analysis of the algorithms here; however, our algorithms are efficient allowing for the piggybacking schedule to be easily recomputed even on computationally constrained sensor nodes. See [14] for a full analysis.

```
$G$ : the input graph
$F_i$ : forest consisting of nodes ($N \geq i \geq 0$)
$F_{i,m}$ : the current set of nodes in the forest $F_i$
$F_{i,s}$ : the set of neighbors to the forest $F_i$

procedure GREEDY-PRIM($G$)
 1: for each forest $F_i$ do
 2:     $F_{i,m}$ ={edge with max weight}, $F_{i,s}$ ={}
 3:     To $F_{i,s}$, add neighbors of nodes in $F_{i,m}$ not in any forest
        $F_0, .., F_{i-1}$ and not present in $F_{i,m}$.
 4:     From $F_{i,s}$, select node $n$ with largest edge.
 5:     if number of nodes in the current forest $F_i$ does not exceed
        $k$ then
 6:         Add $n$ to set $F_{i,m}$, mark it as being seen.
 7:         Go back to step 2
 8:     else
 9:         start a new forest $F_{i+1}$
10:     Repeat until all nodes have been seen
```

Figure 5: Pseudo-code for the Greedy-Prim heuristic algorithm.

5.1 Greedy-Prim Algorithm

Prim's MST algorithm builds a spanning tree starting with a randomly selected vertex in the graph. Every iteration a vertex is added to the tree by selecting the best incoming edge (lowest weight) into the existing tree [2]. The algorithm stops when all vertices are part of the tree.

Our Greedy-Prim algorithm behaves in a similar way by greedily growing subgraphs until the number of vertices in the subgroup reaches the k-constraint. The algorithm starts by selecting the best edge (in our case highest weight) in the entire graph. Then starting with the two vertices from that edge continues to grow the "tree" by selecting the best incoming edges. The algorithm continues to grow the current tree until adding any edge would cause the k-constraint to be violated. Then the algorithm starts over with the best edge not contained in any existing subgraph. The pseudo-code for our Greedy-Prim algorithm is given in Figure 5.

5.2 Greedy-Kruskal Algorithm

Our second greedy algorithm is based on Kruskal's MST algorithm [2]. Kruskal's algorithm starts with each vertex of the graph in a tree by itself. At each iteration the best edge (lowest cost) from the entire graph is examined. If the edge connects two trees previously unconnected it is added to the MST. Otherwise the edge is discarded. The algorithm continues until every vertex is included in the MST.

Our Greedy-Kruskal algorithm also examines edges on a global basis. The algorithm starts with the best edge (highest weight). For each iteration the next highest weight edge is examined. The algorithm checks to make sure the edge

G : the input graph
F_j : the forest consisting of nodes

```
procedure GREEDY-KRUSKAL($G$)
 1: Sort edges of $G$ based on edge weight.
 2: Take the unselected edge $E_i$ with maximum weight.
 3: if adding $E_i$ causes size of any forest to exceed $k$ then
 4:     throw it out.
 5: else
 6:     if $E_i$'s endpoints are members of two separate forests then
 7:         merge forests unless merged forest exceeds $k$
 8:     if $E_i$'s endpoints are not members of any forest then
 9:         create a new forest with $E_i$.
10: Repeat for every edge
```

Figure 6: Pseudo-code for the Greedy-Kruskal heuristic algorithm.

does not combine two subgroups that together would violate the k-constraint. If the combined subgroup obeys the k-constraint then the edge is added to the partition. Otherwise the edge is discarded. The algorithm continues until all edges have been examined. Figure 6 shows the pseudo-code of Greedy-Kruskal.

5.3 Comparison of Algorithms

To compare the heuristics we generated over 5000 semblance graphs with varying configurations. We compared the schedules generated by the new algorithms with the optimal schedule from a brute-force search. A schedule was scored by summing the weights of the edges contained in the schedule subgroups. Overall the Greedy-Kruskal algorithm achieves better performance over a larger set of input graphs than the Greedy-Prim. The score of the Greedy-Kruskal schedules were on average within 3.5% of the optimal score. Varying the number of streams in the system had the biggest effect on performance of the algorithms. Figure 7 shows the variation in results as the number of streams increases. As the stream periods varied there was also slight degradation in accuracy. Changing the k value or the magnitude of the stream periods had little effect.

6. SIMULATION

To evaluate our piggybacking approach we implement the canonical and our new gossip layer in the TinyOS simulator, TOSSIM. We compare the canonical gossip flooding protocol of Figure 2 (labeled “Flood” in our experiments) with our piggybacking scheme (“PgFlood”) from Figure 4. Each individual sensor in the network locally uses the Greedy-Kruskal algorithm to create a schedule. The schedule assigns each stream to a certain subgroup which is maintained by the piggybacking layer as a single gossip stream. The message forwarding probability was 70% based on the optimal range of 60-80% given in previous work [6]. Each trial had a network of 225 nodes and 24 publishers selected randomly from the network, unless otherwise noted. Nodes are arranged on a 3 meter square shaped grid. The default k value used in our experiments was 7 because the default TinyOS maximum payload size is 28 bytes and one single message in our gossiping system is 4 bytes.

We first estimate the power savings using the specs from the Mote 2 data sheet [1]. We are using a basic mote configuration and basic simulator without advanced MAC protocols and radio models in order to study our scheduling in isolation.

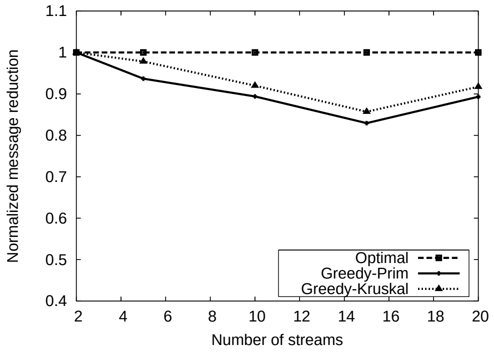


Figure 7: Performance of algorithms while changing the number of streams

According to the data sheet transmission of data draws 2.7 times more current than receiving. In order to estimate the power consumption, we used the following formula:

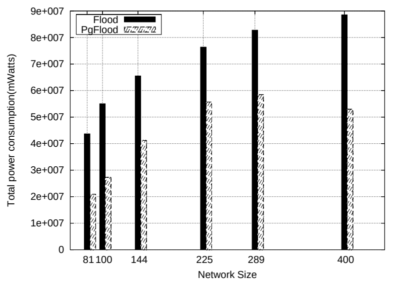
$$P = (bytes\_sent * I_{transmission\_coeff} + bytes\_rcvd * I_{receive\_coeff}) * V$$

where $I_{transmission\_coeff}$ is 27 mA, $I_{receive\_coeff}$ is 10 mA, and $V = 3.0$ volts. Using this equation, Figure 8(a) compares the power consumption of the two different schemes. As can be observed from the figure, the PgFlood scheme can achieve significant amount of power saving as compared to the Flood scheme. Surprisingly, at network size of 400 nodes, the power consumption in PgFlood drops. We believe that the reason for this is that more collisions are occurring in the system as the network size grows and messages are not being received by all of the neighboring nodes. Before the sender can retransmit it detects the message delivery by some other node avoiding the energy cost to receive the original message at each node. The reason this becomes visible only in the PgFlood and not in the Flood scheme is that the piggybacked messages are larger than normal messages and the cost of one packet is much higher.

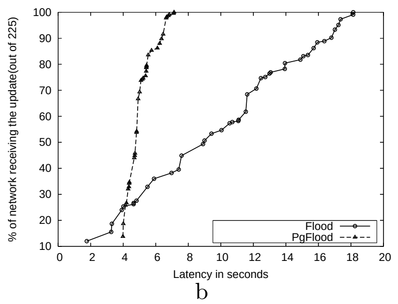
Another interesting and unexpected result is shown in the network latency, Figure 8(b). The experiment shows the cumulative distribution function of a specific update's dissemination latency to every node in the network. The update was generated at 10.7 seconds into the simulation. As can be seen, the piggybacking scheme delivers the update to the whole network with a significantly *lower* latency than normal gossip, despite delays from the piggybacking schedule. The likely explanation is that the higher number of messages sent under the default scheme caused a higher number of collisions and therefore higher delays from timeout and retransmission. Under piggybacked flooding there is less contention and fewer dropped messages.

To test the effect of piggybacking on reliability, random nodes were removed from the simulation to mimic fail-stop behavior. Figure 8(c) shows that as node failure increases the number of messages delivered follows the same decline for both piggybacking and regular gossip. Piggyback reliability is always slightly under the normal gossip because of the initial delay to deliver messages (as seen in Figure 8(b)).

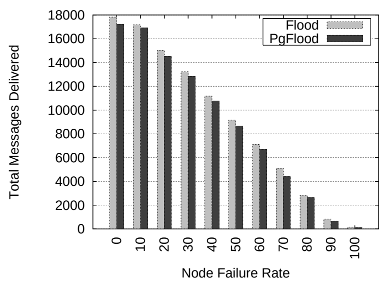
7. CONCLUSION AND FUTURE WORK



a



b



c

Figure 8: (a) Power consumption during a 30 second period. (b) CDF of an update's arrival time at each sensor in the network. (c) Total messages delivered with increasing node failure.

As gossip based protocols become more and more popular for delivering messages in sensor networks with multiple publishers, the overhead will grow with the number of publishers in the system. Using our "semblance graph" based gossip scheduling approach for piggybacking messages much of the overhead from supporting multiple publishers in gossip systems can be eliminated while the original scalability, reliability and latency of canonical gossip is maintained. Our algorithms can be incorporated into existing sensor network gossiping layers with little change to the original protocol. One scheduling approach which we do not address here is how to schedule aperiodic streams, either dynamically or statically. Periodicity makes our scheduling problem simpler. A second approach would be to temporally aggregate messages from the same stream, queuing them for delivery at once. The scheduling problem would be to enqueue as many messages as possible while still delivering them within a system wide maximum time delay. Techniques such as ours will be important to maintain consistent energy efficiency in the future as sensor networks continue to grow and demand multiple data streams.

8. REFERENCES

[1] Mica2 datasheet. http://www.xbow.com/.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (Second Edition).* MIT Press, 2001.

[3] A. Dimakis, A. Sarwate, and M. Wainwright. Geographic gossip: efficient aggregation for sensor networks. In *The Fifth International Conference on Information Processing in Sensor Networks*, 2006.

[4] M. Elhadef and A. Boukerche. A failure detection service for large-scale dependable wireless ad-hoc and sensor networks. In *The Second International Conference on Availability, Reliability and Security*, 2007.

[5] E. Ertin. Distributed multimodal data fusion for large scale wireless sensor networks. *Intelligent Computing: Theory and Applications IV*, 6229(1), 2006.

[6] Z. J. Haas, J. Y. Halpern, and L. Li. Gossip-based ad hoc routing. *IEEE/ACM Transaction on Networking*, 14(3):479–491, 2006.

[7] J. Kulik, W. Rabiner, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *1999 International Conference on Mobile Computing and Networking*, 1999.

[8] P. Kyasanur, R. R. Choudhury, and I. Gupta. Smart gossip: an adaptive gossip-based broadcasting service for sensor networks. In *2006 IEEE International Conference on Mobile Adhoc and Sensor Systems*, 2006.

[9] L. H. Lee and M. H. Wong. Aggregate sum retrieval in sensor network by distributed prefix sum data cube. In *19th International Conference on Advanced Information Networking and Applications*, 2005.

[10] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *First Symposium on Network Systems Design and Implementation (NSDI)*, 2004.

[11] M. Medidi, J. Ding, and S. Medidi. Data dissemination using gossiping in wireless sensor networks. *Digital Wireless Communications VII and Space Communication Technologies*, 5819:316–327, 2005.

[12] S. Mehta, W.-S. Yoon, S.-W. Min, and S. Yu. Topology generation algorithms for home sensor networks. *Second IEEE Workshop on Software Technologies for the Future Embedded and Ubiquitous Systems*, 2004.

[13] M. J. Miller, C. Sengul, and I. Gupta. Exploring the energy-latency trade-off for broadcasts in energy-saving sensor networks. In *25th IEEE International Conference on Distributed Computing Systems*, 2005.

[14] E. Ucan. Scheduling of multi-stream gossip systems. Master's thesis, University of Illinois at Urbana-Champaign, 2007.

[15] G. Wang, D. Lu, W. Jia, and J. Cao. Reliable gossip-based broadcast protocol in mobile ad hoc networks. In *First International Conference on Mobile Ad-Hoc and Sensor Networks*, 2005.

[16] L. Wang and S. Kulkarni. gossip based multi-channel reprogramming for sensor networks. In *Second IEEE International Conference on Distributed Computing in Sensor Systems*, 2006.

[17] Y. Zhang and L. Cheng. Flossiping: A new routing protocol for wireless sensor networks. In *IEEE International Conference on Networking, Sensing and Control*, 2004.