SCHEDULING OF MULTI-STREAM GOSSIP SYSTEMS

BY

ERCAN UCAN

B.S., Bilkent University, 2005

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2007

Urbana, Illinois

Abstract

Many distributed applications are employing gossip-based message dissemination, where the burden of message distribution is placed on recipient nodes. We are concerned with emerging systems (e.g., peer-to-peer designs for RSS dissemination, queries and code propagation in stationary wireless sensor network systems) that are multiple-source, multiple-recipient systems, where each recipient node is interested in all the streams. Default gossip-based approaches tend to treat each stream independently of the others, overloading each node with message overhead summed from all streams. In this thesis, 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" among gossip streams, based on streams' frequencies. Our solution consists of two new heuristic algorithms to solve the semblance graph problem. Both heuristics are inspired by Minimum Spanning Tree algorithms. Our semblance graph evaluation shows that the performance of these two heuristics is within 3.5% of the optimal solution. Our distributed systems simulations show that these scheduling strategies reduce the message overhead, bandwith usage, overall latency (in wireless sensor networks) and power consumption (in wireless sensor networks), while still maintaining the original scalability and reliability of the underlying gossip schemes. Our experiments show message overhead reduction up to 82%, total bandwith usage reduction up to 47%, and reduction in power consumption (in wireless sensor networks) up to 50% compared to the default gossiping schemes.

To my father Ibrahim, mother Turkan and brother Musa Can.

Acknowledgments

This project would not have been possible without the support of many people. Many thanks to my advisor, Indranil Gupta, who helped me with his valuable research ideas and guidance throughout the preparation of this thesis and the projects that we have worked on together. Also, thanks to my previous project partner, Nathanael Thompson, for the guidance and help he provided.

With no doubt, I am thankful for having such a great family, my father, my mother and brother, for them understanding and supporting me at all the good times and bad times that I have had throughout my life and career.

I would also like to thank all my friends here, especially Onur and Ozgul Pekcan, for making the life at Urbana-Champaign as enjoyable as possible for me.

Finally, I would like to thank Mr. Yavuz Cetin, rest in peace and Mr. Kazim Koyuncu, rest in peace, for the inspiration they have been to me and for their beautiful music that kept me sane and inspired at all times.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

DHT	Distributed Hash Table
GSP	Gossip-based Sleep Protocol
GTP	Gossiping Time Protocol
MANET	Mobile Ad-hoc Networks
MST	Minimum Spanning Tree
P2P	Peer-to-peer
RSS	Really Simple Syndication
SPIN	Sensor Protocols for Information via Negotiation
TDMA	Time Division Multiple Access
TGSP	Traffic-aware Gossip-based Sleep Protocol
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol

Chapter 1

INTRODUCTION

Gossip (epidemic-based) algorithms are emerging as attractive choices for peer to peer(P2P) publish-subscribe and code propagation and query dissemination in wireless sensor network systems(throughout this thesis, when we use the term sensor network or wireless sensor network, we mean stationary wireless sensor networks), primarily because they democratize the overhead of content distribution. For example, in an RSS delivery system, the updates from each publisher constitute a single stream. In the canonical gossip protocol, for a single content stream, each node periodically, once every *protocol period* (or *gossip period*) seconds, gossips (i.e., sends copies of) the latest message(s) it has received for that stream. The gossips are sent to a small set of other nodes, selected either at random or in a topologically-aware manner. In a wireless sensor network, code updates and queries are generated at a certain rate by certain sensor nodes. These nodes can be visualized as publishers that generate updates periodically where the period information would be derived from the rate of code updates or query generations. The dissemination mechanism could either be based on site percolation[22] or bond percolation[38] in these sensor network systems. Analysis of such gossip protocols has revealed that a scalable per-node overhead that is (poly-)logarithmic(for P2P systems) in system size disseminates the message(s) quickly and reliably to a large fraction of the entire system (close to 100%). In sensor networks, per-node overhead is a broadcast to the surrounding sensor nodes.

While good in distributing single messages, or single streams to the whole network in a timely and reliable manner, still there is a high message overhead due to multiple sources that publish information in the aforementioned situations. In reality, many P2P publish-subscribe

systems and sensor network code update protocols often disseminate *multiple streams* in the underlying gossip communications layer(e.g. RSS subscribers fetch data from multiple streams, queries can come from different sensor nodes). Yet, most existing gossip mechanisms handle each stream independently, with each node typically generating separate gossip messages for each stream in the system. Although each publisher is using the same network or overlay [1], multiple messages are being sent to the same destinations from different sources. All of this causes the P2P canonical gossip, sensor network flooding, site and bond percolation approaches mentioned above to create a per-node overhead that is the sum from all the streams.

This thesis adopts a general and effective approach to address this overhead problem– that of piggybacking gossip messages from one stream atop gossip 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 reliability, scalability, and latency as canonical gossiping or 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 contain more than a given number of constituent streams inside, with one message per stream. This is specified as a parameter k in our problem formulation. In sensor networks, this constraint comes from the maximum message payload size limitation. In P2P systems, this constraint can come from a maximum message payload size that is required by the underlying network protocol or the users of the system. Having a maximum message size would be preferable in the case that message fragmentation is not desired. The second constraint comes from the fact that we would like to, in spite of piggybacking, maintain the scalability, reliability,

[1] An overlay is a computer network that is built on top of another existing network. Nodes in the overlay can be thought of as being linked to each other by virtual or logical links, each of which corresponds to a path, perhaps through many physical links, in the underlying network.[3]

and latency of the underlying gossip scheme, *for each individual stream*. These properties depend on the the stream's *gossip period* (protocol period), which we assume is fixed and pre-specified to us a priori (for each stream). In P2P systems, a stream's gossip period is a length of time (fixed system-wide) such that a given node should send out, per gossip period time units, an expectation of one instance of the latest update from that stream. In sensor networks, a stream's period is the rate at which code updates or queries are sent out from the source node. Although we assume the gossip period length(P2P), code update or query rates(sensor networks) are given a priori for each stream, in practice, these values may be derived from characteristics of the streams, e.g., they could be set inversely proportional to the average rates of new updates produced by those streams' publisher nodes.

Thus, given a set of streams, each specified with a gossip period length (thus gossip frequency) in P2P settings and a gossip rate in sensor network settings, 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.

In approaching this problem, we first notice that clearly, 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; the latter will have fewer collated gossip messages containing updates from the lower gossip period stream's.

We formalize these two constraints via a new concept called the "semblance graph". The semblance graph is a complete logical graph, with each node representing a distinct stream, and an edge between two nodes carrying a weight indicating the end-streams' "relatedness", as determined by their specified gossip periods. The relatedness metric lies in the real interval $[0, 1]$. The scheduling problem then 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 to solve this problem. Our heuristic algorithms have low running time compared to a brute force examination of the graph and achieve average 3.5% degradation in accuracy. We then integrate our semblance graph based scheduling into two different gossip overlay systems(one being a standard network and the other being a sensor network) 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 thesis 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 P2P network implementation and simulation of a standard and piggybacked gossiping system and its evaluation; (iv) present a sensor network implementation and simulation employing a standard and piggybacked flooding and also one that employs probabilistic gossiping and piggybacked probabilistic gossiping system along with their evaluations.

The rest of this thesis is organized as follows. Chapter 3 discusses related work. In chapter 2, we give an overview of gossip protocols and illustrate how the piggybacking problem can be converted into a semblance graph. The formalization of the semblance graph problem is given here. Then in chapter 4, we describe our heuristic algorithms and discuss their simulated comparison against each other. In chapter 5, we provide pseudocodes for our simulations and explain the design details of the gossiping mechanism that we use. Our PeerSim system simulation evaluation goes in Chapter 6. Chapter 7 discusses our TinyOS sensor network simulation. Finally we conclude in Chapter 8.

Chapter 2

PROBLEM

In this chapter, we identify and present the problem that we are solving. We describe the multiple-stream gossip systems and present the "semblance graph" in section 2.1. Then, we give a formal definition of the problem in section 2.2.

2.1 Scheduling of Multi-stream Gossip Systems

We begin this section with an overview of existing gossiping approaches such as canonical gossip in P2P systems, flooding and probabilistic gossiping in sensor network systems. In canonical gossiping in P2P systems, when a new message arrives, the receiving node delivers the message to the application layer and then schedules the message for forwarding. The node will gossip about the message for a certain number of neighbors called the $fanout$. The $fanout$ is $\log(N)$ where N is the approximate network size. The length of a round is determined by the period of the gossip which is a system configurable parameter. In each round the member selects another member from the overlay at random to forward the message. Messages are usually sent with an unreliable transport layer like UDP as the randomized message forwarding compensates for failed members and dropped messages. The probability of a given member receiving the message is $1 - \frac{1}{N}\dot{(}1 + o(1))$. The latency for a message to spread throughout the network is $O(\log(N))$. Figure 2.1 gives the pseudo code for the canonical gossip protocol.

Figure 2.2 shows the pseudo code for the probabilistic gossiping protocol that we used in sensor networks. In this version, there is a certain probability value, and messages are

Figure 2.1: Pseudo code for gossiping a message in the P2P canonical gossip protocol.

```
GOSSIP
1: View V of network
2: for each new message m do
3:    while gossips < (fanout = O(log N)) do
4:       q ⇐ some non-contacted neighbor from V
5:       send m to q and sleep gossip_period
6:    end while
7: end for
```

broadcast every so often based on the probability. At each round, for every new message, a random real number between 0 and 1 is picked and if it is greater than the selected probability value, then that message is broadcast to the sensor network. Otherwise, this message is not gossiped during that round. We use the probability to be equal to 70% in our case. [22] shows that using gossiping probability between 60-80% suffices to ensure that almost every node gets the message in almost every execution.

Figure 2.2: Pseudo code for site percolation-style gossip for sensor networks.

```
GOSSIP
1: probability = 0.7
2: for each new message m do
3:    select a random number between 0.0 and 1.0
4:    if random number < probability then
5:       broadcast message m
6:    end if
7:    sleep gossip_period
8: end for
```

A gossip stream is a sequence of messages from a single source. For example, in a gossip based publish-subscribe system, each update from one publisher constitutes the gossip stream of that publisher. Publishers generate messages at a certain *injection rate* which may or may not be periodic. New updates supersede previous updates, that is updates are not incremental. The gossip sublayer for a given stream is configured to have a period just right to ensure that each message is forwarded enough times before the next message is injected into the sublayer. Messages may be received asynchronously, but each outgoing gossip obeys

Figure 2.3: A sample message distribution system using a Gossip overlay with 5 different publishers.

the configured stream gossip period and thus is sent synchronously.

Emerging publish subscribe systems have multiple publishers injecting messages into the gossip overlay resulting in multiple gossip streams to support all of the publishers. Each of the streams is handled independently including timer generation, state maintenance and gossiping itself. The number of messages generated each cycle in such a system is the sum from each stream.

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. Nodes use a static schedule to determine from which streams to piggyback messages and periodically 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 combining gossip streams while at the same time preserving the reliability, latency and scalability of the independent 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. The limit may come from underlying network constraints, for example maximum UDP packet size in P2P networks, maximum message payload size in sensor network systems or bandwidth limit in both systems. The limit may also be enforced by the gossip system itself, for example in order to reduce the processing overhead from unpacking multiple mes-

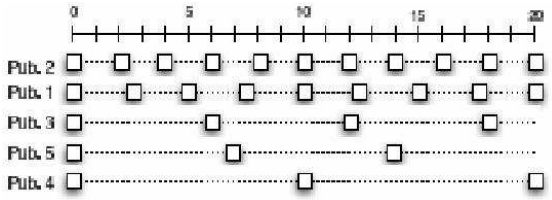

Figure 2.4: Timeline of message delivery for the overlay in Figure 2.3

sages at once. Because each constituent stream contributes a maximum of one message to any piggyback message the maximum number of streams in a piggyback is determined by the system message size limit. We use k to express this limit. k is enforced system wide and is the same at every node in the gossip overlay.

(2) After piggybacking streams together we want to ensure that the original reliability, scalability and latency of the individual streams is maintained. As described earlier the gossip period is configured to control these properties for each stream. 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. As the difference in periods increases the piggybacking decreases. 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:

$$R(i,j) = \frac{\min(t_i, t_j)}{\max(t_i, t_j)} \tag{2.1}$$

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

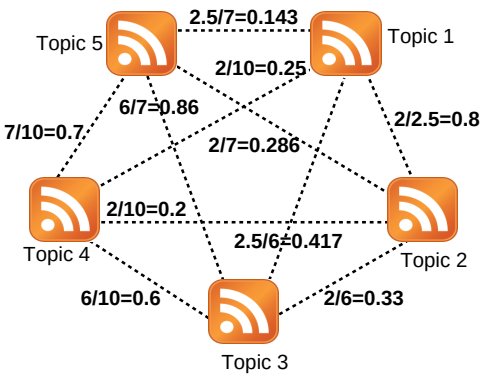

Figure 2.5: The semblance graph for the overlay in Figure 2.3. Edges labeled using equation 2.1.

relatedness using equation 2.1.

To enable efficient stream scheduling a logical graph is maintained which describes potential piggybacks for the entire system. 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 represent the relatedness between the two streams represented by the edge endpoints. The graph is fully connected as every member of the overlay subscribes to each stream and thus can piggyback messages from any stream with those of any other stream. Selecting the best schedule is equivalent to partitioning the semblance graph into connected subgroups by selecting the edges with maximum weight 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. There are many different ways of expressing the relatedness metric for weighting edges. We use equation 2.1. Another option would be to weight the edges with the number of messages sent in some interval by each stream. For example, define a cycle as the lowest common multiple of all stream periods. Then weight the edges with the number of messages sent by the less productive stream in one cycle. This gives one indication of the number of messages piggybacked. We chose to use 2.1 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.

Each node in the system uses the schedule to configure its own streams. The piggybacking layer maintains a single gossip stream for each subgroup in the semblance graph solution. Each combined stream has a gossip period that is the lowest period from any stream in that component. The periods of the original streams are used to determine when a message from that stream is available to be gossiped. When the combined stream is ready to gossip each constituent stream is queried for available messages which are combined into a single message which is forwarded normally according to the gossip protocol.

The parameters for the semblance graph can be gathered in a number of ways. Firstly the original gossip layer can be queried to determine the configuration of each stream. If no such interface exists then the piggybacking layer can easily monitor each stream to determine its period. The k constraint can either be set by a system administrator or determined through active probing. If new streams were introduced into the system or existing streams removed the semblance graph is automatically updated to reflect the changes requiring a new semblance graph solution to be calculated.

Example of a Multi-Stream Gossip System Figure 2.3 shows a sample overlay with multiple streams. There are five publishers with configured gossip periods of 2, 2.5, 6, 7 and 10 seconds. Figure 2.4 shows the timeline of message delivery in the system. Note that very few messages are sent in the exact same time slot. It is clear to see that piggybacking the two lowest period streams would result in higher messages savings then piggybacking them with higher period streams. Using the gossip rates from Figure 2.3 the semblance graph shown in Figure 2.5 is constructed according to equation 2.1. The most related streams are 3 and 5, 1 and 2, and 4 and 5. Therefore we should piggyback those combinations. With a k value of 3 we would collate the streams 1 and 2 into a single gossip stream and streams 3, 4 and 5 into another gossip stream. If we define a gossip cycle as the minimum time in which every stream has gossiped at least once, the total message reduction per cycle is 6.67 out of 14 total messages - nearly half. If the k value was only 2 then we would collate streams 1

and 2 again, but this time only 3 and 5 for the second piggyback. Stream 4 would not be piggybacked. In that case the savings would be 5.43 messages per cycle.

2.2 The Semblance Graph Problem

In this section we offer a formal definition of the semblance graph problem and heuristic algorithms to solve it in the next section. Our algorithms are efficient allowing for the piggybacking schedule to be easily recomputed when the configuration of streams in the system changes. 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 $s_1 \ldots s_n$, 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 $V = A_1 \cup \ldots \cup A_m$ with $A_i \cap A_j = \emptyset$, $i = 1, \ldots, m \wedge j = 1, \ldots, m \wedge i \neq j$, and $|A_i| \leq k$, and $\Sigma\{\{u, v\} \in E : u \in A_i, v \in A_j\}$ is minimum where $i = 1, \ldots, m \wedge j = 1, \ldots, m \wedge i \neq j$.

A brute force solution to the problem has a prohibitively high cost. Such a solution should examine 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 $O(\frac{e!/(k!\frac{e}{k})}{(\frac{e}{k})!})$.

Chapter 3

RELATED WORK

In this chapter, we describe the related work for our research. We discuss the related work for P2P gossip systems in section 3.1 and for wireless sensor networks gossip systems in section 3.2.

3.1 P2P Gossip Systems

Gossip protocols have their origins in the study of infectious diseases. [4]. They have been widely used in distributed systems including, amongst others, multicast [6, 17], database maintenance and replication [13, 28], group membership and state maintenance in DHTs [20] and in publish subscribe systems [11, 23, 43, 48, 51].

In Sub2Sub[48], the problem of constructing scalable content-based publish/subscribe systems is addressed. This work is a collaborative self-organizing publish/subscribe system deploying an unstructured overlay network. The key approach in this work is that subscribers to the same events are automatically clustered. Our work differs from [48] in the sense that we are aiming to improve the dissemination of updates, namely, reducing the number of messages and keeping the delivery time and resiliency the same, in already existing overlays.

One example of a multiple-source, multiple-recipient system is Corona [23] addresses the problem of providing communication protocol support for large-scale group collaboration systems for use in environments such as the Internet. It provides two classes of services: the publish/subscribe service and the peer group service. In order to achieve the former, it proposes a two-level architecture in which a publisher multicasts data to a set of intermediate

nodes, referred to as distributors and then distributors route the data to other distributors, which in turn send the data to the local subscribers.

In [30] gossip broadcasts are made more efficient by using message age to schedule messages for delivery. By removing older messages sooner the probability increases of delivering "useful" previously unseen messages to neighboring hosts. In [44] the approach is used to create adaptive gossip which dynamically adjusts the gossip buffer at each host to prevent buffer overflow and thus message loss. We also schedule the gossip streams focusing instead on message content rather than message age. Our piggyback technique can be used in conjunction with adaptive gossip to further reduce overhead.

In [21] the authors reduce network overhead by constraining the selection of gossip targets based on the underlying network hierarchy. Router overhead is decreased while maintaining the reliability of canonical flat gossip. In [42] the authors develop a network friendly epidemic algorithm that uses congestion control and selective message dropping to reduce the impact of epidemic algorithms on network performance.

A graph based approach to reducing network cost is developed in [35]. Each host calculates the weights of its neighbors based on the *link cut set* of the local topology graph. Neighbors with low connectivity are given higher weights. The protocol dynamically switches between flooding and gossiping depending on the weight of the neighbor. The approach maintains reliability while lowering overall message count. Our approach creates a topology of *streams* called the semblance graph which is used to find the best schedule of piggybacking.

In [5], the authors discuss the surprising power of epidemic communication. It focuses on the most appropriate form of middleware to offer in support of distributed system management, control, information sharing and multicast communication.

Article [46] describes some of the common problems that arise in scalable group communication systems and how epidemic techniques have been used to successfully address these problems. It emphasizes that a new generation of gossip-based or epidemic communication primitives can overcome a number of these scalability problems, offering robustness and

reliability even in the most demanding settings.

In [12], the authors presents Autonomous gossiping, which is a new genre epidemic algorithm for selective dissemination of information in contrast to previous usage of epidemic algorithms which flood the whole network. A/G is a paradigm which suits well in a mobile ad-hoc networking environment because it does not require any infrastructure or middleware like multicast tree and (un)subscription maintenance for publish/subscribe, but uses ecological and economic principles in a self-organizing manner in order to achieve any arbitrary selectivity (flexible casting).

Article [27] is about gossip-based clock synchronization for large decentralized systems. They present the gossiping time protocol (GTP), a completely self-managing epidemic time synchronization algorithm for peer-to-peer networks. In GTP, each node synchronizes its time by gossiping with other nodes. The decisions regarding sample evaluation and gossiping frequency are purely local, yet they result in consistent behavior of the whole system.

In [2], the authors give a new scalable gossip-based algorithm for local view maintenance, together with a proof that the expected time until a network partition is at least exponential in the square of the view size. The work also develops probabilistic bounds on the in-degree (hence the load) of individual nodes, and argues that protocols lacking their reinforcement component eventually converge to star-like networks, whose connectivity depends on a small set of overloaded nodes.

In [18], the authors study epidemic schemes in the context of collaborative data delivery. In this context, they explore the inter-operation between the gossip of multiple, simultaneous message-chunks. They provide an efficient solution that possesses the inherent robustness and scalability of gossip. Their approach maintains the simplicity of gossip, and has low message, connections and computation overhead.

Article [19] presents GosSkip, a self organizing and fully distributed overlay that provides a scalable support to data storage and retrieval in dynamic environments. The structure of GosSkip, while initially possibly chaotic, eventually matches a perfect set of Skip-list-like

structures, where no hash is used on data attributes, thus preserving semantic locality and permitting range queries.

The scalability and resilience of epidemic multicast, also called probabilistic or gossip-based multicast, rests on its symmetry. Each participant node contributes the same share of bandwidth thus spreading the load and allowing for redundancy. On the other hand, the symmetry of gossiping means that it does not avoid nodes or links with less capacity. One cannot naively avoid such symmetry without also endangering scalability and resilience. In [41], the authors point out how to break out of this dilemma, by lazily deferring message transmission according to a configurable policy.

In [40], the authors present a middleware for ad hoc networking, which uses epidemic-style information dissemination techniques to tune the reliability of the communication in mobile ad hoc networks. The approach is based on recent results of complex networks theory.

In [29] the question of 'How does the underlying low-level gossip mechanism (the means by which communication partners are chosen) affect one's ability to design efficient high-level gossip-based protocols?' is addressed. They establish one of the first concrete results addressing this question, by showing a fundamental limitation on the power of the commonly used uniform gossip mechanism for solving nearest-resource location problems.

Article [39] describes an algorithm to replicate and retrieve data items among nodes in a mobile ad-hoc network(MANET) that is based on an epidemic dissemination scheme. Their approach is tailored to the concrete network environment of MANETs and, while embedding several ideas from existing gossip protocols, takes into account the topology, scarcity of resources, and limited availability of both the devices and the network links in this sort of networks.

In [45], the authors outline a method for distributed Monte Carlo optimization of computational problems in networks of agents, such as P2P networks of computers. The optimization and messaging procedures are inspired by gossip protocols and epidemic data dissemination, and are decentralized.

In [47], authors present the newscast model and report on experiments using a Java implementation.The newscast model is a general approach for communication in large agent-based distributed systems. The two basic services - membership management and information dissemination - are implemented by the same epidemic-style protocol.

3.2 Sensor Network Gossip Systems

Article [22] proposes a gossip-based approach, where each node forwards a message with some probability to reduce the overhead of the routing protocols. We employ this kind of an approach in our sensor network experiments as well as flooding in order to implement our gossiping mechanism.

Gossip based approaches are also widely used in sensor network systems. [34] is about a Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. [32] proposes 'smart gossip', a probabilistic protocol that offers a broadcast service with low overheads. Smart gossip automatically and dynamically adapts transmission probabilities based on the underlying network topology. It is capable of coping with wireless losses and unpredictable node failures that affect network connectivity over time. Also, it is completely decentralized.

In [24], authors present a family of adaptive protocols, called SPIN, Sensor Protocols for Information via Negotiation, that efficiently disseminates information among sensors in a wireless sensor network that has energy constraints.

Article [9] proposes an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. Span, a power saving technique for multi-hop ad hoc wireless networks that reduces energy consumption without significantly diminishing the capacity or connectivity of the network.

In [38] explores the energy-latency-reliability trade-off for broadcast in multi-hop WSNs, by presenting a new protocol called PBBF (probability-based broadcast forwarding). PBBF

works at the MAC layer and can be integrated into any sleep scheduling protocol. For a given application-defined level of reliability for broadcasts, the energy required and latency obtained are found to be inversely related to each other.

In [26], the authors present an energy conservation scheme for wireless ad hoc and sensor networks using gossiping to place nodes in an energy saving sleep state. The technique is termed the gossip-based sleep protocol (GSP). With GSP, each node randomly goes to sleep for some time with gossip sleep probability p.

The study [14] is called Geographic Gossip, which is an efficient aggregation for sensor networks. propose and analyze an alternative gossiping scheme that exploits geographic information. By utilizing a simple re-sampling method, they can demonstrate substantial gains over previously proposed gossip protocols. In particular, for random geometric graphs, their algorithm computes the true average to a better accuracy using less radio transmissions, which reduces the energy consumption.

Article [50] presents Gappa, a gossip based multi-channel reprogramming for sensor networks. To reprogram a sensor network with the help of an unmanned aerial vehicle(UAV), one can either communicate the entire new program to one (or a few) sensor in the field, or let the UAV communicate parts of the code to a subset of sensor nodes on multiple channels at once. In the latter approach, the nodes need to communicate with each other to receive the remaining parts of the program. In this paper, they propose a protocol for such gossip between nodes. To better utilize the multi-channel resources and reduce contention, their protocol provides a multi-channel sender selection algorithm.

Disseminating Data among sensors is a fundamental operation in energy-constrained wireless sensor networks. In [36], the authors present a gossip-based adaptive protocol for data dissemination to improve energy efficiency of this operation. To overcome the data implosion problems associated with dissemination operation, the protocol uses meta-data to name the data using high-level data descriptors and negotiation to eliminate redundant transmissions of duplicate data in the network.

In [52], the researchers evaluate the lifetime of various network sizes employing GSP. The largest percentage improvement in network lifetime occurs for smaller networks. On the other hand, the larger networks provide the higher average remaining energy.

Article [25] is about traffic-aware gossip-based energy conservation for wireless ad hoc and sensor network routing. In this paper, the authors extend the Gossip-based Sleep Protocol(GSP) to consider the ongoing traffic before changing the radio mode of a node. They call it Traffic-aware GSP (T-GSP) and show its advantages through simulations.

Article [49] proposes a novel reliable broadcast protocol that uses clustering technique and gossip methodology. They combine local retransmission and gossip mechanisms to provide reliability in mobile ad hoc networks. The proposed protocol can dynamically change system parameters for reliable broadcast communication in order to improve the adaptability in the rapidly changing network environment.

In work [7], the authors analyze the averaging problem under the gossip constraint for an arbitrary network graph, and find that the averaging time of a gossip algorithm depends on the second largest eigenvalue of a doubly stochastic matrix characterizing the algorithm.

Article [15] proposes a failure detection service for large-scale dependable wireless ad-hoc and sensor networks. The authors present a new implementation of a failure detection service for wireless ad-hoc and sensor systems that is based on an adaptation of a gossip-style failure detection protocol and the heartbeat failure detector.

In [31] the authors present a simple time division multiple access (TDMA) algorithms for assigning time slots to sensors and show that it provides a significant reduction in the number of collisions incurred during communication. They present TDMA algorithms customized for different communication patterns, namely, broadcast, convergecast and local gossip, that occur commonly in sensor networks.

Article [16] studies distributed fusion of multimodal sensor data for extracting target information from a large scale sensor network. Their approach to solving the fusion problem with large number of multimodal sensors is construction of likelihood maps.

Article [8] is about measuring energy consumption in wireless sensor networks using Gossip-based Sleep Protocol (GSP). Simulations show that GSP can conserve energy. They expand on this effort by building a prototype system and measuring energy consumption rates.

Article [53] presents Flossiping, a new routing protocol for wireless sensor networks. It can be seen as an enhancement to existing flooding and gossiping approach by using a single branch gossiping with low-probability random selective relaying in order to achieve a better overall performance. By letting each sensor node decide its own activity in the routing procedure, they implement a simple zero-overhead resource-aware routing protocol. It has the advantages of flexibility in delay-power trade-off, affordability in terms of resource consumptions, and reliability in terms of packet lost-free.

Article [37] focuses on sensor network topology discovery problem in home environment. They propose an algorithm for self-configuring sensor networks, which is based on flooding and gossiping methods with some new parameters. The algorithm is simple and energy efficient compared to the flooding method.

Article [33] proposes a novel and data-centric technique for the fast retrieval of aggregate sums from multiple regions in a sensor network, using only one single distributed data structure. The idea here is to construct a distributed data cube in the sensor network. The distributed data cube construction algorithm they propose makes use of the inclusion-exclusion principle and it can build a distributed prefix sum data cube in a sensor network in O(N) worst case time.

Chapter 4

NEW APPROACHES FOR GOSSIP SCHEDULING

In this chapter 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. Rather than building a fully connected spanning tree the semblance graph problem is concerned with selecting the highest weighted edges in the graph and partitioning the graph around those edges. However, the mechanism of examining edges for inclusion in a subgroup is very similar to the approach of building a spanning tree taken both in the Kruskal and Prim MST algorithms [10]. Our heuristics follow the edge selection techniques used in those algorithms. We do note that the semblance graph problem is similar to the graph bisection problem for optimization and processor scheduling. Unlike the graph bisection problem, the semblance graph problem is not concerned with creating equally sized partitions, but rather focused on maximizing the sum of edge weights. Therefore we focus our attention on MST-like heuristics.

4.1 Greedy-Prim Algorithm

Prim's MST algorithm builds a spanning tree starting with a randomly selected vertex in the graph. Each iteration a vertex is added to the tree by selecting the best incoming edge (lowest weight) into the existing tree [10]. 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

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_m ={node with max edge weight among remaining}, F_s ={}
3: To F_s, add the neighbors of newly added node F_m not seen before by any forest $F_0, .., F_{i-1}$ and not be present in F_m.
4: From F_s, select the node that has the largest edge.
5: **if** number of nodes in the current forest F_i does not exceed k **then**
6: Add this node to set F_m, mark it as being seen.
7: Go back to step 2
8: **else**
9: start a new forest F_{i+1}
10: **end if**
11: Repeat until all nodes have been seen
12: **end for**

Figure 4.1: Pseudo-code for the Greedy-Prim heuristic algorithm.

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 4.1.

4.2 Greedy-Kruskal Algorithm

We also developed a greedy algorithm based on Kruskal's MST algorithm [10]. 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

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

procedure GREEDY-KRUSKAL(G)

```
1: Sort the edges of G based on edge weight.
2: Take the unselected edge E_i of maximum value.
3: if adding E_i causes the size of any forest F_j to exceed k then
4:    throw it out.
5: else { add E_i to the forest of it's endpoints }
6:    add E_i to the forest of it's endpoint.
7:    if E_i's endpoints are members of two separate forests then
8:       merge the two forests.
9:    end if
10:   if E_i's endpoints are not members of any forest then
11:      create a new forest with E_i.
12:   end if
13: end if
14: Repeat for every edge
```

Figure 4.2: Pseudo-code for the Greedy-Kruskal heuristic algorithm.

Parameter Name	Range of values
streams (graph vertices)	2 - 20
stream period homogeneity	0.1 - 1.0
stream period range	10 - 40
k-constraint	1 - 5

Table 4.1: The range of values used to generate sample semblance graphs.

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 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 4.2 shows the pseudo-code of Greedy-Kruskal.

4.3 Analysis

The running time of our Prim-based algorithm is $O(V^2)$ where V is the number of vertices in the input graph, same as the original Prim's algorithm. At each iteration, the algorithm tries to find the minimum weight edge to add a new vertex to a group. Using a simple binary heap data structure at line 4 of Figure 4.1, the running time could be reduced to $O(E \log(V))$ where E is the number of edges in the semblance graph.

The running time of our Greedy-Kruskal algorithm is $O(E \log(E))$ where E is the number of edges in the input graph. The cost of the algorithm is dominated by the sorting step which can be done in $O(E \log(E))$ time. Once sorted, each edge is examined once and each vertex in the containing subgraph is examined which is $O(k \times E)$.

4.3.1 Simulated Comparison of the Algorithms

In order to evaluate the relative effectiveness of the proposed algorithms, Greedy-Kruskal and Greedy-Prim, we first performed a graph based simulation study. For the sake of comparison we implemented a brute-force algorithm to compute the optimal solution. The brute force algorithm enumerates all the possible subsets of the edges which do not violate the k-constraint (starting with subsets of size n continuing to subsets of size 2). The sum of edges contained in the subsets is calculated and the maximum value of all configurations returned.

We compared our two heuristics to the optimal solution on different semblance graphs to analyze how the different algorithms responded to different parameter settings. We randomly generated over 5000 sample networks by varying one of four parameters: the number of streams (vertices) in the semblance graph, the homogeneity of periods amongst streams (low homogeneity means most streams have different periods, high homogeneity means most streams have the same period), the variance in stream periods (range of potential period values) and the k-constraint of maximum partition size.

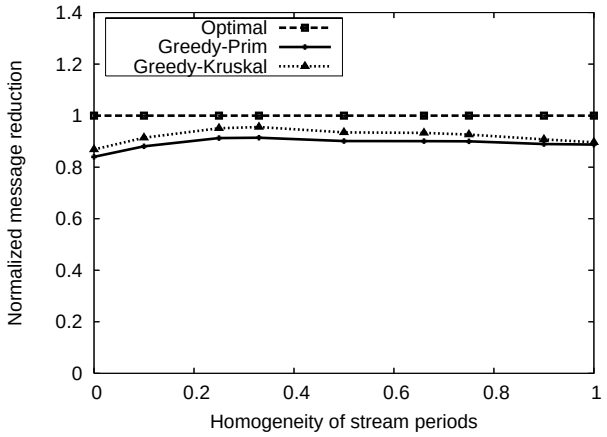

Figure 4.3: Performance comparison of heuristic algorithms and the distribution of stream periods

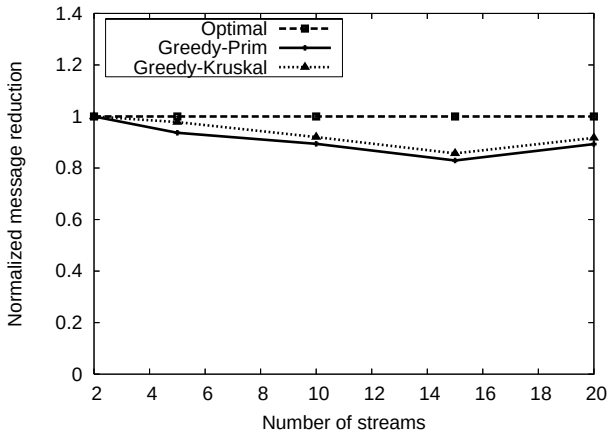

Figure 4.4: Performance comparison of heuristic algorithms while changing the number of streams

Table 4.1 summarizes all of the variables and the range of values used. The range of values is small because the brute force algorithm took too long to complete (over several days) for higher values, especially for higher number of streams. Semblance graphs were automatically generated for all permutations of the four parameters. Edge weights were assigned according to equation 2.1 using the periods from each edge endpoint.

We processed each graph using all three algorithms (Greedy-Prim, Greedy-Kruskal and Brute-force, labeled "Optimal" in our plots). The output from each algorithm is the "best" schedule of piggybacks for the particular semblance graph. The resulting schedules were scored using the sum of all edge weights contained in a subgroup. We compared the relative performance of each algorithm to the optimal score as computed by the brute force algorithm.

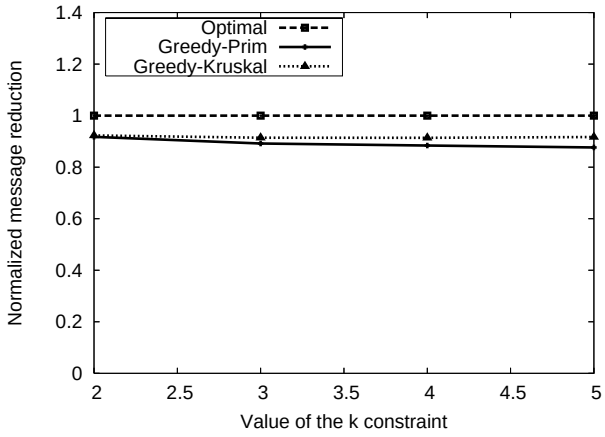

Figure 4.5: Performance of the different heuristic algorithms compared to the optimal vs. k.

Figures 4.3, 4.4 and 4.5 show the behavior of the different algorithms as different parameters are varied. The greedy algorithms show a strong decline in performance as the number of vertices (streams) increases. As the homogeneity of stream periods approaches the end points (0 and 1) the greedy algorithms degrade in performance. Changing the k-constraint had little effect on the performance of the greedy algorithms.

For 39.8% of the graphs the Greedy-Kruskal algorithm outperformed the Greedy-Prim algorithm. Greedy-Prim was better for 7.3% of the graphs while the two algorithms were even for the remaining 52.9% of the graphs. On average the Greedy-Kruskal algorithm had a 5.94% improvement over Greedy-Prim. Compared to the optimal score the Greedy-Kruskal scores were on average 96.5% of the optimal. Greedy-Prim was on average 94.6% of the optimal score. For 51% of the graphs the Greedy-Kruskal algorithm found the optimal score compared to only 44% with the Greedy-Prim algorithm. The results are expected because Greedy-Kruskal selects edges on a global basis compared to Prim which includes edges on a local basis. In general the performance of the greedy algorithms decreased as the number of vertices and edges in the semblance graph increased.

Chapter 5

SYSTEM IMPLEMENTATION

In this chapter we describe our system implementations in detail. Section 5.1 explains the implementation of our gossip scheduling and section 5.2 describes the whole system implementations as well as providing pseudocodes for them.

5.1 Piggybacking Implementation

This section describes the details of our implementation of a piggybacked gossip system. We implement piggybacking as a modification of the existing gossiping layer. The piggyback gossip replaces the functionality for preparing new messages for sending and handles receiving new piggybacked messages. Each node is configured with a semblance graph describing the current state of the system. When the gossip layer is initialized the Greedy-Kruskal algorithm is run to determine the gossip schedule (the piggyback subgroups). Then the node begins receiving and forwarding messages.

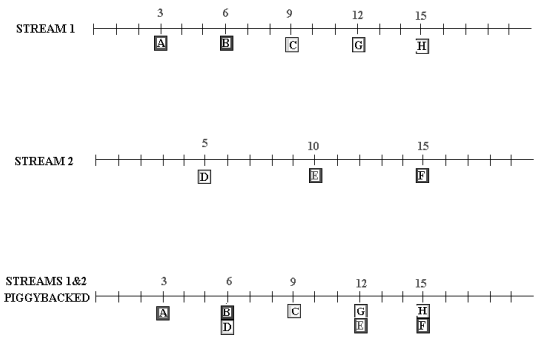

Figure 5.1: Illustration of piggybacking with the messages from two different streams.

The original gossip layer maintains a message buffer for each stream which is unmodified by the piggyback layer. Each message has associated with it a timer which expires when the message should next be gossiped. The piggyback maintains a separate timer for each subgroup, the primary group timer. The group timer uses the period from the stream with shortest period in the subgroup. When the primary timer expires the timers of each constituent stream are queried to see if any have expired. If so, the message from those streams is appended to the new outgoing message. Once the piggyback message has been formed it is forwarded using the original gossip forwarding. Figure 5.1 is an illustration of the piggybacking of messages.

All incoming gossip messages are first passed to the piggyback layer where the messages are decomposed into the individual stream messages. The messages are then delivered to the original gossip layer and processed accordingly.

5.2 Gossip Implementation

In this section, we explain how our simulations work. We provide pseudocodes for each different system in order to show the flow of simulation at every node in that system. Figure 5.2 shows how the whole simulation runs at a single node in the P2P system. This is the commonly known P2P canonical gossiping. Whenever a gossip timer expires for a message in the local message buffer, that message is gossiped to the neighbors as it was described in Figure 2.1.

Figure 5.3 shows the pseudocode for the piggybacked version of canonical gossiping in the P2P system. In this case, all the period information about the streams is read in, and then the semblance graph is formed by calculating the weights. After that, the semblance graph is solved locally using the Greedy-Kruskal algorithm. Then, whenever a gossip times expires in a component(group) of the semblance graph, available messages in that group are piggybacked on top of each other and gossiped out.

```
procedure P2P_GOSSIP()
 1: Get period information about all the streams
 2: Start the gossip timers
 3: while true do
 4:    Update(Decrement) the gossip timers
 5:    for Every message in the message buffer do
 6:       if The gossip timer expires(becomes 0) for the message then
 7:          Gossip that message
 8:          Reset the corresponding gossip timer
 9:       end if
10:    end for
11: end while
```

Figure 5.2: Pseudocode for the P2P standard gossip.

```
procedure P2P_PGOSSIP()
 1: Get period information about all the streams
 2: Calculate the weights and form the semblance graph
 3: Solve the semblance graph(locally) using Greedy-Kruskal algorithm
 4: Start the gossip timers
 5: while true do
 6:    Update(Decrement) the gossip timers
 7:    for Every component in the semblance graph do
 8:       if A gossip timer expires(becomes 0) in a component of semblance graph then
 9:          Piggyback and gossip the corresponding messages in that component
10:          Reset the gossip timer of the component
11:       end if
12:    end for
13: end while
```

Figure 5.3: Pseudocode for the P2P piggybacked gossip.

```
procedure TOS_FLOOD()
 1: Get period information about all the streams
 2: Start the gossip timers
 3: while true do
 4:    Update(Decrement) the gossip timers
 5:    for Every message in the message buffer do
 6:       if The gossip timer expires(becomes 0) for the message then
 7:          Broadcast that message to neighbors
 8:          Reset the gossip timer of the component
 9:       end if
10:    end for
11: end while
```

Figure 5.4: Pseudocode for the sensor network flooding.

```
procedure TOS_PgFLOOD()
 1: Get period information about all the streams
 2: Calculate the weights and form the semblance graph
 3: Solve the semblance graph(locally) using Greedy-Kruskal algorithm
 4: Start the gossip timers
 5: while true do
 6:    Update(Decrement) the gossip timers
 7:    for Every component in the semblance graph do
 8:       if A gossip timer expires(becomes 0) in a component of semblance graph then
 9:          Piggyback the corresponding messages in that component and flood
10:          Reset the gossip timer of the component
11:       end if
12:    end for
13: end while
```

Figure 5.5: Pseudocode for the sensor network piggybacked flooding.

Figure 5.4 demonstrates the pseudocode that belongs to our sensor network system simulation. This approach is the standard flooding whereas Figure 5.5 is the piggybacked version of it. Different from 5.2, in this version, the messages are broadcast to the sensor nodes whenever a timer expires.

The last two pseudocodes in this chapter belong to probabilistic gossip forwarding approach that we implemented in our sensor network systems simulation. Figure 5.6 is the standard probabilistic gossiping pseudocode and 5.7 is the piggybacked version of it. Whenever a gossip timer expires, a random real number between 0 and 1 is picked and if that value

```
procedure TOS_GOSSIP()
 1: Get period information about all the streams
 2: Start the gossip timers
 3: while true do
 4:    Update(Decrement) the gossip timers
 5:    for Every message in the message buffer do
 6:       if The gossip timer expires(becomes 0) for the message then
 7:          Pick a random real number r between 0.0 and 1.0
 8:          if  r ≤ gossip_probability then
 9:             Broadcast that message to neighbors
10:          end if
11:          Reset the gossip timer for the message
12:       end if
13:    end for
14: end while
```

Figure 5.6: Pseudocode for the sensor network gossip.

```
procedure TOS_PGOSSIP()
 1: Get period information about all the streams
 2: Calculate the weights and form the semblance graph
 3: Solve the semblance graph(locally) using Greedy-Kruskal algorithm
 4: Start the gossip timers
 5: while true do
 6:    Update(Decrement) the gossip timers
 7:    for Every component in the semblance graph do
 8:       if A gossip timer expires(becomes 0) in a component of semblance graph then
 9:          for Every message that belongs to this component of semblance graph do
10:             Pick a random real number r between 0.0 and 1.0
11:             if  r ≤ gossip_probability  then
12:                Piggyback this message
13:             else
14:                Do not piggyback this message
15:             end if
16:          end for
17:          Broadcast the piggybacked message to neighbors
18:          Reset the gossip timer for this component
19:       end if
20:    end for
21: end while
```

Figure 5.7: Pseudocode for the sensor network piggybacked gossip.

is greater than the *gossip_probability*, then the message is broadcast to the sensor nodes in the system. In the piggybacked version of this implementation, this probability determines whether a single message is going to be included in the piggybacked message, which is going to be sent out when the gossip timer expires for the component that contains the corresponding stream of that particular message. In our simulations, the *gossip_probability* was chosen to be 70%. Article [22] shows that a probability value between 60-80% is sufficient enough to ensure that a message is disseminated to all the network(with a probabilistic guarantee).

Chapter 6

PEERSIM SYSTEM SIMULATION EVALUATION

To evaluate the piggyback gossip effectiveness we implemented the default gossiping ("Gossip" in our plots) and piggybacked gossip ("PGossip" in our plots) in the Peersim 1.0 simulator. We used the cycle-driven engine of Peersim to perform our experiments. A fixed number of publishers were created which inject new messages into the network periodically at a constant rate. Each node gossiped the stream's message at a gossip period faster than the injection rate to ensure that the message is gossiped at least $\log(N)$ times. Table 6.1 shows the parameters that are used in the experiments and the typical values. The Kruskal algorithm was the semblance graph algorithm used in all simulations unless otherwise marked.

We first varied the size of the gossip overlay. Figure 6.1 shows the growth in total number of messages sent as the network size grows. In this experiment, the number of streams(publishers) was 25. The ratio between Gossip and PGossip seems to remain constant but the graph highlights the savings of PGossip as the number of nodes increases. Neither line in the plot is completely linear, having some dips where the total number of messages goes down as the number of nodes increases. Most likely this is the result of having slightly different injection rates (assigned randomly) between iterations of the experiment.

Parameter Name	Default values
k	5 streams
fanout	5 neighbors
gossip period	3 cycles or injection period
injection period	random from 1 to 10 cycles

Table 6.1: Parameters used in the experiments and their default values.

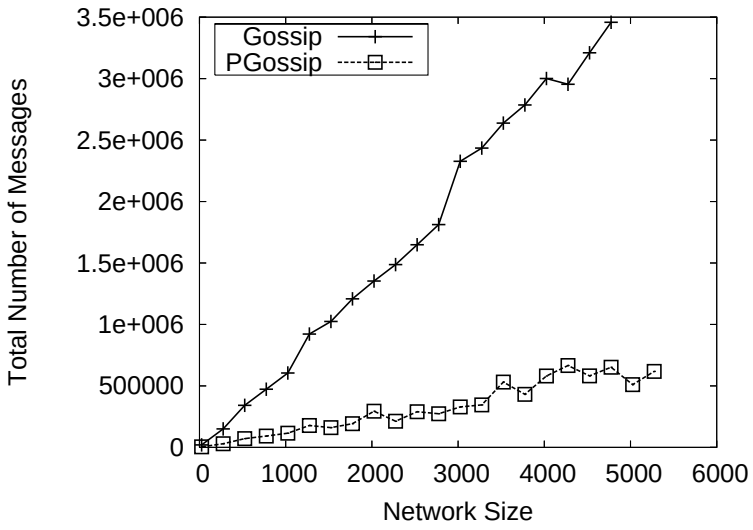

Figure 6.1: The number of messages sent under Gossip and PGossip as network size increases.

In order to observe the gains regarding the total bandwidth usage, we measured the total number of bytes transferred in the network. All of the settings were the same except that the simulations were run for 40 cycles in this experiment. Figure 6.2 shows the total number of bytes sent in the network for different network sizes. This figure shows that canonical gossiping scheme uses up to twice as much bandwith as compared to the piggybacked gossiping. While the total number of data bytes are the same in both schemes, this plots shows that by reducing the total number of messages sent out, we reduce the message overhead significantly, and thus use less bandwidth.

Figure 6.3 shows the effect of adding more streams to the system by increasing the fraction of nodes which publish new streams. The total size of the network used in this experiment was 250 with a k value of 5. Clearly there is a linear increase in messages generated compared to number of streams. The ratio of total messages sent by Gossip and PGossip is close to the k value suggesting that our scheduling algorithm performs efficiently. The growth is very similar to that seen when increase the size of the network. Overall the maximum reduction in message count by PGossip from both experiments was 82%.

Figure 6.4 shows the ratio of messages generated under PGossip compared to regular

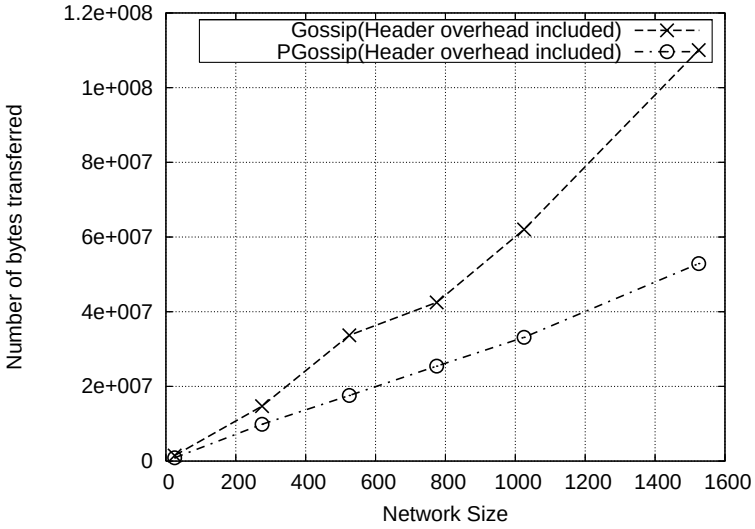

Figure 6.2: The total number of bytes transferred under Gossip and PGossip.

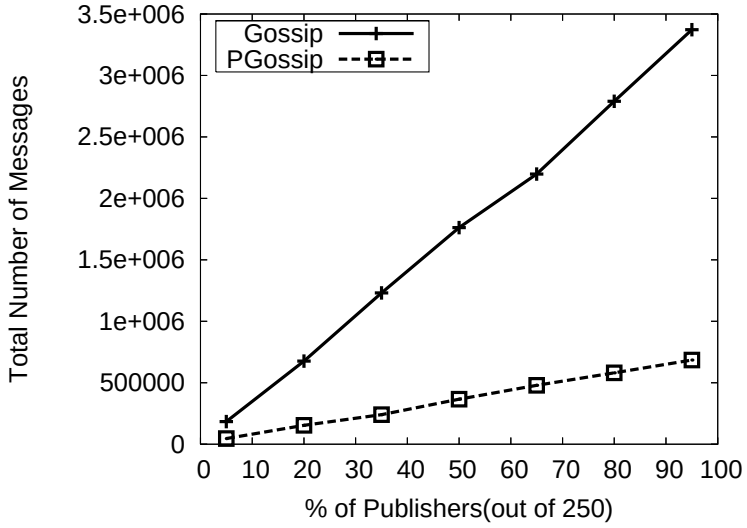

Figure 6.3: Total number of messages send with increasing fraction of publisher nodes.

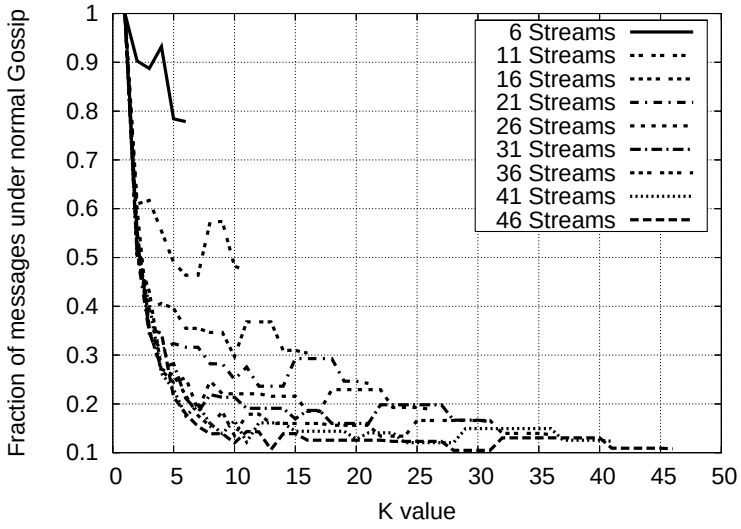

Figure 6.4: Fraction of messages sent under PGossip vs. regular Gossip for different values of k.

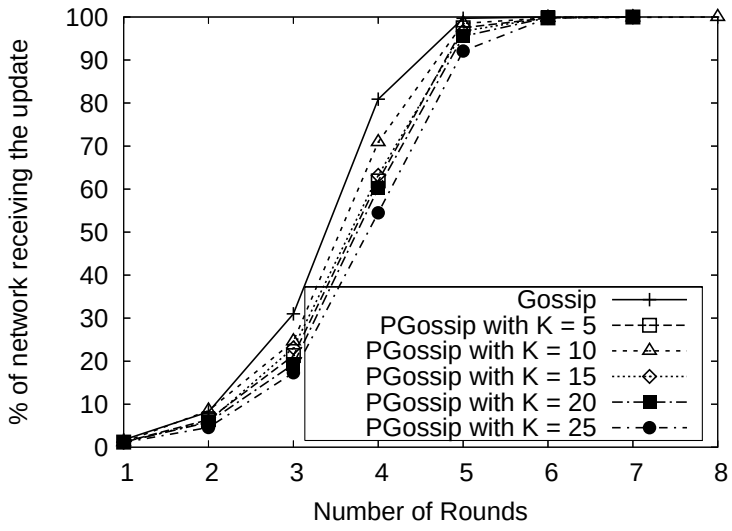

Figure 6.5: Cumulative distribution of an update's dissemination time using Gossip and PGossip.

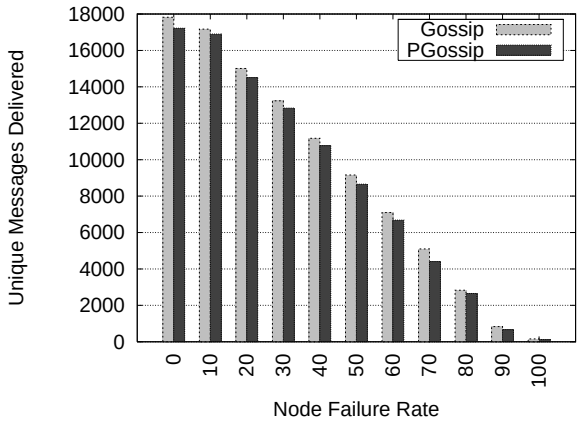

Figure 6.6: The number of unique gossip messages delivered with failing nodes.

Gossip as the value of k changes. It is interesting to note that a knee occurs in the graphs at k values around 5-10. This suggests that having an unlimited k value does not add much to the system and that there is an optimal k value depending on the number of streams in the system. Using these results we can determine that a k value ≈ 10 is sufficient to provide maximum messages savings for any number of streams.

We also compared the latency of propagating a single message throughout the network using PGossip compared to Gossip. Figure 6.5 shows the cumulative distribution function of dissemination latency of a single update, namely update number 3 from publisher 0, when using Gossip and PGossip. The network size in this experiment was 1000 and the number of streams was 25. We ran the piggybacked gossip several times with different k values, each scenario shown with a different line in the graph. As expected, higher k values result in longer latency as more streams are piggybacked together and therefore the average relatedness between all streams in the group goes down. However, the drop in latency, even for high k values, is only one round.

To measure the effects of piggybacking on resiliency we counted the number of unique messages delivered in the system as the failure rate increased. Figure 6.6 shows the results for a system with 250 nodes and 20 streams. Interestingly, PGossip always delivers fewer messages, even when failure rate is 0. This is due to the delay that PGossip adds to messages. When a new update arrives for a given stream it replaces the old message. In PGossip, depending on the piggybacking schedule, some stream messages may be delayed slightly longer than the stream period. Thus, while the message is delayed, occasionally the next update arrives, replacing the old message and effectively reducing the number of unique messages delivered in the system. When this effect is taken into account the performance of PGossip is equivalent to Gossip.

Conclusion from this chapter Through PeerSim full-system simulations, we showed that our algorithms have a very significant impact on P2P gossip systems by reducing the message count by up to 82%. Total bandwith usage reduction can go up to 54%.

Chapter 7

TINYOS SYSTEM SIMULATION EVALUATION

In order to evaluate the piggyback gossip effectiveness in sensor network platforms, we implemented the default gossiping and piggybacked gossiping in TinyOS 2.0. We used TOSSIM, the TinyOS mote simulator. TOSSIM simulates entire TinyOS applications. It works by replacing components with simulation implementations. TOSSIM is a discrete event simulator. When it runs, it pulls events of the event queue (sorted by time) and executes them. Depending on the level of simulation, simulation events can represent hardware interrupts or high-level system events (such as packet reception). TOSSIM supports two programming interfaces, Python and C++. We use the Python interface to program and run the simulation. In the PeerSim implementation, a message is gossiped to $fanout$ number of randomly chosen neighbors. In the TinyOS implementation, we use flooding instead, which is the common ad hoc routing protocol. Each sensor node gossips a message by flooding it over the network, which allows the nearby neighbors get the message if they can hear it. We also use the standard flooding and gossiping(probabilistic flooding) that is discussed in [22]. In our experiments, we set the probability to 70%. Again, the Kruskal algorithm is the semblance graph algorithm that we use in all simulations. Various number of streams(publishers) and sensor nodes were used throughout the experiments. Publisher nodes were selected randomly among the nodes of the network. Most common number of publishers used our experiments was 24 and the number of nodes in the system is 225 unless otherwise noted in the experiment plots. The default k value used in our experiments was 7. This is because a default TinyOS message's maximum payload size is 28 bytes and one single message in our gossiping system is 4 bytes. In order to build a network topology for TOSSIM, we used the link-layer model

Environment Parameters	Meaning	Value
PATH_LOSS_EXP	rate at which signal decays	3.0
SHADOWING_STD_DEV	randomness of received signal due to multipath	4.0 dB
D0	reference distance	1 m
PL_D0	power decay for the D0	55.0 dB
Radio Parameters	Meaning	Value
WHITE_GAUSSN_NOISE	standard deviation of additive white gaussian noise	4
NOISE_FLOOR	radio noise floor	-105.0 dBm
S11	variance of noise floor	3.7 dB
S12	covariance between noise floor and output power	-3.3 dB
S21	same as S12	-3.3 dB
S22	variance of output power	6.0 dB

Table 7.1: Link-layer parameters and their values used in the experiments.

by Marco Zuniga which comes with TOSSIM. The topology that we use in our experiments is a square shaped grid. So, the number of nodes that we use in the experiments is always a square of some integer. All the sensor nodes are placed 3 meters away from each other. Table 7.1 shows the link-layer sensor network parameters.

In our plots, "Flood" and "PgFlood" refer to the schemes that use flooding and piggybacked flooding respectively as their message sending scheme. Similarly, "Gossip70" and "PGossip70" respectively refer to the schemes that use probabilistic flooding and probabilistic piggybacked flooding(with 70% probability).

As in the PeerSim simulation, we started with examining the effect of network size on the total number of messages sent over the network. Figure 7.1 shows the plot that belongs to this experiment. In this experiment, the number of streams(publishers) is 24. Their gossiping periods were randomly selected from the range of [1,4] seconds. The injection periods for each publisher nodes are selected randomly from the range of [9,12] seconds. The plot shows the first 30 seconds of both simulations. As can be seen from the plot, the PgFlood approach is about 3-4 times better than the Flood approach.

The second experiment that we have done was to observe the total bandwidth usage of two different approaches. Figure 7.2 shows the comparison of total bandwidth usage

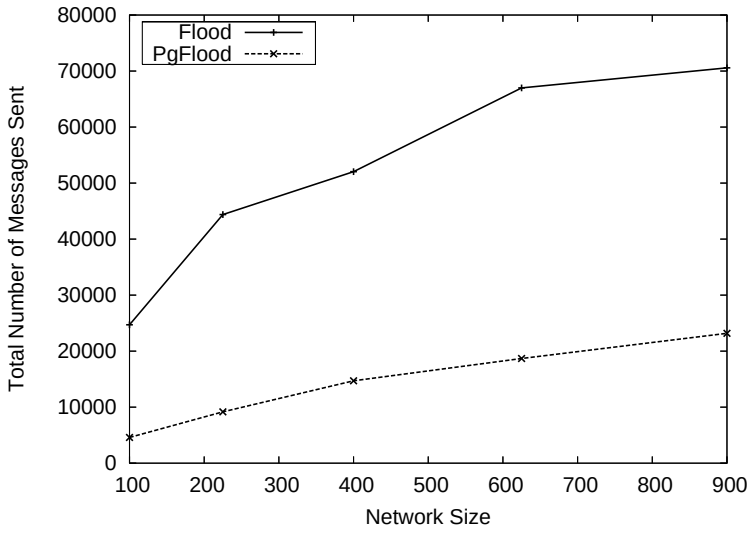

Figure 7.1: The number of messages sent under Flood and PgFlood as network size increases.

between Flood and PgFlood schemes. As can be seen from the plot, PgFlood can reduce the bandwidths usage down to about 57% of the one that is consumed by the Flood scheme. In sensor network systems, message overhead consume a significant amount of bandwith due to their large headers and Figure 7.2 shows that avoiding or reducing the message overhead will help with the total bandwith usage in the whole system.

The next experiment in Figure 7.3 compares the power consumption of two different schemes. We made a power consumption estimation using the Mote2 data sheet that is available at the URL in [1]. According to the data sheet transmission(broadcasting in our case) of data draws 2.7 times more current than receiving does. 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. As can be observed from the figure, PgFlood scheme can achieve significant amount of power saving as compared to Flood scheme. For 400 nodes, the power consumption in PgFlood falls. We believe that the reason for this is too many collisions occuring in the system as the network size grows and too many messages not being heard(received) by the neighboring nodes in

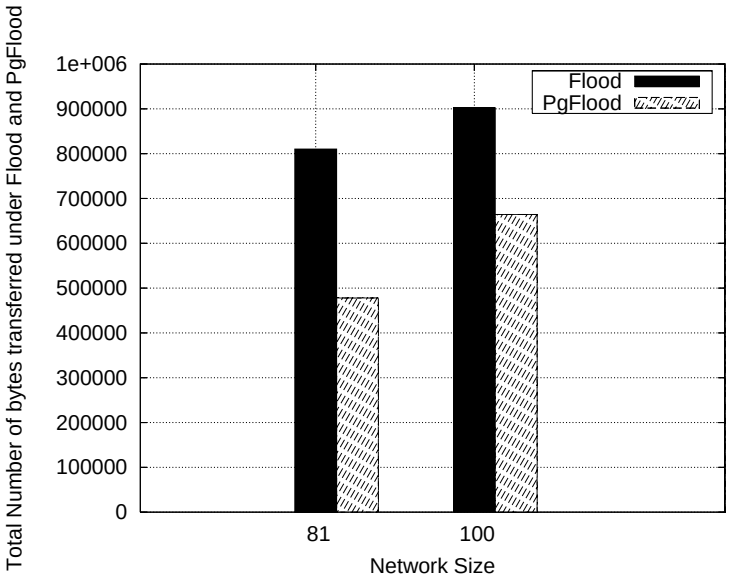

Figure 7.2: The number of bytes sent under Flood and PgFlood with k=7.

the system because of the heavy transmission traffic. The reason this becomes visible only in the PgFlood and not in PFlood scheme is the size of the messages that are getting lost. Piggybacked messages are 7 times as big as the messages in Flood.

Figure 7.4 shows the effect of k on the total number of messages sent over the network. Every possible k value for our TinyOS setup was tried out with the networks that have 225 nodes and varying number of publishers. As the plot shows, the total number of messages sent over the network decreases as the value of k increases. It is important to observe that there are knees of the curves mostly around $k = 5$. So, although picking the $k = 7$ would give the least number of messages sent, picking $k = 5$ would give a result which is close to that of $k = 7$'s. While going from $k = 1$ to $k = 5$ reduces the total number of messages significantly, going from $k = 5$ to $k = 7$ does not reduce the number of messages with the same level of significance.

All the above mentioned figures were sort of what we have expected before the experiments. However, the result that is presented in figure 7.5 needs some discussion. The experiment shows the cumulative distribution function of a specific update's dissemination latency to the whole network. The update is generated at 10.7 seconds in the simulation.

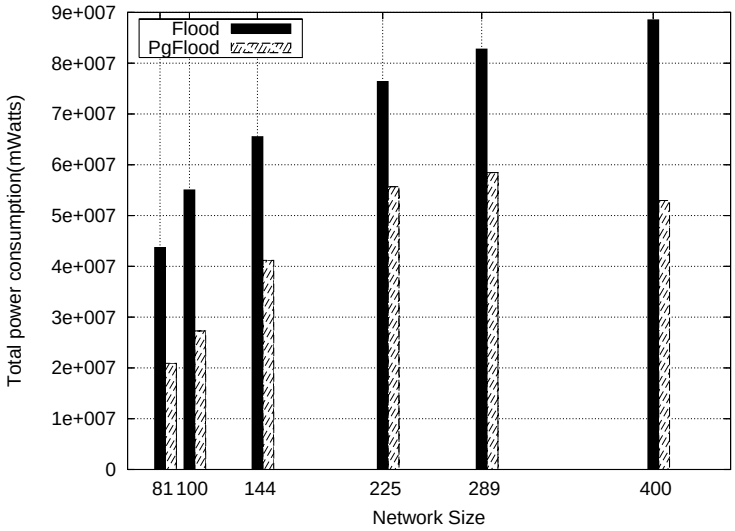

Figure 7.3: Power consumption comparison of Flood and PgFlood.

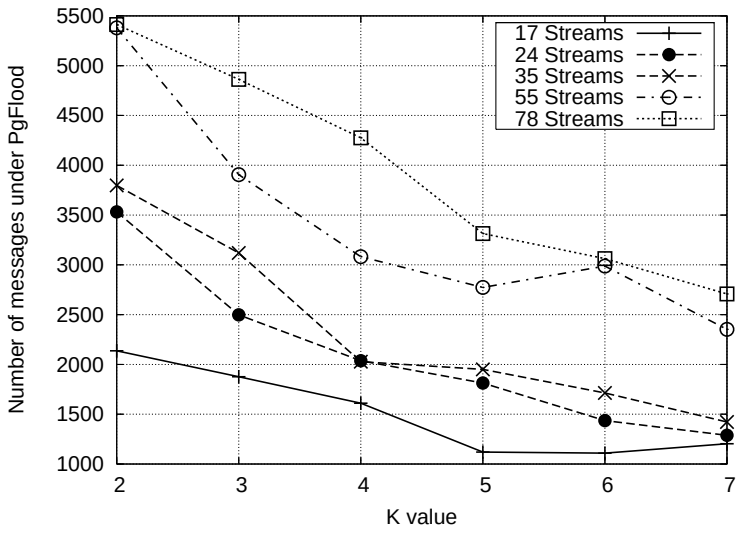

Figure 7.4: Number of messages sent under PgFlood vs. Flood for different values of k.

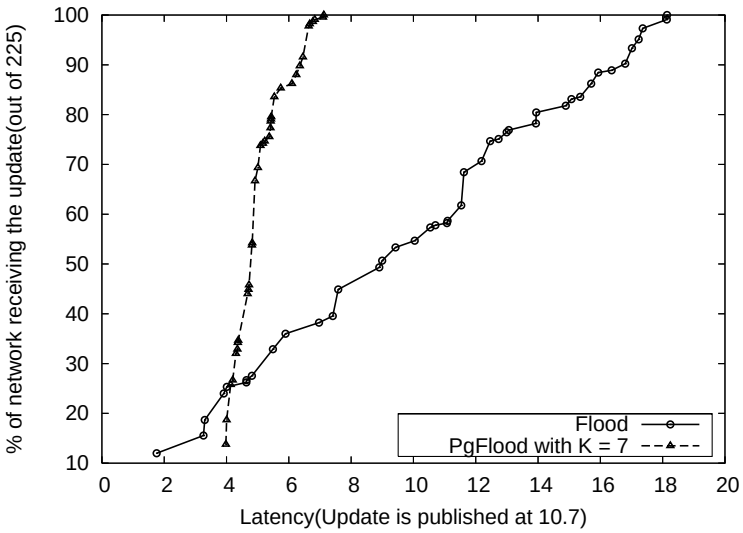

Figure 7.5: An update's arrival time at the sensor nodes in a system of 225 nodes.

As can be seen, the piggybacking scheme delivers the update to the whole network with a significantly better latency. We think that this is because of the nature of the standard scheme that floods the messages one by one which takes more time and causes alot more traffic as compared to the piggybacked version. Also, we hypothesize that due to the high number of messages trying to be sent over the network, too many collisions occur and nodes back off until they find a suitable time to transmit the message. In case of piggybacked flooding, this does not happen as much since the number of messages trying to be sent at a time is not as many as in the standard flooding scheme.

Figure 7.6 demonstrates the effect of increasing number of streams on total number of messages in a 225 node network. As expected, the total number of messages sent out in the network increases linearly with the number of publishers. PgFlood scheme uses about 3-4 times less number of messages than the Flood scheme does.

Figures 7.7, 7.8 and 7.9 refer both to the reliability and latency of the systems. They all belong to a 30 second trace of both standard and piggybacked systems. Figure 7.7 shows the dissemination percentage of each message that was published between the simulation seconds 5 and 10. x axis shows the unique IDs that were assigned in chronological order to

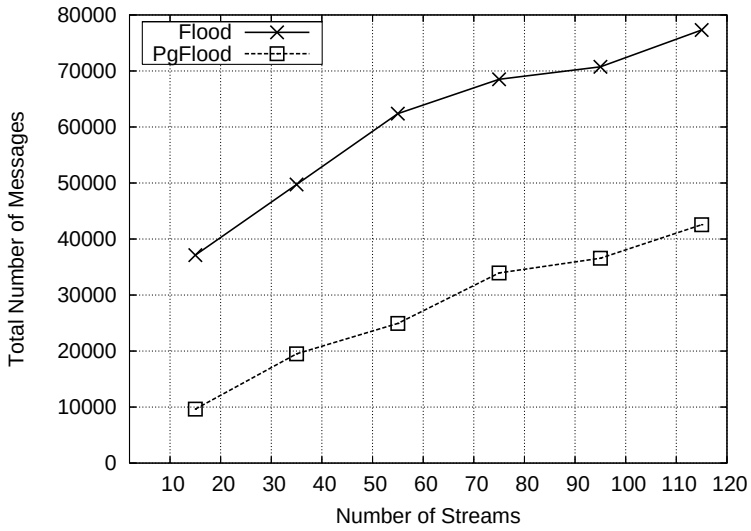

Figure 7.6: The total number of messages sent with increasing number of publisher nodes.

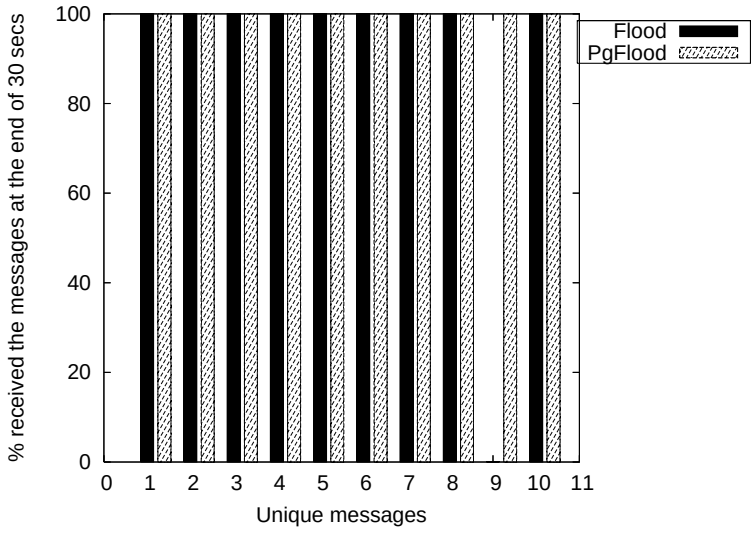

Figure 7.7: Delivery of updates generated during 5-10 secs

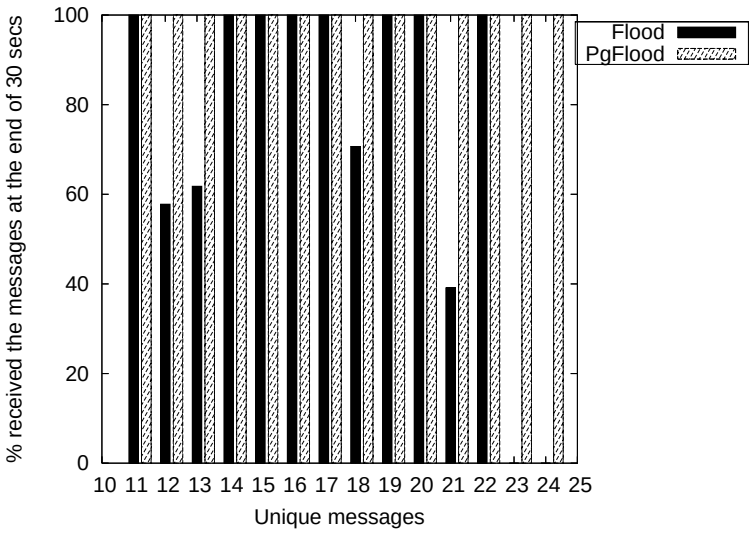

Figure 7.8: Delivery of updates generated during 10-15 secs

the updates that were published. At the end of 30 seconds, almost all the updates generated between 5 and 10 are disseminated to the whole network in both systems. Piggybacked approach looks even better since standard approach has one missing message that was not been able to be disseminated to the network at all.

Similarly, figure 7.8 shows the dissemination percentage of each message that was published during simulation seconds 10 and 15. As can be seen from the plot, at the end of 30 seconds, the piggybacked system has disseminated almost all of the messages where standard gossip is still working on it. Figure 7.9 shows similar information for the updates generated during simulations seconds 15 and 20. Dissemination of the messages in standard gossiping scheme is just starting while piggybacking is almost done.

So far we presented results with the experiments that use flooding for their message dissemination scheme. The rest of the plots are related to the "Gossip70" and "PGossip70" schemes that we have also implemented. Figure 7.10 shows the four approaches in one graph as the network size is being experimented. As expected, gossiping approaches have less number of messages in total as compared to flooding ones.

Similarly, figure 7.11 compares the effect of different approaches on latency. As the plot

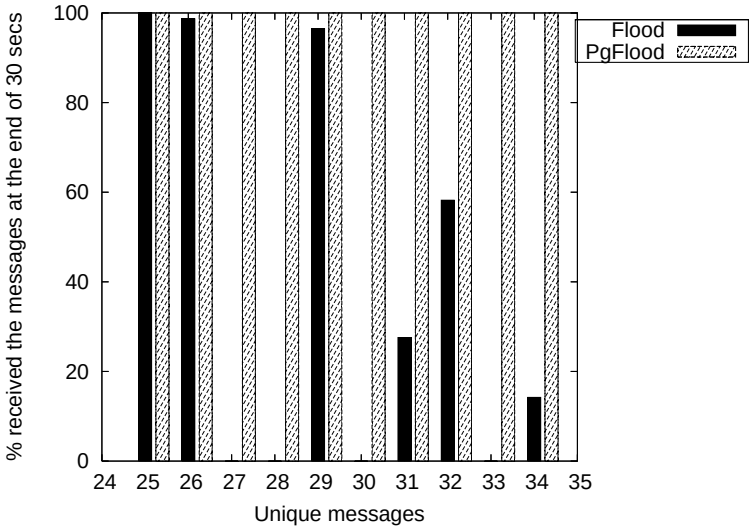

Figure 7.9: Delivery of updates generated during 15-20 secs

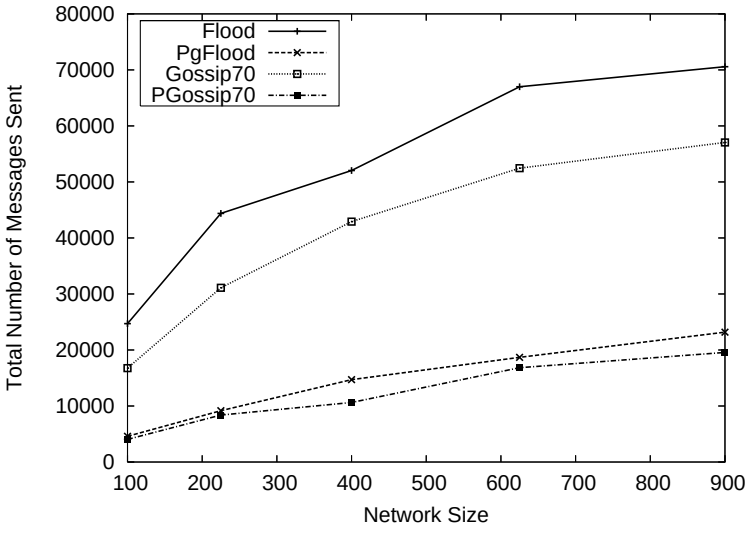

Figure 7.10: Number of messages in Flood,PgFlood,Gossip70 and PGossip70 as network size increases.

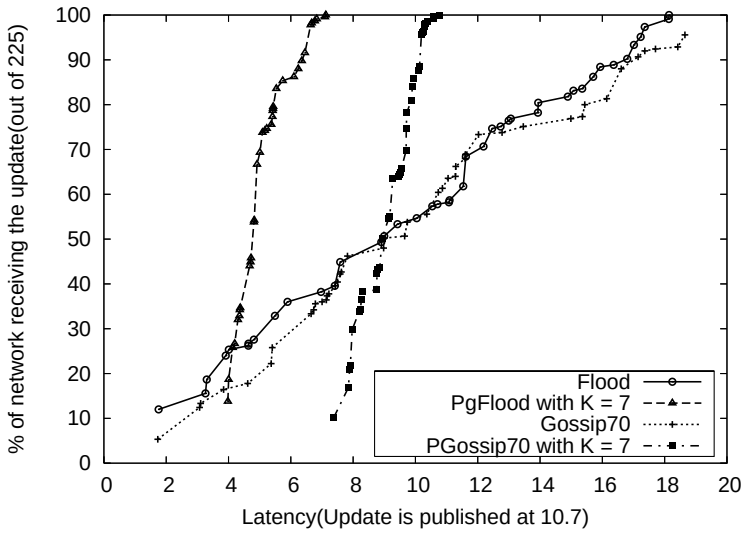

Figure 7.11: Update's arrival time at sensor nodes using Flood, PgFlood, Gossip70 and PGossip70.

shows, there is almost no difference between Gossip70 and PgFlood approaches. There is about 2 seconds of an additional latency in PGossip70 as compared to PgFlood approach. However, still both PGossip70 and PgFlood show smaller overall latency as compared to non-piggybacked approaches.

Figures 7.12 and 7.13 demonstrates the distribution percentage of the updates generated during 5-10 and 10-15 seconds of the 30 second simulation. As the plots demonstrate, Gossip70 and PGossip70 perform almost as good as their corresponding flooding approaches.

Conclusion from this chapter Through TOSSIM(TinyOS) full-system simulations, we showed how our algorithms have a real impact on wireless sensor network gossip systems by reducing the message count by up to 76%. Total bandwith usage reduction in can go upto 47% and power consumption reduction can go upto 50%.

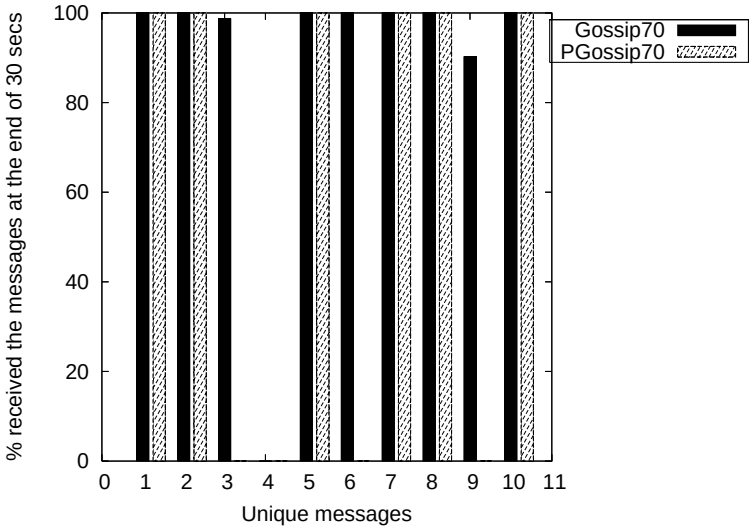

Figure 7.12: Delivery of updates generated during 5-10 secs for Gossip70 and PGossip70

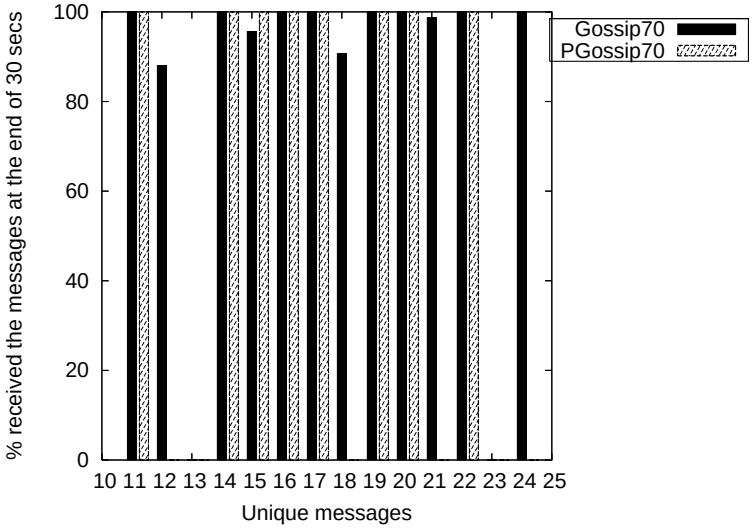

Figure 7.13: Delivery of updates generated during 10-15 secs for Gossip70 and PGossip70

Chapter 8

CONCLUSION

In this thesis, we showed that the system and message overhead in multi-stream gossip based P2P and sensor network publish-subscribe systems can be reduced through effective stream scheduling techniques that we introduced. We proposed a *piggybacking* approach to send messages from related streams together and thus reduce both the message count and system resource demands. Our new concept "semblance graph" gives a representation of the relationship between different streams and the potential benefit from each piggyback. We presented our two new heuristic algorithms, Greedy-Kruskal and Greedy-Prim, which achieve near optimality (within 3.5%) in solving the semblance graph problem via algorithm comparison simulations. Through PeerSim and TOSSIM(TinyOS) full-system simulations, we showed how our algorithms have a real impact on gossip systems by reducing the message count by up to 82%. Total bandwith usage reduction in both systems can go upto 47%. Power consumption reduction in sensor network systems can go upto 50%.

At the core of our approach is the new concept that we call semblance graph problem. Regardless of whether it is a P2P or a sensor network system, the semblance graph problem still exists in both and it is very important to solve this problem as good as possible in order to perform the piggybacking of streams efficiently.

Future Work In this thesis, we discussed one mechanism for creation and maintenance of the semblance graph. We chose to use one metric, stream relatedness, to weight the graph. Since weight calculation is a very important component of solving the semblance graph problem, we believe that investigation of other metrics to weight the graph would

we worthwhile. Even though our heuristic algorithms perform really well, looking for new algorithms to solve the semblance graph problem would be a good research direction. In addition to exploring different metrics for the semblance graph, investigation for collecting the semblance graph parameters k and gossip period would also be useful.

References

[1] Mica2 datasheet. Technical report, Crossbow Technology, 2007. Also available as `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf`.

[2] A. Allavena, A. Demers, and J. E. Hopcroft. Correctness of a gossip based membership protocol. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, 24:292 – 301, 2005.

[3] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. volume 35, pages 131 – 145, Banff, Alta., Canada, 2002.

[4] N. T. J. Bailey. *Epidemic Theory of Infectious Diseases and its Applications.* Hafner Press, second edition edition, 1975.

[5] K. Birman. The surprising power of epidemic communication. *Future Directions in Distributed Computing. Research and Position Papers (Lecture Notes in Computer Science Vol.2584)*, pages 97 – 102, 2003.

[6] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, 1999.

[7] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory*, 52(6):2508 – 2530, 2006.

[8] M. Calle and J. Kabara. Measuring energy consumption in wireless sensor networks using gsp. *2006 IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications (IEEE Cat. No. 06TH8881)*, pages 5 pp. –, 2006.

[9] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, pages 85 – 96, 2001.

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

[11] P. Costa, M. Migliavacca, G. P. Picco, and G. Cugola. Epidemic algorithms for reliable content-based publish-subscribe: An evaluation. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS)*, pages 552–561, 2003.

[12] A. Datta, S. Quarteroni, and K. Aberer. Autonomous gossiping: a self-organizing epidemic algorithm for selective information dissemination in wireless mobile ad-hoc networks. *Semantics of a Networked World. Semantics for Grid Databases. First International IFIP Conference, ICSNW 2004. Revised Selected Papers. (Lecture Notes in Computer Science Vol.3226)*, pages 126 – 43, 2004.

[13] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, New York, NY, USA, 1987. ACM Press.

[14] A. Dimakis, A. Sarwate, and M. Wainwright. Geographic gossip: efficient aggregation for sensor networks. *The Fifth International Conference on Information Processing in Sensor Networks (IEEE Cat. No. 06EX1353)*, pages 69 – 76, 2006.

[15] M. Elhadef and A. Boukerche. A failure detection service for large-scale dependable wireless ad-hoc and sensor networks. *2007 2nd International Conference on Availability, Reliability and Security*, pages 8 pp. –, 2007.

[16] E. Ertin. Distributed multimodal data fusion for large scale wireless sensor networks. *Proceedings of the SPIE - The International Society for Optical Engineering*, 6229(1):1 –, 2006.

[17] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems*, 21(4):341–374, 2003.

[18] C. Fernandess and D. Malkhi. On collaborative content distribution using multi-message gossip. *Proceedings. 20th International Parallel and Distributed Processing Symposium (IEEE Cat. No.06TH8860)*, pages 9 pp. –, 2006.

[19] R. Guerraoui, S. Handurukande, A.-M. Kermarrec, F. Le Fessant, K. Huguenin, and E. Riviere. Gosskip, an efficient, fault-tolerant and self organizing overlay using gossip-based construction and skip-lists principles. *2006 6th IEEE International Conference on Peer-to-Peer Computing (IEEE Cat. No. 06PR2679)*, pages 8 pp. –, 2006.

[20] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[21] I. Gupta, A.-M. Kermarrec, and A. J. Ganesh. Efficient epidemic-style protocols for reliable and scalable multicast. In *IEEE Symposium on Reliable Distributed Systems (SRDS '02)*, pages 180–189, Oct 2002.

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

[23] R. W. Hall, A. Mathur, F. Jahanian, A. Prakash, and C. Rassmussen. Corona: a communication service for scalable, reliable group collaboration systems. *ACM Transactions on Computer Systems*, pages 140–149, 1996.

[24] W. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. *MobiCom'99. Proceedings of Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 174 – 85, 1999.

[25] X. Hou, D. Tipper, and S. Wu. Traffic-aware gossip-based energy conservation for wireless ad hoc and sensor network routing. *2006 3rd IEEE Consumer Communications and Networking Conference, CCNC 2006*, 1:341 – 345, 2006.

[26] X. Hou, D. Tipper, and S. Wu. A gossip-based energy conservation protocol for wireless ad hoc and sensor networks. *Journal of Network and Systems Management*, 14(3):381 – 414, Sept. 2006.

[27] K. Iwanicki, M. van Steen, and S. Voulgaris. Gossip-based clock synchronization for large decentralized systems. *Self-Managed Network Systems, and Services. Second IEEE International Workshop, SelfMan 2006. Proceedings (Lecture Notes in Computer Science Vol. 3996)*, pages 28 – 42, 2006.

[28] R. M. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *IEEE Symposium on Foundations of Computer Science*, pages 565–574, 2000.

[29] D. Kempe and J. Kleinberg. Protocols and impossibility results for gossip-based communication mechanisms. *Proceedings 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 471 – 80, 2002.

[30] P. Kouznetsov, R. Guerraoui, S. B. Handurukande, and A. M. Kermarrec. Reducing noise in gossip-based reliable broadcast. In *IEEE Symposium on Reliable Distributed Systems (SRDS '01)*, page 186, 2001.

[31] S. Kulkarni. Tdma service for sensor networks. *Proceedings. 24th International Conference on Distributed Computing System Workshops*, pages 604 – 9, 2004.

[32] P. Kyasanur, R. Choudhury, and I. Gupta. Smart gossip: an adaptive gossip-based broadcasting service for sensor networks. *2006 IEEE International Conference on Mobile Adhoc and Sensor Systems (IEEE Cat. No.06EX1476)*, pages 10 pp. –, 2006.

[33] L. H. Lee and M. H. Wong. Aggregate sum retrieval in sensor network by distributed prefix sum data cube. *Proceedings. 19th International Conference on Advanced Information Networking and Applications*, vol.1:331 – 6, 2005.

[34] P. Levis, N. Patel, S. Shenker, and D. Culler. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. Technical Report UCB/CSD-03-1290, EECS Department, University of California, Berkeley, Nov 2003.

[35] M.-J. Lin and K. Marzullo. Directional gossip: Gossip in a wide area network. In *European Dependable Computing Conference*, pages 364–379, 1999.

[36] M. Medidi, J. Ding, and S. Medidi. Data dissemination using gossiping in wireless sensor networks. *Proceedings of SPIE - The International Society for Optical Engineering*, 5819:316 – 327, 2005.

[37] S. Mehta, W.-S. Yoon, S.-W. Min, and S. Yu. Topology generation algorithms for home sensor networks. *Proceedings. Second IEEE Workshop on Software Technologies for the Future Embedded and Ubiquitous Systems*, pages 166 – 8, 2004.

[38] M. Miller, C. Sengul, and I. Gupta. Exploring the energy-latency trade-off for broadcasts in energy-saving sensor networks. *25th IEEE International Conference on Distributed Computing Systems*, pages 17 – 26, 2005.

[39] H. Miranda, S. Leggio, L. Rodrigues, and K. Raatikainen. Epidemic dissemination for probabilistic data storage. pages 124 – 45, 2006.

[40] M. Musolesi and C. Mascolo. Controlled epidemic-style dissemination middleware for mobile ad hoc networks. *2006 Third Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (IEEE Cat. No. 06EX1437)*, pages 150 – 8, 2006.

[41] J. Pereira, R. Oliveira, and L. Rodrigues. Efficient epidemic multicast in heterogeneous networks. *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops. OTM Confederated International Workshops and Posters AWeSOMe, CAMS, COMINF, IS, KSinBIT, MIOS-CIAO, MONET, OnToContent, ORM, PerSys, OTM Academy Doctoral Consortium, RDDS, SWWS, and SeBGIS. Proceedings, Part II (Lecture Notes in Computer Science Vol. 4278)*, pages 1520 – 9, 2006.

[42] J. Pereira, L. Rodrigues, M. J. Monteiro, R. Oliveira, and A.-M. Kermarrec. Neem: Network-friendly epidemic multicast. In *IEEE Symposium on Reliable Distributed Systems (SRDS '03)*, October 2003.

[43] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, 2003.

[44] L. Rodrigues, S. B. Handurukande, J. Pereira, R. Guerraoui, and A.-M. Kermarrec. Adaptive gossip-based broadcast. In *International Conference on Dependable Systems and Networks (DSN)*, 2003.

[45] J. Saramaki. A method for decentralised optimisation in networks. *AIP Conference Proceedings*, (776):215 – 23, 2005.

[46] W. Vogels, R. van Renesse, and K. Birman. The power of epidemics: robust communication for large-scale distributed systems. *Computer Communication Review*, 33(1):131 – 5, 2003.

[47] S. Voulgaris, M. Jelasity, and M. van Steen. A robust and scalable peer-to-peer gossiping protocol. *Agents and Peer-to-Peer Computing. Second International Workshop, AP2PC 2003. Revised and Invited Papers (Lecture Notes in Artificial Intelligence Vol.2872)*, pages 47 – 58, 2004.

[48] S. Voulgaris, E. Riviere, A.-M. Kermarrec, and M. van Steen. Self-organizing content-based publish and subscribe for dynamic and large scale collaborative networks. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS '06)*, 2006.

[49] G. Wang, D. Lu, W. Jia, and J. Cao. Reliable gossip-based broadcast protocol in mobile ad hoc networks. *Mobile Ad-Hoc and Sensor Networks. First International Conference, MSN 2005. Proceedings (Lecture Notes in Computer Science Vol.3794)*, pages 207 – 18, 2005.

[50] L. Wang and S. Kulkarni. Gappa: gossip based multi-channel reprogramming for sensor networks. *Distributed Computing in Sensor Systems. Second IEEE International Conference, DCOSS 2006. Proceedings (Lecture Notes in Computer Science Vol. 4026)*, pages 119 – 34, 2006.

[51] C. M. Yang. Deployable techniques to enable cooperative distribution of web content. Master's thesis, University of Illinois at Urbana-Champaign, 2006.

[52] D. Yupho, M. Calle, and J. Kabara. Longer network lifetime when using energy efficient gsp for wireless sensor networks. *2007 International Symposium on Autonomous Decentralized Systems*, pages 6 pp. –, 2007.

[53] Y. Zhang and L. Cheng. Flossiping: A new routing protocol for wireless sensor networks. *Conference Proceeding - IEEE International Conference on Networking, Sensing and Control*, 2:1218 – 1223, 2004.