Scheduling of Multi-Stream Gossip Systems

Nathanael Thompson, Ercan Ucan and Indranil Gupta
Dept. of Computer Science, University of Illinois at Urbana-Champaign
Email: {nathomps,eucan2,indy}@cs.uiuc.edu

Abstract

Many distributed applications are beginning to employ gossip-based message dissemination, where the burden of content distribution is shared democratically among the recipient nodes, e.g., for RSS distribution. However, such systems have many communication channels, i.e., multiple gossip streams may be present within the same application, e.g., an RSS content distribution system involves several publishers sending out streams to overlapping groups of subscribers (i.e., "nodes"). Yet, most existing gossiping approaches tend to have nodes treat each gossip stream independently of one another. This leads to each node being burdened with a message overhead that is the sum from all gossip streams. In this paper, we show that if all nodes instead use a scheduling strategy on the multiple gossip streams that they receive and forward, this leads to a significant reduction in overhead, but without affecting the original reliability, scalability, or latency. Simply put, our approach piggybacks messages from streams atop one another. However, in doing so, we take into account the different frequencies of gossiping for streams, and affinities of streams to one another. We formulate these constraints in a new model that we call the "semblance graph". After formally stating our gossip scheduling problem, we show that finding an optimal solution to it is an NP-complete problem. We then present two greedy heuristics for the problem, one being Prim-like and the other being Kruskal-like (though the MST problem is different from ours). Our experimental results are in two categories – (1) a semblance graph-based evaluation that shows the heuristics come within 3.5% of the optimal solution, and (2) a distributed system simulation that shows the message savings are up to 85% compared to the default gossiping approach without scheduling.

1 Introduction

Gossip (epidemic-based) algorithms are emerging as attractive choices for publish-subscribe systems, primarily because they democratize the overhead of content distribution among the subscribing "nodes". In the canonical gossip protocol, for a single content stream, each node periodically (say once every *protocol 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. Analysis of such gossip protocols has revealed that a scalable per-node overhead that is (poly-)logarithmic in system size disseminates the message(s) quickly and reliably to a large fraction of the entire system (close to 100%).

However, in reality, these publish-subscribe systems often disseminate *multiple streams* in the underlying gossip communications layer. For example, in an RSS delivery system, the updates from each publisher constitute a single stream. Yet, most existing gossip mechanisms handle each stream independently, with each node typically generating separate gossip messages for each stream in the system. In addition, individual membership lists and additional state may be maintained for each stream. All of this causes the canonical gossip approach described above to create a per-node overhead that is a sum from all the streams.

Our approach to addressing this overhead problem is to first observe that streams may be related to one another. However, naive solutions based on mere overlap of streams' subscriber sets may not be the best. For instance, one approach might divide the subscribers into "groups" based on their commonalities of subscription, thus restricting the per-node overhead of forwarding gossip messages to only the sum of streams in that group. However, this division can be a cumbersome approach; further, it would need reconfiguration as nodes' subscriptions change, and it has only limited advantages.

This paper adopts an approach that is simpler, general, and more effective – 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 thus to reduce the message overhead, while achieving the same reliability, scalability, and latency as canonical gossiping with independent streams.

Constraints in the gossip scheduling problem come from two sources – (1) the different gossip periods associated with individual gossip streams, and (2) how gossip streams are "related" to each other. The

former constraint is dictated by each individual stream's publishers' settings on how much overhead it wants imposed on nodes for that individual stream. This affects the amount of message "savings" that can be achieved by a particular schedule.

The latter constraint ((2)) is explained by the following example. Consider a system with four RSS feeds from CNN, BBC, VH1 and MTV. The two popular news sites CNN and BBC are similar, and are also likely to share a large number of common subscribers; thus, these two streams are "related". However, the set of subscribers for MTV is less related to either of these two streams, but it is related to the VH1 stream. Notice that this approach also allows other ways to "relate" streams to one another, beyond subscriber overlap or content type. The gossip layer would clearly benefit by piggybacking messages from the BBC and CNN streams together (similarly the VH1 and MTV streams), thus likely reaching interested subscribers with fewer messages. However, combining messages from the CNN stream with the MTV stream would offer little benefit.

Concretely and in a nutshell, in the gossip stream scheduling problem, each stream is represented by a vertex in a graph that we call the *semblance graph*. An edge exists between two vertices if the streams are related. The relation between two streams is binary: the streams are either related or not.[1] Since each node in this semblance graph is associated with a frequency (i.e., period) of gossiping, edges in the graph represent the potential savings of that piggyback. The final constraint in the problem is that no message that goes out on the network can have more than k constituent stream messages – this restriction limits message sizes from growing too large. k is an a priori fixed constant. Assuming that each message from each stream has the same size, $k =$ maximum gossip message size / stream message size. The gossip scheduling problem then becomes one of componentizing the semblance graph into subgraphs (partitions), each containing k or fewer nodes, in such a way that the the weight of the edges connecting partitions is minimized (in other words, we maximize the savings from piggybacking).

The contributions of this paper are then three-fold: (1) we show that the above gossip scheduling problem is NP-complete (by a reduction with a min-cut problem variant); (2) we propose two new heuristics for gossip scheduling; and (3) our semblance-graph experiments and distributed system simulations show a message reduction of 85% w.r.t. canonical gossiping, and only a mean 3.5% degradation compared to the optimal scheduler.

[1]The more general problem where the relation between streams may be a real number in $[0, 1]$ is currently beyond our scope. However, our NP-completeness result applies to the general problem too.

The rest of this paper is organized as follows. Section 2 discusses related work. In Section 3, we give an overview of gossip protocols and illustrate how the piggybacking problem can be converted into a semblance graph. Next, in Section 4, the NP-completeness proof is given. Section 5 describes our heuristic algorithms followed by our experimental evaluation in Section 6. We conclude in Section 7.

2 Related Work

Gossip protocols have their origins in the study of infectious diseases. [1]. Gossip protocols have been widely used in distributed systems including, amongst others, multicast [2, 3], database maintenance and replication [4, 5], group membership and state maintenance in DHTs [6] and in publish subscribe systems [7, 8, 9].

In [10] 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 [11] 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 [12] 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 [13] 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 [14]. 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. We show that finding the best schedule in the semblance graph is NP-complete by reduction with Min-Cut into Equal-Sized Subsets problem. This min-cut variant was was shown to be NP-complete in [15].

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

Figure 1: Pseudo code for receiving a new message in a typical gossip protocol.

3 Overview of Multi-stream Gossip Systems

Gossip protocols have been developed to address the scalability of message delivery systems while maintaining high reliability The properties of gossip protocols are based on the study of epidemics [1]. The spread of a message in the network can be thought of as the spread of a disease or of a rumor. When a node receives a message (hears the rumor) it forwards the message to a small subset of the total population (i.e., gossips). Messages are forwarded periodically according to the protocol period. A node gossips a message for a certain number of rounds which is also determined by the protocol. Targets for message forwarding are selected either at random or based on network topology. Analysis has shown that messages spread quickly and reliably to a large fraction of the hosts if the message is forwarded to $O(\log N)$ neighbors where N is the network size. Pseudo code for a typical gossip message forwarding is shown in Figure 1.

Nodes in publish-subscribe systems usually maintain a gossip *stream* for each separate data source, i.e. an RSS feed, which obeys the above gossip behavior. Each of the streams is handled independently including generating separate messages and maintaining separate stream state. The cost is the sum of messages from each stream, the memory overhead of storing messages for gossiping and the network overhead of maintaining membership views for each stream. However, streams within an application are often related to one another. One obvious relationship is the content of each stream, as in our RSS example, but it could also be the origin of the message or any thing. It is likely that related streams share an overlapping set of subscribers. Overhead can be reduced by taking advantage of overlap to schedule the gossiping of messages. However, a naïve approach would combine all streams that share overlapping subscriber sets. The approach is cumbersome though and requires constant reconfiguring of the combined groups. Instead messages from one stream can be "piggybacked" on top of another stream. The joint message is then gossiped as usual by the original stream. Upon receiving a message the next node uncouples the two messages and piggybacks

the messages according to its own schedule.

There are two primary constraints for this piggyback approach. The first is the period that each stream uses to send messages. Messages can be piggybacked only if there is a message currently available from both streams. The average period of each stream cannot be changed by either slowing it down or speeding it up to align with another stream. Therefore it is only possible to piggyback as frequently as the slower stream gossips messages. The second constraint is the maximum message size allowed in the system. The maximum message size is system wide and the same for each host in the network. Assuming that the message size of each stream is the same, then the system maximum message size limits the number of streams that can be piggybacked together. At each host a set of piggybacks from all of the subscribed streams is selected to maximize the number of piggybacks - the total message savings. As a basis for the decision is the "semblance graph" which describes the relationships and savings between streams.

3.1 Creating the Semblance Graph

To perform the piggyback selection process a logical graph is maintained at each host which describes the relationships between streams. We call the graph a *semblance graph* because it highlights the relationships and similarities between streams. Each stream is represented by a vertex in the graph. If two streams are related there is an edge in the graph connecting the two vertices. Currently we consider the relation between two streams to be binary. The model could be extended to allow *degrees* of relatedness between $[0, 1]$, but it such a model is beyond the scope of this paper. As shown in the following sections the less general case of binary relationships is nonetheless NP-complete.

The semblance graph also shows the savings for each potential piggyback. The reduction of messages possible from a particular piggyback depends on the periods that the two participating streams. The piggyback can only be performed if there are messages ready for gossip from both streams. Consider two streams with gossip periods of 2 and 5 seconds respectively. In a 10 second span the slower stream only gossips 2 messages while the faster stream gossips 5 messages. Of the 5 messages from the faster stream only 2 of them have messages available for piggyback. The savings of this particular piggyback is $\frac{2}{5}$. In general, for any two streams with periods T' and T'' where $T'' > T'$ the savings for piggyback p is:

$$savings_p = \frac{T'}{T''} \tag{1}$$

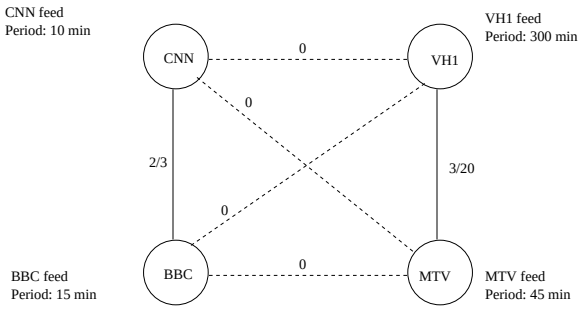


Figure 2: The semblance graph for a single node in a sample RSS aggregator application.

Every edge in the semblance graph is weighted with the savings for the represented piggyback as calculated in Equation 1. The additional parameter k is introduced in the semblance graph to specify the maximum allowed message size. For any combination of piggybacks then number of streams in the piggyback must be less than k.

Figure 2 shows a sample semblance graph from our earlier RSS example. The streams are related by site content seen by the edges between BBC and CNN as well as between MTV and VH1. The majority of RSS fees update within an hour [16]. Reasonable periods for CNN and BBC might be 10 minutes and 15 minutes. Thus the savings is $\frac{10}{15} = \frac{2}{3}$ which is the edge weight between CNN and BBC in Figure 2. The period for MTV is set to 120 minutes and VH1 300 minutes resulting in a savings of $\frac{3}{20}$ for that piggyback. Because of the simplicity of the example the k-constraint is not a factor.

The parameters used to create a semblance graph can be obtained in several ways. The easiest approach is to have a system administrator configure the semblance graph offline. It would be straightforward for a human to assign relationships between streams and stream period can be extracted from the protocol specification or stream source. However, in distributed applications this approach might be infeasible. The subscribed streams might change often at each host or the the stream parameters might fluctuate. A more practical approach is to have a process monitor the streams and dynamically update the semblance graph. As the application subscribes or leaves streams vertices in the semblance graph would be added or removed. Period can easily be estimated using a weighted moving average from recent samples of the stream. Defining the relationships between streams is more difficult, but could be done by comparing the gossip targets of each stream. We do not explore the details of such a monitor in this paper.

4 The Semblance Graph Problem

In this section we formally define The Semblance Graph Problem and prove that it is NP-complete. We first show that the problem is in NP and then make a reduction to the Minimum Cut into Two Equal-Sized Subsets Problem, proven to be NP-complete in [15].

Informally, the Semblance Graph Problem can be stated as divide 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 $1 \leq n \leq |V|$, positive integer k, positive integer W.

Property: There is a partition $V = A_1 \cup \ldots \cup A_n$ with $A_1 \cap \ldots \cap A_n = \emptyset$, $|A_i| \leq k$, $s_i \in A_i$, where $i = 1, \ldots, n$, and $|\{\{u, v\} \in E : u \in A_i, v \in A_j\}| \leq W$ where $i = 1, \ldots, n \wedge i = 1, \ldots, n \wedge i \neq j$.

Theorem. *The Semblance Graph Problem is NP-complete.*

Proof. First, it is easy to see that Semblance Graph Problem is in NP. One can generate a composition and verify in polynomial time whether it is a solution to the Semblance Graph Problem by showing that the k-constraint is satisfied.

Secondly, we need to show that given an algorithm to solve Semblance Graph it would be possible to solve Min-cut into Equal-Sized Subsets. The formal definition of the Min-cut into Two Equal-Sized Subsets Problem from [15] is:

Min-cut into Two Equal-Sized Subsets Problem

Input: Graph $G = (V, E)$, two distinguished nodes s and t, positive integer W.

Property: There is a partition $V = A \cup B$ with $A \cap B = \emptyset$, $|A| = |B|$, $s \in A$, $t \in B$, and $|\{\{u, v\} \in E : u \in A, v \in B\}| \leq W$.

Informally, the above stated problem is about dividing the graph into two equal-sized disjoint sets where the sum of the weights of crossing edges between the two sets is minimum.

Assume that we have an algorithm that solves the Semblance Graph Problem in polynomial time. The algorithm $Semblance(G', k) \rightarrow (A_0 ... A_n)$ takes as input a graph and a value for k and outputs a set of subgraphs of G'. Let G be the input graph for the Min-Cut into Equal-sized Subsets problem with edge

weights of the graph in the interval $[0, 1]$. We convert G into G' such that $w_{G'}(e) = w_G(e) + H$ where $H >> 1$. Non-edges which have 0 weights in G have weight H in G'. Thus G' is a fully connected graph. Now we will prove that if $Semblance(G', n/2)$ returns an optimal solution it will also be the optimal solution to Min-Cut into Equal-size Subsets for graph G.

Claim 1: $Semblance(G', k) = (A, B)$ where $|A| = |B| = n/2$. (i.e. The algorithm for the Semblance Graph generates two equal sized subsets)

We will prove this claim by contradiction. Assume that the solution generated by $Semblance(G', n/2)$ does not have the form (A, B) where $|A| = |B| = n/2$. Then it has to be of the form $(A_1, A_2, ..., A_t), t \geq 3$ and each $|A_i| \leq n/2$. Without loss of generality, assume that $|A_1| = x$, where $0 < x \leq n/2$.

If $|A_1| = n/2$ then we are done. In this case, the min-cut (i.e the sum of weights crossing the subsets) includes at least all the edges going out of A_1. If there were more than 1 subsets in the remaining $n/2$ nodes, then the composition would not be optimal since collapsing those subsets would result in a smaller min-cut value which is a contradiction. Thus, if $|A_1| = n/2$ there must be exactly two subsets in the optimal solution.

For the case that $0 < |A_1| < n/2$ we have a second claim about the nodes $n - x$.

Claim 2: If a graph has m vertices where $n/2 \leq m < n$, then the min-cut satisfying the '$\leq n/2$' constraint has a value $\geq (n/2)(m - n/2)H$. Assume that there is a partition of m nodes into a group that has $y \leq n/2$ and some other group(s) that in total have $(m - y) \geq n/2$ nodes. In this case, the size of the min cut would be

$$\geq y(m - y)H + n/2(m - y - n/2)H \text{ if } m - y \geq n/2; \tag{2}$$

$$\geq y(m - y)H \text{ if } m - y < n/2. \tag{3}$$

This function has a quadratic shape with the minimum values at its end points. For $y = 1$ the min cut would be at least $n/2(m - 1 - n/2)H + (m - 1)H$. For $y = n/2$, the min cut would be at least $(n/2)(m - n/2)H$. These two values satisfy claim 2.

Now we look at the value of the total min-cut for the graph given subsets $(A_1, A_2, ..., A_t)$. In such a case the total min cut for the whole graph would be $\geq (n/2)(n - x - n/2)H + x(n - x)H$ which is equal to $(n/2)^2 + nx/2 - x^2$ where $x = |A_1|$. This is also a quadratic function and with minimum values at the end points. For $x = 1$ the min-cut value would be $[(n/2)^2 + n/2 - 1]H$ and for $x = n/2 - 1$ it is

$[(n/2)^2 + n/2 - 1]H$. Thus the overall minimum cut comes when $x = n/2 - 1$.

Now, we will show that even in the worst case, a partition with more than two subsets will result in a worse (i.e. larger) cut value. With two subsets the cut would have at most a value of $(n/2)^2(H + 1)$ in the worst case where each crossing edge has the maximum value of $H + 1$. In any partition that has more than two subsets the cut cost would be at least $(n/2)^2 H + (n/2 - 1)H$ as calculated above.

By adjusting the value of H such that $(n/2 - 1)H \geq (n/2)^2\ Semblance(G, k)$ will always chose the two-subset partition and thus the optimal solution for Min-Cut. So for large enough values of H $Semblance$ would always solve Min-Cut into Two Equal-size Subsets, which is known to be NP-complete. Therefore the Semblance Graph Problem in NP-complete. □

5 New Heuristic Algorithms

In this section we present two new heuristic algorithms to solve the Semblance Graph problem. Our algorithms are greedy algorithms based on two minimum spanning tree (MST) algorithms. Our problem is not the same as the MST problem. The k-constraint and objective of maximizing savings make our problem different than the MST problem. We developed two new algorithms based on the the well-know Prim and Kruskal MST algorithms.

5.1 Greedy-Prim Algorithm

Prim's MST algorithm builds the tree by starting with a randomly selected vertex in the graph in a tree. Each iteration a vertex is added to the tree by selecting the best incoming edge (lowest weight) into the tree [17]. 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 they reach 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 next best edge not contained in any existing subgraph. The pseudo-code for our Greedy-Prim algorithm is given in Figure 3.

F_i is a forest consisting of nodes $(N \geq i \geq 0)$ $F_{i,m}$ is the current set of nodes in the forest F_i $F_{i,s}$ is the set of neighbors that this forest F_i currently has

procedure GREEDY-PRIM()

1: **for** each forest F_i **do**
2: F_m ={the node that has the max edge among remaining}, F_s ={}
3: To F_s, add the neighbors of newly added node of set F_m. This node should not be 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: Apply same procedures until all nodes are seen
12: **end for**

Figure 3: Pseudo-code for the Greedy-Prim heuristic algorithm.

5.2 Greedy-Kruskal Algorithm

We also developed a greedy algorithm based on Kruskal's MST algorithm [17]. Kruskal's algorithm starts with each vertex of the graph in a tree by itself. At each iteration the best edge (lowest cost) from the entire graph is examined. If the edge connects two trees previously unconnected it is added to the MST. Otherwise the edge is discarded. The algorithm continues until every vertex is included in the MST.

Our Greedy-Kruskal algorithm also examines edges on a global basis. The algorithm starts with the best edge (highest weight). The algorithm checks to make sure the edge does not combine two trees that together would violate the k-constraint. If the combined tree is within 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 shows the pseudo-code of Greedy-Kruskal.

For the sake of comparison we also 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. The running time of this algorithm is $O(2^M)$ in the worst case where M is the number of edges present in the graph.

F_j is the forest consisting of nodes
procedure GREEDY-KRUSKAL()
1: Take the unselected edge E_i of maximum value.
2: **if** E_i causes the size of any forest F_j to exceed k **then**
3: throw it out.
4: **else** { add E_i to the forest of it's node }
5: add E_i to the forest of it's node.
6: **if** it's nodes are part of two separate forests **then**
7: merge the two forests.
8: **end if**
9: **if** E_i's endpoints are not members in any forest **then**
10: create a new forest with E_i.
11: **end if**
12: **end if**
13: Mark E_i as selected
14: Repeat this process until every edge is processed.

Figure 4: Pseudo-code for the Greedy-Kruskal heuristic algorithm.

6 Evaluation

6.1 Comparison of the Algorithms

To evaluate the relative effectiveness of the proposed algorithms, Greedy-Kruskal and Greedy-Prim, we first performed a graph based study. We compared the two algorithms 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 semblance graphs by varying one of five parameters: the number of streams (vertices) in the semblance graph, the degree of relatedness for each stream (number of edges per vertex), 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. Table 1 summarizes all of the variables. Semblance graphs were generated for each combination of values for each parameter. Edge weights were assigned using the periods from each edge endpoint according to equation 1.

We processed each graph using all three algorithms (Greedy-Prim, Greedy-Kruskal and brute-force). The algorithms computed the "best" set of piggybacks for the particular semblance graph which was scored by the total message savings that could be achieved in that configuration. We compared the relative performance of each algorithm to the optimal score as computed by the brute force algorithm. For semblance graphs which had 20 streams and relatedness degree of 10 the brute force algorithm took too long to com-

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

Table 1: The range of values used to generate sample semblance graphs.

plete, so we omitted the results from those graphs for all algorithms.

For 39.8% of the graphs the Greedy-Kruskal algorithm out performed 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.

Figures 5 show the behavior of the different algorithms as each of the five parameters is varied. The greedy algorithms show a strong decline in performance as the number of vertices (streams) and edges (relatedness) increases. As the homogeneity of stream periods approaches the end points (0 and 1) the greedy algorithms degrade in performance. The strong effect of period distribution is also shown by varying the range of period values. As the range increases the distribution of edge weights is uneven, with a few edges having relatively high weight. In those scenarios the greedy algorithms perform better because they aggressively seek out the heavy edges. When the edge weights are more balanced the greedy algorithms struggle more. Changing the k-constraint had little effect on the performance of the greedy algorithms.

6.2 Distributed System Simulator

We implemented our Greedy-Kruskal semblance graph algorithm in a simulated gossip application. The application mimics the behavior of a pub-sub system in which application nodes subscribe to several streams of information and publish to a few streams. Each application node maintains a membership list(i.e. view)

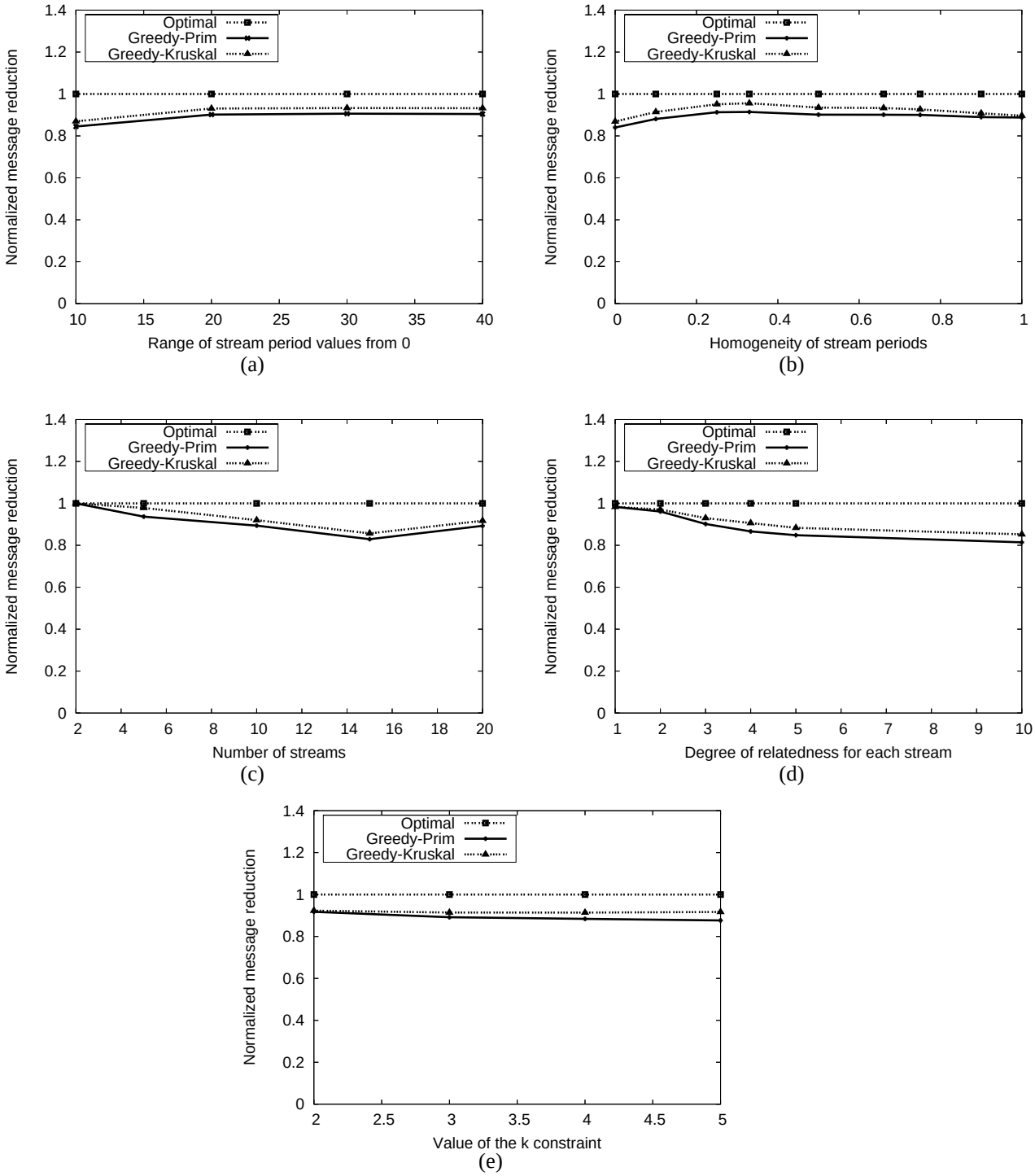


Figure 5: Relative performance of the different algorithms when varying (a) period value, (b) period homogeneity, (c) number of streams, (d) degree of relatedness and (e) k-constraint.

of the other nodes. In this section we use the terms node and host interchangeably. In our implementation, the view size is equal to the total number of nodes in the system. When a new message arrives at the node, either via the network or is generated by the application itself, it starts gossiping about that message. During each round of the execution the epidemic layer sends the message to a subset of b(gossip fan-out) members from the view. Theoretically, a gossip protocol would have enough reliability as it uses $b = log(n)$. In most of our experiments we use a small constant value such as 5 for gossip fan-out value. In order to focus on the performance of our algorithms we ensure perfect delivery of messages, i.e. the packet drop rate is 0%.

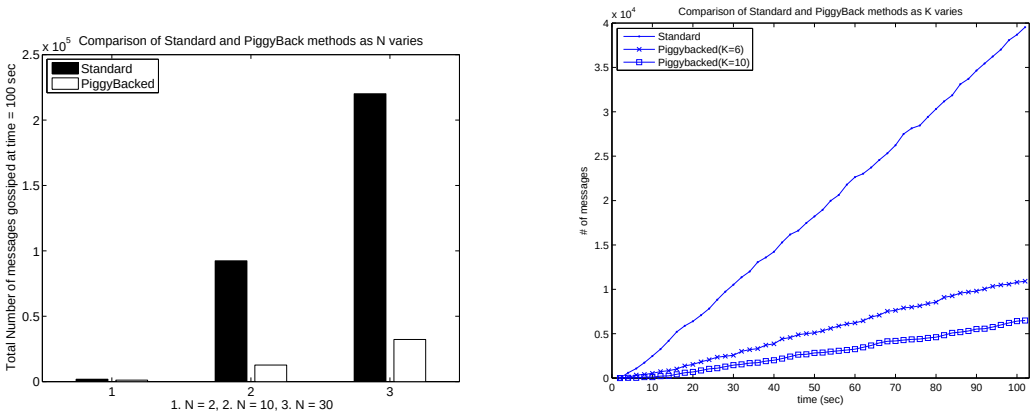
Our semblance gossip replaces the existing gossip layer in the above application. Each application is originally configured with a semblance graph containing each of the streams the application subscribes to. The graph could easily be updated by monitoring application $SUBSCRIBE$ messages to dynamically add new nodes to the graph.

Whenever the semblance graph changes a new set of compositions is calculated, using the Greedy-Kruskal algorithm from 5. For each connected component in the solution a single gossip stream is created which monitors the buffers from all of the application streams specified in that component. The gossip stream has the period of the application stream with the lowest period in the component. At the end of each period, if there are available gossip messages belonging to higher period streams we piggyback all of these messages into one. This is how our scheduling mechanism works. In other words, each round the buffers from each application stream are examined for messages. If a message is available in any buffer it is concatenated onto the existing message buffer. The gossip stream then behaves the same as the original gossip layer, selecting b destinations to send the composed message. Upon receiving a new message the semblance gossip layer examines the message and decomposes the composed contents into the original individual messages. Each message is then delivered to the application independently.

To evaluate the message savings benefit of piggybacking we compared the message count when using semblance gossip compared to canonical gossip. We modified several parameters in our simulator to change the conditions under which each algorithm performed. Table 2 shows the parameters used in the experiment and gives some typical values that we used during our experiments. For each experiment we varied one parameter and kept the others constant. A summary of parameters is: number of nodes, number of randomly selected streams at each node, k-constraint, density of randomly generated semblance graphs, number of rounds that a message is gossiped (TTL), range of the periods of gossiping streams, and message generation period for a node. In all of the experiments, we used the Greedy-Kruskal algorithm to solve the semblance

Parameter Name	Abbreviation	Some Typical Values
Number of nodes(hosts) in the system	N	5 - 50
Number of randomly selected streams each node subscribes to	M	10
k constraint of the Semblance Graph	k	5
Density of randomly generated semblance graphs	density	50%
Number of rounds that a message is gossiped by a gossiper	TTL	5
Range of the periods of gossiping streams	range	[0,8] sec
Message generation period for a host	p	1 sec

Table 2: Parameters, their abbreviations and default values.



(a) (b)

Figure 6: (a) Comparison of number of messages generated as number of nodes in network changes (b) and as k changes.

graphs.

1. In the first experiment we examined the effect the number of network nodes had on number of messages generated in the system. We compared the piggyback gossip against the standard gossip for different network sizes. The results are shown in in Figure 6(a). As shown in the figure the number of messages is reduce to almost 85% of the standard gossip as n increases.

2. In the second experiment we show how the number of messages changes as the k-constraint varies. Figure 6(b) shows that as k gets larger, the number of messages sent in the system gets smaller.

3. The third experiment(i.e. Figure 7) shows how the number of messages change as number of rounds that a message is gossiped(TTL) changes. The TTL has little effect on the ratio of messages between

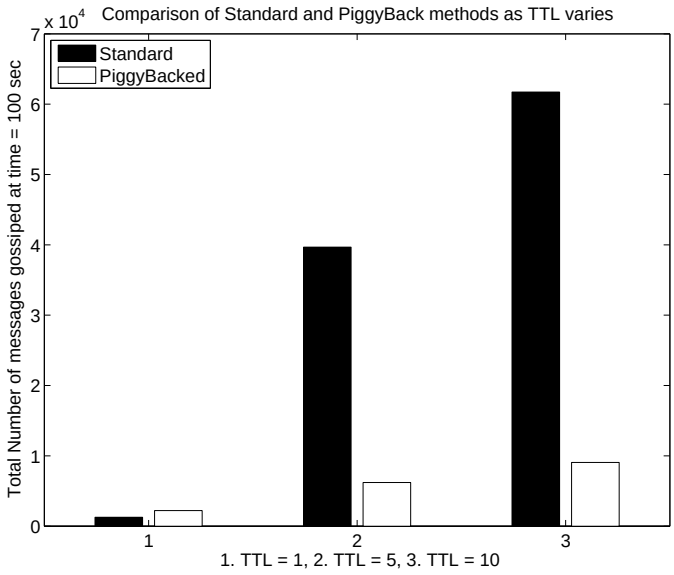


Figure 7: Comparison of number of messages generated as the number of rounds that a message is gossiped changes.

piggyback and standard.

4. The fourth experiment shown in Figure 8 shows how the change in density of the Semblance Graph at each node effects message savings. As seen in the figure, densities of 10% and 25% result in nearly the same message savings. As density increases past 50% the number of messages starts to decrease. Because the semblance graph is more connected there are more possible piggybacks and thus more message savings.

5. Next we examined the effect of stream period on message savings. We assigned periods to each stream using both a uniform and a Zipf distribution from the interval of 0 to 50. The results are shown in Figure 9.

6. Finally we showed the message savings as the average number of subscribed streams increased as each node (Figure 10). The figure shows the increase in message count when hosts all subscribe to the same number of streams (in this case 20) compared to the case when hosts subscribe to a variable number of streams (0 to 20 with a uniform distribution).

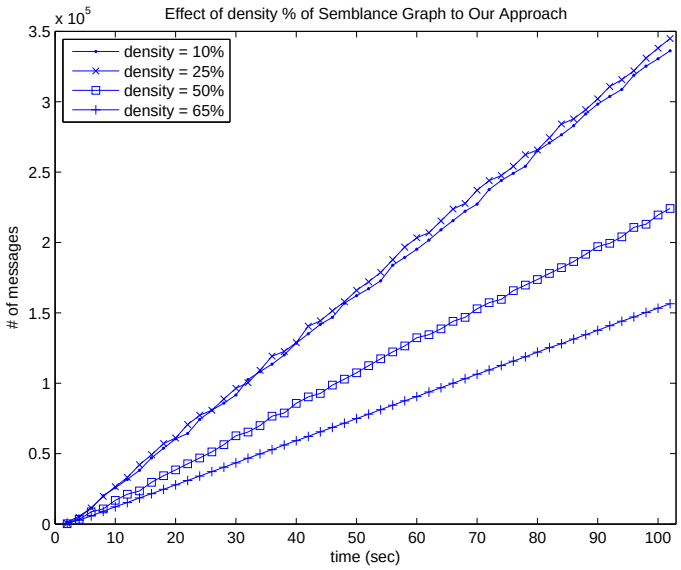


Figure 8: Effect of density of The Semblance Graph on our piggybacking technique. N = 50 was used in this experiment.

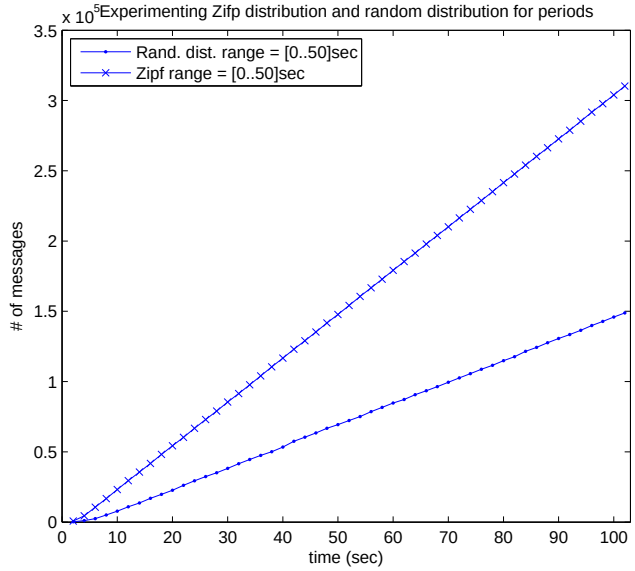


Figure 9: Uniform and Zipf distribution using M = 20, N = 50 and the rest default parameters for this experiment.

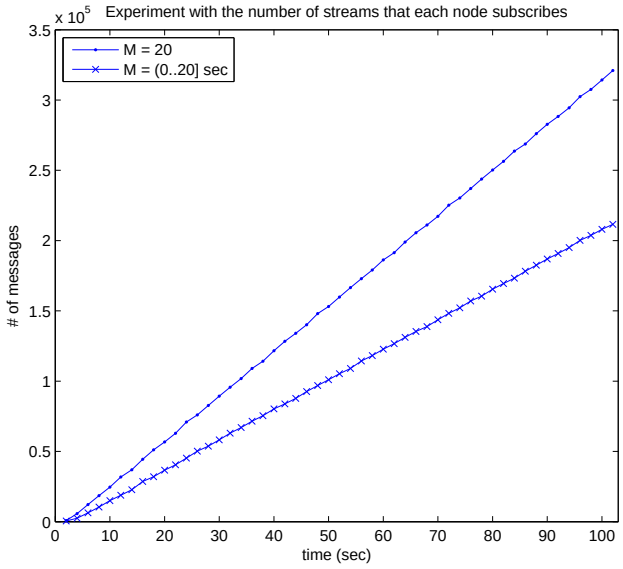


Figure 10: Comparison between M = 20 and random distribution of M for each node in [0,20] interval.

7 Conclusion

In this paper we argued that system and message overhead in multi-stream gossip based publish-subscribe systems can be reduced through effective stream scheduling. We proposed a *piggyback* approach to send messages from streams with overlapping subscriber sets together and thus reduce both the message count and system resource demands. Our "semblance graph" gives a representation of the relationship between different streams and the potential benefit from each piggyback. Although choosing the best combination of piggybacks is NP-complete as we showed in the paper, we developed several heuristic algorithms which achieve near optimality (within 3.5%) and showed through simulation how those algorithms have a real impact on publish-subscribe systems by reducing message count by 85% in some cases. A greedy approach that always tries to add the piggyback with highest savings without violating the constraint on message size performs the best. As part of our on-going work we are looking at other heuristic algorithms from the area of graph partitioning and also considering some non-greedy approaches.

At the core of our approach is the semblance graph. In this paper we only briefly discussed appropriate mechanisms for creation and maintenance of the graph. We are currently investigating the exact details needed to implement piggybacking in existing gossip-based middlewares. Our approach should be easy to deploy with little impact on current systems. The benefit is tangible as shown by our simulation results.

Although we have focused on gossip based systems in this paper we feel our approach would be applicable to any multi-stream publish-subscribe system.

8 Acknowledgments

We would like to thank our friend Viraj Kumar for his guidance in the NP-complete proof of the Semblance Graph Problem.

References

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

[2] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," *ACM Trans. Comput. Syst.*, vol. 17, no. 2, pp. 41–88, 1999.

[3] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec, "Lightweight probabilistic broadcast," *ACM Trans. Comput. Syst.*, vol. 21, no. 4, pp. 341–374, 2003.

[4] 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*. New York, NY, USA: ACM Press, 1987, pp. 1–12.

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

[6] 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.

[7] P. Costa, M. Migliavacca, G. P. Picco, and G. Cugola, "Epidemic algorithms for reliable content-based publish-subscribe: An evaluation," in *Proc. of the 24th Int. Conf. on Distributed Computing System s (ICDCS)*, 2003, pp. 552–561.

[8] R. V. Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Trans. Comput. Syst.*, vol. 21, no. 2, pp. 164–206, 2003.

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

[10] 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)*, 2001, p. 186.

[11] 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.

[12] 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)*, Oct 2002, pp. 180–189.

[13] 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.

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

[15] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer, "Some simplified np-complete graph problems." *Theor. Comput. Sci.*, vol. 1, no. 3, pp. 237–267, 1976.

[16] H. Liu, V. Ramasubramanian, and E. G. Sirer, "Client behavior and feed characteristics of RSS, a publish-subscribe system for web micronews," in *Internet Measurement Conference 2005 (IMC'05)*, October 2005.

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