JetStream: Achieving Predictable Gossip Dissemination by Leveraging Social Network Principles[*]

Jay A. Patel, Indranil Gupta, and Noshir Contractor
University of Illinois at Urbana-Champaign
E-mail: {jaypatel, indy}@cs.uiuc.edu, nosh@uiuc.edu

Abstract

Gossip protocols provide probabilistic reliability and scalability, but their inherent randomness may lead to high variation in number of messages that are received at different nodes. This paper presents techniques that leverage simple social network principles enabling nodes to select gossip targets intelligently. The simple heuristics presented in the paper achieve a more uniform message overhead at each node, lowering the system-wide gossip traffic, while simultaneously reducing the latency of gossip spread (by up to 25%). We experimentally compare our system, called JetStream, against canonical gossip as well as gossip on the Chord overlay. Intuitively, JetStream seeks to make gossip spread more deterministic and predictable, while still inheriting its scale and reliability. JetStream also provides an added benefit by reducing network bandwidth utilization with a low sustained rate of gossip injection.

1. Introduction

The advent of Web feeds such as RSS and ATOM, as well as streaming Web content, has made large-group multicast an important problem. Several systems have been designed for large-scale multicast including FeedTree [17], BitTorrent [7], Bullet [12], etc. However, we believe that gossiping offers the right combinations of probabilistic scalability and reliability to solve these new problems.

One of the primary obstacles limiting the use of gossip in these new settings is its inherent randomness. The random selection of gossip targets at different nodes leads to high variation in incoming message overhead – some nodes may receive 30 copies of the same gossip message while others receive a small number (which is more desirable). We need an intelligent strategy for gossip target selection to reduce the random overhead in a predictable manner. At the same time, we would like the scalability and reliability of the random gossip protocol to be inherited.

This paper presents *JetStream*, a gossip protocol that uses *social network principles* to achieve intelligent gossip target selection. We wish to clarify to the reader that this paper *does not leverage* social network links (e.g., from social network systems like Orkut [1] etc.) to improve epidemics. Instead, we borrow purely algorithmic ideas from social networks research to develop JetStream. Surprisingly, the combination of simple social network heuristics and gossip leads to a more uniform message overhead, reduced latency of gossip spread, and lower system-wide traffic.

The two social network principles used in JetStream are *reciprocity* and *structural holes*. Both are (unconsciously) used by humans while developing their social relationships. Reciprocity theory states that individuals tend to return the affection of others. Structural holes theory states that individuals tend to establish relationships that maximize their "connectivity".

We experimentally compare JetStream's performance to that of canonical random gossip, as well as to gossip spread over the Chord overlay. Our experiments show that JetStream lowers message overhead at nodes by half, reduces gossip latency by up to 25%, and lowers the overall gossip traffic by half. We wish to point out that this paper *does not* present algorithms for topologically-aware gossip, adaptive gossip, or semantic gossip. Although social network rules can be added on to these mechanisms, they are beyond the scope of this paper.

The rest of the paper is organized as follows: we discuss the basic properties of gossip protocols in Section 2. In Section 3, we introduce mathematical heuristics for the theories of reciprocity and structural holes. Section 4 discusses our algorithm and implementation. We provide simulation results in Section 5. And finally, Section 6 concludes.

Related Work: Besides the bulk of work on scalable and reliable multicast (e.g., SRM [9]), several publish-subscribe systems have been developed recently for RSS feeds (e.g., FeedTree [17]), and other content, e.g., Bullet [12]. Several "flat" (canonical) gossiping protocols have been proposed in literature, including Bimodal Multicast [4], work

[*]This research was supported in part by NSF CAREER grant CNS-0448246, and in part by NSF ITR grant CMS-0427089.

by Kermarrec et al [11], work by Kouznetsov et al [13], to name a few. Variants of these gossip protocols that are topologically-aware (e.g., [10]) or semantically-aware (e.g., [16]) have also been proposed. Gossip has been used to design several distributed protocols such as membership mechanisms, e.g., [8] and [19].

Some work has been done on combining social network principles with peer to peer systems. However, none of the work in this area has explicitly drawn mathematical ideas from social networks. Bernstein et al [2] present a policy for selecting peers that lead to better utilization of global resources. Marti et al [14] present a DHT (distributed hash table) that utilizes the friend-of-friend social principle to improve trust among peers.

Monge and Contractor [15] and Wasserman et al [20] are the two primary resources that describe various social network principles and theories using mathematical heuristics.

2. Gossip Networks

In this Section, we describe the flat gossip protocol, and characterize it through simulated experiments. Our network model is composed of n nodes (labeled v_0 through v_{n-1}). Each node v_i may communicate with (i.e., send a message to) any other node v_j. The cost incurred to send a message between any two nodes is constant.

2.1. Flat Gossip

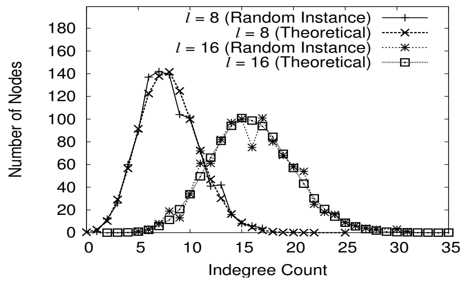
A simple gossip protocol may be used to propagate messages (for example, updates in a publish-subscribe system) within a network. The simplest gossip protocol (named *flat gossip*) is as follows: starting with the message originator, each node chooses c random *forwarding targets* (uniformly from the network *membership list*) during each time interval, to send a message m. The nodes repeat this process for d time intervals after initially receiving a message.[1] A relatively inexpensive optimization is for node v_i to not select node v_j as a forwarding target if node v_j is known to have already received message m (i.e., if node v_j has previously forwarded the message m to node v_i).

Previous research [11] has shown that a gossip protocol with $c \cdot d = O(\log n)$ allows a message to be spread to all nodes with very high probability. To be precise, if each node forwards the message to $\log n + k$ nodes, then the probability that all nodes get the message converges to $e^{-e^{-k}}$.

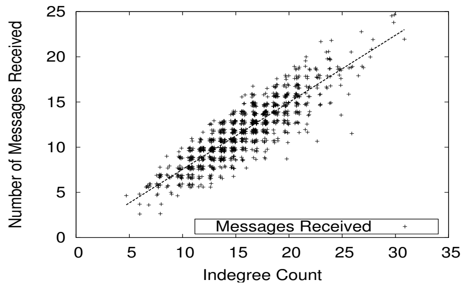
2.2. Overlay Networks

Network protocols generally have a non-zero cost for establishing an initial connection between two nodes v_i and v_j. Hence, it is beneficial for nodes to select the same target candidates for different gossip messages. Doing so amortizes the connection establishment cost over many messages. Stated differently, a node and its *target set* remain connected. The sum of these connections forms an *overlay network*.

Properties of Random Overlay Networks: An overlay where each node has a target set of size $l = O(\log n)$ (selected randomly from all other $n - 1$ nodes) has an interesting property: as shown in Figure 1(a), in a network of size $n = 1,000$, the distribution of node *indegree* (i.e., the number of nodes that have selected a certain node as a forwarding target) follows a binomial distribution.



(a) The indegree distribution of a random overlay (with constant outdegree l) follows a binomial distribution.



(b) There is a tight linear correlation between the number of messages received by a node and its indegree. The data points are plotted up to $\pm(0.25, 0.25)$ for clarity.

Figure 1. Random overlays have an uneven indegree distribution, resulting in unfair gossip workloads.

The variance in the node indegree results in uneven workloads during gossip propagation. A simulated experiment demonstrates the uneven workload property of gossip protocols. Figure 1(b) displays a tight linear correlation between a node's indegree and the total number of messages received, in a network of size $n = 1,000$ and the target set size of $l = 16$. Hence, improving the fairness of indegree distribution may reduce workload variance.

To improve the workload distribution and dissemination speed, we use social network principles. The basics are discussed in the next Section.

[1] In this paper, we use a constant value of $c = 1$, since having $c > 1$ is akin to reducing the time interval duration by $1/c$.

3. Social Networks

A multitude of social network theories attempt to explain the logic behind relationships. In this Section, we focus on two theories that intuitively improve the performance of gossip protocols by specifying a criteria for target selection.

3.1. Reciprocity

Social exchange theory explains dyadic interactions on the basis of resources each actor has to offer. Reciprocity theory [5] states that there is a higher tendency for mutual interactions between members of a social network. Intuitively, reciprocity improves indegree distribution and reduces the total number of messages propagated by flat gossip. If each node has a reciprocal relationship with its forwarding targets, each node's indegree will be exactly equal to its outdegree, l. Furthermore, during gossip, a node need not forward the message back to any node from whom it already has received the message. In essence, this should reduce the number of messages in half. The utility value of node v_i (based on reciprocity) can be calculated as follows:

$$Utility_{Reciprocity}(v_i) = \sum_{j=0}^{n-1} x_{ij}x_{ji} \tag{1}$$

The variable x_{ij} is a boolean value representing the current relationship between nodes v_i and v_j (i.e., if v_j is a target of v_i). A node's utility value improves (only) for each reciprocated relationship. To improve reciprocity, our goal is to maximize this utility value for all nodes.

3.2. Structural Holes

An intriguing social theory based on self-interest is the structural holes theory [6]. The structural holes theory recognizes that there are entrepreneurial actors who actively position themselves in an advantageous positions within a social network.

A structural hole is the position in a network that provides a direct advantage to a network member. An actor in a structural hole connects two disconnected actors. In a competitive world, an individual that fills such a hole draws an advantage from their positioning, both by collecting a higher volume and better quality of information from their contacts, as well as by exercising greater control over them. The utility value (as prescribed by the structural holes theory) of node v_i can be calculated as follows:

$$Utility_{Str.Holes}(v_i) = \sum_{j=0}^{n-1} x_{ij} \sum_{k=0}^{n-1} x_{ik}(1 - x_{jk}) \tag{2}$$

A node's utility value improves only when it forms relationships with nodes that do not have a relationship with each other. Nodes changing forwarding targets based on structural holes theory may improve the dissemination speed of gossip messages.

4. The JetStream Approach

The JetStream algorithm seeks to improve a node's utility. Each node alters its target set strictly based on the local evaluation of the utility function – without considering the effect of the decision on the overall health of the system (i.e., global utility). Our experiments find that this greedy behavior leads the system to converge near the globally optimal configuration.

Utility Function: Our intuition is that the combination of the two aforementioned theories (reciprocity [5] and structural holes [6]) should improve the performance of gossip networks. We derive a net utility function for a node v_i by combining the utility functions for reciprocity and structural holes, as follows:

$$Utility(v_i) = \sum_{j=0}^{n-1} x_{ij}x_{ji} \sum_{k=0}^{n-1} x_{ik}x_{ki}(1 - x_{jk})(1 - x_{kj}) \tag{3}$$

Given l as the number of gossip targets, the maximum attainable utility value is $\frac{l\cdot(l-1)}{2}$ (discounting repetition). Nodes that attain the maximum utility are called *optimal nodes*, whereas, others are labeled as *suboptimal nodes*.

A utility function can shape the overlay without imposing hard constraints and/or deterministic rule sets on target selection. For example, a deterministic rule for structural holes allows a node v_i to select another node v_j as a target if and only if v_j is not a target for any of v_i's other targets. While a deterministic rule such as this may allow nodes to construct their target set iteratively – a newly joined node may not be able to construct its target set quickly. Using a utility function, a new node may simply pick its initial target set at random, and then gradually alter its target set. We describe the process by which nodes improve their target set using utilitarian calculations next.

Generic Algorithm Details: At any given time, each node maintains a membership list (consisting of nodes known to exist in the network), in addition to its gossip target set. Our approach is to have each node continuously attempt to alter its gossip target in order to increase its local utility value. This is achieved by having each node v_i execute a *replacement procedure* once every *update period*. During the replacement procedure, the node v_i (which, for convenience, we shall call the *deciding node*) tentatively changes at most one of its gossip targets, seeking to improve its local utility value.

Initially, the deciding node calculates its current utility value. This value is marked as the *currently highest utility*. Next, the deciding node selects one of its targets (at random) as a potential eviction victim – the *delink candidate*. The node then creates an empty *replacement candidates list*, and adds the delink candidate to this list.

Next, the deciding node iterates through each element v_j in its membership list that is not a gossip target, and calculates its utility value by replacing the delink candidate with the node v_j in its target set. Three cases may arise: (1) If the new utility is lower than the currently highest utility, v_j is removed from consideration. (2) If the utility is higher, the replacement candidates list is set to the singleton $\{v_j\}$, and the value of the currently highest utility is updated. (3) If the utility is the same, then v_j is added to the replacement candidates list.

After the entire membership list has been looked at, a random node from the replacement candidates list is chosen as a replacement for the delink candidate. In essence, the above mechanism implements an evolving *utilitarian overlay*. Next, in Section 4.1, we analyze experimental results from the *global* implementation of JetStream. In Section 4.2, we integrate a membership list protocol in the *local* implementation of JetStream.

4.1. Global Implementation

To study the basic emergent properties of the described algorithm, we implement a system with the following assumptions:

- The network is non-dynamic, i.e., participating nodes do not leave, rejoin, or crash.
- Each node knows the presence of all other nodes, i.e., the membership list is complete and consistent.
- Each node knows the current target set of all other nodes. Stated differently, when a node updates its target set, the update is propagated to all other nodes in the network instantaneously.

The simple implementation has the following important parameters: the update period is 1, i.e., the replacement procedure is executed every time interval. It should be noted that while the implementation is simple, the computation complexity of a single node's replacement procedure is $O(n \cdot l^2)$. Furthermore, each node maintains a memory overhead of $O(n \cdot l)$, to keep track of all other target sets.

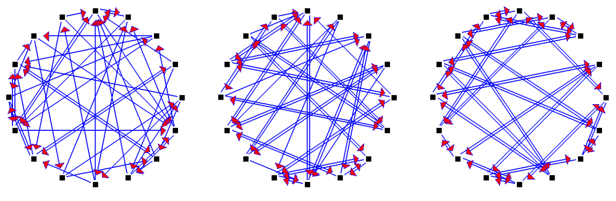
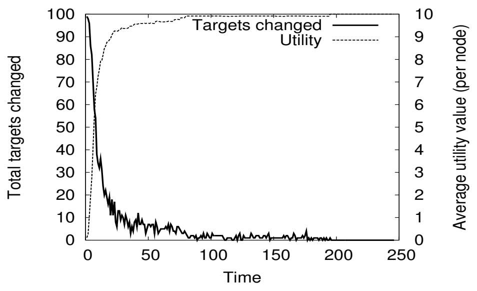
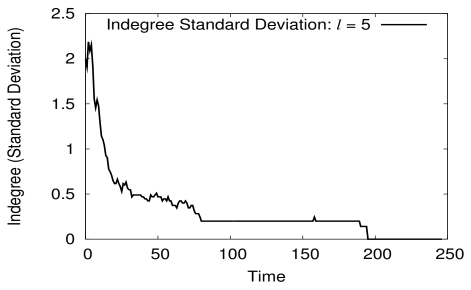

(a) $t = 0$ (b) $t = 10$ (c) $t = 90$

Figure 2. Evolution of a JetStream overlay.

Results: Figure 2 charts the progression of a random overlay (with $n = 16$ and $l = 3$) to an utilitarian overlay with time. The individual nodes select targets that yield higher utility, ultimately resulting in a graph with exactly $\frac{n \cdot l}{2}$ bi-directional edges. Initially, the nodes improve their utility value rapidly, soon reaching a point of diminishing returns (Figure 3(a) shows the phenomenon in a network with $n = 100$ and $l = 5$). However, the algorithm continuously forces nodes to select targets that yield better utility until the network stabilizes, and no more changes to the overlay are required. Here, the network converges when each node reaches its maximum utility value.



(a) With JetStream, nodes in a random overlay converge towards the optimal utility, by selecting better targets.



(b) The converged utilitarian overlay has the same outdegree as the indegree (for each node). The standard deviation of the indegree reduces to zero.

Figure 3. The overlay converges to a set of optimal targets over time.

Reciprocity enforces a degree equality on the utilitarian overlay: each node attains the same indegree value (recall that an initial random network overlay has a fixed constant outdegree l – however, the indegree follows a binomial distribution). Figure 3(b) shows the progression of the standard deviation of node indegree distribution for the overlay. Ultimately, the standard deviation reduces to zero, implying no variation in distribution of node indegree. At this point the network no longer exhibits any tendency to change. However, this characteristic is only exhibited when $n \cdot l$ is even. The network does not stabilize if $n \cdot l$ is odd because of the last remaining "dangling" (i.e., non-reciprocating) target pointer. An easy way to solve this issue is to always use an even value for l.

4.2. Localized Implementation

The global implementation requires perfect knowledge of the network state. Moreover, each node takes $O(n \cdot l^2)$ time per replacement round to find a better candidate for its target set. Furthermore, it requires $O(n \cdot l)$ memory storage at each node. Clearly, this type of computational overhead is not suitable for large scale networks. Hence, we implement a version that requires only partial knowledge of the network state, meanwhile, lowering the computation (and memory) overhead. For the localized implementation, we make much more realistic assumptions about the network. Specifically:

- Each node can communicate with all other nodes, using a reliable underlying messaging mechanism. All messages are delivered within a constant time delay. We model a time delay t_{delay} of 1 time interval.
- A node does not know the consistent state of the network, it must actively probe the network to gather updated information.

We introduce many restrictions in the realistic implementation: the biggest restriction begins with the introduction of the *candidate set*, whose maximum size is set to s (where $s > l$). A node is only required to maintain extended information (i.e., target sets) of the nodes in the candidate set (unlike the global implementation, which kept track of the target set for the entire membership list). An important note is that the candidate set is a superset of a node's target set. This is an obvious requirement because a node can only calculate its utility value correctly if it has access to the target sets of each of its targets. However, limiting the candidate set to s entries reduces the memory overhead per node to $O(s \cdot l)$. For the most compact implementation[2], each node can be represented in 6 bytes by its IP address and port. For a network with $n = 5,000$ and $l = 10$, a candidate set of size $s = 20$ requires as little as 1,200 bytes of memory overhead to maintain this information.

To keep reciprocating nodes updated, a node notifies all affected nodes after a change in its target set (i.e., updates are propagated to the new target set as well as the delinked node). This mechanism ensures that nodes remain up-to-date with changes that affect their utility value. The update period is also varied on a per-node basis to stagger the execution of the replacement procedure. This is required so that nodes do not continually update their target set with stale information (i.e., due to the delay incurred in receiving update messages). For example, with a staggered update period of $t_{stagger} = 5$ time periods, a node picks its next update time randomly between $(t_{rtt}, t_{stagger}]$. The t_{rtt} is the network round trip time (i.e., $t_{rtt} = 2 \cdot t_{delay}$). Waiting for a minimum of t_{rtt} time periods before the next update period allows a node to gather the acknowledgment reply from the newly added node (the reply also contains the latest target set of the newly added node).

[2]We are referring to a simple matrix here. While it would simplify space requirements, it would increase computational requirements. More complicated data structures can be used to improve speed.

Finally, nodes in the candidate set need to be replaced on a regular basis. This allows the network to reach optimal utility, be resilient to crash-stop failures, handle churn, etc. A node in the candidate set is replaced if it sends no message for t_{out} consecutive time intervals. To prevent being timed out, a node send an update message to all its target set every t_{out} time intervals. If a node is timed out from the candidate set, a random node is chosen from the node's membership list to replace it.

A node maintains a membership list consisting of all previously discovered nodes (gathered through update messages). In our implementation, a node's membership list is $O(n)$, however, it can be bounded for large-scale networks. Maintaining a consistent membership list is not required in JetStream.

4.3. Analysis

In this Section, we analyze the overhead required to maintain a JetStream overlay in a static network. Based on this, we provide a gossip injection rate threshold beyond which JetStream utilizes less network bandwidth than a random overlay. Please refer to Table 1 for a review of the notations used to describe the network (and its characteristics).

Table 1. Notations describing the network.

Notation	Meaning
n	Number of active nodes in the network
$n_{optimal}$	Number of optimal nodes
n_{subopt}	Number of suboptimal nodes
l	Size of the target set
s	Size of the candidate set
$t_{stagger}$	The maximum stagger time between update periods
t_{rtt}	Network round trip time
t_{update}	The expected time between update periods
t_{out}	Refresh interval to keep a target entry from timing out
p	Probability of a suboptimal node finding a better target
B	The background traffic overhead (number of packets)
I	The gossip injection rate
w_B	The size of each target set information packet
w_I	The size of each gossip message

Theorem 1: For a sustained gossip injection rate of $I \geq I_{thresh} = \frac{2B \cdot w_B}{n \cdot l \cdot w_I}$, a maximum utility JetStream overlay uses fewer messages than random gossiping (where B is the background traffic overhead of the JetStream overlay, w_B is the average packet size of a target set information packet, and w_I is the size of each gossip message).

Proof. Recall that an optimal node is one that has achieved the highest local utility value, whereas a suboptimal node is still seeking to achieve the highest utility. As the update period is staggered using randomness, its value is expected to be $t_{update} = \frac{t_{rtt}+(t_{stagger}-1)}{2}$. Recall that a suboptimal node sends update packets to its new target set as well as the delinked node on each change in its target set. Given p as the probability of a suboptimal node finding a replacement target, the traffic generated by all suboptimal nodes is approximately $n_{subopt} \cdot p \cdot (l + 1)$ packets every t_{update} time periods. To preserve the reciprocal link, every optimal node sends a refresh packet to its target set. This results in $n_{optimal} \cdot l$ packets every t_{out} time periods. As another consequence of the timeout mechanism, a nodes removes old elements (exclusive of its current target set) from its candidate set after t_{out} time periods. This results in an additional $2n \cdot (s - l)$ packets being sent every t_{out} time periods (each timeout replacement generates two network calls: one for the deciding node to inquire the new candidate set node regarding its target set, and the other for the newly selected candidate set node to respond with its target set). Hence, the total traffic incurred due to the timeout mechanism is $\frac{2n\cdot(s-l)+n_{optimal}\cdot l}{t_{out}}$ packets each time interval. Therefore, the theoretical traffic due to the JetStream protocol is:

$$B = \frac{n_{subopt} \cdot p \cdot (l+1)}{t_{update}} + \frac{2n \cdot (s-l) + n_{optimal} \cdot l)}{t_{out}} \tag{4}$$

A gossip message over a random overlay is forwarded approximately $n \cdot l$ times. In contrast, as discussed in Section 3.1, a gossip message over a JetStream overlay is forwarded only half as many times. Hence, JetStream can provide a benefit over random overlay when $I \geq I_{thresh}$:

$$\begin{aligned} I \cdot w_I \cdot n \cdot l &\geq I \cdot w_I \cdot \frac{n \cdot l}{2} + B \cdot w_B \\ I_{thresh} &= \frac{2B \cdot w_B}{n \cdot l \cdot w_I} \end{aligned}$$

□

5. Results

We present the results of JetStream's localized implementation through synchronous simulation. The simulations were run with $n = 5,000$ and $l = 10$. The simulations were run for either 3,600 or 7,200 time intervals.

5.1. Overlay Characteristics

We study the effect of the various parameters on the progression of the overlay in a static network. The findings are summarized below:

- The localized implementation works almost as well as the global implementation: the network stabilizes with an average utility close to the maximal utility (see Figure 4(a)).
- A small candidate set s is sufficient. For example with $s = O(\log n) = 2l$, approximately 90% of nodes become optimal quickly (see Figure 4(a)). Increasing s provides limited benefits, at the expense of added computational overhead.
- The value of $t_{stagger}$ should be as low as possible. A low value allows suboptimal nodes to choose better targets. Our implementation defaults $t_{stagger}$ to 5 time periods. Figure 4(b) shows that a suboptimal node locates a better target almost every update period using only a candidate set of size $s = 2l$. Specifically, the network stabilizes with $p \geq 0.5$. A value of $p \geq 0.5$ signifies that a deciding node locates at least one suitable replacement node with the same (or higher) utility.
- The value of t_{out} affects the background traffic. Using a t_{out} value of 120 time periods, Figure 4(c) shows that the JetStream overlay stabilizes with an average node sending one packet approximately every 3 time periods.

On Joins and Leaves: While a realistic implementation with a static network preserves the emergent behavior present in the global implementation, experiments involving a large number of sudden joins and crash-stop failures (i.e, leaves) may help uncover other interesting properties. We perform two experiments in a network with initial size of $n = 5,000$: one where 50% of the active nodes leave the network, and another one where an additional 50% nodes join the network. Both the events happen in two separate experiments, at the halfway point of the run.

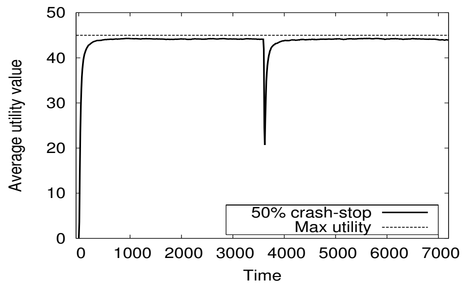
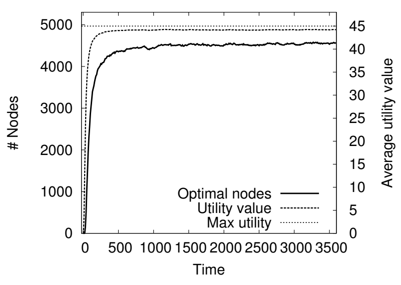


Figure 5. JetStream is resilient to a massive number of sudden leaves.

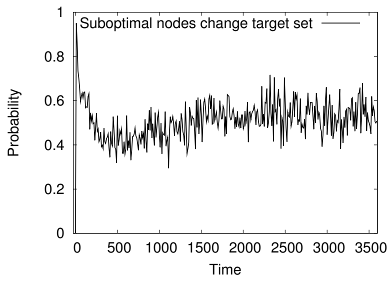
Figure 5 shows the network's quick reconvergence to the near-optimal average utility value after the crash-stop failures. Similarly, joining nodes integrate just as comfortably in the network (not shown here), by providing a greater choice in target selection.

5.2. JetStream with Flat Gossip

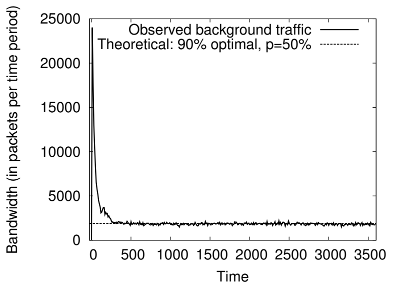
For the next experiment, we propagate a message with flat gossip using three different target selection policies: a random overlay, a Chord overlay, and the JetStream overlay. In the Chord [18] overlay, each node has m finger pointers (in a m-bit key space) as described in the original DHT paper. For the experiment, the nodes are labeled with 13-bit



(a) Nodes achieving optimal utility



(b) Probability of a suboptimal node finding a replacement



(c) Total network activity

Figure 4. Evolution of JetStream overlay for network size $n = 5,000$ and target set size $l = 10$.

keys (as $2^{12} \leq 5000 \leq 2^{13}$), with each node having 13 finger pointers. We select 10 of these 13 pointers randomly as our forwarding targets. The JetStream overlay is simply the random overlay evolved over 3600 time intervals.

The total message overhead imposed by JetStream is lower, and far less varied (see Figure 6(a)) than either the random overlay or Chord. The maximum overhead for a JetStream node was 16 messages, compared to 33 for a node in the random overlay, and 57 for a node in the Chord overlay. Stated differently, JetStream imposes a "fairer" workload to the network participants. Next, we run the same simulation (using different underlying overlays) 10 different times. In Figure 6(b), the total number of messages sent by the flat gossip protocol is between 40% and 50% fewer with the JetStream overlay than the Chord or random overlay. Furthermore, Figure 6(c) shows that gossip propagates approximately 25% faster with JetStream (as measured by the time taken for the message to reach the last node). JetStream was able to deliver the message to all 5,000 nodes in all 10 simulation runs, Chord and random overlay failed to deliver the message to one node during one run each.

5.3. Continuous Gossip

In the next experiment, we evaluate the performance of a static JetStream overlay under continuous gossip injections. In Section 4.3, we showed that when $I \geq I_{thresh}$, the bandwidth used by JetStream is lower than the bandwidth used by a random overlay. Assuming $w_I = w_B$ (i.e., the packet sizes are ignored), we perform the next experimental comparison between the bandwidth utilized by a random overlay and a JetStream overlay. Based on our implementation parameters, $I_{thresh} = 0.076$ based on Theorem 1 (i.e., a new gossip every 13 time periods for the entire network, or a new gossip every 65,000 time periods per node). Our experiment shows that the random overlay actually consumes approximately 4% less bandwidth than JetStream at the calculated I_{thresh}: roughly 3800 packets/second (random overlay) vs. 3650 packets/second (JetStream). This is due to the fact that the localized implementation of the overlay does not attained maximum utility after 3,600 time periods (as assumed by our analysis). However, using only a slightly more aggressive injection rate (i.e, a new gossip packet every 10 time intervals), a reduction in bandwidth utilization is realized by JetStream. This experiment shows that JetStream can provide a substantial reduction in bandwidth utilization when the network expects a gossip injection rate higher than I_{thresh}.

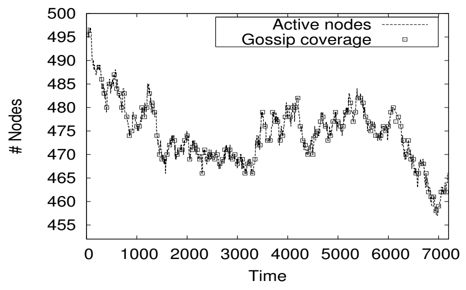
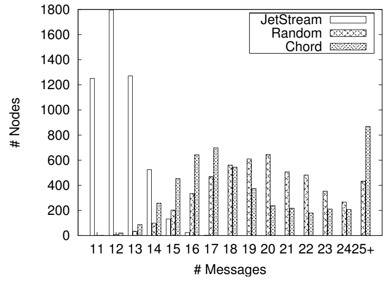


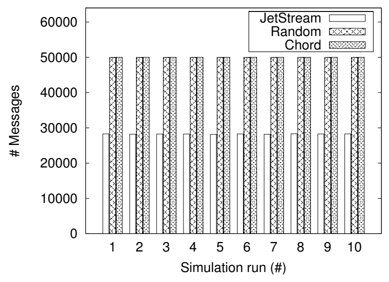
Figure 7. A continuous injection of gossip messages in a dynamic network.

Churn: To evaluate the effect of churn on JetStream, we used the Overnet traces [3] collected from a deployed P2P network. The traces contained availability information of 2400 hosts for 7 days, at a granularity of every 20 minutes (we scaled 1 second = 1 time period for this simulation). Our experiments utilize 2 hours of the traces. While the traces monitored 2400 hosts, the actual number of active hosts fluctuated between 450 and 500 at any given time. A node joins or leaves the network during a random interval spanning the 20 minutes in which the data was collected. A new node is introduced to the system using a bootstrap node provides the joining node with a list of nodes to form an initial candidate set. The joining node then participates in the network as any other node. A departing node is mimicked by a crash-stop failure (i.e., no notification).

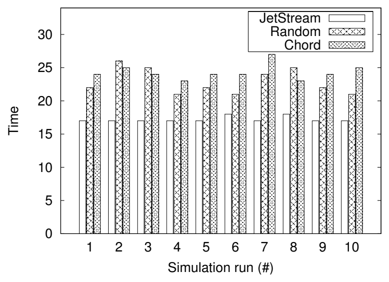
Next, we inject a continuous stream of gossip messages into a JetStream, maintaining $I = 0.2$. A node was chosen



(a) The total number of messages sent and received by each node is fewer with JetStream, and much less varied.



(b) The total number of messages transmitted by JetStream is approximately 40% to 50% fewer than random overlay.



(c) JetStream is approximately 25% faster in spreading a gossip message compared to a random overlay.

Figure 6. The gossip workload is more uniform, and the message propagation faster.

at random to be the originator of each new gossip message. The targets for the gossip were chosen based on the current target list of the JetStream overlay (i.e., the targets possibly changed with time). Figure 7 shows the coverage of a gossip message is close to 100% of active nodes in the system. In fact, some gossip messages were received by more nodes than present during gossip origin time (because new nodes entered the system during the gossip spread).

6. Conclusion

Gossip protocols provide probabilistic reliability and scalability. However, in Section 2, we showed that their inherent randomness leads to high variation in the number of messages received at different nodes. Next, in Section 3 and Section 4, we presented techniques that leverage simple social network principles to select gossip targets intelligently. In Section 5, we showed that these simple heuristics achieve a more uniform message overhead at each node, while lowering the system-wide traffic by up to 50%. We experimentally compared JetStream against canonical gossip, as well as gossip on the Chord overlay. Our results demonstrate that JetStream helps gossip spread in a more deterministic and predictable manner with reduced latencies of up to 25%, while still inheriting scale and reliability. Lastly, we showed that JetStream also utilizes less bandwidth than a random overlay if the continuous gossip injection rate exceeds a low threshold ($I_{thresh} = 0.076$ for a network of size $n = 5000$).

References

[1] Orkut. `http://www.orkut.com/`.

[2] D. S. Bernstein, Z. Feng, B. N. Levine, and S. Zilberstein. Adaptive peer selection. In *Proc. of IPTPS*, 2003.

[3] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Proc. of IPTPS*, 2003.

[4] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, 17(2):41–88, 1999.

[5] D. J. Brass. A social network perspective on human resources management. *Res. in Personnel and Human Resources Management*, 13:39–79, 1995.

[6] R. Burt. *Structural Holes: The Social Structure of Competition*. Harvard University Press, 1992.

[7] B. Cohen. Incentives build robustness in BitTorrent. In *Proc. of Workshop on Economics of Peer-to-Peer Systems*, 2003.

[8] A. Das, I. Gupta, and A. Motivala. SWIM: Scalable weakly-consistent infection-style process group membership protocol. In *Proc. of DSN*, pages 303–312, 2002.

[9] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Trans. on Networking*, 5(6):784–803, 1997.

[10] I. Gupta, A.-M. Kermarrec, and A. J. Ganesh. Efficient epidemic-style protocols for reliable and scalable multicast. In *Proc. of SRDS*, 2002.

[11] A.-M. Kermarrec, L. Massouli, and A. J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Trans. on Parallel and Distributed Systems*, 14(3), 2003.

[12] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proc. of ACM SOSP*, pages 282–297, 2003.

[13] P. Kouznetsov, R. Guerraoui, S. B. Handurukande, and A.-M. Kermarrec. Reducing noise in gossip-based reliable broadcast. In *Proc. of SRDS*, 2001.

[14] S. Marti, P. Ganesan, and H. Garcia-Molina. DHT routing using social links. In *Proc. of IPTPS*, pages 100–111, 2004.

[15] P. R. Monge and N. S. Contractor. *Theories of Communication Networks*. Oxford University Press, 2003.

[16] J. Pereira, L. Rodrigues, R. Oliviera, and A.-M. Kermarrec. Probabilistic semantically reliable multicast. In *Proc. of IEEE NCA*, 2001.

[17] D. Sandler, A. Mislove, A. Post, and P. Druschel. Feedtree: Sharing web micronews with peer-to-peer event notification. In *Proc. of IPTPS*, 2005.

[18] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM*, 2001.

[19] R. van Renesse, Y. Minsky, and M. Hayde. A gossip-style failure detection service. In *Proc. of Middleware*, 1998.

[20] S. Wasserman, K. Faust, D. Iacobucci, and M. Granovetter. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.