JetStream: Achieving Predictable Gossip Dissemination by Leveraging Social Networks Principles

Jay A. Patel, Indranil Gupta and Noshir Contractor*

Abstract

Gossip protocols provide probabilistic reliability and scalability, but their inherent randomness may lead to high variation in (received) message overheads at different nodes. This paper presents techniques that leverages simple social networks principles to enable nodes to select gossip targets intelligently. These simple heuristics achieve a more uniform message overhead at each node, while at the same time reducing the latency of gossip spread (by up to 25%) and also lowering the system-wide gossip network traffic. 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 overall network bandwidth utilization if there is a low amount of sustained 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 [18], BitTorrent [6], Bullet [13], etc. However, we believe that the technique of gossiping offers the right combinations of probabilistic scalability and reliability to solve these new problems.

*Dept. of Computer Science and Speech Communication, University of Illinois at Urbana-Champaign, 201 N. Goodwin Ave., Urbana, IL 61801. Corresponding author's e-mail is jaypatel@uiuc.edu

One of the primary obstacles to 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). In a sense, what we need is an intelligent strategy for gossip target selection so that this random overhead is reduced to being more predictable. At the same time, we would like the scalability and reliability of the random gossip protocol to be inherited (and improved).

This paper presents *JetStream*, a gossip protocol that uses *social networks 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 etc.) to improve epidemics. Instead, JetStream borrows purely algorithmic ideas from the area of social networks research in order to design a new protocol. Surprisingly, the combination of simple social networks rules with gossip leads to more uniform message overhead, reduced latency of gossip spread, and a lower system-wide gossip network traffic.

The two social network rules used in JetStream are called *reciprocity* and *structural holes*. Both these rules are (unconsciously) used by human beings while developing their social links - reciprocity means that an individual does (or does not) establish a link with someone who points a link at her. Structural holes means that the individual attempts to establish links to others who do not already link to each other - this rule maximizes the "expertise" within reach of the individual. For instance, a researcher in the Computer Science department who knows someone in Economics is more likely to prefer a Physicist as a new friend than another Economist.

After describing the core social networks rules in more detail, we describe the structure of the JetStream system. We then experimentally compare its 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 networks rules can be added on to the mechanisms just mentioned, these are beyond the scope of the current paper.

The rest of the paper is organized as follows: we discuss the basics properties of gossip protocols

in section 2. Next, in section 3, we introduce mathematical heuristics for the theories of reciprocity and structure holes. Section 4 discusses our algorithm and implementation. We provide simulation results for our system 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 [18]), and other content, e.g., Bullet [13]. Several "flat" (canonical) gossiping protocols have been proposed in literature, including Bimodal Multicast [3], work by Kermarrec et al [12], work by Kouznetsov et al [14], to name a few. Variants of these gossip protocols that are topologically-aware (e.g., [11]) or semantically-aware (e.g., [17]) have also been proposed. Gossip has been used to design several distributed protocols such as membership mechanisms, e.g., [20] and [8].

Some work has been done on combining social networks principles with peer to peer systems, however none of the work in this area has explicitly drawn mathematical ideas from social networks. Two of these works are as follows. Bernstein et al [1] present a policy for selecting peers that lead to better utilization of global resources. Marti et al [15] present a DHT (distributed hash table) that utilizes the friend-of-friend social principle to improve trust among peers.

Contractor et al [7] discuss social in a workplace, and our paper borrows some of the mathematical formulation from that paper. Monge and Contractor's book [16] and Wasserman et al [21] are two resources we used for social network principles and theories.

2 Gossip Networks

In this section, we describe a type of a gossip protocol, and characterize it through simulated experiments. For the rest of the paper, we assume that our network model is composed of a set 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 (i.e., the same).

2.1 Flat Gossip

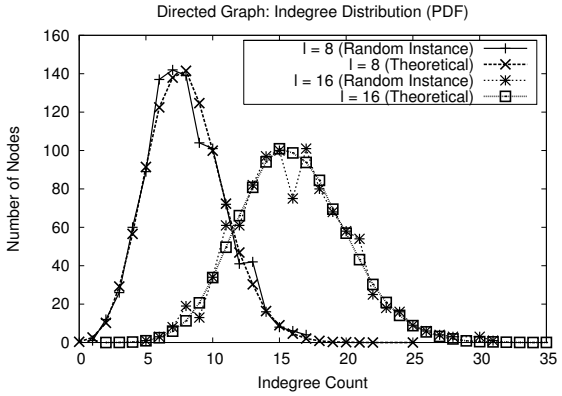
A simple gossip protocol may be used to propagate messages (for example, updates in a publish-subscribe system) within a network. The simplest gossip protocol (known here on as *flat gossip*) can be implemented as follows: starting with the message originator, each node chooses c random *forwarding targets* (from the network *membership list*) during each time interval, to transmit an *outstanding message m*. A message is considered outstanding (on a per node basis) for d time intervals after being (initially) received. A relatively inexpensive optimization (i.e., trading network transactions for local memory) requires a node v_i to unconsider a node v_j as a a candidate for forwarding target if it is known to have already receive the message m previously (for example, if the node v_j has already forwarded the message m to node v_i).

Previous research [10] has shown that a gossip protocol with $c * d = O(\log n)$ allows a message to be propagated to all nodes with high probability. To be precise, if each node communicates with $\log n + k$ nodes on average, then the probability that everyone gets the message converges to $e^{-e^{-k}}$.

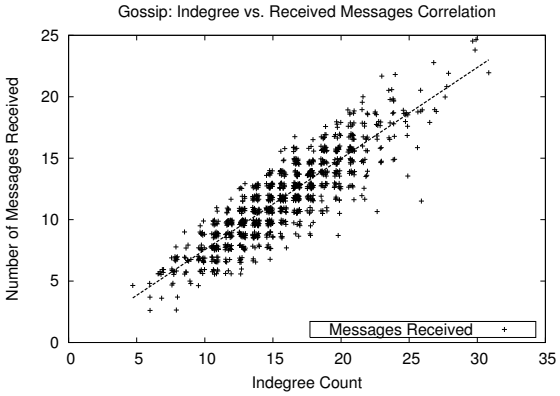
2.2 Overlay Networks

Network protocols generally have a non-zero cost for establishing an initial connection between two nodes v_i and v_j. Hence, it is more efficient for a node to pick the same set of target candidates for different gossip messages. Doing so amortizes the connection establishment cost over many messages. If each node maintains a *target set* of size $l = O(\log n)$, we can guarantee (with high probability) that a message will reach all nodes quickly. Stated differently, with a static network membership, a node v_i and any of its target v_j remain connected for the duration of the network. The aggregate of all these connected nodes form a persistent *overlay network*.

Properties of Random Overlay Networks: As described before, a random overlay network forms the basis of a flat gossip protocol. Choosing l nodes at random from the membership list of n nodes leads to a peculiar property: as shown in Figure 1(a), the distribution of node *indegree* (i.e., the number of nodes that have a given node as a member of its target set) follows a binomial distribution. Such a wide variance in the distribution of node indegree leads to uneven workloads during gossip propagation.



(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 overlay had a constant outdegree $l = 16$. The data points are plotted $\pm random(0.25)$ for clarity.**

Figure 1: **Random overlay networks (here, $n = 1000$) inherently lead to uneven distribution of indegree (with fixed cardinality of the target set, i.e., l is constant). This effect primarily explains the uneven workload distribution of gossip networks.**

We perform a simulated experiment to describe the uneven workload property of gossip protocols. We use a network of size $n = 1000$ for the following experiments. In the experiment, we use a constant value of $c = 1$, as utilizing a value of $c > 1$ is akin to reducing the duration of the time interval by $1/c$. Please refer to Section 5 for further details regarding our simulation methodology.

Figure 1(b) displays a tight linear correlation between a node's indegree and and the total number of messages received (recall that the number of messages sent by each node is a constant d). Improving upon the fairness of indegree distribution may allow a gossip protocol to impose fairer workload requirements per node.

In this section, we demonstrated that while gossip protocols are simple and efficient, they do impose an uneven workload on the individual nodes. To improve the workload distribution and dissemination speed, we discuss social network theories in the next section. Further, we discuss how social network theories may help improve the performance of gossip protocols by forming better overlays (i.e., allowing nodes to choose better forwarding targets).

3 Social Networks

There are a multitude of social network theories that formulate the logic behind human relationships. In this section, we focus on two theories that may (intuitively) lead to improvement of gossip network overlays by specifying a strict methodology for target selection.

In this section, we assume that the network starts with a random overlay, with each node v_i having a target set of size l. The variable x_{ij} is a boolean value representing the current relationship between nodes v_i and v_j. A value of 1 indicates a that v_j is a target of v_i, meanwhile 0 signifies the opposite (i.e., v_j is not a target of v_i). The relationships are not reciprocated by default. In other words, the value of x_{ij} may differ from x_{ji}.

3.1 Reciprocity

Social exchange theory explains dyadic interactions on the basis of resources each actor has to offer. If actor A perceives actor B as potentially having valuable resources, A will initiate a link with B. Reciprocity theory [4] states that there is a higher tendency for mutual interactions between members a social network. In other words, the reciprocity effect states that if actor A communicates with actor B, then B will also (or, is more likely to) communicate with A.

The utility value (as prescribed by the reciprocity theory) of a node v_i can be calculated with the following utility function:

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

We believe that reciprocity will reduce the total number of messages being propagated by the gossip protocol. In an overlay mutated by reciprocity, each node would ideally reciprocate the relationship with its target, thus cutting down the total number of messages sent per node (recall that nodes do not consider targets with whom they have previously exchanged a message with as candidates for forwarding agents).

3.2 Structural Holes

Of the many social theories of self-interest, one that is intriguing and stands out is the structural holes theory [5]. 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. Stated differently, if an actor A communicates in a reciprocal manner with actor B and with C, but actor B and C do not communicate with each other, then actor A is filling a structural hole by bridging the communication between actors B and C.

The utility value (as prescribed by the structural holes theory) of a node v_i can be calculated with the following utility function:

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

We believe that the structural holes theory would allow an overlay to evolve towards a structure that will support faster dissemination of gossip messages.

4 The JetStream Approach

We design a simple utilitarian algorithm that selects a target set (for each node) that provides maximum utility. They key idea is that each node decides to alter its target set strictly based on the evaluation of the utility function at the local node – without considering the effect of the decision to the overall health of the system (i.e., global utility). Our experiments find that this greedy behavior leads to convergence quite close to the globally optimal utility.

Utility Function: Our intuition is that the combination of the two aforementioned theories (reciprocity and structural holes) should lead to an improvement in performance of gossip networks. We can derive a net utility function for a node v_i based on the sum of the utility functions for reciprocity and structural holes described in the previous section.

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

Notice that the aforementioned function calculates the utility value only at node v_i. Each node calculates it utility value based on only the following function, which does not require the complete state of the network.

The key intuition behind using a utility function to shape the overlay is to refrain from imposing hard constraints or deterministic properties on target selection. For example, a deterministic rule for structural holes theory can enforce that a node may select another node as a target i.f.f. the candidate node is not an element of any of its current target's target set. While this deterministic rule may allow a node to construct its target set in an iterative manner – a newly joined node may not be able to quickly build its target set. Using a utility function, a new node may simply pick its initial target set at random and slowly evolve its target set based on the utility function. We describe the process a node takes to move towards the highest utility next.

A node strives to attain maximum utility. The above utility function is calculated for each node v_i. If the maximum number of gossip targets at a node is limited to l, then the maximum utility value can only be $\frac{l\cdot(l-1)}{2}$. A node that attains this maximum local utility value is defined to be an *optimal node*, whereas, the rest of the nodes are known as *unoptimal nodes*.

Generic Algorithm Details: Recall that at any point of time, each node maintains a list of stable gossip targets, and in addition, an extended membership list (consisting of nodes known to exist in the network). The basic idea in our approach is to have each node continuously but slowly attempt to change its gossip target set, so as to keep increasing its local utility value. This is achieved by having each node v_i execute a set of actions at least once every *update period* time units, independent of other nodes. During any of its update periods, the node v_i (which, for convenience, we shall call the *deciding node*) tentatively changes

at most one of its gossip targets, in the hope that its local utility $Utility(v_i)$ will increase.

During the start of the update period, a node calculates its current utility value based on all locally available network information. The initial utility value is marked as the *currently highest utility*. Next, the deciding node picks one of its targets as a potential eviction victim – this target is chosen at random, and we call this the *delink candidate*. The node creates an empty *replacement candidates list*, and adds the delink candidate to this list.

Next, the deciding node runs through each element v_j in its membership list that is not a gossip target, and calculates the utility derived by replacing the delink candidate in the gossip target set with the node v_j. Three cases can arise: (1) If the new utility is lower than that obtained by similar replacement with anyone in the replace candidates list, v_j is dropped from consideration and the replacement candidates list is left unchanged. (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. If the replacement node is different from the delink candidate, the target set is updated by substituting the replacement node for the delink candidate. If the replacement node and delink candidate are one and the same, nothing happens.

In essence, the above mechanism implements a *utilitarian overlay* with respect to gossip target selection. However, maintaining and discovering the membership list has not yet been discussed. We do so in Section 4.2, but before that, Section 4.1 describes some experimental results from a *global* implementation of JetStream, where we assume an external membership protocol that provides this list as a service running on each node.

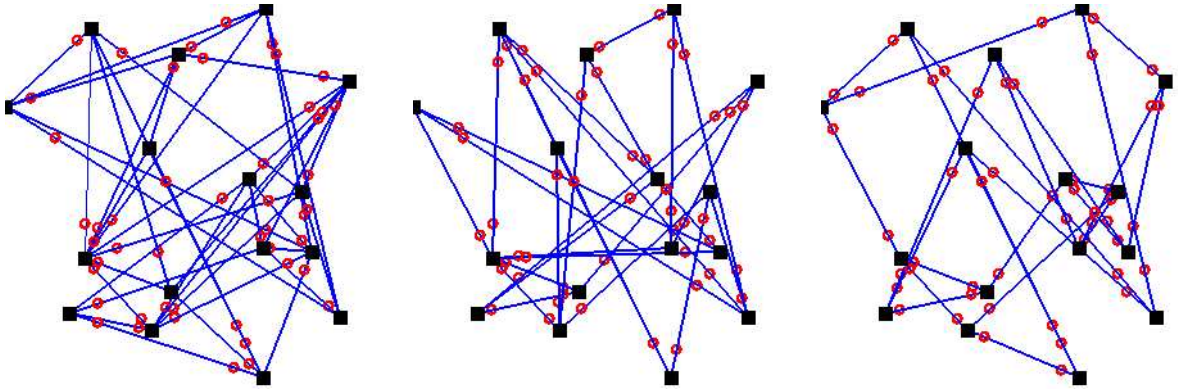
4.1 Global Implementation of the Utilitarian Overlay

To study the basic emergent properties of the described algorithm, we design a simple implementation by assuming that each node can maintain arbitrary amounts of information (specifically, for the membership list). This allows all nodes to know the currently consistent state of the network. In essence, our network

model assumes the following:

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

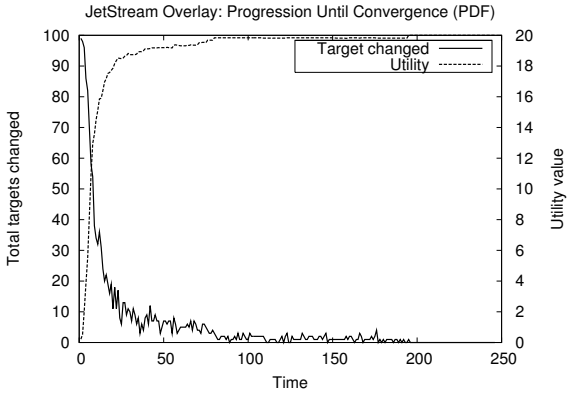
The simple implementation has the following important parameters: the update period is one, i.e., the replacement procedure is run during every time interval on each node. It should be noted that while the implementation is simple, the computation complexity of a single node's replacement procedure is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



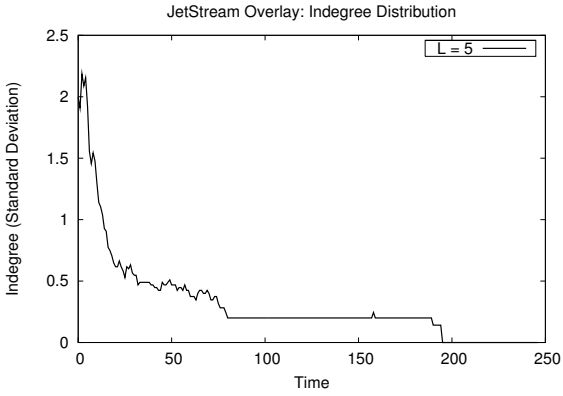
(a) $t = 0$ (b) $t = 10$ (c) $t = 90$

Figure 2: **Progression of a $n = 16$ (with $l = 3$) JetStream overlay. The node closer to the red circle on an edge represents the target.**

Results: Figure 2 charts the progression of the JetStream overlay with time. Starting with a random overlay, the individual nodes select targets that yield higher utility, ultimately resulting in an graph with exactly $\frac{n \cdot l}{2}$ edges. Initially, the nodes improve their utility value rapidly, soon reaching a point of diminishing returns (see Fig 3(a)). However, the algorithm forces nodes to pick targets which yield better utility until the network stabilizes, and no more changes to the overlay are required.



(a) **JetStream starts with random overlay that converges to an optimal utilitarian overlay, by choosing targets that yield a better utility.**



(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. The network here is of size $n = 100$ and target set size $l = 5$.**

Reciprocity enforces a degree equality on the utilitarian overlay: each node has the same indegree value (recall that our 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 the overlay (recall that the average indegree is constant, equaling the outdegree l). At the convergence point of the overlay network, the standard deviation becomes 0, indicating 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 would not stabilize if $n \cdot l$ is odd because there would always be one "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 of the Utilitarian Overlay

The global algorithm takes $O(n \cdot l^2)$ time to locate a better candidate for its target set during each time interval. Furthermore, it requires $O(n^2)$ memory storage at each node. Clearly, this type of computational and memory overhead is not suitable for large scale networks. Hence we implement a realistic version of our algorithm that requires lower memory and computation overhead – while still maintaining the properties of the global algorithm.

In this model, we make much more stringent assumptions about the network. Specifically:

- Each node can communicate with all other nodes, using a reliable underlying messaging mechanism. The messages are delivered within a uniform, 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 idealistic 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 cuts down the memory overhead per node to merely $O(s \cdot l)$. For the most compact implementation[1], each node can be represented by a 6-byte IP address and port combination, i.e., for a network with $n = 5000$ and $l = 10$, a candidate set of size $s = 20$ requires as little as 1200 bytes of memory overhead to maintain this information.

To keep reciprocating nodes updated on target set changes, whenever a node updates its target set, it sends the updated information to all affected note, i.e., the current target set as well as the delinked node. This keeps the target nodes up-to-date with the latest changes that affect its 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 is required, as it allows the node to acquire the latest updates to the network as a result of its previous replacement procedure.

[1]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 sends no message for t_{out} consecutive time intervals (this requires that each node send an update message to all its target set every t_{out} time intervals). A random nodes is chosen from a node's membership list to replace the dropped node from the candidate set.

A node maintains a membership list consisting of all previously discovered nodes (gathered through update messages). In our implementation, a node's membership list grows with $O(n)$, however, it can be bounded for large-scale networks. Maintaining a consistent membership list is not required in JetStream, it is only done so as to have quick replacements for timed out entries in the candidate set.

4.3 Analysis

We briefly analyze the network overhead required to maintain a JetStream overlay in a static network. Furthermore, we provide a gossip inject threshold at which JetStream will outperform a random overlay (in terms of fewer total number of messages transmitted). Please refer to Table 1 for a review of the notations used to describe the network (and its characteristics).

Notation	Meaning
n	Number of active nodes in the network
$n_{optimal}$	Number of optimal nodes
$n_{unoptimal}$	Number of unoptimal nodes
l	Size of the target set (uniform)
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	The probability with which an unoptimal node changes its target set
B	The background traffic overhead (number of packets)
w_B	The size of each target set information packet
I	The gossip injection rate
w_I	The size of each gossip message

Table 1: **Notations used to describe the network state. The terminology is defined in Section 4.**

Theorem: 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, an unoptimal node is still seeking to achieve the highest utility. An unoptimal node will generally change its target set once every update period with probability p. As the update period is staggered using randomness, it is expected to be $t_{update} = \frac{t_{rtt}+(t_{stagger}-1)}{2}$ time periods. Recall that a node sends update packets to its target set as well as the delinked node on each change in the target set. Given p as the probability of a deciding node able to select a target with better utility, the traffic generated by all unoptimal nodes will be approximately $n_{unoptimal} \cdot p \cdot (l+1)$ 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, all nodes will remove old elements (exclusive of its current target set) from their candidate sets after t_{out} time periods. This results in an additional $2n \cdot (s-l)$ packets every t_{out} time periods (recall that 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 traffic incurred due to the timeout mechanism will be around $\frac{2n\cdot(s-l)+n_{optimal}\cdot l}{t_{out}}$ packets each time interval. Hence, the theoretical traffic due to the JetStream protocol should be:

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

Due to built-in reciprocity, a gossip message over JetStream will be sent approximately $\frac{n \cdot l}{2}$. In contrast, a random overlay will sent approximately $n\dot{l}l$ messages per gossip. Hence, JetStream can provide a benefit over random overlay when $I \geq I_{thresh}$:

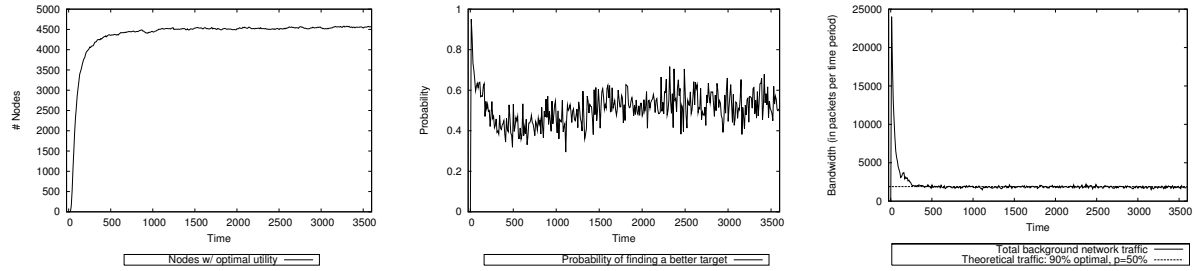
$$\begin{aligned} I \cdot w_I \cdot n \cdot l &\geq I \cdot w_I \cdot \frac{n \cdot l}{2} + B \cdot w_B \\ I_{thresh} &= \frac{2B \cdot w_B}{n \cdot l \cdot w_I} \end{aligned}$$

5 Results

We present the results of our utilitarian network through synchronous simulation. The simulations were run with $n = 5,000$ and $l = 10$. The simulations were run for either 3600 or 7200 time intervals.

5.1 Overlay Characteristics

Implementation Parameters: We briefly study the effect of the various parameters on the progression of the overlay in a static-membership network. The primary parameter that influences the computational and memory overhead is the size of the candidate set. We briefly summarize our findings:



(a) **Nodes achieving optimal utility**

(b) **Probability of unoptimal nodes finding better replacement**

(c) **Total network activity**

Figure 4: **The evolution of JetStream overlay for a network of size $n = 5,000$ and constant target set of size $l = 10$.**

- The localized implementation works almost as well as the global implementation: the network stabilizes with an average utility close to the maximal utility.

- Using a very small value s is sufficient. For example with $s = O(\log n) = 2l$, approximately 90% of nodes reach optimality rapidly (see Figure 4(a)). Increasing s provides limited benefits, at the expense of added computational overhead.

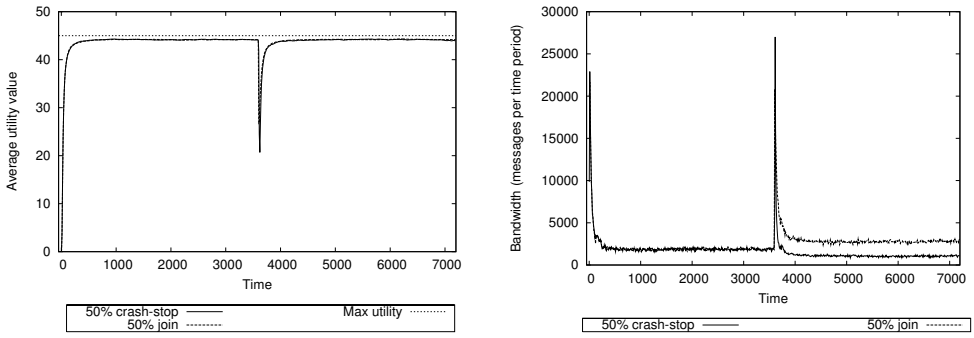
- The value of $t_{stagger}$ should be as low as possible. This allows unoptimal nodes to choose better targets to attain optimality. Our implementation defaults $t_{stagger}$ to 5 time periods. Figure 4(b) shows that an unoptimal node is able to find a better target every update period using only a candidate set of size $s = 2l$. Specifically, the network stabilizes with $p \geq 0.5$. A value of $p = 0.5$ signifies that

the deciding node is able to find at least one other target as a replacement candidate (same or higher utility).

- The value of t_{out} affects the background traffic. Using a value of t_{out} of 120 time periods is sufficient even for systems with high churn (i.e., it only takes the one node to detect a node failure and start a "domino-effect" that will take system back to stability). Figure 4(c) shows that the JetStream overlay stabilizes with an average node sending one packet roughly every 3 time periods.

5.2 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: one with 50% of the active nodes leaving the network, and another one with an additional 50% nodes joining at the halfway point of the simulation.



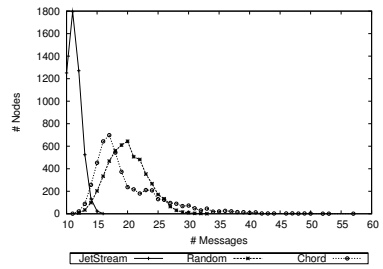
(a) **Average utility value** (b) **Total network activity**

Figure 5: **JetStream is resilient to a massive number of sudden joins and leaves (initial network size $n = 5,000$).**

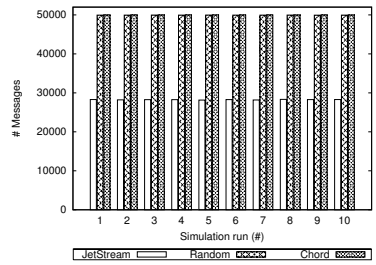
Figure 5(a) shows that the timeout mechanism coupled with the probabilistic design of the replacement procedure results in quick reconvergence to near-optimal average utility value after the crash-stop failures. The reason for the fast recovery is because of the timeout mechanism. As nodes are replaced from the candidate set, a sudden surge in bandwidth utilization is sustained for a brief time period (see Figure 5(b)). Newly joined nodes integrate just as well in the network, by providing a greater choice in target selection.

5.3 JetStream with Flat Gossip

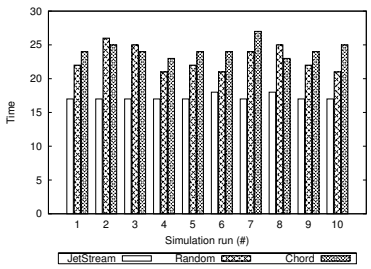
In the next experiment we simulate sending a simple message using flat gossip using three different target selection policies: a random overlay, a Chord overlay, and the JetStream overlay. The Chord [19] overlay is simply pointers to m finger pointers (in a m-bit keyspace) as described in original DHT paper. We define our keyspace in 13-bit (since $2^{13} \geq 5000$), hence each node has 13 total finger pointers. We select l of these randomly as our forwarding targets, before spreading the gossip message. The JetStream overlay is simply the random overlay evolved over 3600 time intervals.



(a) **The total number of messages sent and received by each node is far 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.**

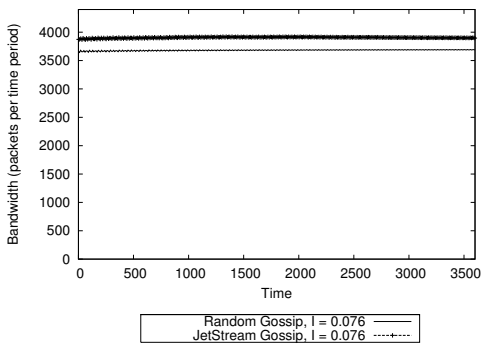
Figure 6: **Compared to a random overlay, the JetStream overlay distributes the flat gossip workload in a more even manner. Furthermore, the JetStream overlay reduces the total workload and improves the latency of gossip spread.**

The emergent behavior of JetStream reduces the total workload imposed by flat gossip. Analyzing the spread of a single gossip message, the total number of messages received by each JetStream node is far fewer, and far less varied (see Figure 6(a)) that either random overlay or Chord. Stated differently, JetStream imposes a "fairer" workload to the network participants.

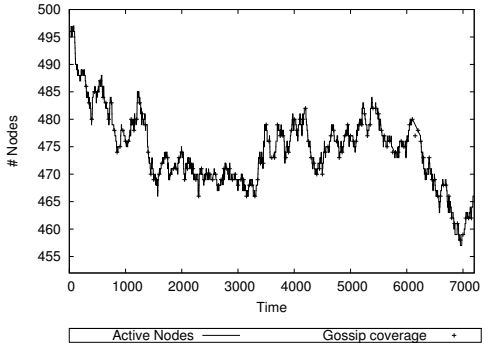
For the next experiment, we ran the aforementioned simulation 10 different times. We can see (in Figure 6(b)) that the total number of messages transmitted by the flat gossip protocol is between 40% and 50% fewer with the JetStream overlay compared to Chord and random overlay. Furthermore, Figure 6(c) shows that the time taken by a gossip message to reach the last node is approximately 25% faster with JetStream on a consistent basis.[2]

[2]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.

5.4 Continuous Gossip



(a) **When gossip injection $I \geq I_{thresh}$, bandwidth utilization using JetStream begins to outperform random overlay.**



(b) **A continuous injection of gossip messages in a dynamic network reaches close to 100% of active nodes.**

Figure 7: **The JetStream overlay performs well with a continuous injection of gossip messages under both static and dynamic networks.**

In the next experiment, we evaluate the performance of a static JetStream network 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. We calculate the value of $I_{thresh} = 0.076$ (i.e., a new gossip every 13 time periods) based on our previously mentioned network assumptions. Figure 7(a) shows that the random overlay actually consumes less bandwidth JetStream at the calculated I_{thresh}. This is due to the fact that the localized implementation of the overlay has not attained maximal utility. However using just only a slightly more aggressive injection rate (i.e, a new gossip packet every 10 time intervals), the bandwidth savings realized by JetStream is clearly noticed (not shown). This experiment shows that JetStream can provide a substantial reduction in bandwidth utilization when the network expects an injection rate higher than I_{thresh}.

Churn: To evaluate the effect of churn on JetStream, we used the Overnet traces [2] collected from a deployed P2P network. We used part of the traces, which monitored the activity of 2400 hosts for 7 days, at a granularity of every 20 minutes (we scaled 1 second = 1 time period for this simulation). While,

the traces monitored 2400 hosts, the actual number of active hosts fluctuated at any give time between approximately 450 and 500. Our experiments utilized only 2 hours of those traces. A node joined or left the network during a random interval spanning the 20 minutes from which the data was collected. A new node was introduced to the system using a bootstrap node that would provide the joining node with enough nodes to complete its candidate set. The joining node would then participate in the network as any other node. A departing node would mimic 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 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(b) 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 time).

6 Conclusion

Gossip protocols provide probabilistic reliability and scalability, but in section 2, we show their inherent randomness may lead to high variation in (received) message overheads at different nodes. Next, in section 3 and section 4, we presented techniques that leveraged simple social networks principles that enable nodes to select gossip targets intelligently. In section 5, we show that the simple heuristics achieved a more uniform message overhead at each node, and also lower the system-wide gossip network traffic by up to 50%. We experimentally compared our JetStream system 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, while still inheriting scale and reliability, and reduced latencies of up to 25%. Lastly, we show 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] D. S. Bernstein, Z. Feng, B. N. Levine, and S. Zilberstein. Adaptive peer selection. In *Proc. of IPTPS*, 2003.

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

[3] 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.

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

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

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

[7] N. Contractor, F. Fonti, C. Steglich, C. Su, and R. Whitbred. Understand the ties that bind: A longitudinal investigation of the evolution of communication network. In *Proc. of Winter Organizational Science Conference*, 2000.

[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] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulie. Peer-to-peer membership management for gossip-based protocols. *IEEE Trans. on Computers*, 52(2), 2003.

[11] I. Gupta, A.-M. Kermarrec, and A. J. Ganesh. Efficient epidemic-style protocols for reliable and scalable multicast. In *Proc. 21st Symposium Reliable Distributed Systems (SRDS)*, 2002.

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

[13] 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.

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

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

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

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

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

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

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

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