Directional Gossip: Gossip in a Wide Area Network

Meng-Jang Lin
University of Texas at Austin
Department of Electrical and Computer Engineering
Austin, TX

Keith Marzullo
University of California, San Diego
Department of Computer Science and Engineering
La Jolla, CA

1 Introduction

A *reliable multicast* protocol ensures that all of the intended recipients of a message m that do not fail eventually deliver m. For example, consider the reliable multicast protocol of [10], and consider a message m, sent by process p_1, that is intended to be delivered by p_1, p_2, and p_3. We impose a directed spanning tree on these processes that is rooted at the message source. For example, for m we could have the directed spanning tree $p_1 \rightarrow p_2 \rightarrow p_3$. The message m propagates down this spanning tree and acknowledgments of the receipt of m propagate back up the tree. A leaf process in this tree delivers m when it receives m, and a non-leaf process delivers m when it gets the acknowledgment for m from all of its children. If a non-leaf process (say, p_1) does not get an acknowledgment for m from one of its children (here, p_2), then it removes the child from the tree and "adopts" that child's children (here, p_3). The process sends m to the newly-adopted children and continues the broadcast. A similar monitoring and adoption approach is used to recover from the failure of the root of the tree.

Reliable multicast protocols are intended for local area networks. Unfortunately, most implementations of reliable multicast do not scale well to large numbers of processes even when all are in the same local area network [3]. For example, with the protocol given above, the sender cannot deliver its own message m until it knows that all non-failed processes have already delivered m. The latency can be reduced by using a bushy directed spanning tree, but doing so increases the overhead of some processes, where by overhead we mean the number of messages a process sends and receives in the reliable multicast of a single m. As the number of processes increases, either the latency or the overhead at some processes increases. Hence, when a multicast is to be sent to a large number of processes or processes located on a wide area network, a protocol like IP Multicast [4] that has been specifically designed for these cases is preferable even though it is not as reliable as reliable multicast.

More recently, *gossip-based protocols* have been developed to address scalability while still providing high reliability of message delivery. These protocols, which were first developed for replicated database consistency management in the Xerox Corporate Internet [5], have been built to implement not only reliable multicast [3, 7] but also failure detection [11] and garbage collection [12]. Gossip protocols are scalable because they don't require as much synchronization as traditional reliable multicast protocols. A generic gossip protocol running at process p has a structure something

like the following:

```
when (p receives a new message m)
while (p believes that not enough of its neighbors have received m) {
   q = a neighbor process of p;
   send m to q;
}
```

Since they lack the amount of synchronization that traditional multicast protocols have, the reliability of gossip-based protocols is evaluated in a different manner. The mathematics of *epidemiology* are often applied, since the spread of a message with a gossip protocol is much like the spread of a disease in a susceptible population. When the mathematics become intractable, simulation is often used.

If one wished to implement gossip-based reliable multicast with as high reliability as possible, then one would use a *flooding protocol* [2] like the following

```
when (p receives a new message m from neighbor q)
for each (r : r neighbor of p)
   if (r != q) send m to r;
```

Flooding can be thought of as a degenerate gossip protocol in which a process chooses all the neighbors that it doesn't know already have the message. Flooding, however, can have a high overhead. Consider the undirected graph $G = (V, E)$ in which the nodes V are processes and edges E connect processes that are neighbors. The total number of messages sent in flooding a single message in G is between $|E|$ and $2|E|$. If the processes are all on a single local area network, then one can consider G to be a clique (that is, all processes can directly communicate with each other), and so the number of messages is quadratic in $|V|$. Gossip protocols are attractive when G is a clique because they provide negligibly less reliability than flooding with a much lower overhead.

If G is not a clique, then the reliability of gossip protocols is less. This is not hard to see, and has already been observed in the context of the spreading of computer viruses [8, 9]. Consider a process p_1 that is in a clique of n processes $p_1, p_2, ...p_n$ and that has a pendant neighbor q: that is, the only neighbor of q is p_1. Suppose that these processes are running a gossip protocol in which p_1 continues to forward a new message m to B of its neighbors that p_1 believes may not yet have m. If p_1 receives a new message m from p_2 and p_1 selects its neighbors uniformly, then the probability that q will receive m is $1 - \binom{n-2}{B}/\binom{n-1}{B} = B/(n-1)$. Thus, B must be close to $n-1$ (and the corresponding overhead high) for the reliability of this protocol to be high. A more intelligent protocol would have p_1 always forward new messages to q and use gossip to communicate with the rest of its neighbors.

We present a protocol that behaves like this more intelligent protocol. Each process determines a *weight* for each of its neighbors. This weight is measured dynamically and is the minimum number of edges that must be removed for the process to become disconnected from its neighbor. For example, assuming no links are down, p_1 would assign a weight of 1 to q and weights of $n-1$ to each of its remaining $n-1$ neighbors. A process floods to neighbors that have small weights and gossips to neighbors that have large weights.

2 Architecture

It has already been observed [11] that the overhead of gossip protocols in a wide area network can be reduced by taking the network topology into account. For example, consider two local area

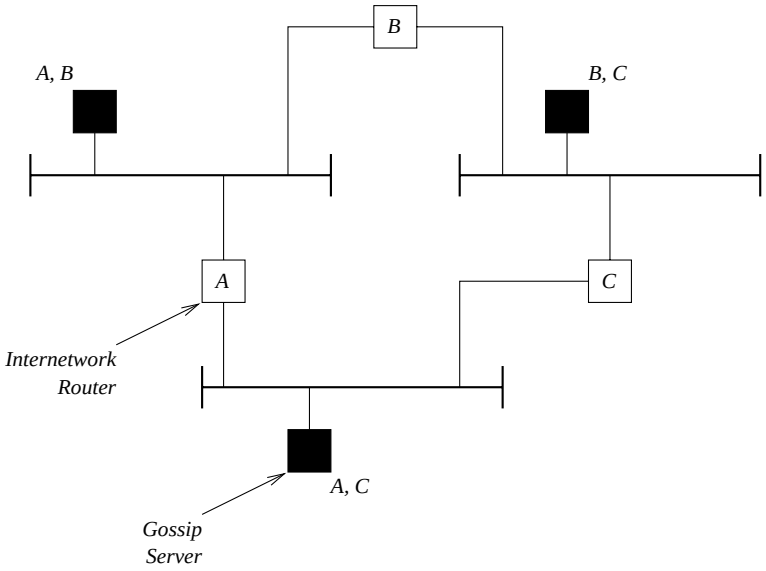

Figure 1: Gossip Server Architecture

networks, each with the same number of processors and that are connected by a single router. If one ignores the network topology, then on average a processor will have half of its neighbors in one local area network and half of its neighbors in the other. Hence, on average half of the gossip messages will traverse the router, which is an unnecessarily high load. The work in [12] addresses this problem by having each processor aware of which local area network each of its neighbors is in. A processor then only rarely decides to send a gossip message to a processor in another local area network. This approach is attractive because it attenuates the traffic across a router without adding any additional changes to the gossip protocol. Its drawback is that it doesn't differentiate between wide area traffic and local area traffic. The performance characteristics and the link failure probabilities are different for wide area networks and local area networks. Hence, we adopt a two-level gossip hierarchy: one level for gossip within a local area network and another level for gossip among local area networks (that is, within a wide area network).

Each local area network runs a *gossip server* that directs gossip to the local area networks that are one hop away. Two gossip servers are neighbors if the local area networks with which they are associated are connected by an internetwork router. For example, Figure 1 shows three local area networks connected by routers A, B and C. Each gossip server is labeled with the routers that are connected to its local area network. Two gossip servers are neighbors if they both have the same internetwork router listed in their label. Hence, the neighbors relation of these three gossip servers in this figure is a three-clique. As will be discussed in the next section, the state that a gossip server maintains is small, and so a gossip server could easily be replicated if the reliability of a single server is not adequately high.

Messages are disseminated to the processes in a local area network, including the gossip servers, using a traditional gossip protocol. When a gossip server receives a message m for the first time via the local area network gossip protocol, it initiates an *wide area network* gossip protocol with message m. When a gossip server receives for the first time a message m via the wide area network gossip protocol, it injects m into its local area network using the local area network gossip protocol.

The protocol that we develop in this paper is the wide area network gossip protocol; we do not address local area network issues further. In Section 1 we argued that to have a high reliability of message delivery, a wide area network gossip protocol needs to have some information about the network topology. Wide area networks can be large and their topology may change frequently, and so we decided not to require each gossip server to have *a priori* knowledge about the entire network topology. Instead, all a gossip server needs to know is its neighbors, which is equivalent to knowing the identity of all local area networks that are one hop away. This is the kind of information that a network administrator will know about a local area network, and so a gossip server can obtain this information from an administrator-generated configuration file.

We believe that the wide area gossip protocol should run on top of IP. Since the gossip protocol determines information about the internetwork connectivity on the fly, it needs to circumvent to some degree the internet routing protocol. As will be described in the next section, a gossip server records the trajectory a gossip message follows to determine the number of link-disjoint paths between itself and a neighbor. Internet routing, on the other hand, abstracts away the notion of a path; routing can change the trajectory of a message as routers fail or become overloaded. Hence, wide area network gossip must thwart routing, which can be done with IP by using either hop counts or source routing.

3 Protocol

In this section we develop a wide area gossip protocol that we call *directional gossip*. We first review some ideas from graph theory and then describe how we use them to measure weights. We then describe the directional gossip protocol in terms of these weights.

3.1 Weights

A *link cut set* of a connected graph G is a set of edges that, if removed from G, will disconnect G. A link cut set with respect to a pair of nodes p and q is a set of edges that, if removed from G, will disconnect p and q. Clearly, the link cut set with respect to a pair of nodes is also a link cut set of the graph.

A gossip server p assigns as a weight to a neighbor gossip server q the size of the smallest link cut set with respect to p and q. If this weight is low, then p will always send new messages to q; else it will send them to q only if p selects q as a neighbor with whom to gossip. The intuition behind this strategy is similar to what was illustrated in Section 1. For example, if this weight is 2, then there are two links, at least one of which must be up and selected when gossiping, for a message to propagate from p to q. As the weight of a neighbor increases, the likelihood of at least one link in the link cut set being up and selected becomes sufficiently large that p and q can exchange information using gossip. Otherwise, p always forwards each new message to q.

Figure 2 gives an example of the weights of a gossip server p. All of the neighbors of p in the four-clique have a weight three, since three edges must be deleted to isolate p from any of these neighbors. The neighbor of p in the three-clique, however, has a weight of two since only the two links connecting the four-clique and the three-clique need be deleted to isolate q from p.

One can imagine other weights that might be interesting. For example, consider the graph in Figure 3 that consists of many long cycles, each distinct except for the (p, q) edge. The weight that p would assign to q is large (in this graph, seven) since there are many link-disjoint paths that connect p and q. Thus, our strategy would most likely have p only probabilistically choose q. If links fail frequently enough, however, then the probability that a message will make it along

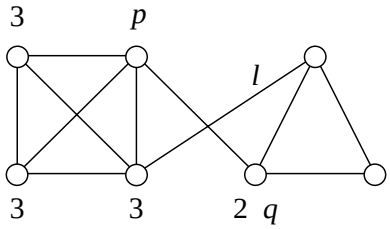

Figure 2: Weights

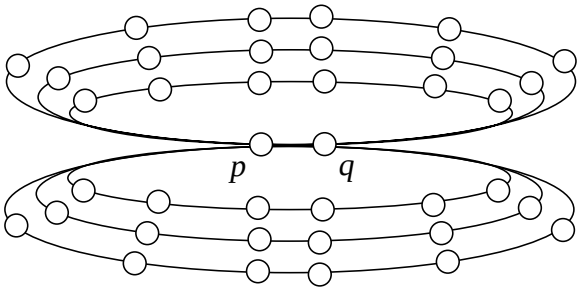

Figure 3: Pathological Graph

one of the long cycles from p to q may be low. Hence, under these conditions p should always forward to q. The benefit of the strategy that we have is that the weights are easy to compute dynamically and the strategy works well for common internetwork interconnection topologies. In addition, our protocol measures the dynamic connectivity between two neighboring nodes. Under the assumptions that the long links are often broken, the weight that p would assign q would in fact be low.

3.2 Measuring Weights

We use the following version of Menger's Theorem, due to Ford and Fulkerson [6], in a method for a gossip server to measure the weights of its neighbors.

> For any two nodes of a graph, the maximum number of link-disjoint paths equals the minimum number of links that separate them.

Thus, a gossip server can maintain for each of its neighbors a list of link-disjoint paths between itself and that neighbor. The size of this set is the weight of the neighbor. A gossip server collects these paths by observing the trajectories that gossip messages traverse, and it ensures through randomization that all such paths are found.

Each gossip message m carries $m.path$ which is the trajectory that m has traversed. Each element in this trajectory identifies an internetwork router that has forwarded m. The internetwork router is implicitly identified by the pair of gossip servers that communicate via that router. Before a gossip server s forwards m to another gossip server r, s adds an identifier for r to the end of $m.path$ if $m.path$ is not empty; otherwise, it sets $m.path$ to the list $\langle s; r \rangle$. Thus, given a trajectory

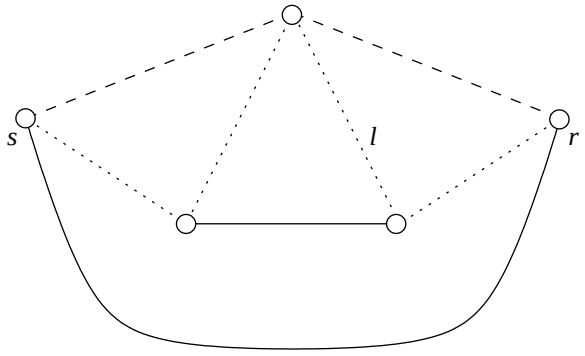

Figure 4: Dynamic Weight Computation

$m.path$ of $g > 1$ gossip servers, we can construct a path of $g - 1$ internetwork routers, which we denote by $INR(m.path)$. Note that the length of $m.path$ is bounded by the diameter D of the wide area network.

Let $Neighbors_s$ be the set of neighbors of a gossip server s. For each neighbor $r \in Neighbors_s$, each gossip server s maintains a list $Paths_s(r)$ of link-disjoint paths that connect s and r. This list contain no more than $|Neighbors_s|$ paths. When a gossip server s receives a gossip message m, for every $r \in Neighbors_s$ such that r is in $m.path$, if for every path $p \in Paths_s(r)$, p and $INR(m.path)$ do not have any common elements, then $INR(m.path)$ is added to $Paths_s(r)$. A simple implementation of this algorithm has $O(D(\log(D) + |Neighbors_s|^2))$ running time for each gossip message that a gossip server receives. The weight a gossip server s computes for its neighbor r is then simply $|Paths_s(r)|$.

The weights that a gossip server computes for its neighbors should be dynamic. For example, consider Figure 2. If the link ℓ fails, then the weight that p assigns to its neighbor q should drop from two to one. Given loosely synchronized clocks, it is not hard to modify the above algorithm to dynamically maintain $Paths_s(r)$ so that failures and recoveries are taken into account. Each element in $m.path$ includes, as well as the identity of a gossip server, the time that the gossip server first received m. Such a time is interpreted, for each element in $INR(m.path)$, as the time that m traversed that internetwork router. Then, when $INR(m.path)$ is compared with a path $p \in Paths_s(r)$, when an element of p is equal to an element of $INR(m.path)$, then the time associated with the link in p is set to the maximum of its current time and the time associated with the same link in $INR(m.path)$. We can then associate a time $Time(p)$ with each element $p \in Paths_s(r)$ as the oldest time of any link in p. If $Time(p)$ is too far in the past, then s can remove p from $Paths_s(r)$.

This simple method of aging link-disjoint paths can result in a temporarily low weight. For example, consider the two gossip servers s and r in Figure 4. Assume that $Paths_s(r)$ contains three paths: the direct path connecting s and r, the path indicated by dashed lines, and the path indicated by dotted lines. Hence, s computes a weight of three for r. Now assume that the link ℓ fails. Eventually, the time associated with the dotted path will become old enough that this path is removed from $Paths_s(r)$, at which point s computes a weight of two for r. This weight is too low: three links must be removed for these two nodes to become disconnected. Eventually, though, s will receive a message following the remaining link-disjoint path, and thus will again compute a weight of three for r. And, as discussed in the next section, computing a too-low weight does not hurt the reliability of the gossip protocol, but only increases the overhead.

3.3 Directional Gossip

The protocol that a gossip server s executes is the following. We first give the initialization. A gossip server only knows about the direct path connecting itself to a neighbor. Thus, s will assign an initial weight of one to each of its neighbors. This weight may be low, and will have s forward new messages to all of its neighbors. As s learns of more paths, it will compute more accurate weights for its neighbors, and the overhead will correspondingly reduce.

init
for each $r \in Neighbors_s : Paths_s(r) = \{INR(\langle s, r \rangle)\}$;

Note that, in order to simplify the exposition, we haven't given a time for the last traversal of this initial path. We assume that whenever a gossip server is added to a trajectory, the current time is also added to the trajectory.

A node starts the sending of a new gossip message by sending it to all of its neighbors. The following code block is executed when s receives a new gossip message m. It first updates $Paths_s(r)$ for each neighbor r that is in $m.path$. It then sends m to all neighbors that s believes may not have m and that have a weight less than K. Gossip server s then chooses enough of the remaining neighbors that may not have m so that at least B neighbors are sent m.

```
when s receives gossip message m for the first time: {
    int sent = 0;
    for each r ∈ Neighbors_s
      if (r ∈ m.path) UpdatePaths(Paths_s(r), INR(Trim(m.path, r)));
    for each r ∈ Neighbors_s AgePaths(Paths_s(r));
    for each r ∈ Neighbors_s
      if (r ∉ m.path && |Paths_s(r)| < K){
        m' = m;
        append r to m'.path;
        send m' to r;
        sent = sent + 1;
      }
    for each r ∈ Choose(B − sent of Neighbors_s − {q : q ∈ m.path}){
      m' = m;
      append r to m'.path;
      send m' to r;
    }
  }
```

The following procedure updates the set of link-disjoint paths between itself and a neighbor based on the trajectory that m has followed. It also updates the times that the links were last traversed. The test for common links can be efficiently implemented by having each path be a sorted list of links, and sorting the trajectory T.

```
void UpdatePaths(ref set of paths P, trajectory T) {
    if (all elements of P have no links in common with T) add T to P;
    else for each p in P:
      for each link ℓ1 ∈ p and link ℓ2 ∈ T:
        if (ℓ1 and ℓ2 name the same internetwork router)
```

```
            set the time ℓ1 was last traversed to
               max(time ℓ1 was last traversed, time ℓ2 was last traversed);
    }
```

The following procedure determines if a path is to be removed because too much time has passed since a link in the path has been traversed.

```
void Age(ref set of paths P) {
    for each p in P:
      if (there is a link ℓ in p: Now() − the last time ℓ was traversed > Timeout)
        remove p from P;
    }
```

Finally, the following function removes a prefix from the sequence of gossip servers a message has traversed.

```
server sequence Trim(server sequence S, gossip server s) {
    return (the sequence S with all servers visited before s removed)
  }
```

4 Simulation

We built a simple discrete event simulator to measure the performance of directional gossip. The simulator takes as input a graph with nodes representing gossip servers and links representing internetwork routers. Messages are reliably sent between gossip servers and are delivered with a time chosen from a uniform distribution. We do not model link failures or gossip server failures, and hence do not implement the aging of links.

We simulated three protocols: flooding, gossip with a fanout B, and directional gossip with a fanout B and a critical weight K. We compared the message overheads of these three different protocols, and when interesting compared their reliability. We also measured the ability of directional gossip to accurately measure weights.

We considered four different network topologies: a ring of 16 gossip servers, a clique of 16 gossip servers, two cliques of eight gossip servers, connected by a single link, and a topology meant to resemble a wide area network.

We show two different kinds of graphs: overhead graphs and and weight graphs. An overhead graph plots for the initialization of each gossip message m the total number of messages gossiping m that were sent. The curves plot the average value computed over 100 runs. A weight graph gives the maximum and the minimum of the weights a node computes for its neighbors against time. We calculate reliability as the percentage of 10,000 runs (done as 100 runs each sending 100 messages) in which all nodes receive all gossip messages.

Ring When the fanout B is at least two, then in a ring all three protocols should behave the same. A node that initiates a gossip message sends the gossip message to its two neighbors, and each neighbor forwards it to its next neighbor. This continues until the last two nodes each send the gossip message to each other. Hence, the last two nodes receive the gossip message twice and the remaining nodes once.

Therefore, 18 messages are sent for a single initiation of a gossip message. The simulator shows this to be the case. The reliability is 1.0 for all three protocols.

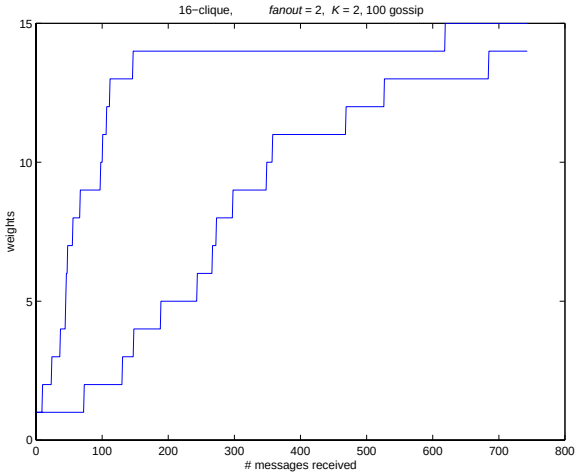

Figure 5: Weight Graph (16-clique, B=2, K=2)

Clique For a clique of size n, a node will eventually learn that there are $n-1$ link-disjoint paths to each of its neighbors. However, learning this will take time. Figure 5 shows how these estimates evolve in a clique of 16 nodes. This graph reflects a run in which 100 gossip messages were initiated and in which $B = 2$ and $K = 2$. A node was chosen at random. The x-axis measures the total number of messages the randomly-chosen node has received, and so is a measure of time. The upper curve shows the highest weight the node has assigned to a neighbor, and the lower curve shows the lowest weight it has assigned to a neighbor. Note that by the end of this run, this node still has not learned that it is in a clique.

However, as soon as the minimum weight a node assigns a neighbor reaches K, then the node will simply use a gossip protocol to disseminate the message. Thus, for this graph the overhead quickly reduces to that of gossip. This behavior is shown in Figure 6. In this figure, the top curve is the overhead of flooding, the middle curve that of directional gossip, and the bottom curve that of simple gossip. The overhead of flooding is expected to be the worst: $(n-1)^2$ messages sent for each initiation of a gossip message, and gossip uses the least number of messages. Directional gossip converges from initially having an overhead less than that of flooding to an overhead of gossip. All three protocols have a reliability of 1.0.

With gossip protocols, the total number of messages sent increases as B increases. Therefore, the difference in message overheads between simple gossip and directional gossip becomes less significant. This is illustrated in Figure 7, although the trend of the overhead for directional gossip is preserved.

Two Cliques For two cliques that have only one link between them, the reliability of gossip protocols can suffer because a node incident on the cross-clique link must always forward that link. Directional gossip overcomes this by identifying this critical link. For example, for two cliques of eight nodes connected by a single link, flooding provides a reliability of 1.0 and directional gossip with $B = 4, K = 2$ a reliability of approximately 0.9963. Gossip with $B = 4$ has a reliability of 0.6329.

Figure 8 shows the corresponding message overheads of the three protocols. As can be seen, initially directional gossip incurs a little more overhead than gossip and gradually decreases to that

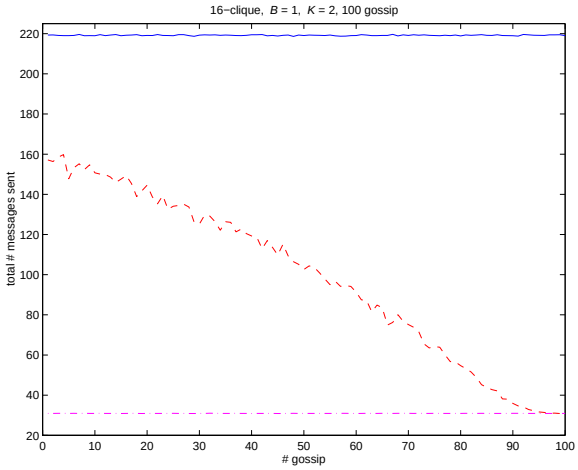

Figure 6: Overhead Graph (16-clique, B=1, K=2)

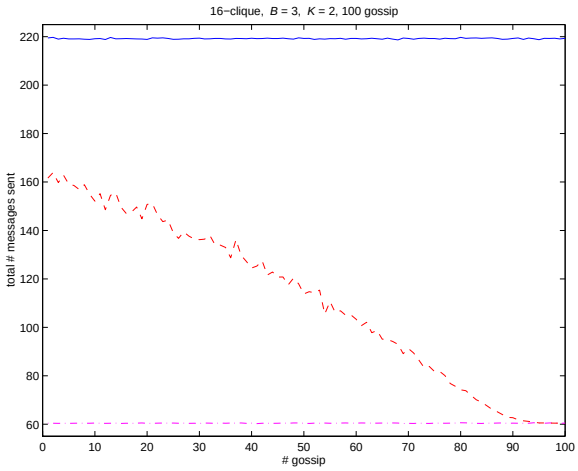

Figure 7: Overhead Graph (16-clique, B=3, K=2)

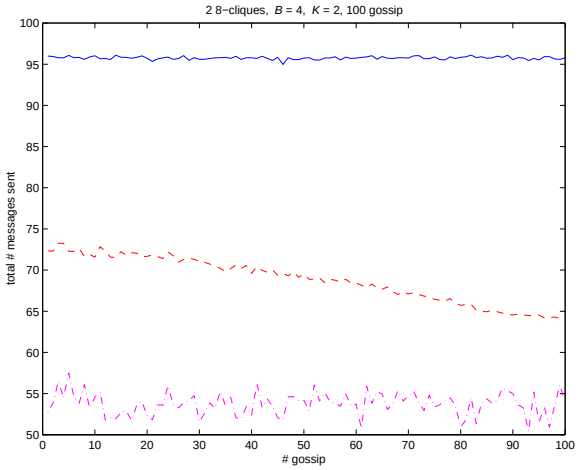

Figure 8: Overhead Graph (2 8-cliques, B=4, K=2)

of gossip.

We have experimented with other values for B and K. The reliability of directional gossip is always significantly higher than that of gossip. Also, the larger the value of B, the better the reliability for both protocols. Increasing K, in general, improves the reliability of directional gossip. However, in the case of two cliques where there is only one critical link, the effect is not as pronounced.

Wide Area Networks We constructed a transit-stub graph to model a wide area network using the technique presented in [1]. We constructed two topologies:

1. A network of 66 nodes. This network consists of two transit domains each having on average three transit nodes. Each transit node connects to, on average, two stub domains. Each stub domain contains an average of five stub nodes. The average node degree within domains is two and is one between domains.

2. A network of 102 nodes. This has the same properties as the previous topology except that stub domains have, on average, eight nodes and each such node has, on average, a degree of five.

Since the flooding protocol always deliver messages to all nodes, it has a reliability of 1.0. Directional gossip in the 66-node WAN with $B = 2$ and $K = 4$ has a reliability of 0.9492, and in the 102-node WAN with $B = 4$ and $K = 4$ has reliability of 0.8994. In contrast, gossip with $B = 2$ in the 66-node WAN has a reliability of 0 and with $B = 4$ in the 102-node WAN has reliability of 0.0597.

Figures 9 and 10 compare the overhead of the three protocols in the two networks. It demonstrates that the high reliability of directional gossip comes at the expense of overhead: the overhead of directional gossip is not far from that of flooding. For the 66-node WAN the overhead of directional gossip is very close to that of flooding. This is not surprising. The average degree of each node is small, and so directional gossip will tend to behave more like flooding. The relatively lower overhead of directional gossip in the 102-node WAN is because the average degree of a node

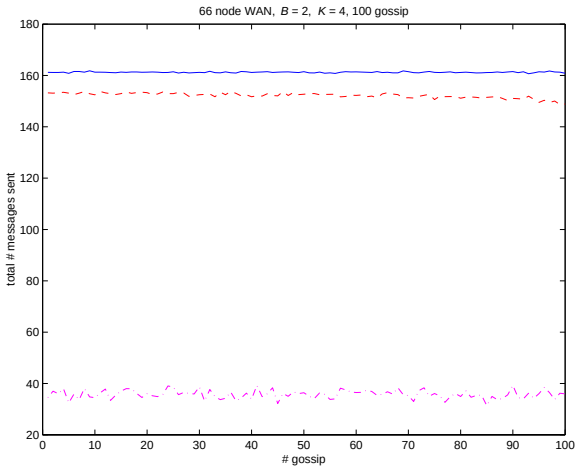

Figure 9: Overhead Graph (66 node WAN, B=2, K=4)

is higher. As we note in Section 5, we believe that the relatively lower reliability in the more richly-connected WAN can be increased.

5 Conclusions

Gossip protocols are becoming an attractive choice to disseminate information in a wide-area network. They are appealing because they are relatively simple yet are robust against common failures such as link failures and processor crashes. They also scale very well with the size of the network and are relatively easy to implement.

However, aimless gossiping does not guarantee good reliability. We have presented a new gossip protocol that can provide higher reliability than traditional gossip protocols while incurring less overhead than flooding. This protocol is part of an architecture that allows one to employ gossip servers and internetwork routers to propagate gossip on a wide area network more efficiently.

Our directional gossip protocol achieves good reliability and low to moderate overhead by having a node identify the critical directions it has to forward gossip messages. A node continuously observes the number of active link-disjoint paths there exist between it and its neighbors. If there are only few paths to a particular neighbor, then it will always pass new messages to that neighbor.

Reliable gossip is based on a simple heuristic: flood messages over links that are critical, and gossip over the other links. The two parameters that are important in these heuristics are K which defines what denotes a link to be critical, and B which defines how broadly the gossip should be. We have looked at the reliability of directional gossip for different values of B and K. For the 66-node WAN, reliability is increased much more by increasing K than by increasing B. For the 102-node WAN and the two-clique examples we have looked at, however, reliability is increased more by increasing B rather than increasing K. We would like to study this further since we believe that understanding the tradeoff of K and B is fundamental to directional gossip.

The gossip protocol that directional gossip is built upon is very simple. There are techniques one can use with gossip protocols to improve its reliability (e.g., see [3]). Understanding which of these techniques can be used for directional gossip, and adapting them to a wide-area setting, are obvious steps that we plan to take.

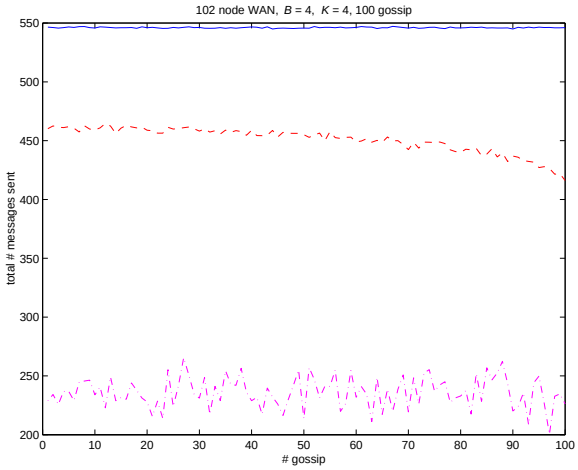

Figure 10: Overhead Graph (102 node WAN, B=4, K=4)

We have only studied directional gossip using a simple simulator and only for a small set of wide area network topologies. We also have not studied how directional gossip performs when links and nodes fail and recover. Our next step is to study directional gossip under more realistic assumptions.

References

[1] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE Infocom '96*, San Francisco, CA, USA, 24-28 March 1996, pp 594–602, Volume 2.

[2] G. R. Andrews. *Concurrent programming: principles and practice*, Benjamin/Cummings, 1991.

[3] K. Birman *et al.* Bimodal multicast. Cornell University, Department of Computer Science Technical Report TR-98-1665, May 18 1998.

[4] S. E. Deering. Multicast routing in internetworks and extended LANs. In *Proceedings of ACM SIGCOMM '88*, Stanford, CA, USA, 16-19 August 1988, pp. 55–64.

[5] A. Demers, *et al.* Epidemic algorithms for replicated database maintenance. In *Proceedings of 6th ACM Symposium on Principles of Distributed Computing*, Vancouver, British Columbia, Canada, 10-12 August 1987, pp. 1–12.

[6] L. R. Ford and D. R. Fulkerson. Maximum flow through a network. *Canadian Journal of Mathematics* 8(1956):399-404.

[7] R. A. Golding and D. E. Long. The performance of weak-consistency replication protocols. University of California at Santa Cruz, Computer Research Laboratory Technical Report UCSC-CRL-92-30, July 1992.

[8] J. Kephart and S. White. Directed-graph epidemiological models of computer viruses. In *Proceedings of IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, USA, 20-22 May 1991, pp. 345-359.

[9] M.-J. Lin, A. Ricciardi, and K. Marzullo. A new model for availability in the face of self-propagating attacks. In *Proceedings of New Security Paradigm Workshop*, Charlottesville, VA, USA, 22-25 September 1998.

[10] F. B. Schneider, D. Gries, and R. D. Schlichting. Fault-tolerant broadcasts. *Science of Computer Programming* 4(1):1–15, April 1984.

[11] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, The Lake District, England, September 1998, pp. 55-70.

[12] K. Guo, *et al.* GSGC: an efficient gossip-style garbage collection scheme for scalable reliable multicast. Cornell University, Department of Computer Science Technical Report TR-97-1656, December 3 1997.