Efficient Epidemic Multicast in Heterogeneous Networks[⋆]

José Pereira[1], Rui Oliveira[1], and Luís Rodrigues[2]

[1] University of Minho
{jop, rco}@di.uminho.pt
[2] University of Lisbon
ler@di.fc.ul.pt

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

1 Introduction

Epidemic multicast protocols, also known as probabilistic or gossip-based, have been proposed for information dissemination due to their scalability to large number of participants and high reliability [BHO$^+$99, EGH$^+$01, PRM$^+$03]. They share at their core a simple procedure: Each item that is received by a node is relayed to a small random subset of other nodes. It can be shown that a message spreads exponentially fast and that the probability that all nodes are informed can be made as large as desired, although not 1, by adjusting configuration parameters [EGKM04]. The same approach has been used for other purposes such as aggregation and topology management [vRBV03, JB06, JKvS03].

The advantages of epidemic multicast stem from its symmetry: Each participant node contributes the same share of bandwidth thus spreading the load and allowing for large redundancy. On the other hand, the symmetry of gossiping means that it does not avoid nodes or links with less capacity, thus being hard to deploy in large and heterogeneous networks. Unfortunately, one cannot naively avoid such symmetry without also endangering scalability and resilience. Existing proposals address this by using explicit knowledge of topology to avoid redundancy on selected links [LM99] or create an additional structured overlay, restricting gossip to subsets of participants [GKG05].

In this paper we point out how to break out of this dilemma without abandoning the simple gossiping procedure. This is achieved by lazily deferring message transmission

[⋆] This work was partially supported by project "P-SON: Probabilistically Structured Overlay Networks" (POS_C/EIA/60941/2004).

R. Meersman, Z. Tari, P. Herrero et al. (Eds.): OTM Workshops 2006, LNCS 4278, pp. 1520–1529, 2006.
© Springer-Verlag Berlin Heidelberg 2006

according to a configurable policy, which has a profound impact in resources used by different nodes and links. A prototype implementation allows the evaluation of several proof-of-concept policies, illustrating the approach.

The rest of this paper is organized as follows. In Section 2 we discuss different gossip strategies. In Section 3 we propose a multicast protocol that combines eager and lazy push gossip and speculate how this can be used to improve resource usage. Section 4 presents a proof-of-concept using the NeEM protocol implementation. Finally, Section 5 concludes the paper by discussing the conclusions and major open issues.

2 Background

Several variants of gossiping exist [KSSV00]. A first variant is *pull gossiping*. A node that wants to know something, contacts another node and asks for the news (e.g. “What's new?”). If the target node has learned something recently it, replies (e.g. “Italy won the world cup.”). A straightforward optimization, called *two-phase pull* is to ask first news titles (e.g. “What's new?”, “The world cup has a winner.”) and then, selectively, pull each interesting item (e.g. “Tell me who it is.”). The lazy variant is useful when the payload is large enough to offset the additional round-trip and control message overhead. This last option has been implemented in the USENET news protocol (NNTP) with the NEWNEWS and ARTICLE commands [KL86].

Alternatively, each node that knows new data contacts another node and tells it the news. Each node does this a limited number of times for each item. This is called *push gossiping*. An optimization of the push strategy, *lazy push*, is to defer the transmission of the payload. A node will therefore contact another node and identify the data (e.g. “I know who won the world cup.”). If the data is unknown, full transmission is required (e.g. “Really? Tell me!”). The originator will then complete the transmission (e.g. “It was Italy.”). Again, this is useful when the payload is large enough and it is likely that the data is already known. Both approaches have been implemented by NNTP [KL86]. Eager push is done with the POST command. Lazy push is done with the IHAVE command.

NNTP has however a very strict policy on which mechanism to use between each pair of hosts. First, for each connection it distinguishes between a server and a client, requiring that only the client issues commands. Namely, between two nodes, the client issues NEWNEWS and ARTICLE to download news and IHAVE to upload recent news. The POST command, and thus push, is reserved for user agent nodes, when uploading a message to a single server. This makes sense in the administratively configured, and mostly static, topology of NNTP.

3 Hybrid Push Gossip

We are now interested in combining the two push gossip strategies in an epidemic multicast protocol to improve its performance in heterogeneous networks. Specifically, during a gossip round, when forwarding a message we choose eager or lazy push independently for each target. Before introducing the algorithm, we discuss the expected impact on reliability and speculate about possible performance advantages.

3.1 Expected Impact on Reliability

The reliability of epidemic multicast is characterized by mathematical formulas that depend on gossip configuration and on fault probability [EGKM04]. Although the proposed changes don't have an impact on gossiping at an abstract level, lazy transmission impacts fault probability, as the additional round-trip and resulting increased latency, widens the window of vulnerability to network faults.

In detail, lazy transmission requires 3 individual packets exchanged in contrast with the single transmission required for eager push. Considering that each packet transmission, or eager push, is lost with a non-zero probability ϵ_e regardless of its size, and assuming that each transmission is an independent event, the probability of a lazy transmission being dropped is $\epsilon_l = 1 - (1 - \epsilon_e)^3$ and thus $\epsilon_e < \epsilon_l$. This should be accounted for when configuring protocol parameters, but has a very low impact for realistically small values of ϵ_e.

However, the probability of small control messages being lost might be smaller that the probability of loosing the entire payload. Furthermore, the probability of several messages being dropped between the same pair of nodes in a short period of time cannot be considered independent, as messages are dropped due to congestion or faulty links. Therefore it is likely that $\epsilon_l < 1 - (1 - \epsilon_e)^3$ by a fairly large margin, although $\epsilon_e < \epsilon_l$ should always hold. This is especially true in the NeEM protocol used in Section 4, due to its use of TCP/IP [PRM$^+$03], which will perform retransmission of spurious dropped packets.

3.2 Expected Impact on Performance

Intuitively lazy gossip is useful as it reduces the likelihood of transmitting the payload multiple times to the same target node. Considering the exponential nature of epidemic dissemination, which starts slowly, has a fast expansion phase and then terminates slowly again, eager gossip is less interesting during the last rounds, when a large share of target nodes have already received the message. Eager gossip is at the last rounds responsible for a large overhead, as each node receives multiple copies of the message. Likewise, lazy push is less useful during the first rounds when only a small share of target nodes has received the message and thus will certainly have to request transmission.

An obvious conclusion is that one should switch from an eager to a lazy strategy based on round number. This has been exploited in the original *pbcast* protocol [BHO$^+$99], although the preferred eager mechanism in *pbcast* is not gossip and the lazy mechanism is two-phase pull gossip. Based on identical arguments, it has been proposed that an eager push mechanism is used initially, and then switched to a pull mechanism [KSSV00].

In contrast, our proposal is based on independently configuring each transmission of a single gossip round: For each of the nodes chosen as targets for gossiping some message, one wants to query a configurable policy module for the appropriate strategy. Roughly, given a set of targets, the policy module partitions it in a lazy set and an eager set, basing its decision on some knowledge it possesses about the system and on the message itself.

```
proc Multicast($d$) do
  Forward(MkId(), $d$, $0$)

proc Forward($i, d, r$) do
  Deliver($d$)
  $K = K \cup \{i\}$
  $W[i] = \emptyset$
  if $r < m$ do
    $P$ = PeerSample($f$)
    $(E, L)$ = Split($P, d, r$)
    for each $p \in E$ do
      Send(MSG($i, d, r + 1$), $p$)
    $C[i] = (d, r)$
    for each $p \in L$ do
      Send(IHAVE($i$), $p$)

upon Receive(MSG($i, d, r$), $s$) do
  if $i \notin K$ then
    Forward($i, d, r$)

upon Receive(IHAVE($i$), $s$) do
  if $i \notin K$ then
    $W[i] = W[i] \cup \{s\}$

periodically, for some $(i, s) : s \in W[i]$ do
  Send(IWANT($i$), $s$)
  $W[i] = W[i] \setminus \{s\}$

upon Receive(IWANT($i$), $s$) do
  $(d, r) = C[i]$
  Send(MSG($i, d, r + 1$), $p$)
```

Fig. 1. Hybrid push gossip protocol

This approach has a more subtle impact on performance, which is not as easy to grasp. To help your intuition we make two assumptions, that we will not use later when evaluating the protocol. The first assumption is that the latency of a lazy round is much larger than the latency of an eager round.[1] If a source node s transmits a message m eagerly to a target t_1 and lazily to another target t_2, it is likely that t_2 ends up receiving m relayed eagerly through t_1 after several hops, instead of directly from s. This means that nodes such as t_2 will not request lazy transmission from s.

The second assumption is that message payload is so large that the overhead of lazy push, namely, the advertisement and request messages, is negligible.[2] Therefore, we conclude that traffic in the link from s to t_2 is negligible when compared to traffic in the link from s to t_1. Gossip strategy selection can therefore be used to divert the bulk of the traffic from lesser capable network links, as long as each node knows the cost of each link according to some interesting metric.

3.3 Algorithm

The algorithm to achieve this is presented in Figure 1. It uses the following variables in each node: a set K of known messages, initially empty; a map W of known source node sets for each message identifier, initially empty; a map C, holding the payload and round number for each message identifier, initially empty. It assumes also a peer sampling service providing an uniform sample of other nodes [JGKvS04]. A message m can be sent to a node p using the Send(m, p) primitive and received by handling the Receive(m, p) up-call. The strategy for each transmission is encapsulated in the Split primitive described below.

[1] This is true to a certain extent, due to the additional round-trip required to request and perform the actual transmission plus some implementation dependent scheduling delay.

[2] This is most likely true in practice, although it depends on the application.

In detail, the algorithm works as follows. The application calls procedure MULTICAST(d) to multicast a message with payload d (line 1). This simply generates an unique identifier and forwards it (line 2). The identifier chosen must be unique with high probability, as conflicts will cause deliveries to be omitted. A simple way to implement this is to generate a random bit-string with sufficient length. Handling of messages received from other nodes is similar (line 15), although it is now necessary to check for and discard duplicates using the set of known identifiers K (line 16) before proceeding.

The forwarding procedure FORWARD(i, d, r) (line 3) uses the message identifier i, the payload d and the number of times, or rounds, the message has already been relayed r, which is initially 0. It starts by delivering the payload locally using the DELIVER(d) up-call. Then the message identifier is added to the set of previously known messages K (line 5) and the set of known sources $W[i]$ is cleared (line 6). These avoid multiple deliveries, as described before, and stop all retransmission requests.

Actual forwarding occurs only if the message has been forwarded less than m times (line 7) [Kol03]. Usually, using only an eager push mechanism, this would reduce to querying the peer sampling service to obtain a set of *f* target nodes and then sending the message, as in lines 8 and 11. Constants m and f are the usual gossip configuration parameters [EGKM04]. However, we use the SPLIT primitive (line 9) to partition the set P of node addresses in two, an eager set E and a lazy set L. Peers in set L will be sent only a message advertisement that does not contain the payload (line 14). Note that the only correctness requirement on the implementation of this primitive is that if $(E, L) =$ SPLIT(P, d, r), then $E \cup L = P$ and $E \cap L = \emptyset$. This ensures that each message is gossiped exactly *f* times, although some of them lazily.

Upon receiving a message advertisement for an unknown message, its source node is recorded (line 20). Periodically, a message identifier and source pair (i, s) is chosen (line 21) and a message requesting its transmission is sent (line 22). The source node is removed from $W[i]$, thus ensuring that it is used only once. Finally, when a node receives a retransmission request (line 24) it looks it up in the cache and transmits the payload (line 26). Note that a retransmission request can only be received as a consequence of a previous advertisement and thus the message is guaranteed to be locally known.

For simplicity, we do not show how identifiers are removed from set K or messages from C, preventing them from growing indefinitely. This problem has been studied before, and efficient solutions exist ensuring with high probability that no active messages are garbage collected [EGH$^+$01, Kol03].

4 Proof-of-Concept

To assess the viability of our proposal we built and experimentally evaluate a prototype. We start by lifting the assumptions used in the previous section, on latency and on size of payload, by using a homogeneous network and a relatively small message of 256 bytes. Experiments are then conducted with several selection strategies.

4.1 Experimental Setting

The protocol is built on an open source and lightweight implementation of the NeEM protocol [PRM$^+$03] that uses the java.nio API for scalability and performance[SP06].

Briefly, NeEM uses TCP/IP connections between nodes in order to avoid network congestion. When a connection blocks, messages are buffered in user space, which then uses a custom purging strategy to improve reliability. The result is a virtual connectionless layer that provides improved guarantees for gossiping.

This implementation was selected as NeEM 0.5 already supports eager and lazy push, although the later is selected only based on a message size and age threshold. Message identifiers are probabilistically unique 128 bit strings. The change required was to remove the hard-coded push strategy and query the strategy module during each gossip round.

The assumptions used in the previous section, on latency and on size of payload, are lifted by using a homogeneous network in which all nodes and links are equal, and a relatively small message payload of 256 bytes. To evaluate the impact of the proposed approach, we do however force nodes to behave as if they had diagnosed an heterogeneous network by implementing several strategies, which impose different static assumptions on network capacity:

ADSL Assume that half of the nodes have asymmetric connections with limited uplink bandwidth to all other nodes. These nodes use only lazy push. The other, assumed have symmetric connections, use only eager pushing.

"Reverse" ADSL Assume that half of the nodes have asymmetric connections with limited down-link bandwidth to all other nodes. Messages directed at these nodes always use lazy pushing. Messages to others are eagerly pushed.

Two ISPs Assume that each half of the nodes is in a different network, with a costly connection between them. Messages traversing to a different network are lazy pushed. Otherwise, within the network, are eagerly pushed.

We have also tested NeEM in its original configuration, which captures the obvious intuition that eager push should be used in the initial rounds. As a baseline, we have also configured the protocol to always do lazy push and always do eager push.

We then ran 200 protocol nodes in a machine with 2 AMD Opteron processors and 4GB RAM. The protocol was configured with gossip fanout of 11 and overlay fanout of 15. These correspond to a probability 0.995 of atomic delivery with 1% messages dropped, and a probability of 0.999 of connectedness when 15% of nodes fail [EGKM04]. Request for retransmissions are done with a uniform random interval between 0 and 200 ms. Each run consists of a 30 second warm-up period, while the overlay is formed and settles down, a 100 second test period, while 200 messages are transmitted with a 500ms interval, one by each node, and finally a 10 second cool-down period. All bytes transmitted, messages relayed and messages delivered during the 100 second period are recorded for later processing.

While tests run, there are at least 1500 TCP/IP sockets active, corresponding to 3000 open file descriptors. As each NeEM instance uses only one thread, there are 200 protocol threads plus 200 application threads running. We observe that this corresponds to approximately 50% processor load and about 1GB memory. This includes extensive logs performed on every I/O operation.

Results were confirmed by repeating them and running similar experiments in different hardware and software configurations with identical conclusions. Significance of the results is also improved by the size of the samples: Each run considers 40000 message deliveries, over 20000 TCP/IP connections, and close to half a million network packets transmitted.

4.2 Results

We collected several different metrics presented in Figure 2. All are presented as empirical cumulative distribution functions: The x-axis shows the metric and the y-axis shows the ratio of samples that were measured less or equal any specific value. A small variance is thus depicted as a close to vertical line and a multi-modal distribution as staircase function.

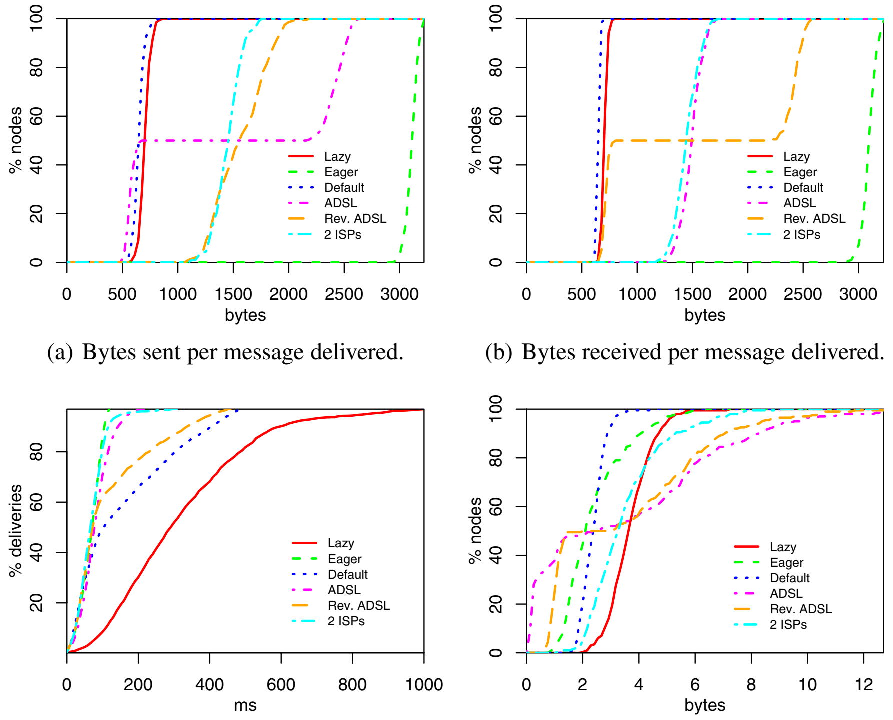
In detail, Figures 2(a) and 2(b) show the amount of network resources used by each message delivery, respectively, the average number of bytes sent and received. Given that each message carries 256 bytes payload, this clearly shows the amount of overhead. Figure 2(c) shows end-to-end delivery latency measured at the application level, considering all deliveries to all nodes. As a side effect, a large slant indicates high jitter. This provides a measurement of quality of service.

Figure 2(d) is computed as follows. For each delivered message, we take the path from sender to delivery. Then, we truncate the first and last nodes (which are invariably the sender and the target). We then count how many times each node appears in such paths that lead to delivery. The result for each node is divided by the number of messages that it has delivered. This gives us a measure of how much a node contributes to the effort required to spread messages. This should be compared with the amount of bytes transmitted, to determine whether a large amount of bytes is a large contribution or simply overhead.

We start by discussing the symmetric protocols: baseline eager and lazy push, as well as NeEM 0.5 default. These use the same strategy for all transmissions in a gossip round. The lazy push strategy achieves low network traffic at the expense of offering also the worst latency. In contrast, the eager push strategy results in a very large number of bytes transmitted, while achieving second best latency. Most interestingly, the relay count is close, showing that most of bytes transmitted are simply redundancy.

The NeEM 0.5 default is an interesting compromise, as it is the strategy that generates the lowest network traffic while providing good latency to half of deliveries, i.e. those that are served during the initial push gossip phase. It provides also extremely good fairness when distributing the load. Actually, the average number of bytes transmitted is very close to the absolute minimum of a single payload transmission (256 bytes) plus 11 headers (11×20 bytes). When compared with lazy push, half of the processes getting the entire payload earlier should also improve reliability.

The *Two ISPs* test shows how effective the technique can be: Compared with the eager push run, the overhead is cut in half with little impact in delivery latency. Better yet, the traffic on assumed inter-ISP links is severely reduced, even when compared to lazy push. This is shown in Table 1, which depicts the amount of data transmitted during the lifetime of a connection. Only the *Two ISPs* produces a statistically significant mean difference, as can easily be confirmed by running a test with a confidence level of 0.95.



(a) Bytes sent per message delivered.

(b) Bytes received per message delivered.

(c) End-to-end delivery latency.

(d) Relevant relays per message.

Fig. 2. Experimental results

This happens as these links are seldom used to convey actual data, but only lazy push advertisements.

We now look at the *ADSL* and *"Reverse" ADSL* strategies. As expected, there is a sharp contrast between the two halves of nodes, but there are some unexpected results. In the first case, *ADSL*, half of the processes send a much lower amount of bytes, while the others produce a large amount of overhead. Actually the later use the exact same strategy as in the eager push experiment, but surprisingly, the amount of data transmitted is lower. The fact that some processes are using lazy push reduces the amount of data transmitted by the remaining processes that use always eager push.

This can be explained by deliveries occurring after a larger number of rounds, in average, as more hops through symmetric processes are required. This happens because asymmetric processes contribute later than they would in the eager push scenario, thus implicitly increasing the share of the remaining. A larger number of processes thus delivers messages in round m and does not gossip again, reducing network usage. A small number of processes has to wait for lazy transmissions, resulting in a longer tail in latency distribution. This is confirmed by Figure 2(d), where a number of processes does not contribute at all to deliveries, while others contribute a much larger share.

Table 1. Bytes transmitted in each link type for each strategy

	Intra-ISP			Inter-ISP			Statistical
	Samples	Mean	Std. Dev.	Samples	Mean	Std. Dev.	sign. (0.95)
Lazy Push	11264	1195.98	1147.79	11154	1192.83	1159.21	no
Two ISPs	11348	4119.17	3720.95	11207	859.10	908.83	yes
Eager Push	11545	5364.82	4781.49	11260	5337.21	4812.57	no

In the second case, *"Reverse" ADSL*, half of the processes receive almost no overhead, as they are targeted always using lazy gossip. This has also a mild impact in number of bytes transmitted, as receiving messages earlier causes more effective transmissions to be performed. This is confirmed by Figure 2(d) which surprisingly shows a result very similar to the ADSL scenario. This leads to the interesting conclusion that the effect of a node p itself choosing eager gossip is similar to the effect of all other nodes choosing eager gossip when targetting p.

Also surprising is that the *"Reverse" ADSL* results in the best overall latency, even than always using eager push. This might be explained by the reduced overhead while still performing a large number of eager transmissions and thus deserves some further exploration as an alternative to the intuitively obvious approach used by default by NeEM 0.5.

5 Discussion

In this paper we propose that eager and lazy push gossip are combined within the same round by encapsulating the choice in a configurable strategy module. This proposal is aimed at better matching resource usage with resource availability in epidemic multicast, thus improving performance in heterogeneous networks.

Although the presented experiments use simple policies, that build on static global knowledge of the network, they show that relevant results can be obtained (i.e. reduction of backbone traffic in the *Two ISPs* test) and that it is possible to control both the amount of data received and sent (i.e. the *ADSL* and *"Reverse" ADSL* tests). They also point out some surprising consequences of the approach, namely, how nodes are indirectly affected by other node's choices and how even a seemingly trivial policy produces a large impact in latency.

Note also that the proposed experimental framework, despite simple, is not the best case scenario. An heterogeneous network would help, for instance, by delaying lazy push on slower links thus further highlighting the impact of laziness. This is being addressed by testing on a realistic large scale network infrastructure.

Finally, what is the best policy and how to implement it deriving the required knowledge about the system in a scalable and efficient fashion, is still an open question. There are previous examples of how this can be achieved, namely, by building on local knowledge extracted from the network [PRPO04] and by using gossip itself to build global knowledge [RSP$^+$03, JB06, MKG03, GKG05].

References

[BHO$^+$99] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Trans. Computer Systems*, 17(2), May 1999.

[EGH$^+$01] P. Eugster, R. Guerraoui, S. Handrukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. In *Proc. IEEE Intl. Conf. Dependable Systems and Networks (DSN)*, 2001.

[EGKM04] P. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulie. From epidemics to distributed computing. *IEEE Computer*, May 2004.

[GKG05] I. Gupta, A.-M. Kermarrec, and A.J. Ganesh. Efficient and adaptive epidemic-style protocols for reliable and scalable multicast. *IEEE Trans. Parallel and Distributed Systems*, 2005.

[JB06] M. Jelasity and O. Babaoglu. T-Man: Gossip-based overlay topology management. In *Proc. 3rd Intl. Ws. Engineering Self-Organising Applications (ESOA'05)*. Springer-Verlag, 2006.

[JGKvS04] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *Proc. 5th ACM/IFIP/USENIX Intl. Conf. Middleware*, 2004.

[JKvS03] M. Jelasity, W. Kowalczyk, and M. van Steen. Newscast computing. Technical Report IR-CS-006.03, Vrije Universiteit Amsterdam, 2003.

[KL86] B. Kantor and P. Lapsley. RFC 977: Network News Transfer Protocol. Internet Engineering Task Force, 1986.

[Kol03] B. Koldehofe. Buffer management in probabilistic peer-to-peer communication protocols. In *Proc. IEEE Symp. Reliable Distributed Systems (SRDS)*, 2003.

[KSSV00] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *IEEE Symp. Foundations of Computer Science*, 2000.

[LM99] M.-J. Lin and K. Marzullo. Directional gossip: Gossip in a wide area network. In *Proc. European Dependable Computing Conf. (EDCC)*, 1999.

[MKG03] L. Massoulié, A.-M. Kermarrec, and A. Ganesh. Network awareness and failure resilience in self-organising overlays networkss. In *Proc. IEEE Symp. Reliable Distributed Systems (SRDS'04)*, 2003.

[PRM$^+$03] J. Pereira, L. Rodrigues, M. J. Monteiro, R. Oliveira, and A.-M. Kermarrec. NeEM: Network-friendly epidemic multicast. In *Proc. IEEE Symp. Reliable Distributed Systems (SRDS)*, 2003.

[PRPO04] J. Pereira, L. Rodrigues, A. Pinto, and R. Oliveira. Low-latency probabilistic broadcast in wide area networks. In *Proc. IEEE Symp. Reliable Distributed Systems (SRDS'04)*, October 2004.

[RSP$^+$03] L. Rodrigues, S.Handurukande, J. Pereira, R. Guerraoui, and A.-M. Kermarrec. Adaptive gossip-based broadcast. In *Proc. IEEE Intl. Conf. Distributed Systems and Networks (DSN)*, 2003.

[SP06] P. Santos and J. Pereira. NeEM version 0.5. http://neem.sf.net, 2006.

[vRBV03] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Computer Systems*, 21(2):164–206, May 2003.