A Robust and Scalable Peer-to-Peer Gossiping Protocol*

Spyros Voulgaris[1], Márk Jelasity[2], and Maarten van Steen[1]

[1] Department Computer Science, Faculty of Sciences, Vrije Universiteit Amsterdam, The Netherlands
{spyros,steen}@cs.vu.nl
[2] Department of Computer Science, University of Bologna, Italy
jelasity@cs.unibo.it

Abstract. The newscast model is a general approach for communication in large agent-based distributed systems. The two basic services—membership management and information dissemination—are implemented by the same epidemic-style protocol. In this paper we present the newscast model and report on experiments using a Java implementation. The experiments involve communication in a large, wide-area cluster computer. By analysis of the outcome of the experiments we demonstrate that the system indeed shows the scalability and dependability properties predicted by our previous theoretical and simulation results.

1 Introduction

The popularity of peer-to-peer systems in the last couple of years illustrates how the Internet is gradually shifting toward a distributed system that supports more than only client-server applications. A key issue in peer-to-peer systems is that distribution of data and control across processes is symmetric. Moreover, this distribution is done in such a way that processes are highly autonomous and independent of each other. The important advantage of this approach is scalability. A well-designed peer-to-peer system can easily scale to millions of processes, each of which can join or leave whenever it pleases without seriously disrupting the system's overall quality of service.

A crucial aspect of large-scale peer-to-peer systems is that they are easy to manage. Any system that attempts to centrally manage how processes connect to each other and distribute data and control will fail, notably when processes join and leave all the time. Instead, it should be a property of the design itself that the distribution of data and control takes place in an automated fashion that requires no global management at all. In effect, we are looking at the design of self-managing systems.

There are many different types of peer-to-peer systems. In most cases, these systems can be divided into two separate layers. The lowest layer consists of protocols for handling group membership and communication, whereas the highest

* This work was partially supported by the Future & Emerging Technologies unit of the European Commission through Project BISON (IST-2001-38923).

G. Moro, C. Sartori, and M.P. Singh (Eds.): AP2PC 2003, LNAI 2872, pp. 47–58, 2004.

layer implements the required functionality for a specific application. The lowest layer thus forms the core of the peer-to-peer system. Roughly speaking, there are three types of core systems.

The first, and most popular type is designed to efficiently support content-based searching. In many cases, these systems operate with centralized index servers that keep track of where content is located. The index servers are often constructed dynamically in the form of super peers [15]. Examples include Gnutella and KaZaa. The second type aims at efficiently routing a request to its destination through an overlay network formed by the collection of peers. Examples of such systems are Chord [12], Pastry [11], Tapestry [16], and CAN [10]. A third type deploy epidemic protocols [3]. In these systems, the goal is not so much enabling point-to-point communication between peers, but rather the rapid and efficient dissemination of information. Examples in this class include Scamp [6], and probabilistic reliable broadcasting [4,5,9].

In this paper, we concentrate on information-dissemination based systems that deploy epidemic protocols. A crucial element in an epidemic protocol is that a participating peer can randomly select another peer to exchange information. Traditional protocols supported this random selection by providing a list of all other participating peers. Clearly, such an approach cannot scale to large networks. As an alternative, approaches such as described in [4,7] ensure that a peer always has a list that represents an independent and randomly selected sample from the entire set of peers.

We have recently developed an epidemic protocol for disseminating information in large, dynamically changing sets of autonomous agents. This, so-named, *newscast protocol* solves two problems inherent to large sets of agents: (1) information dissemination, and (2) efficient membership management. The main distinction in comparison to similar epidemic-based solutions, is that agents can join and leave at virtually no cost at all, and without affecting the information-dissemination properties of the protocol.

The associated model of *newscasting*, that is, the model of information dissemination and membership management as presented to agents, is described in detail in a separate paper [8], along with theoretical analyses partly based on simulations. To better substantiate our claims regarding scalability, we have implemented the newscast protocol (in Java). We subsequently used this implementation to conduct a series of experiments that *emulate* large-scale agent-based applications on a real network. In particular, we set up a series of experiments with 128,000 agents scattered across a hierarchically organized cluster of 320 processors. These processors, in turn, are geographically spread over four different sites in the Netherlands.

An important and interesting aspect of these experiments is that the underlying communication network is heterogeneous. It includes interprocess communication facilities on a single workstation, point-to-point local-area high-speed links, as well as wide-area links. In this way, we are better able to measure the effect that a real communication infrastructure has on the properties of our dissemination model.

In this paper, we describe the newscast protocol and report on our experiments involving emulation of large networks of agents. We show that the theoretical results, which are based on an idealized underlying communication infrastructure, still hold when dealing with a realistic infrastructure, thus further substantiating our claims that newscasting is a highly robust and scalable model for information dissemination in large and rapidly changing sets of agents. In the following we discuss our protocol, the experimental setup, and the results of our experiments, to end with conclusions.

2 The Newscast Protocol

In our implementation of the newscast model, a large group of agents is connected through a simple peer-to-peer data exchange protocol. The protocol is extremely simple: each agent knows only a (continuously changing) small set of peers of which one is randomly chosen to exchange information. In this section, we start with explaining how the protocol works, after that we explore some remarkable theoretical properties of its emerging behavior. These properties are further investigated in Section 3 when we report on our large-scale emulation experiments.

2.1 Principal Operation

The two main building blocks of our newscast model are a *collective of agents* and a *news agency*, as shown in Figure 1. The basic idea is that the news agency asks all agents regularly for *news* by means of a callback function getNews(). In addition, the news agency provides each agent with news about the other agents in the collective, again through a callback function newsUpdate(news[]).

The definition of what counts as news is application dependent. The agents simply live their lives (perform computations, listen to sensors and the news, etc.) and based on the computations they have completed and the information they have collected they must provide the news agency with news when asked.

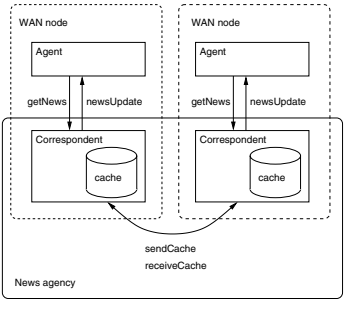


Fig. 1. The organization of a newscast application.

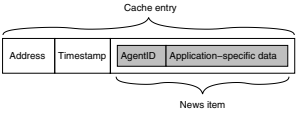


Fig. 2. The format of news items and cache entries.

The model itself can be fully specified in terms of the functional and statistical properties of the operations getNews() and newsUpdate(news[]). Instead of this definitial style of specification, we take a much simpler approach in this paper by describing the semantics of the model in terms of the newscast protocol, of which we have shown that it meets the model's specifications [8].

Each agent has an associated *correspondent* that runs on the same machine hosting the agent. The correspondents jointly form the distributed implementation of the news agency. Each correspondent maintains a *fixed-sized* cache of c news items. Whenever an agent passes a news item to its correspondent, the latter timestamps the item, adds its own network address, and subsequently caches the item. A news item itself consists of an agent identifier and the actual news as provided by the agent, as shown in Figure 2.

Correspondents regularly exchange caches as follows. Omitting specific details (which are found in [8]), each correspondent executes the following five steps once every ΔT time units (ΔT is referred to as the *refresh interval*).

1. Request a fresh news item from the local agent by calling getNews(). Add the item to the cache.
2. Randomly select a peer correspondent by considering the network address of other (and available) correspondents as found in the cache.
3. Send all cache entries to the selected peer, and, in turn, receive all the peer's cache entries. Merge the received entries into the local cache.
4. Pass the received cache entries from the peer agent to the local agent by calling newsUpdate().
5. The correspondent now has $2c$ cache entries; it subsequently throws away the c oldest ones.

The selected peer correspondent executes the last three steps as well, so that after the exchange both correspondents have the same cache. Note, however, that as soon as any of these two correspondents executes the protocol again, their respective caches will most likely be different again.

The protocol does not require that the clocks of correspondents are synchronized, but only that the timestamps of news items in a single cache are mutually consistent. We assume that the communication time between two correspondents is negligible in comparison to ΔT (which is generally in the order of minutes). When a correspondent A passes its cache to B, it also sends along its current local time, T_A. When B receives the cache entries, it subsequently adjusts the timestamp of each entry with a value $T_A - T_B$, effectively normalizing the time of each new entry to those already cached.

2.2 Properties of Newscasting

As it turns out, this simple model of communication has desirable statistical properties. To understand the behavior of newscasting, we consider the communication graphs G_t at different time instants t that are induced by maintaining caches at each correspondent. Each such graph is constructed from a corresponding directed graph D_t as follows. The vertex set V_t of D_t contains the correspondents. For correspondents a and b in V_t we have the link $a \to b$ if and only if the address of b is in the cache of a at time t. The cache-exchange algorithm leads to a series of directed graphs, given an initial directed graph D_0. The communication graph G_t is now simply constructed by dropping the orientation in D_t. G_t expresses the possibility of cache exchanges.

Now consider the series of graphs $G_0, G_{\Delta T}, G_{2\Delta T}, \ldots$. Note that during a time interval ΔT each correspondent initiates the cache-exchange algorithm. In other words, after ΔT time units, all correspondents will have fetched a news item from their agent, exchanged caches with at least one of their neighbors (and possibly more), and have passed c news items to their agent. We say that a *communication cycle* has completed.

We have conducted simulations with up to 50,000 correspondents, assuming an idealized communication infrastructure with no communication delays and packet losses. Our simulations show that even for relatively small cache sizes (say, $c = 20$) each graph $G_{k\Delta T}$ stays connected. Moreover, it turns out that the average length of each shortest path between two nodes is small, as shown in Figure 3(a).

In fact, further investigations revealed that the induced communication graphs have many properties in common with what are known as *small worlds* [1, 14]. An important property of these types of networks is that they show a relatively high *clustering coefficient*, which, for a given node, is the ratio of the number of edges between the neighbors of the node and the number of all possible edges between the neighbors. For example, in a complete graph all nodes have

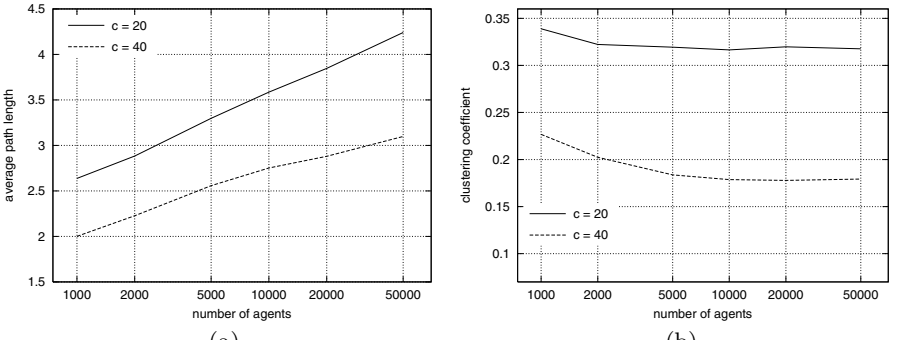


Fig. 3. (a) Average shortest path length between two nodes for different cache sizes. (b) Average clustering coefficient taken over all nodes.

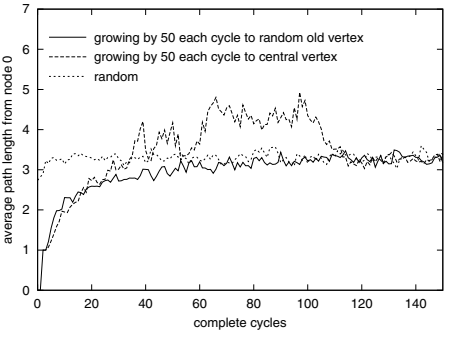


Fig. 4. Average shortest path length while adding 50 agents every cycle until 5000 agents have been added.

a clustering coefficient of 1 while in random graphs this coefficient is typically small (if our graphs were random, the clustering coefficient would be expected to be c/n). Figure 3(b) shows the clustering coefficient for different cache sizes c and communication graphs $G_{k\Delta T}$.

Simulations also show that we need only an extremely simple way of handling membership, which is an important improvement in comparison to other epidemic models. Consider the worst solution to handling membership that could possibly disrupt the emergent behavior of our protocol: an agent contacts a well-known central server and simply initiates the cache-exchange protocol with that server. This approach systematically biases the content of caches, which now all depend on what is stored at the central server.

We conducted a simulation experiment in which we admitted 50 new agents at every communication cycle until 5000 agents had joined the network, after which no new agents were allowed to join. When measuring the average path length again, we obtain the results shown in Figure 4. What is seen, is that shortly after the last 50 agents have been added (i.e., after 100 completed cycles), the average path length quickly converges to the one we would expect in a stable graph. We can conclude that even this worst-imaginable membership protocol does not affect the properties of newscasting. In effect, when a node wants to join, it needs to know only the address of a single other node and can simply start with executing the newscast protocol. Leaving is done by simply stopping communication.

3 Validation of the Newscast Protocol

Using theoretical analyses and simulations, we are able to show that the statistical properties of the protocol meet the specifications of the newscast model. However, in the world of large-scale systems, theory and practice often diverge. Therefore, to investigate how the protocol would behave in practice and to further substantiate our claims, we conducted a series of *emulation* experiments.

Emulation, as opposed to simulation, involves implementing the protocol and conducting experiments on a real network of computers. In our case, we carried out experiments with a collective of up to 128,000 agents distributed across a 320-node wide-area cluster of workstations.

3.1 The Implementation

In order to experiment with the newscast model described earlier, we implemented its underlying protocol. Java was chosen for portability reasons, allowing us to easily execute the protocol in heterogeneous environments. Our implementation is organized as three modules: the core, the application, and the utility module. The *core module* implements a correspondent materializing the epidemic protocol described in Section 2.1. The core module has no dependencies on the other two modules. It is a self-contained implementation of the newscast protocol. All communication is based on UDP. Multiple instances of the core module can coexist in a single Java virtual machine, behaving as separate, independent correspondents.

The *application module* provides the implementation of an agent. One instance of the core module has exactly one associated instance of the application module. Our experiments were focused on the properties of the epidemic protocol itself without considering any particular application. Therefore the agent we defined has only basic functionality. It returns empty content in the getNews() operation, and ignores any content delivered to it through the newsUpdate(news[]) operation.

The *utility module* serves the specific needs of our experiments, such as batch running, coordinating the experiments, and logging. Exactly one utility module instance exists in each virtual machine. In particular, the utility module takes care of starting multiple agent-correspondent pairs each running on a separate thread within the same Java virtual machine to allow emulation of a large network. It coordinates with utility modules running on other Java virtual machines (possibly on remote hosts) to determine initial connection addresses for the correspondents. The utility module also contains logging functionality. It periodically freezes the agent-correspondent pairs running in the Java virtual machine, logs their state, and then resumes operation. Utility modules coordinate to ensure that freezing and resuming for logging occur simultaneously on all the Java virtual machines spread across the different hosts.

3.2 The DAS-2

We conducted our experiments on the *Distributed ASCI Supercomputer* (DAS-2), a wide-area distributed cluster-based system consisting of five clusters of dual-processor PCs located at different sites across the Netherlands. The cluster at the Vrije Universiteit consists of 72 nodes, while the other clusters consist of 32 nodes each, giving a total of 200 nodes (400 processors). Each node has two 1-GHz Pentium-III processors, and at least 1GB of RAM.

Nodes within a single cluster are connected by a Fast Ethernet (100Mbps) network dedicated to their cluster. Clusters, in turn, communicate over wide-area links, which are shared for all traffic between the universities and which have shown to support an aggregated bandwidth of 20 Mbps.

3.3 Experimental Setting

We carried out experiments with a network of 128,000 agents distributed across 160 dual-processor nodes on four of the five DAS-2 clusters. We recorded and analyzed the behavior of the newscast model for three different cache sizes c: 20, 30 and 40. In all three cases the refresh interval ΔT was 10 seconds.

The presented series of experiments was conducted to examine the possible impact of the underlying network's heterogeneity on the operation of the newscast model. It is, therefore, worth describing the deployment of agents across the DAS-2 nodes. We used 160-dual processor nodes, selecting 64 nodes from the cluster at the Vrije Universiteit, and 32 nodes out of three other DAS-2 clusters. We executed two Java virtual machines per node (one per processor), each Java virtual machine running 400 agents.

The deployment of agents described above presents a desirable property for our experiments: network heterogeneity. Four different types of communication were involved, depending on the relative location of the agents communicating:

- *Intraprocess* communication for agents running in different threads within the same Java virtual machine.
- *Interprocess* communication for agents run by separate Java virtual machines, but on the same DAS-2 node.
- *Local-area* (or *intracluster*) communication for agents residing on different nodes, but within the same cluster. These agents were communicating through a 100Mbps Fast Ethernet network.
- *Wide-area* (or *intercluster*) communication for agents belonging to different clusters. This type of communication was carried out over the wide-area links shared with other wide-area traffic.

This diverse environment (with respect to networking) provided us with a valuable testbed for studying the newscast model.

It is important to observe that even though 800 agents run within each DAS-2 node, more than 99% of the communication between agents is either across wide-area or local-area links. For any given agent, 799 other agents run on the same node, and 127,200 run on other nodes, which account for 0.6% and 99.4% of the total 128,000 agents respectively. As we observed in our experiments, the items in an agent's cache are randomly distributed over *all* the participating agents, irrespective of their location. Therefore we expect only 0.6% of the total communication to be within or between processes on the same node, and all the rest to be across local-area or wide-area links. In particular, agents in the three 32-node clusters are expected to experience 80% wide-area and 19.4% local-area traffic, while agents in our 64-node cluster are expected to have 60% wide-area and 39.4% local-area traffic.

Another parameter of our experiments that is worth noting, is the *bootstrapping* mechanism. By bootstrapping we refer to the procedure of providing agents with the information required to jump-start the newscast network's formation. In principle, a new agent joins by contacting *any* existing agent and exchanging caches. When the whole network starts from scratch, a systematic way has to be present to provide one or more initial communication points to each agent. In our experiments this task was handled by the utility module. All agents were provided with the single address of one selected agent. Providing agents with a choice of (possibly random) agents to connect to initially, enhances the randomness of the network in the early cycles. However, a bootstrapping mechanism as simple and centralized as the one we chose further substantiates our claims of the protocol's convergent behavior, as discussed in the following section.

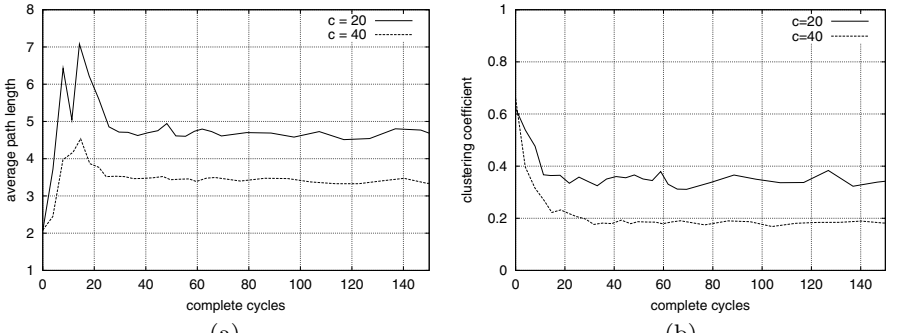
4 Results

This section presents a thorough analysis of the output of our three large-scale experiments with 128,000 correspondents using cache sizes of 20, 30, and 40, respectively. We will often compare the emerging communication graphs to random graphs. In all cases, the random graphs we refer to are generated by selecting exactly c undirected edges randomly for each node. For example, in such a graph each node has at least c edges (but usually more).

4.1 Statistical Properties of the Communication Graph

Figure 5 illustrates the two most important properties of the emergent communication graphs. The number of cycles actually performed was over 5000, however, only the initial cycles are depicted because the values remain the same throughout the experiment indicating a convergent behavior.

The *average path length* from a node is defined as the average of the minimal path lengths of that node to all other nodes. To get a finite value we have to



(a) (b)

Fig. 5. (a) The evolution of average path length from a node. (b) Clustering coefficient.

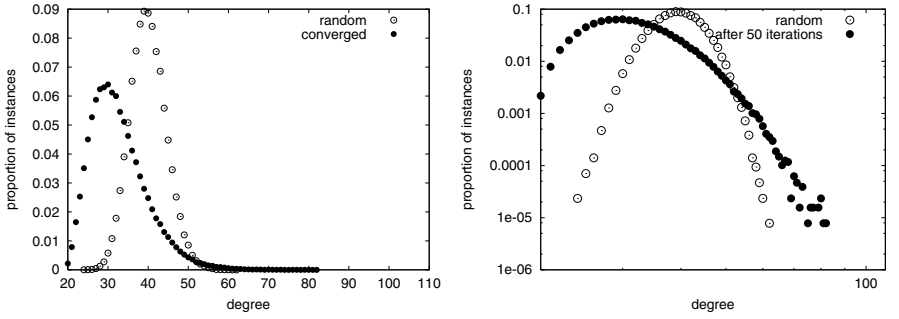
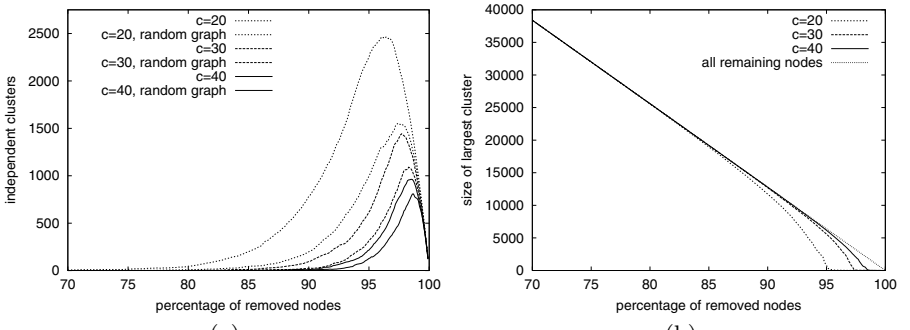


Fig. 6. Degree distribution on (a) linear and (b) log-log scale using the same range. The depicted values belong to a single converged communication graph with $n = 128000$ and $c = 20$ (converged) and a graph with c random edges generated for each vertex (random).



(a) (b)

Fig. 7. Partitioning of the communication graph as a function of the percentage of removed random nodes (node failures). The curves belong to a single graph. The largest clusters of random graphs are omitted for clarity; their relationship to the communication graphs is similar to the relationship in the case of the number of clusters.

have a connected graph. We can observe very low average path lengths which coincide with the expected lengths after extrapolation of the simulation data shown in Figure 3(a). The initial peak is explained by the applied bootstrapping mechanism described in Section 3. This mechanism results in an initially unbalanced neighborhood structure. However, after all correspondents get connected to the collective, the average path length converges quickly to its final value.

The *average clustering coefficient* taken over all nodes is shown in Figure 5(b), and again corresponds to our simulation results. Together with the values found for average path lengths, we can indeed conclude that our communication graphs are *small-world* graphs.

Small-world graphs come in very different flavors however. One interesting property to investigate is whether our graphs are *scale free* or not. The degree

of a random node defines a random variable. If this variable is exponentially distributed (linear on the log-log scale) then the graph is scale free. Figure 6 shows the distribution of the node degree for the case of $c = 20$ which deviates the most from the random case.

It can be seen clearly that our communication graph is not scale free. From a dependability point of view this is an advantage since scale-free graphs are sensitive to the removal of highly connected nodes (even though they are less sensitive to random node removal). The effect of node removal in our graphs is discussed next.

4.2 Robustness to Node Removal

Figure 7 shows the effect of node removal to the connectivity of the communication graph. Note that the number of clusters decreases when approaching 100% removal because the remaining graph itself becomes small. The graph shows very similar behavior to a random graph, especially if the cache is large. These results indicate considerable robustness to node failures especially considering the size of the largest cluster which indicates that most of the clusters are in fact very small and most of the nodes are still in a single connected cluster.

5 Conclusions

In this paper we presented experiments with a Java implementation of the newscast model. The experiments involved 128,000 agents communicating with each other over a wide-area, large-scale heterogeneous cluster of processors.

The outcome of these experiments is particularly valuable since it represents the real implementation of our model as opposed to previously conducted simulations, yet the size of the system is comparable with the scale of typical simulation results as well. The results are in complete agreement with the theoretical predictions and simulations presented in [8] providing practical evidence concerning the correctness of our algorithm and of the statistical properties of the emerging communication graphs.

As we demonstrated, the series of the communication graphs show stable small-world properties which make it a dependable and effective device for information dissemination and membership management. Most importantly, these properties are not maintained explicitly, but they are emergent from the underlying simple epidemic-style information exchange protocol.

References

1. R. Albert and A.-L. Barabasi. "Statistical Mechanics of Complex Networks." *Reviews of Modern Physics*, 74(1):47–97, Jan. 2001.
2. M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. "Exploiting Network Proximity in Peer-to-Peer Overlay Networks." Technical Report MSR-TR-2002-82, Microsoft Research, Cambridge, UK, June 2002.

3. A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. "Epidemic Algorithms for Replicated Database Management." In *Proc. Sixth Symp. on Principles of Distributed Computing*, pp. 1–12, Aug. 1987. ACM.
4. P. Eugster, R. Guerraoui, S. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. "Lightweight Probabilistic Broadcast." In *Proc. Second Int'l Conf. Dependable Systems and Networks*, pp. 443–452, June 2001. IEEE Computer Society Press, Los Alamitos, CA.
5. P. T. Eugster and R. Guerraoui. "Probabilistic Multicast." In *Proc. Int'l Conf. Dependable Systems and Networks*, June 2002. IEEE Computer Society Press, Los Alamitos, CA.
6. A. Ganesh, A.-M. Kermarrec, and L. Massoulié. "Scamp: Peer-to-Peer Lightweight Membership Servic for Large-Scale Group Communication." In *Proc. Networked Group Communication Workshop*, volume 2233 of *Lect. Notes Comp. Sc.*, pp. 44–56, Nov. 2001. Springer-Verlag, Berlin.
7. A. Ganesh, A.-M. Kermarrec, and L. Massoulié. "Peer-to-Peer Membership Management for Gossip-based Protocols." *IEEE Trans. Comp.*, 52(2):139–149, Feb. 2003.
8. M. Jelasity and M. van Steen. "Large-Scale Newscast Computing on the Internet." Technical Report IR-503, Vrije Universiteit, Department of Computer Science, Oct. 2002.
9. A.-M. Kermarrec, L. Massoulié, and A. Ganesh. "Probabilistic Reliable Dissemination in Large-Scale Systems." *IEEE Trans. Par. Distr. Syst.*, 14(3):248–258, Mar. 2003.
10. S. P. Ratnasamy. *A Scalable Content Addressable Network*. PhD thesis, University of California at Berkeley, Oct. 2002.
11. A. Rowstron and P. Druschel. "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems." In R. Guerraoui, (ed.), *Proc. Middleware 2001*, volume 2218 of *Lect. Notes Comp. Sc.*, pp. 329–350, Nov. 2001. Springer-Verlag, Berlin.
12. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications." In *Proc. SIGCOMM*, Aug. 2001. ACM.
13. I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications." *IEEE/ACM Trans. Netw.*, 2003. To appear.
14. D. J. Watts. *Small Worlds, The Dynamics of Networks between Order and Randomness*. Princeton University Press, Princeton, NJ, 1999.
15. B. Yang and H. Garcia-Molina. "Designing a Super-Peer Network." In *Proc. 19th Int'l Conf. Data Engineering*, Mar. 2003. IEEE Computer Society Press, Los Alamitos, CA.
16. B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing." Technical Report CSD-01-1141, Computer Science Division, University of California, Berkeley, Apr. 2001.