Gossiping in Distributed Systems

Anne-Marie Kermarrec
INRIA, Rennes, France
Anne-Marie.Kermarrec@irisa.fr

Maarten van Steen
Vrije Universiteit Amsterdam, Netherlands
steen@cs.vu.nl

ABSTRACT

Gossip-based algorithms were first introduced for reliably disseminating data in large-scale distributed systems. However, their simplicity, robustness, and flexibility make them attractive for more than just pure data dissemination alone. In particular, gossiping has been applied to data aggregation, overlay maintenance, and resource allocation. Gossiping applications more or less fit the same framework, with often subtle differences in algorithmic details determining divergent emergent behavior. This divergence is often difficult to understand, as formal models have yet to be developed that can capture the full design space of gossiping solutions. In this paper, we present a brief introduction to the field of gossiping in distributed systems, by providing a simple framework and using that framework to describe solutions for various application domains.

1. THE GOSSIP REVIVAL

Gossiping in distributed systems refers to the repeated probabilistic exchange of information between two members. Probabilistic choice is a key element of gossiping, and in general refers to the choice of member pairs that communicate. Repetition is also crucial: in principle, gossiping is the *endless* process of randomly choosing two members and subsequently letting these two exchange information. The effect of gossiping is that information can spread within a group just as it would in real life. In a sense, this is a strongly related to epidemics, by which a disease is spread by infecting members of a group, which in turn can infect others. In distributed systems, epidemics refers to information dissemination where a node randomly chooses another member, to which the information (i.e., the "disease") can be communicated (unless both had already been "contaminated"). Following common practice, we shall not make a distinction between gossip and epidemics in this paper.

The first use of gossip in distributed systems appeared some twenty years ago in [5], where it was applied to ensure consistency in replicated databases. Since then, it has been steadily applied to solve a variety of problems, but has never received so much attention as in the past few years. There is a good reason for this increased interest. Just as in real life, gossiping is characterized by the fact that it can spread quickly and be extremely persistent, notably also in situations in which groups change with respect to their memberships and relations between group members.

In the last decade, we have seen a dramatic shift in the scale of distributed systems: they have become larger, geographically more dispersed, and often cross multiple administrative boundaries. This shift in scale has forced us to revisit many of the assumptions underlying distributed systems, and hence the solutions we have developed for them. More specifically, we need to deal with near-continuous changes regarding the collection of nodes constituting a distributed system, as well as the existence and quality of the connections between those nodes. As a consequence, by being forced to pay attention to a system's *behavior under continuous change*, we have been forced to focus more on *convergent behavior*. Gossiping solutions are often indifferent to changes in the group of communicating nodes, while at the same time exhibit strong convergent behavior. Moreover, many solutions are designed to let nodes take local-only decisions. As such, they are attractive for developing large-scale distributed systems.

In this light, we present a brief overview of gossiping in modern distributed systems. We aim to accomplish two goals. First, we want to provide a framework that is easy and general enough to understand and compare gossiping solutions. Furthermore, although gossiping has been traditionally used for reliable information dissemination, its applicability goes way beyond in distributed systems. Our second goal is to provide an overview of traditional gossip-based dissemination, but also to have a look at other applications such as those for data aggregation and systems management.

The rest of this paper is organized as follows. We start by providing a framework for gossiping protocols in Section 2. Section 3 provides the main results of probabilistic gossip-based data dissemination; in Sections 4 and 5 we show that the same set of algorithms may be applied to build and maintain overlay networks from fully unstructured (random-based) to fully structured. In Section 7, we also show that the same techniques can be used to compute aggregates in a fully decentralized way and in Section 6 to slice the networks according to a given attribute metric. We conclude in Section 8 by stating that we believe that such approaches can be used in many other contexts which are either out of the scope of this paper (such as application to ad-hoc networks) or yet to be discovered.

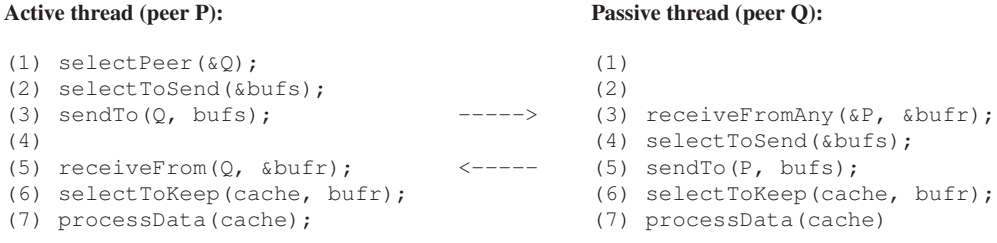

Figure 1: The general organization of a gossiping protocol.

2. GOSSIPING FRAMEWORK

As with so many concepts in computer science, providing a precise characterization of gossiping solutions is difficult. The common model underlying gossip-based solutions is that of a (possibly dynamically changing) set of processes, called *peers*, each of which regularly exchanges information with other peers. This information exchange can be modeled by means of two separate threads: an active thread which takes the initiative to communication, and a passive thread accepting incoming exchange requests. Note that each peer consists of these two threads.

Figure 1 shows the general organization of a gossiping protocol. We consider two peers, P and Q, of which P has just selected Q to exchange information with. Each peer is equipped with a **cache**, consisting of references to other peers in the system. A cache is also capable of storing (peer specific) information. Peer P first selects a target peer (in our example Q) to setup communication. It then decides on what data that it will send to Q, and stores this in a send buffer. The information is then sent to Q, which, in turn, selects information to return to P. After the exchange has completed, both decide separately which exchanged information will actually be stored in the cache. In case the cache is limited in size, this may imply that cache entries will need to be replaced.

The crucial aspects of this framework are the following:

Peer selection: A peer such as P needs to select another peer (Q in our example) to exchange information with. Gossiping protocols differ in their selection process of Q. For example, in many systems it is assumed that Q can be chosen uniformly from the complete set of currently available peers. Of course, this is generally not a realistic assumption and special measures need to be taken to approximate its validity. In this respect, we point to the peer sampling service [12] which is a distributed implementation allowing a peer to uniformly randomly select another available peer.

In wireless systems, the situation becomes completely different. In this case, it is often necessary to choose a peer only from those nodes that are in range of the peer initiating a gossip. Of course, it is also possible to mimic the behavior of wire-line systems (see, e.g., [1]), but this may be costly, and even unnecessary. For example, consider the gossiping protocol for wireless environments as described in [11]. Here, a node P selects one of its neighbors, say Q, to exchange data with. P and Q are in each other's range, and execute a synchronous protocol. That is, P will send data to Q, and expects data to be returned from Q. As it turns out, this synchronous behavior can be more or less mimicked using an asynchronous local broadcast protocol as deployed in wireless sensor networks. In that case, P simply sends data to all its neighbors, of which one will keep it (with high probability). Likewise, later one of P's neighbors will issue a broadcast, and P may be the one to accept it for further processing. At application level, there is hardly any difference between the synchronous and the asynchronous data exchange. Note however, that although the *effect* of both protocols is the same, one could question whether the asynchronous version can actually be qualified as gossiping. As such, it illustrates that drawing a hard line around gossiping protocols may not be that easy.

Data exchanged: As explained earlier, both peers decide on which data they exchange with each other. This exchange is highly application-dependent and we will see many differences depending on application domains. One interesting example of data exchange is the one used for implementing the above mentioned peer sampling service. Instead of exchanging application data, peers may also decide to exchange references to other peers as stored in their local cache. Gossiping in that case leads to a situation in which the network topology, as induced by the references that peers hold to each other, changes at each gossip exchange.

Data processing: This part of the framework describes how a peer deals with the received information. Again, data processing is highly application dependent. An interesting case is when references are exchanged, which are then subsequently used by the application to build lists of nearby nodes, where the notion of proximity may vary. For example, we shall see that this approach allows to build distributed data structures, but that it can also be used to cluster nodes according to mutual interests.

In the following we will have a look at various application domains where gossiping can be successfully applied.

3. DISSEMINATION

Traditionally, gossip-based solutions have been used for dissemination purposes. A standard approach toward dissemination is to simply let peers forward messages to each other. As explained in [9], this dissemination process is roughly steered by three parameters: every process can store up to c messages in its local cache, a message is forwarded up to t times, and each time a peer selects f other peers to forward the message to. A crucial element in this process is the selection of the f peers, and much work on gossip-based dissemination concentrates on how peers can be selected such that they represent a uniformly drawn sample from the entire set of current members. We will return to this problem below.

When focusing only on the data dissemination part, the basic approach can be characterized as follows:

- **Peer selection:** each peer P periodically chooses $f \geq 1$ peers $Q_1, \ldots, Q_f$ uniformly at random from the entire set of currently available (i.e., alive) peers.
- **Data exchanged:** a message is selected from the local cache and copied from one peer to another. In a *push* model, P forwards a message to each Q_i; in a *pull* model, each Q_i sends a message to P. A combination of the two is also possible.
- **Data processing:** effectively, nothing special is done except storing the received message for a next iteration, or passing it to a higher (application) layer.

The essence of this approach is described in the seminal paper by Demers et al. [5]. In this case, dissemination of several data elements takes place at the same time by letting randomly chosen pairs of nodes exchange new information. At the end of the exchange, the two nodes forming a pair should each have the same information (effectively reducing the entropy of the entire system).

An important observation of gossip-based dissemination is that the data gets spread exponentially fast through the network. In general, it takes $O(\log N)$ rounds to reach all nodes, where N is the number of nodes. A round completes when every peer has initiated a gossip exactly once. More specifically, if a peer repeatedly gossips to f other peers chosen uniformly at random among all other peers, the proportion ρ of nodes that will eventually get the message satisfies the equation $\rho = 1 - exp(\rho f)$ and is actually independent of the system size.

Now, consider the probability that not one specific peer should get the message, but that all peers should get it. In a later work [21], it has been shown that the probability of achieving such an *atomic* reliable broadcast can be related to the system size, the number of link and peer failures and the number of gossip targets. In a system where each peer, receiving for the first time a given message, gossips it to $f = log(N) + \gamma$ other peers chosen uniformly at random, N being the size of the system and γ a parameter of the system, the probability that eventually all peers receive the message is $exp(-exp(-\gamma))$. This guarantee holds if the fanout f (number of gossip targets) is *on average* $O(log(N))$ regardless of the distribution.

Although using only gossiping for data dissemination is perfectly feasible, it has the drawback of potentially introducing much redundancy. This is certainly the case with multicasting, when a message is intended to be delivered only to a specific subset of all peers. An interesting combination of multicasting and gossiping was introduced by Birman et al. [2] to achieve *probabilistic reliable multicasting*. Assuming that unreliable multicasting has already been taken care of, for example, through IP multicasting, reliability can be achieved by having nodes gossip information about missed messages. The important improvement is that only meta information is gossiped, and not the multicast messages themselves.

An approach along the same lines is followed in the CREW system [6]. In this case, the information to be disseminated is partitioned into chunks, and only the list of chunks is initially broadcast to all peers. Each node then attempts to get all the chunks by pulling in the missing ones from randomly selected peers. In this way, we at least avoid that chunks are sent out to nodes that already have them.

However, the problem remains that inherent flooding nature of the approach imposes that notably those nodes having a high indegree are at risk of receiving the same message over and over again. With random selection of peers, it can be shown that such nodes will indeed exist. There are several approaches to tackle this problem, but the basic idea is always to reduce the variance in the indegree distribution. As a consequence, the construction of the overlay becomes an important issue. We will return to this matter below. It is worth mentioning that Patel et al. [24] propose to borrow a solution from the theory of social networking by having a node A give higher preference to neighbors that (1) also refer to A, or (2) for which A can act as a bridge between two otherwise disconnected nodes. Property (1) is the equivalent of social reciprocity, while (2) is also known as filling a structural hole. The net effect is that dissemination improves while the number of redundant messages drops.

The main results regarding gossip-based dissemination that we want to emphasize here are that solutions are (1) simple: they rely on a simple exchange of message between peers, (2) scalable: the number of times a peer needs to gossip or the number of targets a peer needs to gossip to are logarithmic in the size of the system, and (3) provide some probabilistic guarantees of message delivery. To this end, they rely on randomization and a proactive approach to recovery by ensuring *a priori* a high degree of redundancy. These protocols are therefore extremely resilient to a large number of failures in the system and cope well with the high level of dynamism of large-scale distributed systems. At the same time, redundancy may impose a problem that requires additional solutions.

4. PEER SAMPLING

Virtually all gossip algorithms rely on the fact that a node can uniformly at random select a peer from the current set of nodes. Obviously, scalability demands that we employ decentralized membership protocols for gossip-based algorithms. Although alternative protocols have been proposed to create a neighbor set for each node [1, 10, 23], it turns out that gossiping itself can be used to provide randomly selected peers in a highly efficient and effective way.

Again, the same substrate is used except that instead of exchanging a message to be broadcast, membership information (i.e., a list of peers) is gossiped. The first protocol which introduced such a gossip-based membership algorithm is Lpbcast [8] and combined it with the actual dissemination algorithm. Newscast was then introduced in [13] and provided a gossip-based algorithm alone, achieving strong connectivity properties. Along these lines, Cyclon was proposed [30], resulting in topologies close to those of regular directed random graphs.

In [14], a common framework has been defined fitting those approaches in order to study the impact of the protocol parameters on the resulting graphs of connections. An interesting conclusion from this work was that a large number of gossip-based membership protocols can be used to provide each node with a local cache, such that randomly selecting a peer from a local cache is very close to randomly selecting a peer from the entire network. We can thus build a scalable *peer sampling service* using gossiping. In this case, the characteristics of an implementation are as follows:

- **Peer selection:** each peer P chooses periodically a gossip target Q from its current set of neighbors.
- **Data exchanged:** the data exchanged between peers is are simply lists of peers.

- **Data processing:** upon receipt of the list, the receiving peer merges the list of peers received with its own list to compose a new list of neighbors. Some peers may need to be dropped from the new list due to size limitations.

It turns out that some parameters have a significant impact on the resulting overlay. For example, it may be decided to guarantee that a number of exchanged peers are kept in the new neighbor lists of both gossiping peers. This so-called *shuffle* parameter specifies the diversity of the local caches and limits the *loss* of information over gossip. Likewise, the strategy to drop old references to peers (when deciding on which references to keep in the procedure `selectToKeep` in Figure 1) may have a large impact with respect to how quickly failed nodes are forgotten. It is interesting to note that although different implementations show similar *functional behavior* (i.e., they can all provide randomly selected peers), their overall convergent behavior in terms of connectivity, convergence speed, and even vulnerability to attacks [18], differs widely. The peer sampling service is now considered a building block for many other applications.

Peer sampling in many cases assumes the underlying network has a high degree of homogeneity, in the sense that the cost for contacting a node and exchanging information is almost the same for each node. Of course, this is not true and ignoring heterogeneity may adversely affect performance, or even the intended behavior of the protocol.

One of the first papers in which heterogeneity was taken into account concerned *directional gossiping* [22]. An important observation is that in many networks neighbors may be reachable through multiple paths. As a consequence, well-reachable neighbors have a higher chance of receiving a message than neighbors with only one or a few paths. Consequently, it is more efficient to give a higher probability to selecting peers that are less reachable for the node that is initiating a gossip exchange.

Equally important is that when gossiping between nodes of which some are operating behind firewalls, we need to take into account that links become asymmetric. In other words, although a node behind a firewall can initiate a gossip exchange, it will generally be impossible to select such a node for a data exchange. This situation has been studied by Drost et al. [7]. The problem with many gossip implementations is that a nonreachable node is considered to be faulty, which need not be the case with firewalled nodes. Instead of dropping a reference when a connection setup fails, another node is selected from a *fallback cache*, which consists only of references to peers to which a connection has once succeeded. The effect is that nodes behind firewalls are continued to be seen by other peers, and can thus be selected by applications.

5. TOPOLOGY CONSTRUCTION

For practical reasons, each node in a gossip-based system maintains only a *partial view* of the complete set of nodes currently in a system. This partial view is stored as a local cache and is maintained by exchanging entries from caches of neighboring nodes. The result is an overlay network in which a directed link from A to B exists only if A has a reference to B in its local cache. To guarantee that the selection of a peer (through the operation `selectPeer` in Figure 1) follows a uniform distribution, implementations of a peer sampling service show that induced overlays strongly resemble traditional random graphs [3].

However, as we also mentioned, we sometimes need to exert a stronger control when constructing overlays. Again, gossiping turns out to be an excellent vehicle to do so. By introducing a proximity metric, which may be application specific, it turns out that many different overlay topologies can be constructed.

Jelasity et al. [15] propose to use a ranking function that reflects a preference for keeping certain references in the local cache. For example, if we assume that every node has a unique numerical identifier $i \in \{0,\ldots,N-1\}$, a simple ranking is to let a peer with ID j keep only a reference to the peer with the largest IDs $i < j$, and one to the peer with the smallest IDs $k > j$ (with modulo N arithmetic). In effect, we are creating a bidirectional ring. Other (and more robust) schemes can easily be envisaged.

Note that in order to preferentially select peers, it is essential that a node is continuously offered peers from the whole network. For this reason, gossip-based topology construction may be layered on top of a peer sampling service that returns uniformly and randomly selected peers. Feeding the topology construction algorithm enables a node to keep on discovering relevant peers.

This brings us to the following classification:

- **Peer selection:** the set of peers is ranked according to a given ranking function and the gossip target is chosen at random among the first half of peers from the local cache.
- **Data exchanged:** the data exchanged between peers are lists of peers.
- **Data processing:** upon receipt of the message, the receiving peer merges the received list with its own, ranks the elements according to the given ranking function, and keeps the first elements (up to the size required).

Structured overlays can be constructed by using geometry-based proximity metrics applied to the identifier space for nodes. Clearly, these metrics are application independent. Along the same lines, one can focus on node-specific properties, such as availability. Sacha et al. [25] propose to assign a utility to each node and to then subsequently let nodes with similar utility keep references to each other. The effect is a *gradient network* [27], which is characterized by the fact that directed paths tend to end in the node with the highest utility. Again, note that also these networks require to be "fed" by an underlying peer sampling service.

The same approach may be applied to cluster peers according to application-specific metrics, such as *semantic similarity*. Such similarity can, for example, be represented by the fraction of files that two peers have in common, expressing that the two peers have similar taste (see, e.g., [31]). Such clustering can help in speeding up search queries, or passing notifications as in publish-subscribe systems. The main difference with many application-independent metrics is that it may be impossible to maintain connectivity of the overlay as peers may have strong mutual preferences, leading to "islands" of peers. In addition to feeding the topology construction algorithm, the underlying peer sampling service also prevents the network from becoming disconnected.

6. RESOURCE MANAGEMENT

Topology construction can be viewed as a special case of systemwide resource management. More specifically, gossiping has also been

used to monitor the state of nodes and resources in large-scale distributed systems. The most common monitoring service has always been failure detection [29,32], but it has also been shown useful for monitoring other aspects of resources [26]. The most important aspect in gossip-based monitoring is that nodes build up a consistent view on specific nodes. The applications can be characterized as follows:

- **Peer selection:** each peer P chooses periodically a gossip target Q from its current set of neighbors.
- **Data exchanged:** the data exchanged between peers is status information on other peers (e.g., last reported *alive* message).
- **Data processing:** upon receipt of the message, the receiving peer merges the received information with its own status information on nodes, effectively updating its view of other nodes.

From this classification, one can view resource monitoring as a specific form of data dissemination. An interesting case is failure detection, which can be either explicit or implicit. In the explicit approach [26, 29], nodes send heartbeat messages to each other from which they draw conclusions regarding failures. In an implicit approach, whenever a peer cannot be contacted after its selection from the local cache (through the `selectPeer()` function), a node will drop that peer from its local cache. The net effect is that eventually references to failed nodes will have been completely removed from the system. The speed by which this removal converges depends strongly on the actual gossiping strategy [14].

Strongly related to resource monitoring is aggregation of resource information, which has been studied extensively for systems such as Astrolabe [28] and GEMS [26]. We return to aggregation in the next section.

Besides monitoring, gossiping can also be used for *resource allocation*. The underlying idea is that several applications should be able to be executed simultaneously on a given collection of nodes. The main problem that needs to be addressed is allocating nodes to applications. When dealing with very large collections of nodes, taking a traditional centralized approach may turn the allocation component into a serious bottleneck. It has been shown that gossiping can provide a fully decentralized solution to this problem.

For example, a gossip-based approach to estimate to which *slice* of a collection a node belongs has been proposed in [16] (built on top of the peer sampling service for the reasons invoked earlier). Each node is expected to hold a specific value related to a given attribute (bandwidth, storage, uptime, etc.) along with an associated random $ID \in [0,1]$. The basic idea is to sort the nodes according to both the random ID and the attribute value. The gossip-based protocol is used to have peers swapping their random ID until the order of IDs reflects the order of the attribute value. Assuming that IDs are actually distributed uniformly, eventually a peer holding a random ID larger than 0.9 knows that it belongs to the top 10% of the peers.

In case an absolute number is required (i.e., an application requires N nodes to be allocated), the relative positioning of a node can be combined with an estimate of the network size using, for example, the gossip-based aggregation techniques we discuss below. In that case, a node can determine its absolute ranking position and decide whether it should be allocated to the application or not.

7. COMPUTATIONS

As we mentioned, resource management is closely related to aggregation. Gossiping has also found its way in the area of aggregate computations in very large distributed systems. Such systems are generally characterized by the fact that (1) centralized components cannot be deployed, (2) the set of members changes almost continuously, and (3) the communication topology that ties the nodes together may not be globally known. In addition, when dealing with wireless systems (such as sensornets), computational power may be limited. Gossiping has shown to be an efficient tool for computing aggregates, such as sums, averages, and maximum and minimum values. It is also an appropriate tool when one is not interested in the results of a single node, but when aggregate data (in a part) of the network is what matter.

The main difference with the gossiping solutions we have discussed so far, is that the data processing is now a crucial element to consider. This brings us to the following characterization:

- **Peer selection:** each node periodically chooses one other peer uniformly at random from the entire set of currently available (i.e., alive) peers.
- **Data exchanged:** an application-specific data element is copied from one peer to another.
- **Data processing:** a new data value is computed from exchanged information, and which will then be used in a next gossip exchange.

As an example, consider the computation of an average value. Each node i stores a single numeric value v_i and initializes the average value with its initial value. Upon gossip a peer i and j exchange their local value v_i and v_j and adjust it to

$$v_i, v_j \leftarrow (v_i + v_j)/2$$

Such operations guarantee the conservation of the weight (i.e., $\sum v_k$ remains the same) while bringing each value v_i closer to the average. It can be shown that the variance decreases exponentially.

Several groups have devised gossip-based solutions for aggregate information, with first results published by Kempe et al. [20] and Jelasity et al. [17]. Dynamically configuring aggregation functions has been explored in the aforementioned Astrolabe system [28]. Recent publications have concentrated on optimizing algorithms for improving convergence speed and number of messages to exchange (see, e.g., [4, 19]). Again, in most cases it is assumed that peers can uniformly at random select another live peer from the entire network.

8. DISCUSSION

The purpose of this paper was to demonstrate that using a simple generic substrate of gossip, arbitrary functions might be ensured by just changing the way gossip targets are chosen and the content and processing of the information exchanged. Gossip-based algorithms are now at the point that they represent a mature technology that has been identified as a powerful tool to build and maintain distributed systems. Introduced for very specific reliable dissemination purposes, it turns out that the key to scalability in distributed computing is to keep disseminating information around. Yet many challenges remain in this area, many of which are discussed in accompanying papers in this special issue and for which reason we shall not address them here.

9. REFERENCES

[1] Z. Bar-Yossef, R. Friedman, and G. Kliot. "RaWMS - Random Walk based Lightweight Membership Service for Wireless Ad Hoc Networks." In *Proc. Seventh Int'l Symp. Mobile Ad Hoc Networking and Computing*, pp. 238 – 249, May 2006. ACM Press, New York, NY.

[2] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. "Bimodal Multicast." *ACM Trans. Comp. Syst.*, 17(2):41–88, May 1999.

[3] B. Bollobas. *Random Graphs*. Cambridge University Press, Cambridge, UK, 2nd edition, 2001.

[4] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. "Randomized Gossip Algorithms." *IEEE Transactions on Information Theory*, 52(6):2508–2530, June 2006.

[5] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. "Epidemic Algorithms for Replicated Database Maintenance." In *Proc. Sixth Symp. on Principles of Distributed Computing*, pp. 1–12, Aug. 1987. ACM.

[6] M. Deshpande, B. Xing, I. Lazaridis, B. Hore, N. Venkatasubramanian, and S. Mehrotra. "CREW: A Gossip-based Flash-Dissemination System." In *Proc. 26th Int'l Conf. on Distributed Computing Systems*, July 2006. IEEE Computer Society Press, Los Alamitos, CA.

[7] N. Drost, E. Ogston, R. V. van Nieuwpoort, and H. E. Bal. "ARRG: Real-World Gossiping." In *Proc. 16th Int'l Sympòn High Performance Distributed Computing*, July 2007. IEEE Computer Society Press, Los Alamitos, CA.

[8] P. Eugster, R. Guerraoui, S. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. "Lightweight Probabilistic Broadcast." *ACM Trans. Comp. Syst.*, 21(4):341–374, Dec. 2003.

[9] P. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. "Epidemic Information Dissemination in Distributed Systems." *IEEE Computer*, 37(5):60–67, May 2004.

[10] 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.

[11] D. Gavidia, S. Voulgaris, and M. van Steen. "A Gossip-based Distributed News Service for Wireless Mesh Networks." In *Proc. Third Int'l Conf. Wireless On-demand Network Systems & Services (WONS)*, Jan. 2006. IEEE Computer Society Press, Los Alamitos, CA.

[12] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. "The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations." In *Proc. Middleware 2004*, volume 3231 of *Lect. Notes Comp. Sc.*, pp. 79–98, Oct. 2004. Springer-Verlag, Berlin.

[13] M. Jelasity, W. Kowalczyk, and M. van Steen. "Newscast Computing." Technical Report IR-CS-006, Vrije Universiteit Amsterdam, Department of Computer Science, 2003.

[14] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. "Gossip-based Peer Sampling." *ACM Trans. Comp. Syst.*, 25(3), Aug. 2007.

[15] M. Jelasity and O. Babaoglu. "T-Man: Gossip-based Overlay Topology Management." In *Proc. Third Int'l Workshop Eng. Self-Organising App.*, volume 3910 of *Lect. Notes Comp. Sc.*, pp. 1–15, June 2006. Springer-Verlag, Berlin.

[16] M. Jelasity and A.-M. Kermarrec. "Ordered Slicing of Very Large-Scale Overlay Networks." In *Proc. Sixth Int'l Conf. Peer-to-Peer Comput.*, pp. 117–124, Sept. 2006. IEEE Computer Society Press, Los Alamitos, CA.

[17] M. Jelasity, A. Montresor, and O. Babaoglu. "Gossip-based Aggregation in Large Dynamic Networks." *ACM Trans. Comp. Syst.*, 23(3):219–252, Aug. 2005.

[18] G.-P. Jesi, D. Gavidia, C. Gamage, and M. van Steen. "A Secure Peer Sampling Service." In *Proc. Fourth Int'l Conf. Autonomic Comput.*, 2007.

[19] S. Kashyap, S. Deb, K. V. M. Naidu, R. Rastogi, and A. Srinivasan. "Efficient gossip-based aggregate computation." In *Proc. 25th Symp. on Principles of Database Systems*, pp. 308–317, 2006. ACM Press, New York, NY.

[20] D. Kempe, A. Dobra, and J. Gehrke. "Gossip-Based Computation of Aggregate Information." In *Proc. 44th Symp. Foundations Computer Science*, pp. 482–491, Oct. 2003. IEEE Computer Society Press, Los Alamitos, CA.

[21] 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.

[22] M.-J. Lin and K. Marzullo. "Directional Gossip: Gossip in a Wide-Area Network." In *Proc. Third European Dependable Computing Conf.*, volume 1667 of *Lect. Notes Comp. Sc.*, pp. 364–379. Springer-Verlag, Berlin, Sept. 1999.

[23] R. Melamed and I. Keidar. "Araneola: A Scalable Reliable Multicast System for Dynamic Environments." In *Proc. Third Int'l Symp. Network Computing and Applications*, pp. 5–14, 2004. IEEE Computer Society Press, Los Alamitos, CA.

[24] J. Patel, I. Gupta, and N. Contractorn. "JetStream: Achieving Predictable Gossip Dissemination by Leveraging Social Network Principles." In *Proc. Fifth Int'l Symp. Network Computing and Applications*, pp. 32–39, July 2006. IEEE Computer Society Press, Los Alamitos, CA.

[25] J. Sacha, J. Dowling, R. Cunningham, and R. Meier. "Discovery of Stable Peers in a Self-Organising Peer-to-Peer Gradient Topology." In *Proc. Sixth Int'l Conf. Distributed Applications and Interoperable Systems*, volume 4025 of *Lect. Notes Comp. Sc.*, pp. 70–83, June 2006. Springer-Verlag, Berlin.

[26] R. Subramaniyan, P. Raman, A. D. George, and M. Radlinski. "GEMS: Gossip-Enabled Monitoring Service for Scalable Heterogeneous Distributed Systems." *Cluster Comput.*, 9(1):101–120, 2006.

[27] Z. Toroczkai and K. Bassler. "Network dynamics: Jamming is limited in scale-free systems." *Nature*, 428:716, Apr. 2004.

[28] R. van Renesse, K. Birman, and W. Vogels. "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining." *ACM Trans. Comp. Syst.*, 21(2):164–206, May 2003.

[29] R. van Renesse, Y. Minsky, and M. Hayden. "A Gossip-Style Failure Detection Service." In *Proc. Middleware '98*, pp. 55–70, Sept. 1998. IFIP.

[30] S. Voulgaris, D. Gavidia, and M. van Steen. "CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays." *J. Netw. & Syst. Mgt.*, 13(2):197–217, June 2005.

[31] S. Voulgaris and M. van Steen. "Epidemic-style Management of Semantic Overlays for Content-Based Searching." In *Proc. 11th Int'l Conf. Parallel and Distributed Computing (Euro-Par)*, volume 3648 of *Lect. Notes Comp. Sc.*, pp. 1143–1152, Sept. 2005. Springer-Verlag, Berlin.

[32] S. Q. Zhuang, D. Geels, I. Stoica, and R. H. Katz. "On Failure Detection Algorithms in Overlay Networks." In *Proc. 24th INFOCOM Conf.*, Mar. 2005. IEEE Computer Society Press, Los Alamitos, CA.