Gossip-based Networking for Internet-Scale Distributed Systems

Etienne Rivière[1] and Spyros Voulgaris[2]

[1] Computer Science Department, Université de Neuchâtel, Switzerland.
[2] Vrije Universiteit Amsterdam, The Netherlands.
etienne.riviere@unine.ch, spyros@cs.vu.nl

Abstract. In the era of Internet-scale applications, an increasing number of services are distributed over pools of thousands to millions of networked computers. Along with the obvious advantages in performance and capacity, such a massive scale comes also with challenges. Continuous changes in the system become the norm rather than the exception, either because of inevitable hardware failures or merely due to standard maintenance and upgrading procedures. Rather than trying to impose rigid control on the massive pools of resources, we should equip Internet-scale applications with enough flexibility to work around inevitable faults. In that front, gossiping protocols have emerged as a promising component due to their highly desirable properties: self-healing, self-organizing, symmetric, immensely scalable, and simple.

Through visiting a representative set of fundamental gossiping protocols, this paper provides insight on the principles that govern their behavior. By focusing on the rationale and incentives behind gossiping protocols, we introduce the reader to the alternative way of managing massive scale systems through gossiping, and we intrigue her or his interest to delve deeper into the subject by providing an extensive list of pointers.

1 Introduction

With the advent of worldwide networks and the Internet, computer systems have been going through an unprecedented shift in scale and complexity. Services that are distributed on thousands, if not millions, of machines, are gradually becoming commonplace.

Peer-to-peer systems are a well known example of massively distributed services. They employ end-user computers, often in the order of thousands or even millions. Each node acts both as a client of the provided service and at the same time as a server, collaborating with other peers to provide this same service. Examples of this first class of massive-scale systems include file sharing networks [1–3], collaborative search engines [4], multicast systems [5, 6], publish/subscribe [7, 8], etc.

A second area in which scale has grown from large to massive is that of *data centers*, providing services in a more traditional client-server fashion. For the major companies providing Internet services, this is largely due to the need to serve more and more complex applications to an ever-increasing number of clients. Note that the shift in scale does not concern only large companies. The advent of data center externalization and virtualization offered by the *cloud computing* paradigm allows short- and medium-scale companies to benefit from potentially very large infrastructures. In the following section we describe the important characteristics of large-scale distributed systems.

1.1 Challenges in Massive-Scale Distributed Systems

Both peer-to-peer systems and large data centers constitute large-scale distributed systems. Their massive scale, while allowing for an unprecedented capacity and performance potential, also comes with certain challenges. These challenges must be addressed from the very conception of the system design, supporting software, and applications.

First, *centralized management* is impractical for systems of such scale, due to the large number of entities involved, be they computers or data items. Tracking membership (which nodes join or leave the network), locating data and services among millions of nodes, and generally monitoring the system, are no longer possible to realize in a centralized manner. Such an omniscient node would have to maintain a *global and consistent view* of the system, becoming a bottleneck for system performance. Additionally it would constitute a single point of failure and a perfect target for attacks. Spreading the load of such operations on multiple nodes is much more appropriate in large-scale systems. Then, each node is responsible for only a fraction of global system knowledge, called the node's *local view* of the system. Maintaining individual local views is less complex than maintaining a single, centralized, global, and consistent view. It allows for greater scalability due to the elimination of single points of failure.

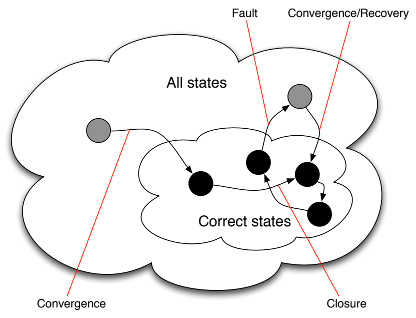


Fig. 1: Principle of self-stabilization.

Second, systems of such scale are inherently of *highly dynamic nature*, either due to nodes leaving, joining, or merely failing. If in small- and medium-scale distributed systems faults were considered as exceptions and were mitigated by traditional reparation mechanisms (e.g., checkpointing and restarting individual nodes or the whole application), in large-scale systems faults must be considered as the norm. The number of nodes joining and leaving the system during any period of time is expected to be high. The rate at which nodes join and leave is often referred to as the *node churn* (or *churn*) of the system. High level of churn imposes that the removal of failed/leaved nodes and the insertion of newly joined ones must be integrated at the core mechanisms used for building large-scale applications and systems. The reader may find experimental studies of churn in real systems in [9, 10].

Third, the *high complexity* of large-scale distributed systems make their explicit management in case of misconfiguration or faults totally impractical. It is thus vital that algorithms and protocols involved exhibit self-* properties: self-configuration, self-stabilization and self-optimization are some but few examples of these properties. The self-organization property, illustrated by Figure 1 ensures that divergence from a correct state of the system (e.g., only valid and non-failed nodes are available as potential communication partners in nodes' views) is possible but automatic recovery eventually happens as part of the protocols' operation and not due to the help of some external mechanism (e.g., human operation or restarting of the system).

This paper focuses on the use of the gossip-based communication paradigm for building large-scale applications.[1]

1.2 Outline

We seek to survey, motivate, and exemplify a representative set of gossip-based building blocks for large-scale systems. These mechanisms are meant to be integrated as components of large and complex systems, and we give examples of such integration whenever applicable.

The remaining of this document is organized as follows.

- Section 2 introduces gossiping, from its seminal use to modern version.
- Section 3 presents gossiping protocols that emerge random overlay networks.
- Section 4 presents gossiping protocols that emerge structured overlay networks.
- Section 5 presents gossiping protocols for overlay slicing.
- Section 6 presents gossiping protocols for data aggregation.
- Section 7 presents additional uses of gossiping protocol for large scale distributed systems.
- And finally, Section 8 concludes our work.

Note that for each section, we do not only provide the description and motivation of the corresponding protocol but also provide additional links that are meant to guide the readings of a reader wishing to delve deeper into the subject.

2 Gossiping Protocols: From Traditional to Modern

Gossiping, also known as *epidemic*, protocols are not a new concept in computer science. They have been around for nearly three decades. However, the daunting Internet growth has created new challenges, and has shaped gossiping protocols in a new way.

This section introduces the seminal gossiping system, Clearinghouse, as well as the modern ones, explaining the reasons that led to the latter.

[1] Another introduction to gossip-based networking can be found in [11].

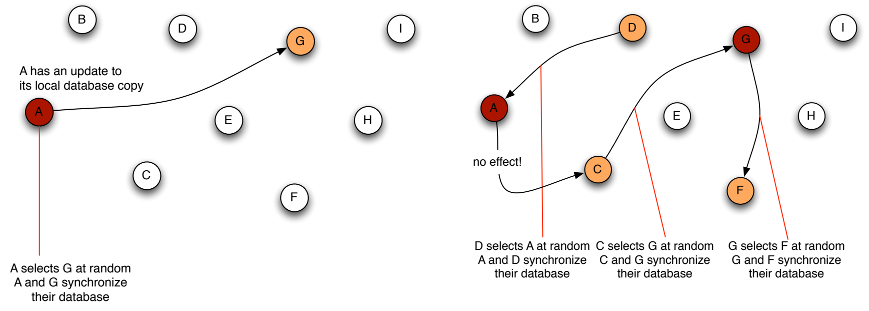
2.1 Clearinghouse: Synchronization of Database Replicas

The seminal paper by Demers *et al.* on the Clearinghouse project [12] introduced the use of gossiping in medium-scale networks for propagating updates to replicas of a database. The mechanisms introduced in this work still lie at the core of all proposals for gossip-based data dissemination.

The Clearinghouse project [12] involved a database replicated across a set of a few hundred replicas, dispersed across diverse geographic areas, where updates were allowed at any one replica of the system. Maintaining consistency across all replicas in the face of updates was a major objective. More accurately, the objective was to maintain consistency across all *alive* replicas, given that individual replicas would occasionally fail, as is the norm in any large scale system. Furthermore, the system should be highly failure resilient, that is, the failure of any node or set of nodes should not hinder the propagation of updates across the remaining set of alive replicas.

This work introduced two gossiping algorithms for the propagation of updates to all replicas, namely Anti-Entropy and Rumor Mongering. In *Anti-Entropy*, each node "gossips" periodically, that is, it periodically picks a *random* other node among all alive ones, and they exchange some data to synchronize their replicas. Figure 2 illustrates an example of anti-entropy.

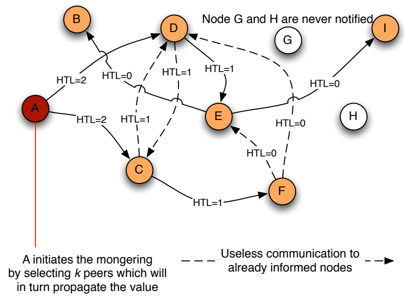
In *Rumor Mongering*, nodes are initially "ignorant". When a node has a new update, it becomes a "hot rumor". While a node holds a hot rumor, it periodically selects a random node among the alive ones, and forwards the update to it. After having forwarded the update to a number of nodes that were already hot rumors, it stops being a hot rumor and, thus, maintains the update without forwarding it further. Figure 3 shows an example of rumor mongering.



(a) Initial state: node A has a local update that needs to be propagated

(b) Propagation of the update through anti-entropy operations

Fig. 2: Propagation of the update from node A using *anti-entropy*.



B
D
Node G and H are never notified
I
G
HTL=2
HTL=0
HTL=1
HTL=1
HTL=0
A
HTL=2
HTL=1
E
H
HTL=0
HTL=0
C
F
HTL=1
A initiates the mongering by selecting *k* peers which will in turn propagate the value
Useless communication to already informed nodes

Fig. 3: Propagation of the update from node A using *rumor mongering*.

2.2 Today's challenges

Gossiping protocols have shown to possess a number of desirable properties for data dissemination, notably fast convergence, symmetric load sharing, robustness, and resilience to failures. The same applies for data aggregation, node clustering, network slicing, and other forms of decentralized data manipulation, as we will see in subsequent sections. We will be referring to these gossiping protocols as *traditional gossiping*.

However, traditional gossiping protocols are based on a common assumption: the *complete view of the network* by every node. This is in fact dictated by the need of every node to periodically sample the network for a random other node to gossip with. Although this assumption is acceptable for fixed sets of up to a few hundred machines, it becomes a serious obstacle in networks that scale to tens of thousands or millions of nodes. This is clear given the dynamicity inherent in systems of such scale, given the probability of nodes to crash or to voluntarily leave or join. Imagine the join of a single node triggering the generation of millions of messages, to inform the millions of other nodes of its existence.

Ironically enough, the solution to the complete network view assumption of traditional gossiping protocols is given by a new generation of gossiping protocols that handles overlay management. These protocols are generally known as the Peer Sampling Service and will be studied in Section 3.

In the Peer Sampling Service, nodes maintain just a *partial view* of the network, rather than a complete view. Periodically, a node picks a neighbor from its partial view. They exchange some data, which more specifically is *membership information*. That is, they send each other some of the neighbors they have. This way nodes refresh their partial views, and update them with new information on participating nodes. Deferring certain details to Section 3, these partial views can provide nodes with links to other nodes picked uniformly at random out of the whole network, bypassing the need for complete view of the network.

Figure 4 presents the model of executing traditional gossiping protocols (such as gossip-based dissemination) on very large scale systems. Each node executes

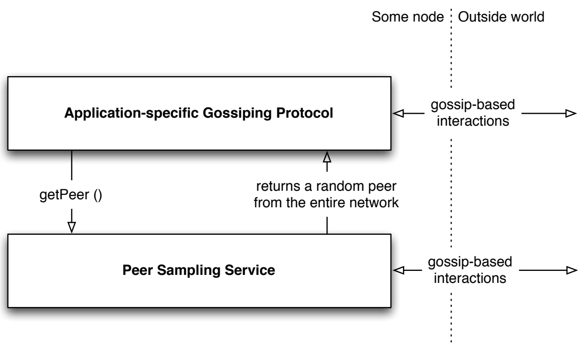

Some node
Outside world
Application-specific Gossiping Protocol
gossip-based interactions
getPeer ()
returns a random peer from the entire network
Peer Sampling Service
gossip-based interactions

Fig. 4: Gossip-based dissemination using a Gossip-based Peer Sampling Service to implement its selectPartner() operation.

(at least) two gossiping protocols. The first one (top) is the traditional gossiping protocol needed by a certain application. The second one (bottom) is the Peer Sampling Service, used to manage membership and serving as a source of uniformly randomly selected nodes from the whole network for the first gossiping protocol.

2.3 The Gossiping Framework

We define a gossiping framework, which is generic enough to apply to all gossiping protocols, both traditional and peer sampling ones. Each node (or peer) in the system maintains a local, often partial, view of the system. This view can be of various types: a replica of a database, a set of published events, localization information, or even sets of other peers participating in the system.

Gossip interactions are pair-wise periodic exchanges of data among peers. Each peer periodically selects a partner to gossip with, amongst the nodes it knows in the system (selectPartner() function). Then, it selects the information from its local view that will be exchanged with this partner (selectToSend() function). The partner proceeds to the same operation, which results in a bidirectional exchange between the partners. Thereafter, each of the two decides on its new local view based on the information it had before the exchange (available in its view) and the one received (in *resp* or *req* buffers). Additionally, extra actions (e.g., notifying the upper layers that the local view has changed) can be taken depending on the protocol considered.

All the protocols we present in this document follow this very simple algorithmic framework, where no global vision of the system is assumed whatsoever, and where only local update decisions based on the local view and the received information are key to local convergence, and as we shall see, to the global convergence in the system as a whole.

Gossip-based networking is often based on probabilistic decisions (e.g., for the selection of the partner, the selection of the data to send, etc.). The local

```
(on P) do every δ time units                 (on Q) reception of request from P
    // select exchange partner
    Q ← selectPartner()
    // select exchange content
    buf ← selectToSend()
    // proceed to exchange                       // receive request
    send buf to Q                        →       receiveFrom(P,req)
                                                 // select exchange content
                                                 buf ← selectToSend()
    // wait for response                         // proceed to exchange
    receiveFrom(Q,resp)                  ←       send buf to P
    // decide on a new view                      // decide on a new view
    view ← selectToKeep(view,resp)               view ← selectToKeep(view,req)
    // (optional) specific actions               // (optional) specific actions
    processView(view)                            processView(view)
end                                          end
```

Algorithm 1: Gossip-based interaction framework

decisions made by each node are often driven by local convergence criteria. The convergence to a *better local view* according to these criteria leads to global convergence: the state of the system as a whole, when carefully engineered, converges to an expected property that in fine allows implementing the desired service, without any assumptions on one node having a global view of the system. Moreover, the many interactions between nodes in the system support a certain level of redundancy, which is key to robustness: the loss of some of the interactions (due to failed nodes, message loss, etc.) can impact the convergence speed but seldom impact the eventual convergence. The local convergence vs. global state is the key for the self-stabilization, self-repair and self-configuration offered by gossip-based protocols.

2.4 Further reading

A number of systems employ gossiping techniques. Many are focused on scalable group communication and multicast [13–19]. Others have focused on data aggregation [20], live streaming of video [21], maintenance of Distributed Hash Table routing tables [22], social network links to propagate data more efficiently [23], or specific network characteristics for gossiping with lower cost [24]. Finally, a number of researchers have worked on theoretical analysis of gossiping properties [25, 26].

Dynamo [27] is a distinguished example of a gossip-based system applied in an industrial environment. More specifically, it is used in Amazon's infrastructure to spread indexing information across all servers involved in a Distributed Hash Table handling crucial data, such as customer records.

3 Random Overlays

As explained in the previous section, a number of gossiping protocols rely on the ability to select random samples of alive nodes from the network. Clearly, providing each node with a complete view of the network is unrealistic for very large scale networks, particularly in the face of node churn, that is, nodes joining and leaving. Similarly, building a centralized service for maintaining such information is not a viable solution either.

Along these lines, a class of entirely *decentralized* protocols has emerged to collaboratively maintain membership information. These protocols are collectively known as the Peer Sampling Service [28], and they are based on a gossiping framework themselves.

In a nutshell, each node maintains a small (e.g., a few dozen nodes) *partial view* of the network, and periodically refreshes its partial view by gossiping with one of its current neighbors. It turns out that, by following this gossiping paradigm with certain policies, the partial view of each node constitutes a periodically refreshed sliding random subset of all nodes in the network. Making a random selection out of a random subset of all nodes is equivalent to making a random selection out of *all* nodes. This is exactly the assumption which traditional gossiping protocols are based on: sampling peers from the whole network at random. It becomes now evident that, by employing a Peer Sampling protocol, a node is able to select peers at random out of the whole network by means of a *local operation*. This essentially overcomes the scalability barrier for executing traditional gossiping protocols in very large scale networks.

Sampling peers uniformly at random is not the sole utility of Peer Sampling protocols. It turns out that by running a Peer Sampling protocol on a large set of nodes, the nodes self-organize in an overlay that shares a lot of similarities with *random graphs* and inherits most of their properties. Namely, the overlay becomes very robust and extremely resilient to failures, in the sense that failures, even large scale ones involving much more than half of the nodes do not put the *connectivity* of the overlay at risk.

Peer Sampling refers to a *family* of protocols, whose design space is extensively analyzed in [28]. This analysis is out of the scope of this paper. Instead, we will focus on the two most prominent instance protocols of the Peer Sampling Service, namely Newscast [29,30] and Cyclon [31].

3.1 The Newscast Protocol

In Newscast each node maintains a small partial view of the network of length ℓ, and periodically picks a random node from it to gossip with. The two nodes share with each other their views, including newly generated links to themselves.

The principal design objective in Newscast is to keep overlay links *fresh* by giving newer links priority over older ones. In doing so, Newscast policies consider the *age* of a link, that is, the time elapsed since the link was injected into the network by the node it points at.

Link ages can be precisely determined if links are timestamped at generation time, assuming network-wide time synchronization is possible. Otherwise, they can be sufficiently approximated by associating each link with an *age* counter. Here we follow the second—more realistic—approach.

In terms of our gossiping framework shown in Algorithm 1, NEWSCAST implements the following policies:

selectPartner() Select a random node from the view. Also, increase the age of all nodes in the view by one.
selectToSend() Select all nodes from the view, and append own link with age 0.
selectToKeep() Merge received links with own view, sort by the age, and keep the ℓ freshest ones, including no more than one link per node.

Note that after a gossip exchange between nodes P and Q, the two nodes have the same view, except for a link to each other. This, however, is a temporary situation, as next time they gossip (either initiating it or being contacted by others) their views will most likely be mingled with views of different other nodes.

3.2 The CYCLON Protocol

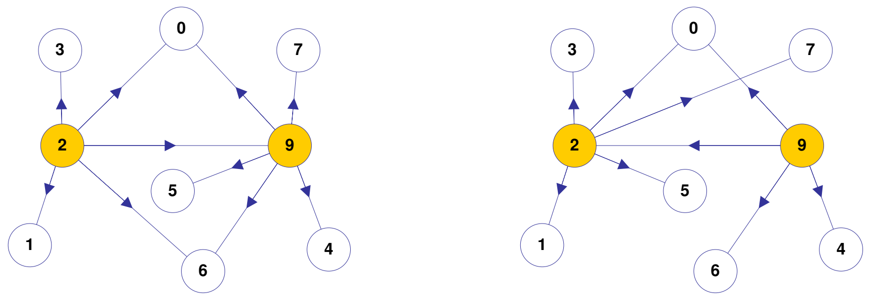
CYCLON [31] is a Peer Sampling protocol, where view refreshing is based on *exchanging contacts*. That is, a node sends a few links to its gossiping partner, and receives the same number of links in return. Each node accommodates *all* received links by discarding the links it just sent away. The main intuition behind this operation is to mix links, resulting in overlays resembling random graphs.

Like in NEWSCAST, the age of a link is also utilized in CYCLON, alas in a different way. It is used to select which neighbor to gossip with, rather than to select which links to keep in the view. This serves two fundamental goals, that will be discussed later in this section.

Let g denote the number of links traded in a gossip exchange. In terms of our generic gossiping framework shown in Algorithm 1, CYCLON employs the following policies:

selectPartner() Select the node whose link has the *oldest* age. Also increase the age of all links in the view by one.
selectToSend() Select g random links, and *remove* them from the view. If this is the initiating node, the link selected in selectPartner() should be among these g links, and after removal it should be substituted by a link to itself, with age 0.
selectToKeep() Add all g received links to the view, by replacing the g links selected in selectToSend().

Note that after a gossip exchange, the link between the two nodes involved changes direction, as illustrated in Figure 5. E.g., if node P knows node Q and selects it as a gossip partner, after the gossip exchange P will have discarded Q from its view, while Q will deterministically know P. In other words, P's indegree



(a) Before swapping $2 \rightarrow 9 : \{2, 0, 6\}$ $2 \leftarrow 9 : \{0, 5, 7\}$ (b) After swapping

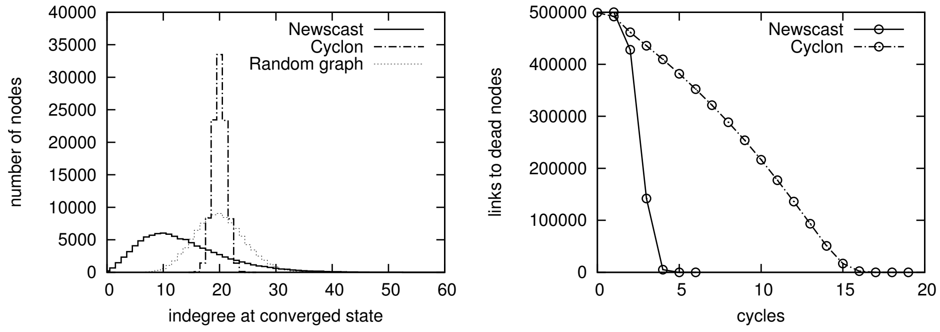
Fig. 5: An example of swapping between nodes 2 and 9. Note that, among other changes, the link between 2 and 9 reverses direction.

was increased by one, while Q's indegree dropped by one. Third nodes' indegrees have not been altered, even if some nodes changed from being neighbors of P to being neighbors of Q or the other way around.

This provides for an interesting self-adaptive mechanism for indegree control. A node's indegree increases when it initiates gossiping, which happens at constant intervals (due to CYCLON's periodic operation). However it decreases when it is contacted by another node, which happens at a frequency proportional to the node's indegree: the more known you are, the more gossip exchanges you will be invited to in a given period. Statistically, if exactly ℓ other nodes know you, you will be contacted exactly once per gossiping period, and as you will also initiate exactly one gossip exchange, your indegree will remain stable and equal to ℓ. However, the lower a node's indegree is below ℓ, the faster it will grow higher, while the higher it is above ℓ, the faster it will drop lower. This leads to a natural equilibrium of indegrees, a self-adaptive mechanism for balancing links evenly across all nodes.

Regarding the selection of the *oldest* link for a gossip exchange, as mentioned earlier it serves two goals. The first one is to limit the time a link can be passed around until it is chosen by some node for a gossip exchange. Since by selecting a link for a gossip exchange also removes the link from the network, selecting always the oldest one prevents links to dead nodes from lingering around indefinitely. This results in a more up-to-date overlay at any given moment.

The second—and far less obvious—goal is to impose a predictable lifetime on each link, in order to control the number of existing links to a given node at any time. During one gossiping period, a node P initiates *one* gossip exchange, therefore pushing its oldest age link out of its view, and increasing all other links' ages by one. Also, P is contacted on average by *one* other node for a gossip exchange, thus accepting a new link of age 0 in its view. As a result, a node's view contains on average one link of each age, from 0 to $\ell - 1$. This means that links selected for gossip exchanges are typically of age around $\ell - 1$.



(a) Indegree distributions (b) Links to dead nodes

Fig. 6: Some main properties of NEWSCAST and CYCLON.

In other words, a pointer has a lifetime of about ℓ cycles. This implies that besides the constant birth rate of links, their death rate is also close to constant, which results in an almost constant population of ℓ links for each node. This is validated by extensive simulations.

3.3 Properties

Although a detailed discussion on properties of these protocols is out of the scope of this paper, it is worth noting the effect that certain policies have on some of the properties.

Figure 6(a) shows the indegree distribution for NEWSCAST and CYCLON, running on 100K nodes with view length $\ell = 20$ for both protocols. The self-adaptive mechanism for indegree control in CYCLON detailed in Section 3.2, becomes evident in this graph. Indegrees follow a very narrow distribution, centered around the nodes' outdegree (i.e., their view length). Each node has an indegree of $\ell \pm 3$. This results in higher robustness of the overlay in the face of errors, as no node has indegree of lower than 17, which means there are no "weak links" in the topology. In NEWSCAST, the indegree distribution is very much spread out, which is an expected outcome of the nature of gossip interactions: a link to a node can either be duplicated on both gossiping partners or completely discarded, which results in high fluctuations of node indegrees. Specifically, we note that there are several nodes with indegree 0, becoming more vulnerable than others in the face of failures.

Another implication of the indegrees is that CYCLON offers much better load balancing, as nodes are invited to the same number of gossip exchanges per time unit. Contrary to that, in NEWSCAST there is a long tail of nodes with indegrees up to 60, which receive proportionally more gossip requests (and therefore load) per time unit. Nevertheless, extensive simulations in [28] showed that in NEWSCAST nodes fluctuate across the whole spectrum of indegrees withing a few gossiping periods, therefore in the long run load is fairly balanced among Newscast nodes as well.

Figure 6(b) shows the efficiency of each protocol in relieving the overlay of links to dead nodes. More specifically, in this experiment an overlay of 100K nodes with views of length $\ell = 20$ was let emerge by each protocol. At some point a very large failure was simulated by killing half of the network, and letting exactly 50K nodes survive. At that point, statistically $\ell/2$ links of each surviving node was pointing at dead nodes, accounting to 500K links. The graph in Figure 6(b) shows how the total number of links to dead nodes drops in each of the protocols, as a function of cycles (a time unit representing one gossiping period). Newscast's eagerness on keeping the *freshest* links is evident in this graph, as links to dead nodes vanish within six cycles. Contrary to that, Cyclon's self-adaptive control of links' lifetimes to ℓ gossiping periods is clear in this figure, as it takes exactly ℓ cycles for eliminating all links to dead nodes. With respect to this metric, Newscast is more efficient, and shows it is better at handling overlays of very high node churn.

3.4 Further reading

Work on overlay management for random overlays assumes the understanding of fundamental concepts such as random graphs [32], scale free networks [33], and small worlds [34, 35].

With respect to computer networks, Lpbcast [17] is a peer sampling protocol targeted at broadcasting messages. Scamp [36] is a reactive protocol that creates a static overlay that resembles a random graph. HyParView [37] is a gossiping protocol that creates overlays targeted at disseminating data in the face of high node churn.

4 Structured Overlays

Besides creating random overlays as a basis for massively decentralized systems, many distributed applications require structured overlays to operate on. Examples include, but are not limited to, clustering nodes based on interest (e.g., for a file sharing system), sorting nodes based on some metric (e.g., ID, load, memory, etc.), forming more complex structures (e.g., distributed hash tables, publish/subscribe systems, etc.) and more.

T-Man [38, 39] and Vicinity [40, 41] are two very similar gossiping protocols that provide a generic *topology construction* framework, suitable for the construction of a large variety of topologies. Through such a framework, nodes flexibly and efficiently self-organize in a completely autonomous fashion to a largely arbitrary structure. The advantages of these frameworks are their generic applicability, flexibility, and simplicity.

The target topology is defined by means of a *selection function*, which selects for each node the set of ℓ neighbors it should be linked to. This selection function is executed locally by every node to determine its neighbors. The selection is made based on some application-specific data associated with each node, which is called the node's *profile*. In topology construction protocols, each link to a node also carries that node's profile.

When fed with the complete list of nodes and their profiles, the selection function returns a node's optimal neighbors for the target topology. When fed with a subset of the nodes, it returns a selection of neighbors that brings the overlay as close to the target topology as possible.

Typically, the selection function is based on a globally defined peer proximity metric. Such a metric could include semantic similarity (see Section 4.1), ID-based sorting, domain name proximity, geographic- or latency-based proximity, etc.

Like in other gossiping protocols for overlay management, each node maintains a partial view of the network of length ℓ. As mentioned above, each link to a node also carries that node's profile. The protocol framework is similar to that of Peer Sampling Service protocols, except that nodes decide which links to keep in their views based on the selection function.

In the context of the hooks defined in our generic gossiping framework of Algorithm 1, topology construction protocols employ the following policies:

selectPartner() Select a random link from the view.
selectToSend() Select all links from the view, and append own link with own profile.
selectToKeep() Merge received links with own view and apply the selection function to determine which ℓ links to keep in the view, including no more than one link per node.

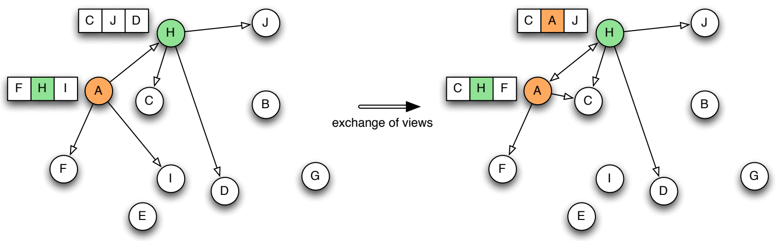


Fig. 7: Example of a Vicinity exchange and resulting views. The distance here is the 2D Euclidean distance and the nodes seek to find the $\ell = 3$ closest nodes in terms of this distance metric.

Figure 7 depicts a sample gossip exchange for topology construction, assuming a selection function that opts for minimizing the 2D Euclidean distance between nodes.

A key point in topology construction protocols is the *transitivity* exhibited by the selection function. In a selection function with high transitivity, the "better" a selection node Q is for node P, the more likely it is that Q's "good" selections are also "good" for P. This transitivity is essentially a correlation property between nodes sharing common neighbors, embodying the principle "my friend's friend is also my friend". Surely, this correlation is fuzzy and generally hard to quantify. It is more of a desired property rather than a hard requirement for topology construction. The higher the transitivity, the faster an overlay will converge to the desired topology.

There are two sides to topology construction. First, assuming some level of transitivity in the selection function, a peer should explore the nearby peers that its neighbors have found. In other words, if P_2 is in P_1's view, and P_3 is in P_2's view, it makes sense to check whether P_3 would also be suitable as a neighbor of P_1. Exploiting the transitivity of the selection function should then quickly lead to high-quality views. The way a node tries to improve its view resembles *hill-climbing* algorithms. However, instead of trying to locate a single optimal node, here the objective is to optimize the selection of a whole set of nodes, namely the view. In that respect, topology construction protocols can be thought of as distributed, collaborative hill-climbing algorithms.

Second, it is important that *all* nodes are examined. The problem with following transitivity alone is that a node will be eventually searching only in a single cluster of related peers, possibly missing out on other clusters of also related—but still unknown—peers, in a way similar to getting locked in a local maximum in hill-climbing algorithms. This calls for randomized candidate nodes to be considered too in building a node's view. This points directly at the two-layered approach depicted in Figure 4.

survey of overlay networks [42]

4.1 Test case: Interest-based Overlays

A direct application of VICINITY and T-MAN is the self-organization of nodes participating in a social network in a way that reflects their interests. Social networks inherently exhibit interest locality, that is, a user encompassing content on a certain topic is highly likely to address additional content on that same topic or related ones. Additionally, social networks tend to share many properties with Small Worlds [43, 44], that is, highly clustered networks of short diameter. High clustering (i.e., the friend of a friend is likely to be a friend) implies a highly transitive selection function in a topology construction protocol.

In [40], Voulgaris and van Steen apply VICINITY (with CYCLON as the underlying Peer Sampling Service instance) on nodes participating in the e-Donkey [45] file sharing network, to cluster them based on the degree of overlapping in their shared file collections. The selection function sorts a node's neighbors based on the number of shared files in common to the node's own collection, and selects the top ℓ ones. The experiments on 12,000 nodes indicate that by setting the view length to $\ell = 10$ neighbors, over 90% of the optimal relationships between nodes are established within 50 rounds of the protocol, starting from an arbitrarily connected initial topology. Furthermore, these 10 "semantic neighbors" per node prove to be capable of serving on average *one third* of each node's queries for new files, a ratio that significantly boosts decentralized search performance.

The same setting can be applied for automated *recommendations* in a decentralized file sharing system. Files that are popular among someone's "semantic neighbors" are likely to be interesting for that user as well.

4.2 Further Reading

GosSkip [46] employs gossip to implements skip lists [47] in a distributed fashion similar to SkipNets [48]. RayNet [49] uses gossiping to create structured overlays inspired from Voronoi diagrams. Other work, like [50] create overlays that resemble Small World networks [43, 51]. Rappel [52] uses gossip to leverage clustering of interests to build dissemination trees.

5 Overlay Slicing

Overlay slicing is an alternative form of overlay management to the clustering mechanisms introduced in Section 4. It relates to the problem of overlay *provisioning*. Slicing allows to separate a network in relative-sized groups in a self-organizing manner.

Here, we are no longer interested in creating a graph of links between nodes in the network that form a given structure, but rather to split the network in a set of *slices*. Each slice has a size that is expressed as a proportion of the total network size, and that can aggregate the nodes with the highest value for a given, node-specific metric. It is important to note here that the actual size of the network does not need to be known to proceed to the slicing operation.

Slicing has many useful applications. It allows to provision parts of the network to dedicate each such parts to a particular applications, or to different services pertaining to one application:

– One may want to provision the most powerful nodes to support the critical services of some application. In this case, each node is attached to a metric that depicts its relative power. One can think of the available bandwidth, the available storage capacity, the processing power, among others. The nodes that have a smallest value for this metric can be dedicated to less critical operations of the applications, e.g., the powerful nodes can support a naming mechanism that allows to locate data or services, while the less powerful nodes can support a less critical part of the application, such as caching or monitoring mechanisms.
– One may want to split the network into groups regardless of the nodes' characteristics, in order for each slice to be used for a different application with the same nodes' characteristics distribution for each slice. For instance, if a network needs to support three different applications, and the nodes supporting each application must be dedicated to only this application, one can express the size of each slice to be $\frac{1}{3}$ of the network regardless of the nodes' characteristics.

An example of slicing based on nodes' characteristics is given by Figure 8(a). Here, we are interested in creating three slices. The first slice shall contain half (50%) of the network, and contain the nodes that have the smallest value for the considered metric (say, the available disk capacity). The second slice must contain the 30% nodes that have intermediate values for this same metric. Finally, the third slice must be composed of the 20% nodes with the highest values for the metric, say, with the highest available disk capacity. We note that we do not want the assignment of nodes to slices to be static. If the metric changes for some node, e.g., the disk capacity is reduced due to the use of this resource by another application running on the same machine, then the self-organization objective requires that the assignment of nodes to slices reflects this change, and so on for the lifetime of the application (unless one needs to fix the assignment once and for all due to application requirements, which would require freezing the assignment).

The assignment of one node to some slice is autonomous. After the gossip-based overlay slicing mechanism has run for enough cycles, each node must be able to determine which slice it belongs to. We consider that the parameters and relative size of slices are known by all nodes, that is, each node knows what metric to consider for itself and its peers, and what are the relative sizes of the slices (e.g., {50%,30%,20%} in the case of Figure 8(a)).

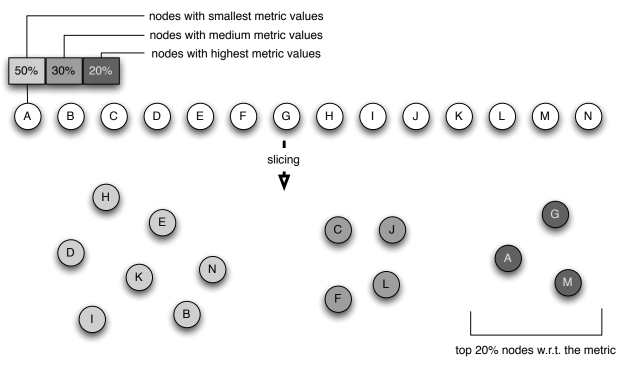
The overlay slicing protocol based on gossip-based networking that we present in this document is the one proposed by Jelasity and Kermarrec in [53]. In order to determine which slice they do belong to, nodes must determine what is the relative position of their metric value in the set of all metric values, if such an ordered set was known. Obviously, due to the large-scale of the considered network, it is impractical to consider that any one node will know this sorted set. The relative position must be known without globally sorting all the nodes' metrics in order to decide on any one node's position. This relative position determination is illustrated by Figure 8(b). Let us consider node J. Here, in order for J to determine that it belongs to the second slice, the node must determine an approximation of:

- the number of nodes that have a higher value than J for the metric;
- the number of nodes that have a smaller value than J.

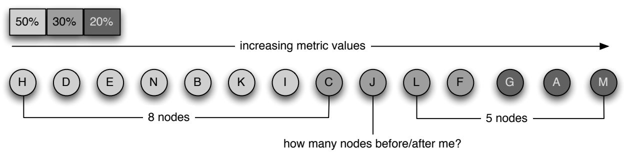
The position of nodes in the set is approximated by a value in [0:1]. More specifically, this position is the *relative position* of the node in the set of all metric values. Note that while the relative positions are contained in a bounded range, this is not the case for the metric values, which can range over any space.

The case where the network must be split into relative-sized slices but independently of any metric is simply a special case. In order to allow a random sampling of nodes in each slice (according to the slice size relative to the size of the network of course), each node simply emulates a metric by picking a random value in [0:1] as its metric value.

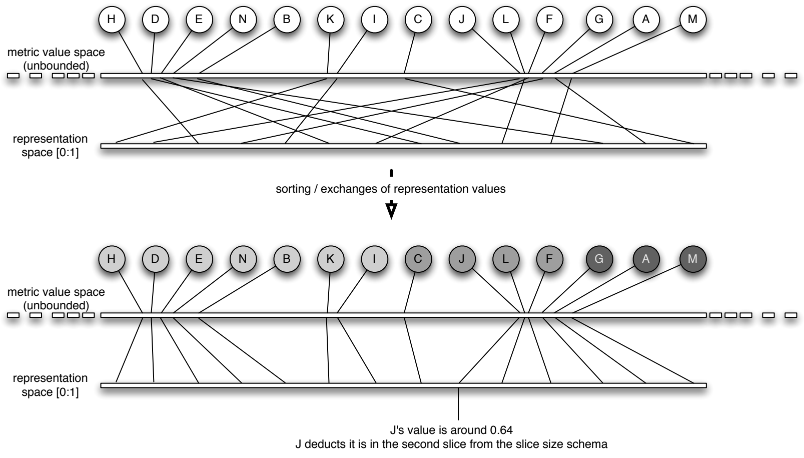
Each node starts with a random relative position in [0:1], which obviously is unrelated to the final expected position. This starting position is illustrated by the upper representation of Figure 8(c). We can see here that the initial relative


nodes with smallest metric values
nodes with medium metric values
nodes with highest metric values
50%
30%
20%
A B C D E F G H I J K L M N
slicing
H E D K N I B
C J F L
G A M
top 20% nodes w.r.t. the metric

(a) Problem definition



(b) Using the relative position of a node to autonomously determine its slice

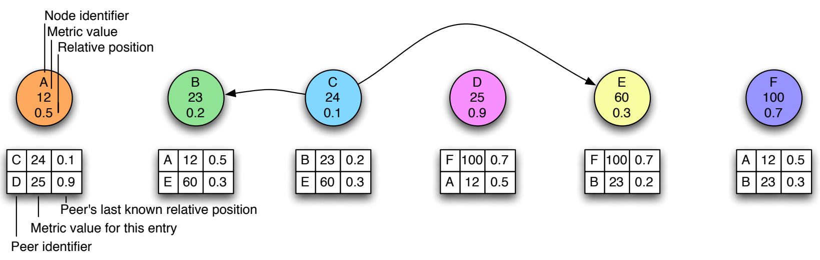

H D E N B K I C J L F G A M
metric value space (unbounded)
representation space [0:1]
sorting / exchanges of representation values
H D E N B K I C J L F G A M
metric value space (unbounded)
representation space [0:1]
J's value is around 0.64
J deducts it is in the second slice from the slice size schema

(c) Gossip-based sorting to determine relative positions

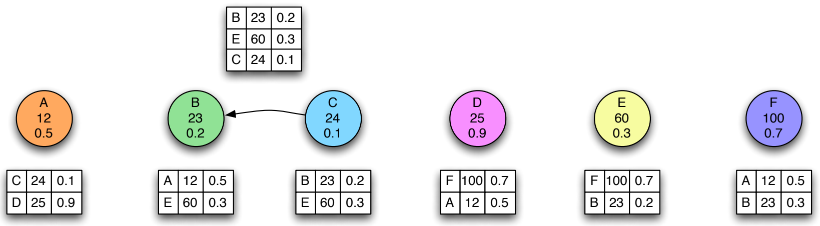
Fig. 8: Gossip-based *slicing*: problem representation and determination of the relative position using gossip-based sorting.

position of J, which is around 0.3, has nothing to do with the expected relative position, that is, around 0.6.

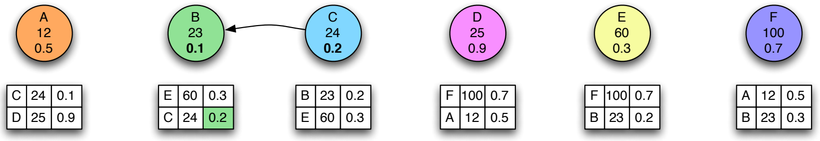
After deciding randomly on these initial values, nodes engage in a gossip-based self-organization, that must lead to each node holding a relative position that reflect its slice, as illustrated on the bottom representation of Figure 8(c). Here, one can see that the relative position of J, being 0.64, reflects correctly the slice it must belong to. The final positions are obtained by pair-wise *gossip-based sorting* of the relative position with respect to the metric of the two nodes. Each node has a view, of bounded size c, and can pick random nodes from a Peer Sampling Service.


Node identifier
Metric value
Relative position
Peer's last known relative position
Metric value for this entry
Peer identifier

(a) selectPartner() operation: there is no order violation for peer E, but there is an order violation for peer B (B's metric value 23 is smaller than C's metric value 24 but their metric value have an opposite order: 0.2>0.1). B is selected.



(b) selectToSend() on peer C: selection of the entire view, plus an entry for C.



(c) selectToKeep() on peer B: after removing duplicates entries (not shown), C sees that there still is an order violation. Henceforth, it takes the current relative position of C. C does the same upon receiving the view from B.

Fig. 9: A pair-wise interaction for gossip-based sorting that exchange the relative position of nodes B and C and resolves an order violation w.r.t. their metric values.

Figure 9 represents a single gossip-based interaction between two nodes B and C. This operation, following the framework of Algorithm 1, is as follows:

selectPartner() (Figure 9(a)) The peer selects at random a partner for the exchange among the peers it knows, including the peers it obtains from the Peer Sampling Service (see Section 3), and for which an order violation exists (that is, the metric of the peer is higher than the metric of the initiating node but their relative values follow a different order, or conversely). This peer is selected in order to proceed to an exchange of the relative values and to resolve the order violation.[2]

selectToSend() (Figure 9(b)) The node sends its entire view, and includes itself in the sent view (as links are unidirectional, there is indeed no guarantee that B would know the metric value and the relative position of C that it needs for the exchange).

selectToKeep() (Figure 9(c)) If there still is an order violation between the gossip initiator and the partner, the partner exchanges its own relative position with the one of the initiator, and returns its view in the very same way. The initiator, upon receiving the view, also trades its relative position with the one of the partner. Henceforth, the order violation is resolved. Note that it can happen that the order violation that was witnessed by the initiator did not longer exist. Indeed, the partner node may have already exchanged its value with another node since the last update of the view entry, resolving the violation. In this case, the gossip exchange is simply cancelled.

Each such gossip step reduces the global *disorder metric*, that is, the average squared distance between a nodes' relative position and its "correct" position. This metric is simply the standard deviation over the relative position incorrectness. Interestingly, the convergence of the gossip-based sort is empirically independent from the size of the network. Within 20 cycles (during one cycle, each node exchanges with one other node if there exist one with an order violation in its view or in the Peer Sampling Service view), the average error is no more than 1% of the network size, already allowing a very good estimation of the slice a node belongs to: in a 10,000 nodes network, nodes are on average 100 positions away from their ideal position (which means that, if they are not considered in the correct slice, they still have very similar metric values to the correct nodes for that slice). If one lets the protocol converge for 40 cycles, the average error becomes 0.1%, which is a negligible value in a dynamic and large-scale systems such as the ones considered.

5.1 Further reading

In [54], the authors improve over the original slicing protocol presented in [53] in two ways. First, they propose measures to speed up the convergence of the

[2] Note that an additional aging mechanism can allow to ensure that nodes are contacted within a bounded amount of time, in order to detect changes/failures in a timely manner. For the sake of simplicity, we do not consider this optimization in this paper and encourage the reader to refer to [53] for details.

protocol by using a local disorder measure for the selectPartner() operation: peers choose the gossip partner with which an exchange is the most likely to reduce the global disorder measure. Second, they revisit the use of random values sorting for the slicing operation. The rationale is that, when the metric value is based on the nodes' characteristics, e.g., the uptime or the available bandwidth, there is a correlation between this metric and the relative position that tends to bias the distribution of relative values. Instead of using random values sorting, the authors propose to estimate the rank of nodes based on the history of recently-seen values for the relative positions, which proves to be more robust to churn and bias. In [55], the Sliver protocol is presented, that integrates further optimizations to gossip-based slicing.

6 Distributed Aggregation

Aggregation is the collective estimation of system-wide properties, expressed as numerical values. It is a key functionality to a number of large-scale distributed systems, in particular to implement monitoring mechanisms.

The properties that can be aggregated by gossip-based distributed aggregation can relate to a large variety of metrics. One can mention the average system load, the identity of the node with the lowest or highest load or disk capacity, the total available disk capacity in a distributed storage system, among others. As we shall see in the Subsection 6.1, aggregation can also be used to solve in an elegant and autonomous way a difficult problem in decentralized large-scale systems: network size estimation.

Each node starts with its own value for the metric that is to be aggregated. Thereafter, aggregation should be carried out collectively by all participating nodes in a purely distributed fashion, and the result(s) of the aggregation should become known to all nodes.

We present as an example in this section a basic aggregation protocol that follows the push-pull gossip-based networking paradigm. This protocol appeared in [56]. Each node has a local *estimate* of the property being aggregated and a set of *neighbors*. At random times, but once every δ time units, a node picks a random neighbor and they exchange their local estimates. This random neighbor is typically obtained by calling the getPeer() operation of the Peer Sampling Service (see Section 3), as we assume that no global view of the system exists at any node. After the exchange, each node updates its local estimate based on its previous estimate and the estimate of the partner it has received.

Averaging constitutes a fundamental aggregation operation, in which each node is equipped with a numeric value, and the goal is to estimate the average, or arithmetic mean, of all nodes' values. We start by describing the calculation of the average, and later show how it can form the basis for the computation of other aggregates, as detailed in Jelasity et al. [56].

In averaging, a node updates its estimate to the average between its previous local estimate and the estimate received. That is, when nodes p and q with estimates s_p and s_q proceed to a gossip exchange, their estimates are updated as follows:

$$s_p = s_q = \frac{s_p + s_q}{2}$$

Note that the sum of the two nodes' estimates does not change, therefore neither does the global average. However, the variance:

$$V = \sqrt{\frac{1}{N}\sum_{p=1}^{N}\left(s_p - \sum_{q=1}^{N} s_q/N\right)^2}$$

decreases after each exchange, unless s_p and s_q were already equal, in which case it remains unaltered. (N denotes the size of the system.) Experiments and theoretical analysis in [56–60] show that the variance V converges to zero. Moreover, it converges at an exponential rate, whose exponent depends on the communication graph defining the nodes' neighbors. The rule of thumb is that the higher the link randomization in an overlay, the faster the aggregation convergence. We propose to illustrate this fact by some experimental figure based on simulation.

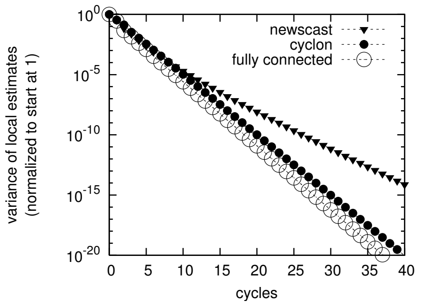


Fig. 11: Performance of gossip-based aggregation, for different peer selection mechanisms: realistic with Peer Sampling protocols Cyclon and Newscast, and ideal, with a purely random selection of gossip partners among all nodes (*fully connected* graph).

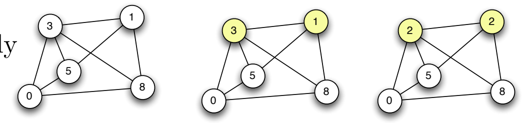


Fig. 10: An exchange in average calculation.

We consider a 100,000 nodes network, with the availability of a Peer Sampling Service [28], namely the two instances CYCLON [31] and NEWSCAST [61] described in Section 3. In this case, the gossip exchange partner for the aggregation is obtained from the Peer Sampling Service's view. For the sake of simplicity, we consider only a static network in which either CYCLON or NEWSCAST has converged and where the views are frozen for the duration of the aggregation. Similar conclusions as the ones we present for a static network can be made with a dynamic network where nodes' views keep evolving at each exchange cycle. Figure 11 presents the evolution of the variance as a function of the aggregation cycles (δ time units) elapsed. To have a point of reference, we plot the variance evolution for averaging over a fully connected graph, in which a node exchanges estimates with a node picked randomly out of the whole network.

First, we observe that in all cases the variance converges to zero at an exponential rate. Second, we record a clear difference between the aggregation efficiency of static Cyclon and Newscast Peer Sampling protocols, the former converging significantly faster. This is a direct consequence of Cyclon's narrow in-degree distribution and very low clustering. Each node has roughly the same number of incoming links, and therefore participates in roughly the same number of estimation exchanges as all other nodes. Moreover, the very low clustering ensures that each node's initial value is uniformly spread across "all directions" of the network, not being confined to any highly clustered subset. This leads to faster convergence of nodes' estimates to the global average. On the other hand, Newscast's skewed in-degree distribution results in an uneven distribution of estimation exchanges across nodes. Also, due to high clustering, local estimates spread quickly within highly clustered communities, but take longer to spread globally.

These observations illustrate that the type of Peer Sampling Service used can have an important impact on the efficiency of the protocols that use them as a basis, and, as pointed out in Section 3, one must carefully consider the impact of the parameters of the Peer Sampling Service on the served protocols.

Other aggregates. The computation of the geometric mean is similar to the computation of the arithmetic mean (average) we just presented. It also exhibits the same convergence properties. One simply needs to replace the update function by: $s_p = s_q = \sqrt{s_p \times s_q}$. The computations of the the average (arithmetic mean) and the geometric mean serve as a basis for the computation of most aggregates. Let us consider however for now that we know the number of nodes in the network N. Obviously, knowing this number in a large-scale system is all but a trivial task; however we explain in the next Subsection how we can actually obtain it based on a gossip-based aggregation calculation.

Based on N and gossip-based aggregation, the following aggregates can be composed:

- The sum of all values in the system, which is useful for instance if one needs to know the total available disk space in a distributed, collaborative data storage system, is simply obtained by multiplying the locally available arithmetic average by the number of nodes: $S = s_p \times N$.
- Similarly, the product can be composed with the geometric mean and the size of the network: $P = {s_p}^N$.
- Finally, the variance can be composed with the computation of the arithmetic average of initial values (here, denoted as $avg(s_p)$) and the arithmetic average of the squares of the initial values (here, $avg({s_p}^2)$):

$$V = avg({s_p}^2) - (avg(s_p))^2$$

Finally, the computation of the minimal and maximal values is also possible using gossip-based aggregation. These computations are simply performed by replacing the update function by: $s_p = s_q = min(s_p, s_q)$, or $s_p = s_q = max(s_p, s_q)$ respectively. One can note here that the propagation of the minimal or the maximal value from the node where it was present initially, to all nodes in the system, will be strictly similar to what a propagation of a single value using the *anti-entropy* mechanism introduced in Section 2 would be, if only one value (that is, this minimum or maximum) is considered and each node periodically polls another, randomly chosen node for the availability of this value.

6.1 Decentralized System Size Estimation using Aggregation

We have considered in the previous description of the *sum* and *product* composed aggregations that an important parameter, the total size of the system N, was supposedly available but without explaining how it was obtained. Knowing the total size of a large-scale system is not a trivial task. As no node has a global view of the system, and as the population of nodes is dynamic, the knowledge of N cannot be based on membership information. It is instead necessary to engage into a specific protocol for determining this number N, or more specifically to determine a sufficiently precise estimation of it (as we consider inherently dynamic networks, an exact size estimation would be impractical anyway).

In this Section, we present the use of aggregation for calculating the size of the network in an autonomous manner [56].

Note however that other techniques exist for large-scale decentralized size estimation, that are not necessarily based on gossip-based networking. For instance, Kostoulas et al. [62] rely on interval density sampling of the history of hashed nodes' identifiers over a bounded range to determine the population of nodes in the system; Massoulié et al. propose to use random walks methods and the principle of the inverse anniversary-problem to determine the size of the network based on the occurrences of collisions amongst random walkers [63]. These techniques are experimentally compared to the one we present in this document in [64].

The idea behind the peer counting based on aggregation is conceptually simple: one single peer in the system starts with the value 1, and all other peers start with the value 0. The average value of all the starting values is thus $\frac{(\sum_{N-1} 0)+1}{N} = \frac{1}{N}$, and this is the value that all local estimates will be equal to after the convergence of the gossip-based aggregation. Thereafter, each peer can autonomously use the inverted value of their local estimate to infer N.

However, there is no direct possibility for a single peer in the system for deciding which one of them will hold this initial value of 1 while ensuring that the others will hold the value 0. There is indeed no pre-existing omniscient peer that can decide that it can act as such a *leader* for the aggregation start, or this would require a global view of the network, contradicting the large-scale characteristics of the network. Several decentralized mechanisms exist to decide upon the initial peer in an autonomous way and without the need to maintain any global information. We simply sketch one such mechanism below.

We allow several concurrent instances of the average computation. Each such instance is associated with a different peer starting with the value 1. This peer is called the *leader* of each instance. The messages exchanged during the gossip-based aggregation are tagged with a unique identifier, e.g., the identifier of the leader for the corresponding instance. We note already that running several instances has the inherent advantage of added stability: each node cannot only base its estimation on one, potentially imperfect aggregate (if the system has not converged, or due to high levels of churn), but on the average of several such aggregates. Nodes decide to act as a leader for a new instance autonomously, based on a parameter i that denotes the target average number of gossip cycles one requires between the start of two aggregation instances. Each node knows the current estimate of the system size (or a reasonable guess for the first round), denoted by N_e. Each node decides at each of the aggregation cycle, that it will start a new aggregation of which it will be the leader with a probability of $\frac{i}{N_e}$.

In order to not let the number of local aggregates grow indefinitely, and in order to support variations in the system size, a periodic restarting mechanism is used. The time is divided in *epochs*, that are local to each node but that use the same duration λ for all nodes. At the end of an epoch, a node delivers the average of the local size estimates obtained during the last epoch to the application, and starts collecting new estimates for the current epoch. The garbage collection of previously known estimates can be done when one entire epoch has passed since their delivery to the application, as no other node will send a gossip exchange request pertaining to these values anymore. The duration of the epochs allows to express the tradeoff between the reactiveness of the size estimation to changes, versus the accuracy of this estimation.

7 The Many Other Uses of Gossip-Based Networking

In this Section, we wish to give the reader a quick overview of the many other uses of gossip-based networking that we did not introduce in details in this paper. Our goal is not to be comprehensive in surveying the usages of gossip (that would require a monograph on its own), but rather to highlight the fact that gossip-based networking can be applied in a wide range of large-scale distributed systems-related problems.

7.1 Self-organizing Publish and Subscribe

The publish and subscribe communication paradigm allows to greatly simplify the design of large-scale applications by providing a *decoupled* communication model. The producers of information do not need to know the interested consumers of the information they produce. Similarly, the consumers do not need to know beforehand which node is likely to issue information that match their interest. Here, producers simply *publish* to the publish and subscribe middleware, and consumers can express their interest in new data by the means of *subscriptions*.

It is then the system's responsibility to match the publications to the existing subscriptions, and to route the messages to all interested subscribers. Publish and subscribe mechanisms are very appealing for the design of large-scale applications as they allow to delegate the management of the application data flow to an external service.

Gossip-based networking is a strong contender to build and operate publish and subscribe services. One typically distinguishes between publish and subscribe mechanisms based on the expressiveness allowed for the subscriptions. The simplest model, topic-based, allows nodes to register their interest to a set of predefined topics, and publications are also attached to one of these topics. TERA [65] is an example of a topic-based publish and subscribe mechanism that leverages the principles of gossip-based structure emergence (Section 4) and a modified Peer Sampling Service. Gossip-based interactions allow to emerge clusters where the nodes interested in the same topic are linked. The publication then requires to reach the correct group, which is made possible by a biased peer-sampling mechanism and random walks, and then to disseminate among the group using gossip-based dissemination. The Rappel system [52] also leverages gossip-based networking to construct dissemination structures that take into account both the network characteristics (delays) and the presence of clustering in the users' subscriptions to reduce the number of links. STaN [66] is another system that leverages gossip for creating a network where nodes with similar interests are grouped together, with the additional guarantee that the subscriptions of nodes are not made public.

Content-based publish and subscribe allows expressing subscriptions based on the content of the events. This is a more powerful model, but since the matching of publications to subscribers shall be done dynamically for each publication, it is typically more complex to support. Sub-2-Sub [8] proposes to let a routing layer emerge from the use of the Vicinity [40] gossip-based networking framework. The very structure of the overlay that emerges allows publications to reach all interested subscribers autonomously, while the system inherits the self-organization allowed by the use of gossip for its construction. DPS [67] leverages gossip-based protocols to group the nodes with overlapping interests (based on their subscriptions) and form the basis of a self-organizing matching and dissemination layer.

7.2 Taming Networked Systems Complexity

Gossip-based networking can also be used to abstract the complexity of the network onto which it operates, in order to ease the development of applications.

A first example is the Nylon [68] NAT-aware Peer Sampling Service. Indeed, in real networks a majority of nodes lie behind NATs and firewalls and cannot be contacted directly, which may have a strong impact on the operation of applications, including the gossip-based protocols we presented in this document. In a similar way to the Peer Sampling Service protocols [28] presented in Section 3, Nylon provides a continuous set of random peers from the network in the view of each node, but each such node is attached with the necessary information for bypassing NATs, be it by opening connections from the destination node behind the NAT or by the use of relay nodes. This is done in a purely gossip-based fashion, with nodes periodically exchanging the information about their peers and the associated contact details. Leitao et al. [69] also propose to leverage gossip-based self-organization to tackle the natural imbalance that arises in networks where some nodes are more difficult to reach than others.

Another example is given by the Dr Multicast [70] system. While IP multicast is an efficient mechanism for dissemination, it is not always available in the whole network, and when it is, it is typically limited in the number of groups supported by the network elements. Vigfusson et al. propose to use gossip-based techniques for propagating information about the memberships and activities of multicast groups, and let a mapping that leverages the available IP multicast resources for as much of the communications as possible, and relying on point-to-point communication for the others.

7.3 Gossip-based networking and security aspects

An important aspect of large-scale systems that we did not mention yet in this introduction paper is that of the security. Indeed, large scale systems are composed of nodes that typically span over multiple administrative domains or end-users, and there is a risk of witnessing byzantine behaviors from part of the nodes.

First, several proposal have been made to handle the case of nodes wishing to bias the Peer Sampling Service, e.g., in order to favor a node over the others or isolate a part of the network. Brahms [71], the *Secure Peer Sampling* [72], and PuppetCast [73] are three examples of PSS protocols that take into account the presence of byzantine nodes wishing to bias the sampling.

The BAR (Byzantine, Altruistic, Rational) model is used by the authors of BAR gossip [74] to support byzantine and rational (selfish) nodes in a gossip-based dissemination of messages in a network. An interesting aspect of the protocol is that the selection of partners for gossip is no longer made at random (e.g., by using a PSS), but by a pseudo-random selection that keeps the properties of randomness that a PSS would typically provide. StarblabIT [75] is another proposal of an intrusion-tolerant gossip-based dissemination protocol. It allows to ensure the complete and authenticated dissemination of messages within a group; while making sure that external attackers cannot hamper the dissemination (by guaranteeing reliability, authenticity and consistency).

Finally, gossip-based mechanisms are used in Whisper [76] for implementing confidential group membership and communications in an autonomous and self-organizing manner. Nodes leverage gossip-based overlay construction principles to exchange alternative paths, that are used as anonymizing routes between members of the group, without the need for a trusted third party for protecting group members' identifies and communications. In the context of authentication, Yan et al. [77] propose to use gossip-based networking to implement group key distribution.

Gossip-based networking has also been leveraged in [78] to implement decentralized failure detection in a large-scale setting. Guo et al. propose to implement garbage collection using gossip in [79].

8 Conclusion

In massive-scale distributed systems, rigid control of the system's operation is inapt and can lead to severe bottlenecks and operational deficiencies. In the face of –inevitable for such systems– continuous errors and failures, a self-adaptive scheme with self-configuring and self-healing properties is more appropriate for working around the errors. Such properties are known as self-* properties.

In this paper we advocated the importance of gossiping protocols in providing self-* properties to distributed systems of very large scale. We classified gossiping protocols in two categories. Peer sampling protocols, serving as sources of randomly selected nodes from the whole network, and the standard ("traditional") gossiping protocols for serving specific application needs (data dissemination, node clustering, data aggregation, etc.).

We identified a number of properties in gossiping protocols that make them particularly attractive for large scale distributed systems. They are remarkably robust and tolerant to faults, even to failures of very large scale. They distribute the load across many nodes, leading to load balanced networks. Gossip-based systems are inherently symmetric, in the sense that no node has special responsibility at a particular task, therefore no fault at any single node may harm the smooth operation for the whole community. They are very scalable, to millions of nodes, and they disseminate, aggregate, or cluster data at exponential speed. Last but not least, they are very simple.

By visiting a set of representative gossiping protocols, we gave insight to the conceptual framework of epidemics, within which elegant, simple, and robust algorithmic building blocks for large-scale systems can be proposed.

References

1. Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: IPTPS '02: Revised Papers from the Second International Workshop on Peer-to-Peer Systems, London, UK, Springer-Verlag (2002) 53–65
2. Loo, B.T., Huebsch, R., Hellerstein, J.M., Shenker, S., Stoica, I.: Enhancing p2p file-sharing with an internet-scale query processor. In: In Proc. 30th International Conference on Very Large Data Bases (VLDB). (August 2004)
3. Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., Shenker, S.: Making gnutella-like p2p systems scalable. In: SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, ACM Press (2003) 407–418

4. Felber, P., Kropf, P., Leonini, L., Luu, T., Rajman, M., Rivière, E.: Collaborative ranking and profiling: Exploiting the wisdom of crowds in tailored web search. In: Proceedings of DAIS'10: 10th IFIP international conference on Distributed Applications and Interoperable Systems, Amsterdam, The Netherlands (jun 2010)
5. Castro, M., Druschel, P., Kermarrec, A.M., Nandi, A., Rowstron, A., Singh, A.: Splitstream: high-bandwidth multicast in cooperative environments. In: SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, New York, NY, USA, ACM Press (2003) 298–313
6. Zhang, C., Jin, H., Deng, D., Yang, S., Yuan, Q., Yin, Z.: Anysee: Multicast-based peer-to-peer media streaming service system. In: Proceedings of the Asia-Pacific Conference on Communications(APCC05), Pernth, Western Australia (October 2005)
7. Rowstron, A., Kermarrec, A.M., Castro, M., Druschel, P.: Scribe: The design of a large-scale event notification infrastructure. In Crowcroft, J., Hofmann, M., eds.: Networked Group Communication, Third International COST264 Workshop (NGC'2001). Volume 2233 of Lecture Notes in Computer Science. (November 2001) 30–43
8. Voulgaris, S., Rivière, E., Kermarrec, A.M., van Steen, M.: Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks. In: Proceedings of IPTPS'06: 5th International Workshop on Peer-to-Peer Systems, Santa Barbara, USA (feb 2006)
9. Bhagwan, R., Savage, S., Voelker, G.M.: Understanding availability. In: Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03). (2003)
10. Stutzbach, D., Rejaie, R.: Understanding churn in peer-to-peer networks. In: IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement, New York, NY, USA, ACM Press (2006) 189–202
11. Kermarrec, A.M., van Steen, M.: Gossiping in distributed systems. SIGOPS Oper. Syst. Rev. **41**(5) (2007) 2–7
12. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing, New York, NY, USA, ACM Press (1987) 1–12
13. Birman, K.P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., Minsky, Y.: Bimodal multicast. ACM Transactions on Computer Systems **17**(2) (1999) 41–88
14. Eugster, P.T., Guerraoui, R., Kermarrec, A.M., Massoulié, L.: Epidemic information dissemination in distributed systems. In: Computer. Volume 37., IEEE (May 2004) 60–67
15. Gupta, I., Kermarrec, A.M., Ganesh, A.: Efficient epidemic-style protocols for reliable and scalable multicast. In: Proceedings of the 21st Symposium on Reliable Distributed Systems (SRDS'02). (2002) 180
16. Khambatti, M., Ryu, K., Dasgupta, P.: Push-pull gossiping for information sharing in peer-to-peer communities. In: Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA). (June 2003) 1393–1399
17. Eugster, P.T., Guerraoui, R., Handurukande, S.B., Kouznetsov, P., Kermarrec, A.M.: Lightweight probabilistic broadcast. ACM Transactions on Computer Systems **21**(4) (2003) 341–374
18. Vogels, W., van Renesse, R., Birman, K.: The power of epidemics: robust communication for large-scale distributed systems. SIGCOMM Comput. Commun. Rev. **33**(1) (2003) 131–135

19. Kermarrec, A.M., Massoulié, L., Ganesh, A.J.: Probabilistic reliable dissemination in large-scale systems. IEEE Transactions on Parallel and Distributed Systems **14**(3) (2003) 248–258
20. Renesse, R.V., Birman, K.P., Vogels, W.: Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. ACM Trans. Comput. Syst. **21**(2) (2003) 164–206
21. Locher, T., Meier, R., Schmid, S., Wattenhofer, R.: Push-to-pull peer-to-peer live streaming. In: Proceedings of DISC 2007: 21st Internation Symposium on Distributed Computing, Lemosos, Cyprus (sep 2007)
22. Rhea, S.C.: Opendht: a public dht service. PhD thesis, Berkeley, CA, USA (2005) AAI3211499.
23. Patel, J.A., Gupta, I., Contractor, N.: Jetstream: Achieving predictable gossip dissemination by leveraging social network principles. In: NCA '06: Proceedings of the Fifth IEEE International Symposium on Network Computing and Applications, Washington, DC, USA, IEEE Computer Society (2006) 32–39
24. Serbu, S., Rivière, E., Felber, P.: Network-friendly gossiping. In: SSS '09: Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems, Berlin, Heidelberg, Springer-Verlag (2009) 655–669
25. Fernandess, Y., Fernández, A., Monod, M.: A generic theoretical framework for modeling gossip-based algorithms. SIGOPS Oper. Syst. Rev. **41**(5) (2007) 19–27
26. Allavena, A., Demers, A., Hopcroft, J.E.: Correctness of a gossip based membership protocol. In: PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing, New York, NY, USA, ACM Press (2005) 292–301
27. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available key-value store. SIGOPS Oper. Syst. Rev. **41**(6) (2007) 205–220
28. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.M., van Steen, M.: Gossip-based peer sampling. ACM Transactions on Computer Systems **25**(3) (aug 2007)
29. Jelasity, M., Kowalczyk, W., van Steen, M.: Newscast Computing. Technical Report IR-CS-006, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands (November 2003)
30. Voulgaris, S., Jelasity, M., van Steen, M.: A robust and scalable peer-to-peer gossiping protocol. In: 2nd Int'l Workshop Agents and Peer-to-Peer Computing,. Volume 2872., Springer-Verlag GmbH (2003)
31. Voulgaris, S., Gavidia, D., van Steen, M.: Cyclon: Inexpensive membership management for unstructured p2p overlays. Journal of Network and Systems Management **13**(2) (June 2005)
32. Erdös, P., Rényi, A.: On the evolution of random graphs. Publications of the Mathematical Institute of the Hungarian Academy of Sciences **5** (1960) 17–61
33. Barabasi, A.L., Albert, R.: Emergence of scaling in random networks. Science **286** (1999) 509–512
34. Barabasi, A.L., Albert, R.: Statistical mechanics of complex networks. Reviews of Modern Physics (2002)
35. Barabasi, A.L.: LINKED: The New Science of Networks. Perseus Books Group (2002)
36. Ganesh, A.J., Kermarrec, A.M., Massoulié, L.: Scamp: Peer-to-peer lightweight membership service for large-scale group communication. In: Proceedings of the Third International COST264 Workshop on Networked Group Communication. NGC '01, London, UK, Springer-Verlag (2001) 44–55

37. Leitão, J., Pereira, J., Rodrigues, L.: Hyparview: a membership protocol for reliable gossip-based broadcast. In: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Edinburgh, UK (June 2007) 419–428
38. Jelasity, M., Babaoglu, O.: T-man: Gossip-based overlay topology management. Engineering Self-Organising Systems **1**(15) (2005)
39. Jelasity, M., Montresor, A., Babaoglu, O.: T-man: Gossip-based fast overlay topology construction. Computer Networks **53**(13) (aug 2009) 2321–2339
40. Voulgaris, S., van Steen, M.: Epidemic-style management of semantic overlays for content-based searching. In Cunha, J.C., Medeiros, P.D., eds.: Euro-Par 2005 Parallel Processing: 11th International Euro-Par Conference. Volume 3648. (September 2005) 1143
41. Voulgaris, S.: Epidemic-Based Self-Organization in Peer-to-Peer Systems. PhD thesis, Vrije Universiteit Amsterdam (2006)
42. Lua, E.K., Crowcroft, J., Pias, M., Sharma, R., Lim, S.: A survey and comparison of peer-to-peer overlay network schemes. In: IEEE Communications survey and tutorial. (March 2004)
43. Kleinberg, J.: The small-world phenomenon: An algorithmic perspective. In: Proceedings of the 32nd ACM Symposium on Theory of Computing, Portland, OR, USA (May 2000) 163–170
44. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature **393** (1998) 440–442
45. eDonkey: (no date) http://www.edonkey2000.com.
46. Guerraoui, R., Handurukande, S.B., Huguenin, K., Kermarrec, A.M., Fessant, F.L., Rivière, E.: GosSkip, an efficient, fault-tolerant and self organizing overlay using gossip-based construction and skip-lists principles. In: P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing, Cambridge, UK, IEEE Computer Society (September 2006) 12–22
47. Pugh, W.: Skip lists: A probabilistic alternative to balanced trees. Communication of the ACM **32**(10) (1990) 668–676
48. Harvey, N.J.A., Jones, M.B., Saroiu, S., Theimer, M., Wolman, A.: Skipnet: A scalable overlay network with practical locality properties. In: the Fourth USENIX Symposium on Internet Technologies and Systems (USITS '03), Seattle, WA (2003)
49. Beaumont, O., Kermarrec, A.M., Rivière, E.: Peer to peer multidimensional overlays: Approximating complex structures. In: Proceedings of the 11th International Conference On Principles Of Distributed Systems (OPODIS'07). Lecture Notes in Computer Science, Guadeloupe, French West Indies (December 2007)
50. Bonnet, F., Kermarrec, A.M., Raynal, M.: Small-world networks: From theoretical bounds to practical systems. In: Proceedings of the 11th International Conference On Principles Of Distributed Systems (OPODIS'07). Lecture Notes in Computer Science, Guadeloupe, French West Indies (December 2007)
51. Milgram, S.: The small world problem. Psychology Today **2** (1967) 60–67
52. Patel, J.A., Rivière, E., Gupta, I., Kermarrec, A.M.: Rappel: Exploiting interest and network locality to improve fairness in publish-subscribe systems. Computer Networks **53**(13) (2009) 2304–2320
53. Jelasity, M., Kermarrec, A.M.: Ordered slicing of very large-scale overlay networks. In: P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing, Cambridge, UK, IEEE Computer Society (September 2006) 117–124

54. Fernandez, A., Gramoli, V., Jimenez, E., Kermarrec, A.M., Raynal, M.: Distributed slicing in dynamic systems. In: Proceedings of the International Conference on Distributed Computing Systems (ICDCS'07), Toronto, Ontario, Canada, IEEE Computer Society (jun 2007)
55. Gramoli, V., Vigfusson, Y., Birman, K., Kermarrec, A.M., van Renesse, R.: Slicing distributed systems. IEEE Transactions on Computers – Special Issue on Autonomic Network Computing (IEEE TC) **58**(11) (jul 2009) 1444–1455
56. Jelasity, M., Montresor, A., Babaoglu, O.: Gossip-based aggregation in large dynamic networks. ACM Trans. Comp. Syst. **23**(3) (2005) 219–252
57. Jelasity, M., Montresor, A.: Epidemic-Style Proactive Aggregation in Large Overlay Networks. In: 24th Int'l Conf. on Distributed Computing Systems. (2004) 102–109
58. Montresor, A., Jelasity, M., Babaoglu, O.: Robust aggregation protocols for large-scale overlay networks. In: DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04), Washington, DC, USA, IEEE Computer Society (2004) 19
59. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, Washington, DC, USA, IEEE Computer Society (2003) 482
60. Kowalczyk, W., Vlassis, N.: Newscast EM. In: Advances in Neural Information Processing Systems (NIPS) 17, Cambridge, MA, MIT Press (2004)
61. Voulgaris, S., van Steen, M.: An epidemic protocol for managing routing tables in very large peer-to-peer networks. In Marcus Brunner, A.K., ed.: Self-Managing Distributed Systems: 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2003. Volume 2867., Springer-Verlag GmbH (2003) 41–54
62. Kostoulas, D., Psaltoulis, D., Gupta, I., Birman, K., Demers, A.: Decentralized schemes for size estimation in large and dynamic groups. In: NCA '05: Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications, Washington, DC, USA, IEEE Computer Society (2005) 41–48
63. Massoulié, L., Merrer, E.L., Kermarrec, A.M., Ganesh, A.: Peer counting and sampling in overlay networks: random walk methods. In: PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing, New York, NY, USA, ACM Press (2006) 123–132
64. Merrer, E.L., Kermarrec, A.M., Massoulié, L.: Peer to peer size estimation in large and dynamic networks: A comparative study. In: Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing, Paris, France (June 2006) 7–17
65. Baldoni, R., Beraldi, R., Quema, V., Querzoni, L., Tucci-Piergiovanni, S.: Tera: topic-based event routing for peer-to-peer architectures. In: DEBS '07: Proceedings of the 2007 inaugural international conference on Distributed event-based systems, New York, NY, USA, ACM Press (June 2007) 2–13
66. Matos, M., Nunes, A., Oliveira, R., Pereira, J.: Stan: Exploiting shared interests without disclosing them in gossip-based publish/subscribe. In: Proc. of IPTPS'10: 9th international workshop on Peer-to-Peer Systems, San Jose, CA, USA (2010)
67. Anceaume, E., Gradinariu, M., Datta, A.K., Simon, G., Virgillito, A.: A semantic overlay for self-* peer-to-peer publish/subscribe. In: Proceedings of the International Conference on Distributed Computing Systems (ICDCS'06). (jun 2006)

68. Kermarrec, A.M., Pace, A., Quema, V., Schiavoni, V.: Nat-resilient gossip peer sampling. In: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems. ICDCS '09, Washington, DC, USA, IEEE Computer Society (2009) 360–367
69. Leitão, J.a., van Renesse, R., Rodrigues, L.: Balancing gossip exchanges in networks with firewalls. In: Proceedings of the 9th international conference on Peer-to-peer systems. IPTPS'10, Berkeley, CA, USA, USENIX Association (2010) 7–7
70. Vigfusson, Y., Abu-Libdeh, H., Balakrishnan, M., Birman, K., Burgess, R., Li, H., Chockler, G., Tock, Y.: Dr. multicast: Rx for data center communication scalability. In: Proceedings of Eurosys'10, Paris, France (apr 2010)
71. Bortnikov, E., Gurevich, M., Keidar, I., Kliot, G., Shraer, A.: Brahms: Byzantine resilient random membership sampling. Computer Networks **53**(13) (aug 2009) 2340–2359
72. Jesi, G.P., Montresor, A., van Steen, M.: Secure Peer Sampling. Elsevier Computer Networks - Special Issue on Collaborative Peer-to-Peer Systems **54**(12) (2010) 2086–2098
73. Bakker, A., van Steen, M.: Puppetcast: A secure peer sampling protocol. In: Proc. of the European Conference on Computer Network Defense (EC2ND 2008), Dublin, Ireland (dec 2008) 3–10
74. Li, H., Clement, A., Wong, E., Napper, J., Alvisi, L., Dahlin, M.: Bar gossip. In: Proc. of 7th Symposium on Operating System Design and Implementation (OSDI '06). (2006)
75. Kihlstrom, K.P., Elliott, R.S.: Performance of an intrusion-tolerant gossip protocol. In: Proc. of the 21st IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS), Cambridge, MA, USA (nov 2009)
76. Schiavoni, V., Rivière, E., Felber, P.: Whisper: Middleware for confidential communication in large-scale networks. In: Proc. of ICDCS'11: 31st Int'l Conference on Distributed Computing Systems, Minneapolis, Minnesota, USA (2011)
77. Yan, Y., Ping, Y., Yi-Ping, Z., Shi-Yong, Z.: Gossip-based scalable and reliable group key distribution framework. In: InfoSecu '04: Proceedings of the 3rd international conference on Information security, New York, NY, USA, ACM Press (2004) 53–61
78. van Renesse, R., Minsky, Y., Hayden, M.: A gossip-style failure detection service. In IFIP, ed.: Proc. of Middleware, the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, The Lake District, UK (1998) 55–70
79. Guo, K., Hayden, M., van Renesse, R., Vogels, W., Birman, K.P.: Gsgc: An efficient gossip-style garbage collection scheme for scalable reliable multicast. Technical report, Cornell University, Ithaca, NY, USA (1997)