On Adding Structure to Unstructured Overlay Networks

João Leitão
INESC-ID / IST
jleitao@gsd.inesc-id.pt

Nuno A. Carvalho
University of Minho
nuno.carvalho@di.uminho.pt

José Pereira
University of Minho
jop@di.uminho.pt

Rui Oliveira
University of Minho
rco@di.uminho.pt

Luís Rodrigues
INESC-ID / IST
ler@ist.utl.pt

Abstract

Unstructured peer-to-peer overlay networks are very resilient to churn and topology changes, while requiring little maintenance cost. Therefore, they are an infrastructure to build highly scalable large-scale services in dynamic networks. Typically, the overlay topology is defined by a peer sampling service that aims at maintaining, in each process, a random partial view of peers in the system. The resulting random unstructured topology is suboptimal when a specific performance metric is considered. On the other hand, structured approaches (for instance, a spanning tree) may optimize a given target performance metric but are highly fragile. In fact, the cost for maintaining structures with strong constraints may easily become prohibitive in highly dynamic networks. This chapter discusses different techniques that aim at combining the advantages of unstructured and structured networks. Namely we focus on two distinct approaches, one based on optimizing the overlay and another based on optimizing the gossip mechanism itself.

1 Introduction

Gossip, or epidemic, protocols have emerged as a highly scalable and resilient peer-to-peer approach to implement several application level services such as reliable multicast [1, 2, 3, 4, 5, 6, 7], data aggregation [8], publish-subscribe [9], among others [10, 11, 12]. This chapter addresses peer-to-peer communication support for reliable and scalable information dissemination. A gossip-based broadcast protocol usually operates as follows: to broadcast a message, a node selects t nodes at random from the system (t is a configuration parameter called *fanout*) and sends the message to them. Upon the reception of a message for the first time, each node simply repeats this procedure.

The gossip approach to data dissemination has several advantages: *i)* it is simple to implement, *ii)* it shares the load evenly across all nodes in the system, making gossip protocols highly scalable, in fact the load imposed by the process in each node of the systems only has to grow logarithmically with the size of the system in order to ensure atomic broadcast with a high probability [1, 13], and finally, *iii)* its inherent redundancy makes gossip protocols highly resilient to node and link failures (for instance, [5] proposes a gossip-based broadcast protocol that can maintain high resilience even in scenarios where 80% of the nodes in the system fail simultaneously).

Gossip-based protocols were originally designed to operate with full membership information [14, 1], by maintaining locally at each node a list with the identifiers of every other node in the system (typically, an identifier is a tuple $(ip, port)$ that allows a node to be reached). However, such approach is not scalable, not only due to the large size of the membership but also (and mainly) due to the cost of maintaining such information up-to-date in dynamic systems. For scalability, nodes may rely on a *peer sampling service* [15, 16, 17, 5], provided by a membership protocol that operates with the goal of maintaining locally, at each node, a small random subset (called a *partial view*) of the full membership list. In this case, nodes use their local partial views to select peers for exchanging messages.

Partial views establish *neighboring* associations among nodes that define an overlay network which can be used for gossiping data. Ideally, the selection of peers from local partial views should be equivalent to a random selection of peers across the full membership. Therefore, the resulting overlay has a random (unstructured) topology.

Although this randomness has some desirable features, it also raises two distinct problems that may impair the efficiency of applications and protocols that operate on top of these unstructured overlay networks. First, it prevents the underlying network topology to be taken into consideration by the peer sampling service. This problem is known as *topology mismatch* [18]: it usually leads to scenarios where many overlay links are suboptimal with regard to a given network efficiency criteria such as bandwidth or latency. Second, because the overlay structure is random, it fails to exploit the *natural heterogeneity* [19] of large-scale peer-to-peer systems, and does not take advantage of nodes and links that have a higher capacity.

Node heterogeneity is easier to take into account in structured multicast protocols, by explicitly building dissemination structures according to a predefined efficiency criteria [20, 21, 22, 23], and then use these structures (such as spanning trees [24, 25]) to disseminate multiple messages. In a structured approach, nodes with higher resource availability can offer a bigger contribution to the global dissemination effort by having larger degrees or by being placed closer to the root of the tree (the reader should notice that nodes located at the leaves of the tree are not required to contribute to the message dissemination effort).

The trade-off between gossip-based and structured approaches is clear: By avoiding the need to build and maintain a spanning tree, epidemic multicast provides extreme simplicity. Moreover the balanced load across all nodes in the system, is a key factor to achieve resilience and scalability. On the other hand, structured multicast provides better resource usage (and thus higher performance when the network is stable) by optimizing the cost of the spanning tree according to efficiency criteria such as network bandwidth and latency. However, structured approaches have to deal with the complexity of rebuilding the structure when faults or network reconfiguration occurs.

In this chapter, we address techniques that aim at combining the best of both approaches, namely, the simplicity, scalability and resilience of unstructured overlay networks with the performance of structured approaches. In order to achieve this, some degree of structure is added to low-cost unstructured overlay networks to improve their performance without impairing the relevant properties of unstructured approaches. We start by presenting a survey of several existing works that aim at improving the topology of unstructured overlay networks. This is followed by a description of key properties of unstructured overlay networks that should be preserved when introducing structure. Then we introduce two approaches that can be used to introduce structure in unstructured overlay networks. The first approach bias the topology of an unstructured overlay according to some performance metric without compromising the resilience of the overlay. The second is based on an emergent behavior, approximating the operation of a structured overlay on top of an unstructured overlay. We present a performance evaluation of both approaches.

2 Adding Structure to Unstructured Overlay Networks

In this section we survey several existing protocols that can be used to add structure to, or improve the locality properties of, unstructured overlay networks. Then we list some key properties of unstructured overlay networks We also enumerate some relevant metrics that can be used to evaluate the benefits of adding structure to unstructured overlays. Finally we identify two distinct methodologies that allow to add some degree of structure to such overlays.

2.1 Existing Protocols

2.1.1 Narada

Narada [24] is a protocol designed to support application-level multicast. Narada aims at minimizing the overhead introduced by implementing multicast at the application layer (as opposed to IP multicast). Namely, Narada aims at minimizing both the stress induced on physical links (due to duplicate packets that transverse the same links) and the end-to-end latency of the multicast process. To address these issues, Narada is based on an unstructured overlay network, whose topology is adapted for improved performance. The goal of the protocol is to build an overlay that is: self-organizing, efficient, self-improving, and adaptive to network dynamics. We now briefly describe how the topology of the unstructured overlay is adapted.

Since Narada is targeted at small and medium sized systems, it is assumed that each node has access to a full membership list containing node identifiers for all participants in their algorithm. Using this information, Narada builds a limited degree unstructured overlay network, named a *richly connected mesh*. The overlay network topology is biased to obtain a majority of low cost links. On top of the resulting unstructured overlay network, a distance vector routing algorithm is executed to build, and maintain, a spanning tree routed at each sender for each multicast group. Each node will therefore maintain a local routing table which is used to disseminate multicast messages.

When a new element joins the system, it contacts a peer already present in the network to obtain the current full membership list. The node then randomly selects a few group members to whom it sends a join message, requesting to be added as their neighbor in the overlay. Nodes rely in the resulting unstructured overlay network to exchange periodic messages which are used to update global membership information, and to detect failed nodes and partitions.

After the execution of the steps described above, nodes form a fully connected unstructured overlay network. However, links in the network have a high probability to be suboptimal for a given set of target efficiency criteria. To improve the overlay, nodes capture information about their execution environment. For instance, in video conferencing applications, the overlay is biased to improve both point-to-point latency and bandwidth. Passive monitoring techniques are used to obtain available bandwidth values for peers in the system. Active monitoring techniques based in the exchange of ping messages are used to extrapolate values for latency. This information is then used in heuristics which bias the overlay topology as follows:

Add links Periodically, each node n selects another random non-neighbor node p and performs measurements to assess the efficiency of the communication with p. Also, p sends back to n a copy of its local multicast routing table. Node n uses both the received information and the expected efficiency of the link between n and p to locally compute a utility function that evaluates the gain of adding such link to the overlay. If the expected gain is above a given threshold value, n will add the link between himself and p to the overlay.

Remove links Periodically, each node selects, and removes, the (local) link with the lowest utility value. The computation of these utility values is done in such a way that the resulting value is an overestimate of the real utility of the link. Moreover, the link is only removed if its utility falls bellow a given threshold value.

This is done to ensure some stability in the overlay. Notice that because the network is dynamic, the utility of a link may also be dynamic. It would not be efficient to allow situations where one is constantly removing and adding the same link to the overlay.

Using this methodology, the unstructured overlay topology can be biased, increasing the efficiency of applications that operate above it.

2.1.2 Localiser

The Localiser algorithm [26] aims at solving the *network mismatch* problem while ensuring that the overlay network remains connected despite failure of large percentage of nodes. It also ensures a fair degree distribution among every peer in the system. The localiser algorithm is fully decentralized and only relies in local knowledge. In [26] the authors show the impact of the algorithm on the operation of the unstructured overlay network maintained by the Scamp protocol [16].

The goal of Localiser is to bias the topology of the overlay network such that the majority of neighbors kept by each node are "close" peers (given a "network distance" criterion). The protocol also aims at biasing the original overlay such that every node has the same amount of neighbors, which also contributes to increase the failure resilience of the overlay.

Localiser was designed based on a metropolis model [27]. This is an iterative model in which an utility function f is minimized. In order to do this, on each iteration, the utility of the current overlay configuration c is compared with the utility of a possible alternative configuration c'. The algorithm is probabilistic given that the acceptance of an adaptation of the overlay configuration from c to c' is determined by a decreasing probability function in $f(c') - f(c)$. The reader should notice that this approach allows to perform adaptations to the overlay topology which increase the value of the function f. This however is required by the algorithm to avoid local minima configurations.

The specific algorithm is based on a periodic operation executed by every node in the system. In each iteration, each node n executes the following steps:

1. Node n selects at random 2 overlay neighbors p_1 and p_2 and computes for each one a local cost function.

2. Node n obtains the node degree of p_1 and p_2. Furthermore it also obtains from p_1 the cost of the link between p_1 and p_2.

3. Node n locally computes the global benefit of exchanging its link with p_1 for a link between p_1 and p_2.

4. Finally, node n uses a probabilistic function, which takes into account the benefit, the expected cost of the adaptation, and node degree, to make the decision of applying, or not, the link exchange. If the exchange is accepted, n coordinates with p_1 and p_2 the steps required to perform the adaptation.

The algorithm can be parameterized to give more weight, in the probability function, to the balance of node degrees or to the proximity of neighbors (notice that this proximity notion is encoded in the link cost function). The probability function can also be tuned to promote maintenance of low cost configurations or to increase the probability of a faster convergence.

This scheme allows an unstructured overlay to self adapt to reach a configuration where most neighbors are "local" (i.e., with a small link cost) while at the same time, improving the degree distribution. This leads to more efficient overlay configurations with increased resilience to faults.

2.1.3 Araneola

Araneola [28] is a protocol for reliable and efficient multicast based on unstructured overlay networks. The protocol is able to build, and maintain, a bounded degree overlay. Moreover, Araneola, includes a mechanism for exploiting network proximity in the overlay.

This mechanism operates independently of the main task of the Araneola protocol. It operates by adding new links to the overlay to promote communication between close peers. The proposed extension to the original protocol is based on two distinct components, namely: a task to locate nearby peers, and another task which establish connections with discovered nearby peers. These tasks operate as follows:

Locating nearby nodes The task operates by capturing network performance values from peers selected from a local partial view. The performance values are used to sort nodes into a candidate list which is then used by the second task. Several techniques can be employed to capture performance values (different metrics require different techniques). For instance, the authors of [28] rely on a network-level hop-count between peers which is extracted using the UNIX *tracepath* utility (aiming at lowering point-to-point latency in the network).

Connecting to nearby nodes This task tries to maintain a number of nearby neighbors equal to a target value *NB* (*NB* is a protocol parameter). Periodically, if the number of nearby neighbors of a node falls bellow the target value, the node issues a CONNECT_NEARBY request to the first peer in its candidate list (the list generated by the previously described component). A node which receives a CONNECT_NEARBY message will accept the connection, and add the issuing node to its nearby neighbors set, if it has a number of nearby neighbors below *NB*. If the node accepts the request it replies with a CONNECT_OK_NEARBY. Upon the reception of a CONNECT_OK_NEARBY the receiving node adds the sender to its nearby neighbors set, unless the number of its nearby neighbors has reached the target value of *NB*. In the later case, the node will send a LEAVE_NEARBY message, which will result in the removal of the newly established connection.

This extension to the original Araneola protocol is able to correlate the topology of the overlay and the topology of the underlying network. As a result, better links are used in the overlay and the latency of the dissemination process is decreased.

2.1.4 GoCast

GoCast [29] is a protocol for reliable group communication that operates by building a multicast tree on top of an unstructured overlay. This overlay network is biased to promote low latency links and a constant degree for all nodes in the system.

GoCast operates by maintaining both near and random neighbors. The protocol also relies in a peer sampling service which is used as a bootstrap overlay, and also as a source for random peers for the protocol operation. The protocol tries to select a sample of C_{rand} and C_{near} nodes such that the sum of these numbers converges to a given value D (all these values are protocol parameters). TCP connections are maintained for every neighbor of each node, and all communication made between such peers is done by relying in these connections. UDP is used for communication for all remaining nodes (for instance, to obtain latency measurements).

Periodically every node in the system performs two operations; the first to maintain random neighbors and the second to maintain a nearby neighbors. We now describe these operations.

Maintaining random neighbors To this purpose, each node p compares its current number of random neighbors with the target value: C_{rand}. If these values are equal, no operation is required. If the number of random neighbors is below the target value, then the node adds a random node (obtained from the peer sampling service) to its neighbors set, and establishes a TCP connection to it. Finally, if the number of random neighbors is above C_{rand}, the node p might take one of the following corrective measures:

- If the current number of ps random neighbors is equal or above $C_{rand} + 2$, p selects two random neighbors, q and r, and asks them to replace their links with p for a link between q and r. This allows to reduce by 2 the number of random neighbors of node p, while preserving the number of random links for all remaining nodes in the system.
- If one of ps random neighbors, q, has a number of random neighbors above C_{rand}, p simply asks q to remove the link between them. This allows for two nodes, in a single step, to approximate their number of random neighbors to the target value of C_{rand}.

Maintaining nearby neighbors GoCast mechanism to maintain nearby neighbors is composed of three sub protocols. The first serves to replace nearby neighbors for other nearby neighbors with a lower latency. The second is used to add nearby neighbors to the partial view of the node, when the number of nearby neighbors is bellow the target value of C_{near}. Finally, there is a protocol to remove nearby neighbors when their number is equal or above $C_{near} + 2$.

- Periodically, a node p, measures its latency to a random peer, say r. If the estimated latency to r is lower than an existing nearby neighbor n, p might exchange its link with n for a link with r if and only if the following four conditions are true: *i)* the number of nearby neighbors of n must at least $C_{near} - 1$; *ii)* the number of nearby neighbors of r must be below $C_{near} + 5$; *iii)* if the number of nearby neighbors of r is above C_{near}, then r must have a nearby neighbor with a higher latency than the estimated latency value between p and r and finally, *iv)* to ensure that there is a relevant gain in the link exchange, the latency between p and r must be at least, half of the latency between p and n.
- In order to add new nearby neighbors, p selects a random peer r and simply adds it as his neighbor if, and only if, the conditions *ii* and *iii* depicted above, are true.
- If node p has a number of nearby neighbors equal or above $C_{near} + 2$ it drops the connection to a nearby neighbor which does not have a number of nearby neighbors below the threshold of $C_{near} - 1$.

GoCast can successfully bias an unstructured overlay network to improve its performance, reducing the overall latency, and also converge to a configuration where all nodes have a degree value between $D - 2$ and $D + 2$.

2.1.5 T-Man

T-Man [30] is a generic topology management scheme for unstructured overlay networks. The goal of the protocol is to reach a given target topology from a pure random overlay. Examples of target topologies are torus, ring, or some user defined topology. The topology is defined by fixed size partial views that are maintained at each node (the size c of these partial views is a protocol parameter).

The protocol relies on a ranking function that, at any give node, is able to sort a set of peers accordingly to some preference. The ranking function must be able to encode, somehow, the desired topology, in the sense that it must be able to provide clues, for every node, concerning the most relevant peers that they should keep as neighbors, in order to generate the desired topology. The operation of the protocol is based on a periodic exchange of information performed by every node, which works as follows:

1. A given node n starts by using the ranking function to select the neighbor p that is closer to itself;

2. then n sends to p a set of peers containing n's identifier, n's partial view, and a random sample of other peers in the system[1];

3. when p receives this information from n it replies with a similar set of peers: p's identifier, p's partial view, and a random sample of other nodes in the system ;

4. after this exchanged is performed, both nodes use a merge function which also relies in the ranking function to return the c best peers from the union of each node partial view and the received set of peers;

5. each node partial view is then updated to contain the c nodes returned by the local execution of the merge function.

This protocol allows the overlay network to converge for the desired topology. Because nodes exchange information with their closest peers, the probability of receiving information concerning other peers which are good candidates to improve the overlay topology is increased, as there is a high probability that nearby nodes will be trying to converge their partial views to contain similar peers.

2.1.6 Plumtree

The Plumtree protocol [25] is a dissemination scheme which relies on a reactive unstructured overlay network to embed a highly resilient low cost spanning tree. The protocol uses this spanning tree to bias the communication pattern of a gossip-based broadcast protocol, in order to lower the inherent overhead of the gossip protocol, without impairing its reliability. To do this, eager push is used in overlay links which belong to the spanning tree while, for both fault-tolerance and support the healing mechanism of the spanning tree, lazy push is used on the remaining overlay links.

The protocol has two main components. The first builds the spanning tree structure by removing redundant links when they exist. The second component is able to heal the spanning tree structure whenever a node fails or leaves the system, and also recover from message loses due to membership dynamics. We now describe the behavior of each component:

Building the spanning tree Initially, Plumtree assumes that every link in the overlay belongs to the spanning tree. The same is true whenever a new link is added to the overlay due to natural dynamics in the peer-to-peer system. When a link is used to transmit a redundant gossip message, the protocol removes that link from the spanning tree. Therefore, when the first broadcast message is disseminated, it is eagerly flooded through the overlay. However, when the dissemination process is concluded, the spanning tree has been completely established, and following broadcast messages are only eagerly transmitted in the links which belong to the tree.

Healing the spanning tree In the presence of node failures the spanning tree may become disconnected. This results in poor reliability, as disconnected nodes will miss broadcast messages. To address this, nodes also send lazy push messages through the remaining links of the overlay (*e.g.* links which are not part of the spanning tree). Messages transmitted by lazy push only carry an unique identifier for broadcast messages and omit the original payload, therefore these messages are in fact announcement messages that a new broadcast message (or messages, as more than one message identifier can be carried in a single IP packet) is available.

When a node receives an announcement for a given broadcast message that it has not received yet, it starts a timer. When the timer expires, if the payload is not yet locally available, the node request the payload

[1]This random sample can usually be extracted from an out-of-band peer sampling service such as Cyclon [17] or HyParView [5].

to the neighbor who sent the announcement. This message implicitly adds the link to that neighbor to the spanning tree, effectively healing the tree.

After a failure, several nodes may concurrently add links to the spanning tree; this may result in the creation of redundant links. However, the mechanism for building the tree will detect such redundancy during the dissemination of the next broadcast message and, as result, will prune existing redundant links (if any).

This protocol is completely decentralized, and by using two distinct transmission modes (eager and lazy push) it can lower the overhead of disseminating broadcast messages to a value comparable to some multicast structured solutions. The use of lazy push ensures, at a low cost, that the natural resilience of gossip protocols is maintained. Also, as a result of the strategy used to select links, links that form spanning tree are those with lower latency.

2.2 Key Properties to Preserve

In the previous paragraphs, we have surveyed a number of protocols to optimize the overlay network to achieve better performance. There are however a number of key topology properties[2] that should be preserved during the optimization, as listed below:

Connectivity The overlay is connected if there is at least one path that allows every node to reach every other node in the overlay. The overlay should remain connected despite failures that might occur. If this requirement is not meet, isolated nodes will not receive broadcast messages.

Degree Distribution The degree of a node is the number of edges of a node, or in other words, the number of neighbors that a given node has[3]. The degree of a node is both a measure of its reachability on the overlay and also a measure of its contribution to maintain the overlay connected. If the probability of failure is uniformly distributed in the node space, for improved fault-tolerance, all nodes should have the same degree value. Nodes that have a small degree will more easily become disconnected from the overlay as the number of faults increases. On the other hand, the failure of nodes with high degree may have an undesired impact in the overall connectivity of the overlay.

Average Path Length A path between two nodes in the overlay is a set of edges that connect one node to the other. We define the average path length as the average of all shortest paths between all pair of nodes in the overlay. The average path length is closely related to the overlay diameter. To promote the overlay efficiency when broadcasting messages, the average path length between nodes should be as small as possible. Large values of average path length have two negative implications: *i)* The number of hops required for messages to reach all nodes increases, with a negative impact in the broadcast latency and, *ii)* the broadcast process becomes more prone to failures, as the time window for failures increases (*e.g.* the number of steps required to fully disseminate a message increases).

Clustering Coefficient The clustering coefficient of a node is the number of edges between that node's neighbors divided by the maximum possible number of edges across those neighbors. The clustering coefficient captures a density of neighbor relations across the neighbors of a given node, having it's value between 0 and 1. The clustering coefficient of a graph is the average of clustering coefficients across all nodes. The clustering coefficient of an unstructured overlay should be as small as possible, and failure to meet this requirement

[2]Some of these properties are intrinsically related with graph properties, as it is, an overlay network can be seen as a graph, where nodes are represented by vertex, and links, or neighboring relations, are represented by edges. Depending on the nature of these relations, graphs can be directed or undirected.

[3]To be precise, usually partial views establish asymmetric neighboring relations, therefore the degree is viewed as two distinct components: in-degree and out-degree. However, in this chapter we will mostly focus on systems which use a gossip-based membership protocol which offers to nodes access to symmetric partial views therefore, we do not consider these components as being distinct.

has the following negative implications: *i)* the number of redundant messages received by nodes when disseminating data increases, especially in the first steps of the dissemination process; *ii)* the diameter of the overlay increases, which in turn will make the average path length increase, and finally *iii)* it decreases the fault resilience of the overlay, as areas of the overlay which exhibit a high value of clustering can more easily became disconnected.

The interested reader can find a more detailed discussion of these and other properties of random overlays in [5] and [31].

2.3 Performance Metrics

Several metrics can be used to measure the performance of a gossip-based broadcast protocol operating on top of a random overlay network. In this chapter we focus mainly on offering high dependability for applications requiring reliable broadcast. Therefore, the metrics that we present here are mostly related with the operation of broadcast protocols, as we specifically aim at biasing the overlay topology to minimize the message dissemination overhead, while preserving the typical reliability of gossip-based broadcast protocols.

Average Link Cost We assume that each link of the overlay may be tagged with a *cost*. Costs may be associated to a concrete (underlay) network metric such as link latency. However, the link cost may also be associated to higher level utility metrics; for instance, in a file sharing peer-to-peer system it could be a measure of the semantic similarity between the data stored at the edges of a link.

Reliability Gossip reliability is defined as the percentage of correct nodes that deliver a given broadcast message. A reliability of 100% means that the protocol was able to deliver a given message to all active nodes or, in other words, that the message resulted in an atomic broadcast as defined in [2].

Latency We define latency of a gossip-based broadcast protocol as the time between the instant when a message is transmitted by its original sender, to the moment when the last peer, which receives the broadcast message, delivers it to the application layer. The reader should notice that one can have good latency values by failing to deliver the message to a large number of nodes. Therefore the goal of a gossip-based broadcast protocol should be to achieve a low latency value while ensuring a high reliability. Moreover, latency values are only comparable between broadcast protocols that exhibit a similar reliability value, for systems composed of the same number of nodes.

Last Delivery Hop The Last Delivery Hop measures the number of hops required to deliver a broadcast message to all recipients. When a message is gossiped for the first time, its hop count is set to 1 and, each time it is relayed in the overlay, the hop count is increased in one unit. The last delivery hop is the hop count of the last delivery for a given broadcast message or, in other words, is the maximum number of hops that a message must be forwarded in the overlay before it is delivered to all participants. This metric is closely related with the diameter of the overlay and with the the latency of a gossip protocol. In other words, it can be seen as an efficiency metric.

2.4 Methodologies

We can distinguish two main methodologies that allow to introduce some degree of structure in unstructured overlay networks. These methodologies operate at distinct levels:

Overlay Optimization This methodology consists in manipulating the neighboring relations among peers, effectively changing the unstructured overlay network topology and changing the communication patterns among peers (*e.g.* by changing the communication peers for each node). This methodology aims at improving the overall overlay network, which is the support for several gossip protocols, by replacing existing links between peers for alternative links which present a better performance given an efficiency criteria (*e.g* such as latency). In Sect. 3 we describe with some detail a protocol based on this approach.

Gossip Optimization This methodology consists in selecting different communication modes for transmitting messages between different peers. The possible modes to transmit messages are: eager push, lazy push and pull [25]. This methodology supports the emergence of structure, from the unstructured overlay, by establishing patterns in the communication modes used among peers. Protocols based on this approach are able to: *i)* make a better usage of network resources; *ii)* reduce the communication overhead of gossip protocols and also, *iii)* address heterogeneity of nodes. In Sect. 4 we describe a protocol which employs this technique. Another protocol which also uses this methodology can be found in [25].

The first technique is used in a larger number of proposed solutions. Protocols such as Narada, Localiser, Araneola, GoCast and T-Man use variants of overlay optimization to bias, or adapt, the topology of random overlay networks. The second technique has only recently been proposed (the protocol presented in Sect. 4 and Plumtree are two of the few protocols employing this technique).

3 Overlay Optimization

3.1 Overview

In this section, we describe a protocol to **B**ias the **O**verlay **T**opology according to some target efficiency criteria **X**, or simply X-BOT. A target efficiency criteria can be, for instance, to better match the topology of the underlying network. However, in X-BOT, biasing the overlay is done without compromising key properties of random overlay networks (such as the node degree, small diameter, and low clustering coefficient), which are essential to ensure the efficiency and reliability of some peer-to-peer applications such as, gossip-based broadcast protocols.

X-BOT relies on the combined use of two distinct partial views, inspired by HyParView [5], a gossip-based membership protocol that illustrated how to achieve a high resilience to faults (as high as 80% of simultaneous nodes failures) in a gossip-based broadcast protocol using a low fanout value. The architecture of this protocol is based in the combination of a small sized active view and a larger passive view. X-BOT relies on a similar architecture in order to optimize the overlay network used for message dissemination.

The goal of the protocol is to reduce, as much as possible, the average link cost of the overlay network defined by the active views. For that purpose, X-BOT actively bias the neighbors in the active view using random peers extracted from the larger passive view. This is feasible because only the active view is used for communication among peers. Moreover, the passive view is maintained by a cyclic strategy [5] which ensures that the contents of this view are periodically updated and therefore, gives access to an increasing number of potential neighbors over time to each node. X-BOT is flexible allowing to bias a topology for different criteria such as, link latency or content similarity as a result of being independent of the cost function. The protocol only requires costs to be comparable and totally ordered.

3.2 Architecture

X-BOT maintains two distinct, and disjoint, partial views: a small sized symmetric active view and a larger cyclic passive view. As in HyParView, the active view is used mainly for communication among peers and TCP connections are maintained to neighbors in this view.

X-BOT assumes that all nodes have access to a local *Oracle*. Oracles are components that export a `getLinkCost(Peer p)` interface, which returns the link cost between the invoking node and the given target node p in the system (since there is a single link to each neighbor, in this chapter we use interchangeably link cost or node cost when referring to the output of the Oracle). The implementation of such Oracles are not discussed in the chapter. However, for completeness, we provide a brief description of three simple Oracles.

3.2.1 Oracles

Latency Oracle This Oracle operates by measuring round trip times (RTT) to peers. This can be performed by exchanging probe messages with Oracles located at other nodes[4]. The Oracle must be aware of the peers which are known at the local host, and it slowly measures the RTT for each know node (this value can be directly used as the cost value).

Internet Service Provider Oracle In a setting where exchanging messages across different ISPs has an increased monetary cost, it might be useful to keep as many neighbors as possible that share the same ISP. Such Oracle can be built by maintaining information concerning the local ISP and a table of costs for each known ISP. When the Oracle becomes aware of a new peer, it simply exchanges local ISP information with the remote Oracle and asserts the cost for the link using the local cost table.

IP-based Oracle X-BOT can also leverage on previous work addressing the use of inexpensive Oracles that do not require the exchange of control information [32, 33]. Such Oracles are able to calculate neighbor proximity values, which can be used as cost, using IP aggregation information (for instance, using a match of common IP prefixes to calculate a measure of proximity between two peers).

Oracles are not required to be perfect for the operation of the protocol, in the sense that provided costs are not required to be 100% accurate. The interested reader can refer to [34] for experimental results that show the effect of unreliable Oracles in the protocol.

3.2.2 Rationale

The rationale of X-BOT is as follows. As in HyParView X-BOT maintains a small active view and a larger passive view. However, unlike HyParView, that strives to ensure the stability of the overlay, X-BOT relaxes stability to be able to continuously improve the overlay. This allows the topology of the unstructured overlay to self adapt to better match the requirements of the application executed on top of it. Periodically, each node starts an *optimization round* in which it attempts to switch one member of its active view for one (better) neighbor of its passive view. In the optimization protocol, a node uses its local Oracle to obtain an estimate of the link cost to some random selected peers of its passive view. The number of nodes π for which the cost is measured in each optimization round is a protocol parameter called *Passive Scan Length*. This parameter limits the maximum number of optimization exchanges started by each node each time it runs the optimization procedure. Similarly to the original HyParView protocol, the passive view is not biased.

X-BOT strives to preserve the connectivity of the overlay. This has two implications: *i)* nodes only make an effort to optimize their active views when they have a full active view (*i.e.*, no bias is applied to active views until connectivity of the nodes is ensured). Furthermore, each node attempts to maintain some unbiased neighbors, as we explain in the next section; *ii)* we try to preserve the degree of nodes that participate in a optimization procedure, given that the node degree has a significant impact on the connectivity of the overlay. To ensure this, each optimization round involves typically 4 nodes in the system as we describe later in the text.

[4]Probe messages can also be piggybacked on application traffic, for instance, when measuring the cost for peers in the active view.

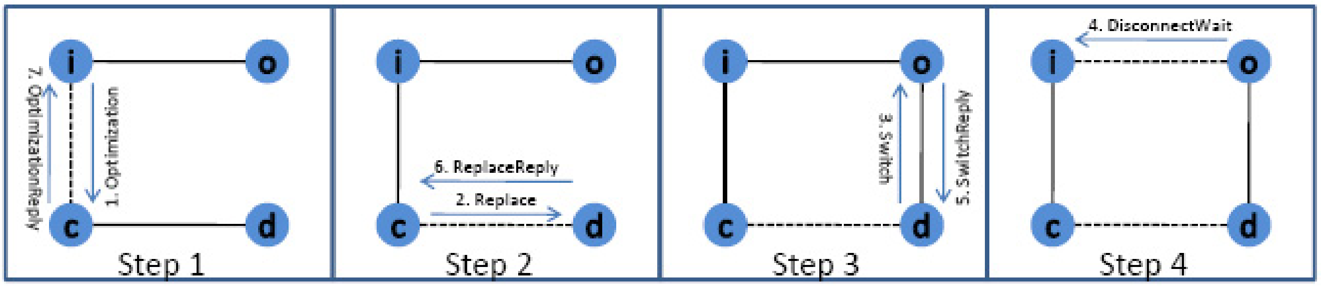


Figure 1. Steps of the optimization protocol

3.2.3 Unbiased Neighbors

By blindingly imposing a bias in the topology of the overlay, one may easily break some of the desirable key properties of a random overlay, such as the low clustering coefficient, low average path length, or connectivity [29]. The negative effect of such bias can be even more notorious in the architecture of X-BOT, that relies on small partial views. To avoid this flaw, X-BOT does not bias all members of the active view. Instead, each node maintains both "high-cost" (unbiased) and "low-cost" (biased) neighbors. The number of unbiased neighbors each node keeps is a protocol parameter called *Unbiased Neighbors* and simply denoted μ.

Unfortunately, it is not trivial to decide which peers have a "high-cost", given that nodes are not expected to have global knowledge of the system, not only regarding membership information but also regarding global metrics, such as the average link cost in the overlay. To circumvent this limitation, X-BOT maintains the active views of each node sorted by link cost, where the first element of each active view is the neighbor with the largest link cost. Therefore, a node never applies any bias to the first μ members of its active view. Also, whenever a change occurs in the elements of a node's active view, the active view is reordered using the same criterion. The same happens if the cost of a node changes due to modifications in the execution environment.

3.3 Algorithm

In this section we briefly describe the operation of the overlay optimization algorithm (a more detailed description can be found in [34]). The algorithm executed at each optimization round is depicted in Algorithm 1 and illustrated in Fig. 1. The algorithm listing has been simplified for clarity, for instance, we omitted some insertions of nodes into passive views and the mechanisms required to ensure the symmetry of active views.

Usually an optimization round involves 4 nodes of the system, and each round is composed of 4 steps, one for each node that participates in the optimization. The goal of these steps is to exchange two of the existing links in the overlay for other two links such that the cost of the two added links is lower than the cost of the original links. To ensure that the overall cost of the overlay is reduced in an optimization round, and because it is assumed that link costs are symmetric, the optimization scheme only requires that two of the four participating nodes in a round consult their local Oracles.

A complete optimization round requires the serial exchange of seven messages. However, in most cases, each node involved in the optimization only has to send and receive at most two messages. Given that the optimization of the overlay can be executed as a background activity, the cost of the adaptive mechanism can be easily tuned to become negligible when compared with the (application) data traffic.

Oracles are not required to be perfect, in the sense that they might provide information that is not fully accurate. Namely, two nodes may obtain different costs for the same link when they consult their local Oracle. For instance, [33] states that longest IP prefix and latency has an approximate correlation of -0.85. In cases where Oracles are not perfect, nodes have to make decisions with inaccurate information. Therefore, the **isBetter** evaluation function

Algorithm 1: Optimization Procedure

```
Data:
    activeView //fixed size sorted list
    passiveView //fixes size list

1:  every $\Delta T$ do
2:      if isFull(activeView) then
3:          candidates $\longleftarrow$ randomSample(passiveView, $\pi$)
4:          for $i := \mu$ ; $i <$ sizeOf(activeView) ; $i := i + 1$
5:              $o \longleftarrow$ activeView[$i$]
6:              while candidates $\neq \{\}$ do
7:                  $c \longleftarrow$ removeFirst(candidates)
8:                  if isBetter($o$,$c$) then
9:                      Send(OPTIMIZATION($o$, myself),$c$)
10:                     break

11: upon Receive(OPTIMIZATION, $o$, $i$) do
12:     if $\neg$ isFull(activeView) then
13:         activeView $\longleftarrow$ activeView $\cup \{i\}$
14:         Send(OPTIMIZATIONREPLY(true, $o$, $\perp$, myself),$i$)
15:     else
16:         $d \longleftarrow$ activeView[$\mu$]
17:         Send(REPLACE($o$, $i$, myself),$d$)

18: upon Receive(OPTIMIZATIONREPLY,answer,$o$,$d$,$c$) do
19:     if answer then
20:         if $o \in$ activeView do
21:             if $d \neq \perp$ then
22:                 Send(DISCONNECTWAIT(myself),$o$)
23:             else
24:                 Send(DISCONNECT(myself),$o$)
25:             activeView $\longleftarrow$ activeView $\setminus\{o\}$
26:         passiveView $\longleftarrow$ passiveView $\setminus\{c\}$
27:         activeView $\longleftarrow$ activeView $\cup\{c\}$

28: upon Receive(REPLACE, $o$, $i$, $c$) do
29:     if $\neg$ isBetter($c$,$o$) then
30:         Send(REPLACEREPLY(false, $i$, $o$, myself),$c$)
31:     else
32:         Send(SWITCH($i$, $c$, myself),$o$)

33: upon Receive(REPLACEREPLY,answer,$i$,$o$,$d$) do
34:     if answer then
35:         activeView $\longleftarrow$ activeView $\setminus\{d\}$
36:         activeView $\longleftarrow$ activeView $\cup\{i\}$
37:     Send(OPTIMIZATIONREPLY(answer,$o$,$d$,myself),$i$)

38: upon Receive(SWITCH,$i$,$c$,$d$) do
39:     if $i \in$ activeView or received(DISCONNECTWAIT from $i$) then
40:         Send(DISCONNECTWAIT(myself),$i$)
41:         activeView $\longleftarrow$ activeView $\setminus\{i\}$
42:         activeView $\longleftarrow$ activeView $\cup\{d\}$
43:     Send(SWITCHREPLY(answer,$i$,$c$,myself),$d$)

34: upon Receive(SWITCHREPLY,answer,$i$,$c$,$o$) do
35:     if answer then
46:         activeView $\longleftarrow$ activeView $\setminus\{c\}$
47:         activeView $\longleftarrow$ activeView $\cup\{o\}$
48:     Send(REPLACEREPLY(answer,$i$,$o$,myself),$c$)

49: isBetter($old$,$new$)
50:     cOld := Oracle.getLinkCost($old$)
51:     cNew := Oracle.getLinkCost($new$)
52:     return cOld > cNew $\wedge \frac{(cOld - cNew)}{cOld} \geq$ THRESHOLD
```

(depicted in Algorithm 1) includes some hysteresis, namely, a given link *new* is only considered to have a lower cost than another link *old*, if the difference between the cost (obtained through the Oracle) offers a gain above a given *Threshold*, which is a protocol parameter introduced to address the inaccuracy of Oracles. The threshold value can be calculated experimentally for each Oracle. This value should be related with medium precision and error drift of Oracles, which can be measured in a semi-controlled environment such as Planet-lab [35].

3.4 Performance

In the following sections we show performance results of the X-BOT protocol. To better understand the impact of this approach, which affects the properties of the unstructured overlay network, we used simulation, which allows us to extract performance metrics in a system composed of a large (10.000) number of nodes.

3.4.1 Experimental Setting

The figures presented are the result of extensive experimental evaluation of this approach in the PeerSim simulator [36] using its cycle based engine. Each cycle is a virtual time slice where each node can execute periodic operations. Additionally, a single broadcast message is also disseminated in each cycle to enable the observation of the reliability of a gossip-based broadcast protocol operating on top of biased unstructured overlays. All experiments were conducted in a system composed of 10.000 nodes. Simulation were run on top of a network model composed of 13.037 routers generated by Inet-3.0 [37] with its default parameters. In order to calculate the cost between neighbors, we calculated the shortest paths in the underlying network among the 10.000 overlay nodes. The cost between two nodes is the sum of the pseudo geographical distance, generated by Inet-3.0, of links between the routers that form shortest paths. All results report an average extracted from 5 independent runs. Each one of these runs used one of 5 distinct random network topologies generated using the methodology described above.

Simulations use the configuration parameters for HyParView reported in [5]. The most relevant configuration parameters are the active view size, which was set to 5 and the passive view size, which was set to 30. Different *Unbiased Neighbors values* (μ) ranging from 0 to 5 (all) were tested; the last configuration corresponds to the operation of the original HyParView protocol (given that no bias is applied to any member). Moreover, in all simulations we used the following parameters:

The period between optimizations was set to 2 simulation cycles. In each simulation cycle each node initiates an exchange with a random peer in the overlay, which results in the update of both nodes passive views. Moreover, in average, a node will also participate in an exchange initiated by other peer. Therefore, setting the period between optimization to two cycles ensures that, between executions of optimization steps the passive view of nodes is updated, increasing the possibility of selecting new nodes that can be used to improve the active view of the node.

Passive Scan Length (π) was set to 2, so each time a node executes the step 1 of the optimization algorithm, it measures, at most, 2 nodes from its passive view. This also limits the number of nodes which are replaced in the active view of any given node in a single round as 2. Setting π to a small value allows to achieve two goals: i) It promotes some stability in the overlay, as we avoid to exchange the majority of nodes in the active view of a single node in the context of a single optimization execution.; ii) It lowers the cost of the overall optimization process.

3.4.2 Stable Environment

First, X-BOT performance results are shown for a stable environment, where no failures were induced. Simulations were run for 250 cycles. Nodes were added to the overlay using the *Join* mechanism provided by the HyParView protocol as described in [31] and had access to local perfect Oracles (*e.g.* their precision was of 100%).

Overlay Properties. Figure 2 shows the average link cost of the overlay as the system evolves. As expected, while the original HyParView protocol ($\mu = 5$) shows a constant link cost in steady state, X-BOT, is able to lower its

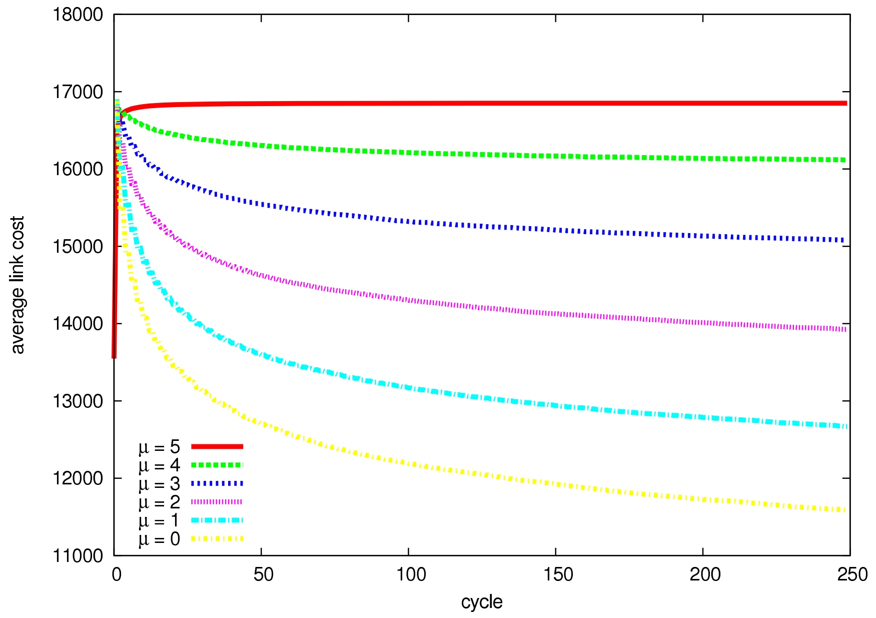


Figure 2. Average Link Cost

average link cost from approximately 5% while maintaining 4 unbiased neighbors to 25% when keeping 1 unbiased neighbor or even 32% when no unbiased member is maintained in the active view. Notice also that, although the optimization process works continuously, 50 simulation cycles is enough to obtain a visible improvement in the average link cost.

Figure 3 depicts results for clustering coefficient. As expected, if the bias is applied to all members of the active view the clustering coefficient of the overlay increases. On the other hand, maintaining a single unbiased neighbor is enough to partially mitigate this effect. However, as it can be observed after 250 simulation cycles, the clustering coefficient of the network with a single unbiased member is still above that of the original HyParView protocol. Interestingly, when 2 to 4 unbiased neighbors are maintained in the active view, the clustering coefficient drops to values below those obtained with 5 random neighbors. This phenomenon can be explained as follows: By maintaining active views sorted by cost, the selected unbiased neighbors are those with a larger cost known by each node during the lifetime of the system. In other words, X-BOT with no extra cost, promotes the maintenance of "long distance" links in each active view. The same effect is also visible in Fig. 4 where we show the average path length values.

Broadcast Reliability. Experiments were also conducted to assert the impact in the reliability of a gossip-based broadcast protocol operating on top of a biased overlays resulting from the operation of X-BOT. Experiments were conducted as follows: in each simulation cycle, we select a random node in the system to broadcast a message. After the dissemination process is complete, we evaluate the reliability of the broadcast by observing the percentage of active nodes that receive that message. The reliability obtained for all configurations of the protocol was of 100% in steady state. This shows that the overlay maintained by the protocol did not became disconnect due to the operation of X-BOT.

Effect of Node Clustering. In order to illustrate some of the benefits of X-BOT, the algorithm was compared with T-Man in a scenario where nodes are highly clustered in the cost function space. Notice that in these scenarios the optimization of the overlay can lead to the creation of disconnected clusters.

The experiment was conducted using a version of T-Man [38] for the PeerSim simulator, which can use the same

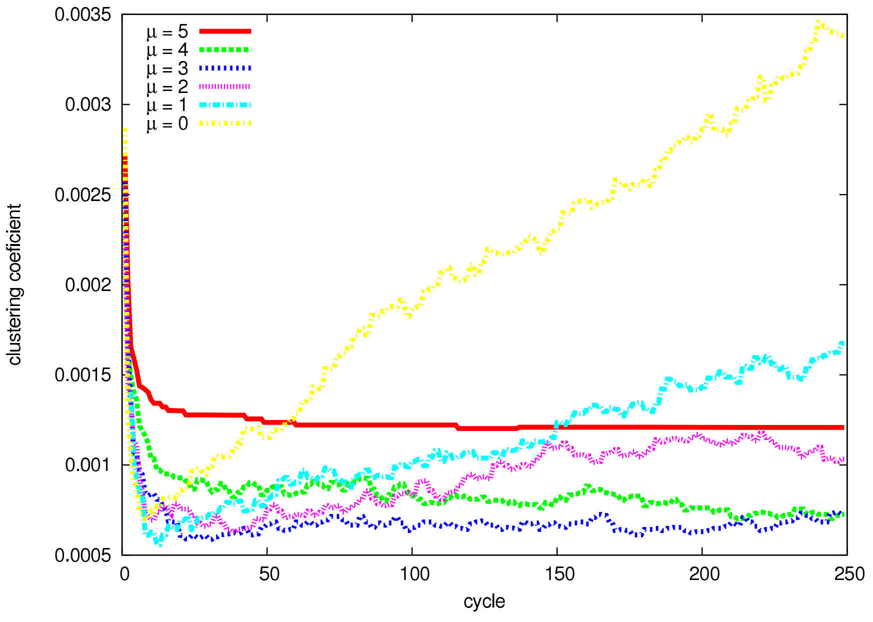


Figure 3. Clustering Coefficient

Oracles as X-BOT. The selection of this version of T-Man was motivated by an additional parameter k present in the protocol which is similar to the μ parameter of X-BOT. The parameter limits the biasing the protocol performs over nodes partial views to $c-k$ neighbors. The remaining k neighbors of each node are selected at random.

The experiment was designed as follows: It starts by positioning 1024 nodes in two spatial clusters. Then HyParView is used to build an overlay connecting these nodes. The resulting overlay is depicted in Figure 5(a) where each node is represented by a cross, and each overlay link is represent by a line. As the reader can observe, the overlay is highly connected.

Then X-BOT, with μ set to 1, and T-Man, with k set also to 1, are executed for 250 simulation cycles to optimize the distance between neighbors, resulting in the topologies depicted, respectively, in Figure 5(b) and Figure 5(c). As expected, X-BOT replaces a large number of links between the two clusters by better links inside each cluster. However both clusters are still highly connected. The same is not true for T-Man, which breaks the connectivity between the two clusters. This happens because the random selection of nodes in T-Man is implicitly biased to nodes that are, at most, at two hops of distance whereas X-BOT extracts distant neighbors from an unbiased low cost passive view. Moreover, as described earlier, X-BOT (unlike T-Man) promoted the maintenance of long cost links in the overlay which, is such scenarios, is essential to ensure the global connectivity of the overlay.

3.4.3 Massive Failures

In this section we provide results concerning the reliability of the gossip-based broadcast protocol on top of the optimized overlay network. Specifically, after the induction of massive node failures in the system that range from 10% to 95% of all nodes. These faults were induced in the system after 250 cycles of simulation to ensure that the overlay had time to converge to a biased version. After the induction of failures, simulations were conducted for an extra 250 cycles and in each cycle a random correct node was selected to initiate the dissemination of a broadcast message. After the completion of the dissemination process, the reliability of the broadcast protocol was measured. Figure 6 shows the average reliability obtained while broadcasting 250 messages after massive failures. The reader should notice that all protocol instances present similar values for reliability. This happens

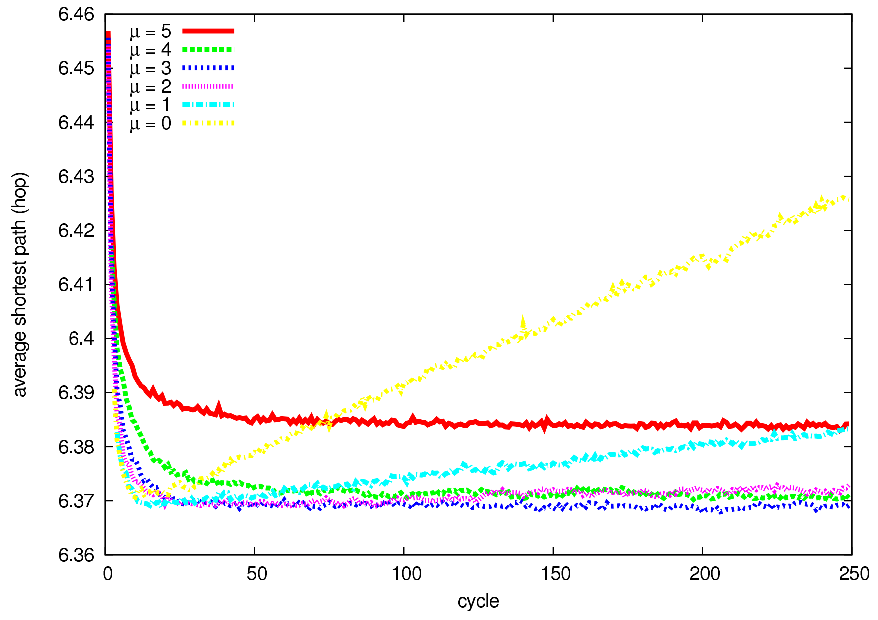


Figure 4. Average Path Length

due to the passive view maintained by HyParView. Notice that, although we use passive views across nodes to bias the topology of the overlay, the properties of the passive view are not affected, thus the healing properties of passive views are not affected by the operation of X-BOT.

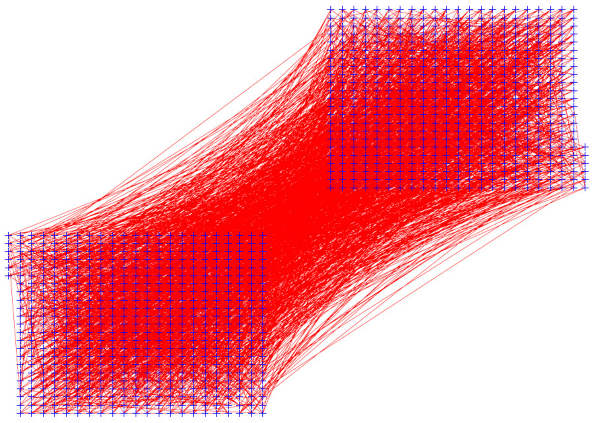
One could expect that maintaining a single unbiased neighbor would decrease the resilience of the overlay to node failures. In fact, one could imagine that if the unbiased member fails, it would be difficult for this member to be replaced by another unbiased member. However, X-BOT only attempts to apply some bias when the active view is complete. Therefore, when a node crashes and needs to be replaced, the replacement is picked at random from the passive view. This policy, combined with the use of a sorted active view explains the effect that we depicted in Sect. 3.4.2 in which unbiased neighbors become implicitly biased for high costs.

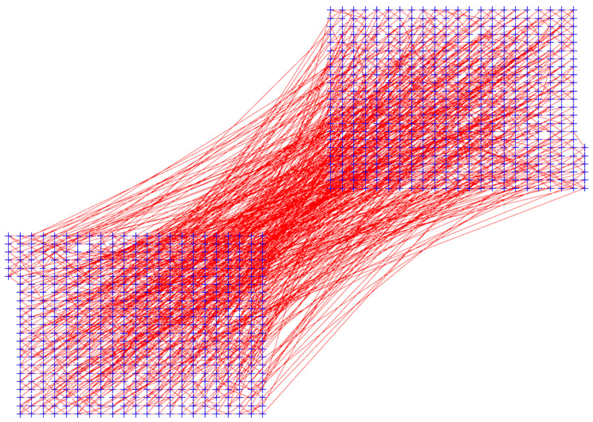
4 Gossip Optimization

4.1 Overview

In this section we describe a protocol which employs the *gossip optimization* methodology. This allows the structure to emerge from the natural operation of a gossip-based broadcast protocol. The approach leverages the fact that two distinct communications modes between peers may be used, namely: eager push and lazy push. A main goal of the protocol is to approximate the efficiency of a gossip-based broadcast protocol to that of a structured broadcast protocol, more specifically, to the efficiency of a dissemination strategy which employs a spanning tree that covers all peers in the system. Moreover the protocol aims at building a solution that exhibits the following characteristics:

- Does not impair the natural resilience of gossip protocols.
- Avoids the necessary overhead to explicitly build and maintain the spanning tree structure. Namely, it avoids the maintenance of additional state for supporting the dissemination structure.


(a) Initial State


(b) Out protocol Result

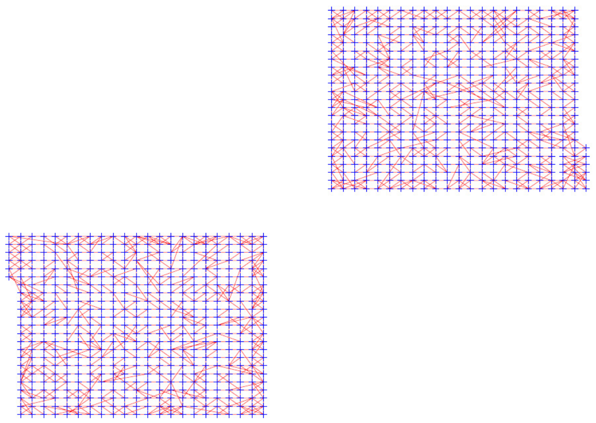

(c) T-Man Result

Figure 5. Our protocol vs T-MAN highly clustered system.

- Uses a decentralized approach, with low requirements in node coordination.
- Optimizes the dissemination process to take node heterogeneity into account in such a way that nodes with higher capacity can contribute more to the dissemination of messages.

In the following sections we describe, in some detail, the operation of the protocol and how it allows to achieve the characteristics listed above.

4.1.1 Background

Gossip-based multicast protocols are often based on an *eager push gossip* approach [3, 7, 12]: A gossip round is initiated by a node that has received a message, relaying it to a number of targets. However, it is well known that this strategy consumes a lot of bandwidth, as the fanout required for atomic delivery leads to multiple copies of each message being delivered to each destination.

A different trade-off can be achieved by using a *lazy push* strategy, which defers the transmission of the payload. In detail, during a gossip round a node will send only an advertisement of the new message. Transmission of the payload is initiated only if the message is unknown to the recipient. This allows the message payload to be transmitted only once to each destination, at the expense of an additional round-trip. Lazy transmission has also

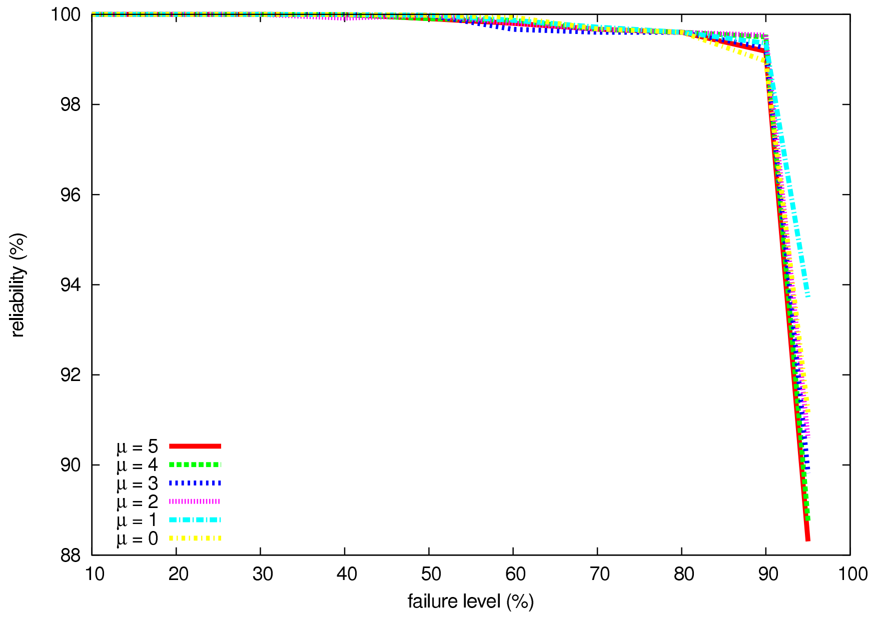


Figure 6. Average reliability of 250 **messages after failures**

an impact on the reliability, as the additional round-trip and resulting increased latency widens the window of vulnerability to network faults. The impact is however small for realistic omission rates and can be compensated by a slight increase in the fanout [19]. Even with a larger fanout, one can significantly improve the use of the network resources namely, in terms of consumed bandwidth.

In fact, one can mix both approaches in a single gossiping round [19], thus providing different latency/bandwidth trade offs depending on how many messages are eagerly transmitted.

4.1.2 Approach

The architecture stems from the observation that, in an eager push gossip protocol, paths leading to deliveries of each message implicitly builds a distinct random spanning tree for each broadcasted message. This tree is composed of links which belong to the overlay network, and therefore, one can say that it is embedded in the underlying random overlay. If one knew beforehand which links are used to transmit messages that lead to deliveries, one could use eager push gossip for those links and lazy push gossip for all others. This would achieve exactly once transmission for each destination. Unfortunately, this is not possible, as one cannot predict which paths in the overlay will lead to message deliveries.

There is however an alternative strategy which is feasible: If one of the embedded trees is selected beforehand for eager push gossip, one increases the probability that the links which compose that tree lead to an increased amount of message deliveries. This happens because lazy push has additional latency, and paths that use it will be outrun by paths that solely rely in eager push. If one assigns nodes and links with higher capacity to support such spanning tree, the performance of the protocol should approach that of a structured approach, where the tree is build beforehand, in such a way that it improves one, or more, efficiency criteria. Note that keeping redundant lazy transmissions is essential to retain the gossip resilience properties. On the other hand, this strategy requires the explicit coordination among peers to maintain a tree structure which imposes additional overhead and complexity.

Instead of selecting a single embedded tree, gossip optimization aims at increasing the probability of implicitly creating spanning trees in a gossip protocol to include nodes and links with higher capacity. The resulting struc-

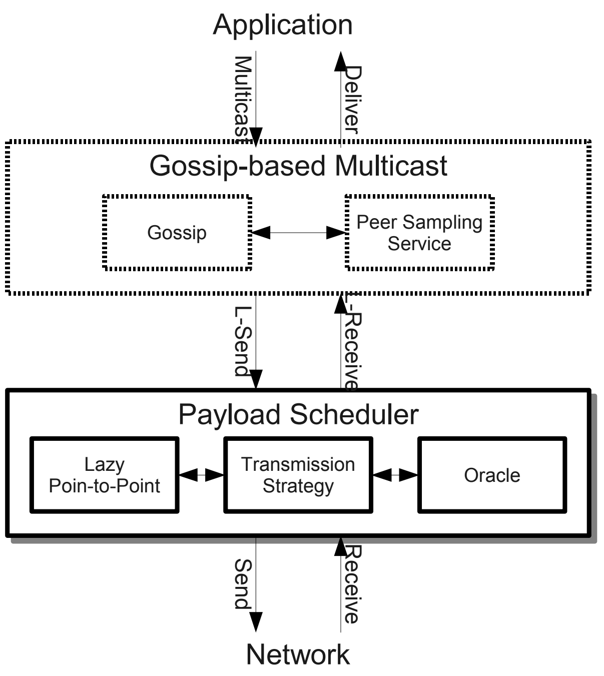


Figure 7. Protocol architecture overview.

tures are therefore probabilistic in nature: Nodes and links are selected with different probabilities for payload transmission (or in other words, for eager push transmissions). Consequently, structure emerges naturally from the strategy used for scheduling message payloads in a combined eager/lazy push gossip protocol. One of the main challenges is to achieve an emergent structure without global coordination, while at the same time obtaining a meaningful performance improvement.

4.2 Architecture

The architecture that supports the operation of the approach described above, is depicted in Fig. 7. Notice that it relies in an additional layer which is located below a pure eager push gossip protocol. This layer, called the *Payload Scheduler*, selects when to transmit the message payload (by using a combination of eager push and lazy push) in a transparent manner for the gossip protocol above. The Payload Scheduler layer can be decomposed into three separate components, also depicted in Fig. 7:

Lazy Point-to-Point The lazy point-to-point module is in charge of intercepting the interaction between the gossip layer above and the transport protocol below. It queries the Transmission Strategy module to decide whether to send the payload immediately (in the case, the exchange is performed in a pure eager push mode) or to delay the payload transmission until a request is received. As we will later describe, this module is also in charge of generating and replying to payload requests.

Transmission Strategy The Transmission Strategy module is the core component of the Payload Scheduler. It defines the criteria that is used to defer payload transmission at the sender and, at the receiver, when to request a specific payload transmission (*e.g.* when to trigger lazy push requests by the receiver). Note that different strategies may be implemented, according to the efficiency criteria is targeted for optimization. Notice that the goal of each strategy is to generate a combined protocol (push gossip plus scheduler) that approximates the behavior of a structured multicast approach.

Algorithm 2: Simple push gossip-based broadcast protocol

```
Data:
    $K$ //known messages

1:  initially
2:      $K \longleftarrow \emptyset$

3:  proc MULTICAST($d$) do
4:      FORWARD(MKID(), $d$, 0)

5:  proc FORWARD($i$,$d$,$r$) do
6:      DELIVER($d$)
7:      $K \longleftarrow K \cup \{i\}$
8:      if $r < t$ do
9:          $P \longleftarrow$ GETPEERS($f$)
10:         for each $p \in P$ do
11:             trigger L-Send($i$,$d$,$r+1$,$p$)

12: upon L-RECEIVE($i$, $d$, $r$, $s$) do
13:     if $i \notin K$ then
14:         FORWARD($i$, $d$, $r$)
```

Oracle The last component of the Payload Scheduler is the Oracle. This component goal, similar to the Oracles described previously in this chapter, is to offer additional information to nodes. This information can be either configured in a static way or extracted, in run-time, concerning the performance data about the operation of the system, for instance, by computing round-trip delays between peers. This data is then used to feed the Transmission Strategy module.

In the remainder of this section, we describe each component of the architecture in detail as well as the existing interfaces among them.

4.2.1 Gossip Protocol Layer

As noted before, a fundamental aspect of this approach is that the Payload Scheduler can operate in a manner that is transparent for the operation of a simple push gossip-based broadcast protocol. Therefore, it can be applied to different gossip protocols, such as [3, 12, 7].

Nevertheless, for self containment, we depict in Algorithm 2 a typical push gossip protocol. This implementation assumes the availability of a peer sampling service [15] providing an uniform sample of f other nodes with the GETPEERS(f) primitive. It assumes also an unreliable point-to-point communication service, such that a message m can be sent to a node p using the L-SEND(m, p) primitive. A message m is received from a peer p by handling the L-RECEIVE(m, p) up-call. The gossip protocol maintains a set K of known messages (line 2), initially empty. This set is used to detect and eliminate duplicates. In more detail, the algorithm works as follows.

- The application calls procedure MULTICAST(d) to multicast a message with payload d (line 3). This simply generates an unique identifier and forwards it (line 4). The identifier chosen must be unique with high probability, as conflicts will cause deliveries to be omitted. A simple way to implement this is to generate a random bit-string with sufficient length.

- Received messages are processed in a similar manner (line 12), with the difference that it is necessary to check for, and discard, duplicates using the set of known identifiers K (line 13) before proceeding.

Algorithm 3: Point-to-point communication

```
Data:
    C[ ] //cached data
    R //known messages

1:  initially
2:      $\forall i : C[i] \longleftarrow \perp$
3:      $R \longleftarrow \emptyset$

4:  Task 1:
5:      proc L-SEND(i,d,r,p) do
6:          if EAGER?(i,d,r,p) then
7:              Send(MSG(i,d,r,p))
8:          else
9:              $C[i] \longleftarrow (d,r)$
10:             Send(IHAVE(i),p)

11:     upon Receive(IHAVE(i),p) do
12:         if $i \notin R$ then
13:             QUEUE(i,s)

14:     upon Receive(MSG(i,d,r),s) do
15:         if $i \notin R$ then
15:             $R \longleftarrow R \cup \{i\}$
16:             CLEAR(i)
17:         trigger L-RECEIVE(i,d,r,s)

18:     upon Receive (IWANT(i),s) do
19:         $(d,r) \longleftarrow C[i]$
20:         Send(MSG(i,d,r),p)

21: Task 2:
22:     forever do
23:         $(i,s) \longleftarrow$ SCHEDULENEXT()
24:         Send(IWANT(i),s)
```

- The forwarding procedure FORWARD(i,d,r) (line 5) uses the message identifier i, the payload d and the number of times, or rounds, the message has already been relayed r, which is initially 1. It starts by delivering the payload locally using the DELIVER(d) up-call. Then the message identifier is added to the set of previously known messages K (line 7). This avoids multiple deliveries, as described before. Actual forwarding occurs only if the message has been forwarded less than t times (line 8) [12] and consists in querying the peer sampling service to obtain a set of f target nodes and then sending the message, as in lines 9 and 11. Constants t and f are the usual gossip configuration parameters [13].

For simplicity, we do not show how identifiers are removed from set K, preventing it from growing indefinitely. This problem has been studied before, and efficient solutions exist ensuring with high probability that no active messages are prematurely garbage collected [3, 12].

4.2.2 Payload Scheduler Layer

The Lazy Point-to-Point module is the entry point to the Payload Scheduler. It controls the transmission of message payload using a simple negative acknowledgment mechanism. The policy used for each individual message is obtained from the Transmission Strategy module using a pair of primitives:

- EAGER?(i,d,r,p) is used to determine if payload d for message with identification i on round r should be immediately sent to peer p. Note that if the method always returns true the protocol operates as a pure eager push protocol. Otherwise, if the method always returns false, the protocol operates as a pure lazy push protocol.

- $(i,s) = \textsc{ScheduleNext}()$ blocks until it is the time for some message i to be requested from a source s. From the correctness point of view, any scheduling policy is safe, and will maintain all gossip protocols properties, as long as it eventually schedules all lazy requests that have been queued.

The Lazy Point-to-Point module also informs the Transmission Strategy of known sources, for each message and also, when payload has been received using the following primitives:

- $\textsc{Queue}(i,s)$ queues a message identifier i to be requested from source node s. The Transmission Strategy module must keep an internal queue of known sources for each message identifier, and eventually schedule them, unless payload is received first.

- $\textsc{Clear}(i)$ clears all requests on message i. Note also that a queue eventually clears itself as requests on all known sources for a given message identifier i are scheduled.

The Lazy Point-to-Point module is depicted in Algorithm 3. It is based on two separate tasks. Task 1 is responsible for processing transmission requests from the gossip layer and message deliveries from the transport layer. Task 2 runs in background, and performs requests for messages that are known to exist (due to the reception of IHAVE messages), but whose payload has not yet been received. Furthermore, the module maintains the following data structures: a set R of messages whose payload has been received and; a map C, holding the payload and round number for the message (if known).

This module operates as follows: When a message is sent (line 5), the Transmission Strategy module is queried to test if the message should be immediately sent (line 7). If not, an advertisement without the payload is sent instead (line 10). Upon receiving a message advertisement for an unknown message, the Transmission Strategy module is notified (line 13). Upon receiving full message payload, the strategy module is informed (line 16) and the message is also handed over to the gossip layer (line 17). Finally, when a node receives a request (line 18) it looks it up in the cache and transmits the payload (line 20). Note that a retransmission request can only be received as a consequence of a previous advertisement and thus the message is guaranteed to be known and stored locally.

Task 2 executes the following loop. The Transmission Strategy module is invoked to select a message to be requested and a node to request the message from (pair (i,s) in line 24). This invocation blocks until a request is scheduled to be sent by the Transmission Strategy module. A request is then sent (line 25).

For simplicity, we again do not show how cached identifiers and payloads are removed from C and R, preventing them from growing indefinitely. This is however similar to the management of set K, discussed in the previous section, and thus the same techniques may be applied.

Finally, the goal of the Oracle module is to measure relevant performance metrics of the participant peers and to make this information available to the Transmission Strategy in an abstract manner. The exported interface of this module is simply composed by the $\textsc{Metric}(p)$ method, that returns the current metric value for a given peer p. This metric is used by the Transmission Strategy to select whether to immediately schedule an eager transmission or when to request lazy push transmissions from each source. Note that, the Oracle module may be required to exchange messages with its peers (for instance, to measure round trip delays). However, this communication does not affect the dissemination process and moreover, for optimization, can be piggy-backed in regular gossip messages.

In the following section we discuss different implementations of the Transmission Strategy and of Oracle modules, which aim at achieving different emergent dissemination structures.

4.3 Strategies and Oracles

The definition of a Transmission Strategy has two main objectives: i) Avoid as much as possible redundant transmissions of the same payload to any given target node; ii) Decrease the global latency for message dissemination and delivery. These goals are however conflicting. The first goal can be achieved by using a lazy push strategy in all peer exchanges. Since nodes only gossip IHAVE messages, the recipient can request the payload only once. Unfortunately, each lazy push exchange adds one additional round trip to the final delivery latency. On the other hand, a pure eager push strategy minimizes latency at the cost of generating a significant amount of redundancy.

The key to obtaining a better latency/bandwidth trade off in applying a clever and decentralized strategy to select which nodes (and therefore, links) should be preferred. To help the reader in assessing the goal of strategies, we start by describing a couple of strategies that do not take advantage of knowledge about the environment, which can also be used as a baseline for evaluating the benefits of more complex strategies.

4.3.1 Strategies

Flat The flat strategy is defined as $\textsc{Eager?}(i,d,r,p)$ returning true with some probability π or false with probability $1-\pi$. When π equal to 1, this strategy implements a fully eager push gossip. On the other hand, with π equal to 0, it provides a fully pure lazy push gossip. with π values between 0 and 1, it provides different latency/bandwidth trade offs, as a different share of gossip messages are handled in a lazy fashion.

When a lazy strategy is used (and IHAVE messages are sent), we need also to consider how retransmissions are scheduled by receivers within the $\textsc{ScheduleNext}()$ procedure. In the Flat strategy, the first retransmission request is scheduled immediately when queued, which means that an IWANT message is issued immediately upon receiving an IHAVE advertisement. Further requests are done periodically every T, while additional sources are known. Notice that, this is required to mask the failure of the node which sent the first received IHAVE advertisement.

The value of T is an estimate of maximum end-to-end latency. This avoids issuing explicit transmission requests until all eager transmissions have been performed, thus optimizing bandwidth. Note that, unless there is a network omission or an extreme transmission delay, there is usually no need to issue a second request. Thus the value of T has no practical impact in the final average latency, and can be set using only an approximation to the real end-to-end latency.

Although the T value is based only on the (current) end-to-end latency, the reader should notice that, as stated before, this transmission strategy logic has a pure probabilistic nature, as it does not rely in any knowledge about the execution environment (*e.g.* the transmission strategy does not rely in information provided by any Oracle).

Time-To-Live (TTL) This strategy uses eager push until some gossip round u and is thus defined as $\textsc{Eager?}(i,d,r,p)$ returning true if, and only if, $r < u$. When $u > t$, this strategy defaults to a simple lazy push gossip. With $u = 0$, it provides pure lazy push gossip. With a u value between 0 and t, it provides different latency/bandwidth trade offs, as it results, similar to the previous strategy, in a different share of gossip messages being handled in a lazy fashion. $\textsc{ScheduledNext}()$ is defined exactly as in the Flat strategy.

Notice that this strategy is intuitively useful: During the first rounds, the likelihood of a node being targeted by more than one copy of the payload is small and thus there is no point in using lazy push. Moreover, if the supporting unstructured overlay network presents a low clustering coefficient, the usefulness of this strategy is increased, as it will lower the number of redundant peers selected for gossip in the initial dissemination rounds.

Radius This strategy is defined as $\text{EAGER?}(i,d,r,p)$ returning true if, and only if, $\text{METRIC}(p) < \rho$, for a given peer p, has a lower value than a given constant radius ρ. As described before, $\text{METRIC}(p)$ is provided by an Oracle. module for node p. $\text{SCHEDULEDNEXT}()$ delays the first retransmission by some time T_0, which is an estimate of the latency to nodes within radius ρ for the given metric. Further retransmissions are scheduled periodically with period T, as in the Flat strategy. However, if multiple sources are known, the nearest neighbor (according to the Oracle) is selected for requesting the payload.

This follows the intuition of gossiping first with close nodes to minimize hop latency. The expected emergent structure should approximate a mesh structure, with most of payload being carried by links between close neighboring nodes.

Ranked This strategy aims at achieving a hubs-and-spokes structure by selecting a set of *best nodes* to serve as hubs, bearing most of the load. Therefore, at some node q, $\text{EAGER?}(i,d,r,p)$ returns true if, and only if, either q or p are considered to be best nodes, meaning that eager push is used whenever a best node is involved. In such a way, more powerful nodes will have an increased probability to receive payload messages through eager push, as they only rely in this transmission mode to forward the message to all their selected peers. $\text{SCHEDULEDNEXT}()$ is defined exactly as in the Flat strategy.

Although some nodes can be explicitly configured as *best nodes*, for instance, by an Internet Service Provider (ISP) that wants to improve performance to local users, a ranking can also be computed using local Oracles and a gossip based sorting protocol [39]. As shown later, this is greatly eased by the fact that the protocol still works even if ranking is approximate.

4.3.2 Oracles

In this section we provide a short description of two simple Oracles, which are used in the evaluation scheme (we will present experimental work further ahead in Sect. 4.4). Notice that, the implementation of these Oracles is orthogonal to the design of the architecture.

Other, and more complex, strategies may require different types of Oracles. Moreover, an Oracle may compute more complex metrics, for instance, that combine in some clever way more than one performance metric. These performance metrics can be both network efficiency metrics, such as latency, but also application efficiency metrics, such as content similarity in file sharing applications.

Latency Oracle Measures the latency to all neighbor nodes. Real-time monitoring of latency has been addressed a number of times, in fact, every TCP/IP connection implicitly estimates round-trip time to perform congestion control [40]. This estimate can be retrieved by applications and used for other purposes.

Distance Oracle Measures geographical distance to all neighbor nodes. This is useful mostly for demonstration purposes, as it allows to plot network usage graphs such that, the resulting emergent structure, is understandable by the reader. Otherwise, it is not useful in optimizing network parameters.

4.4 Performance

In this section we show performance values for the gossip optimization protocol. Unlike the simulation setting used in Sect. 3.4, here we use an experimental setting based on emulation.

4.4.1 Network Emulation

Experimental evaluation of the protocol for building emergent structures by optimizing gossip, was conducted using the ModelNet large-scale emulation infrastructure [41] with a realistic network model generated by Inet-3.0 [37]. In detail, ModelNet allows a large number of virtual nodes running unmodified programs to be configured in a smaller number of physical nodes in a LAN. Traffic is routed through a set of emulator nodes thus applying delay, bandwidth, and loss as specified in the network model. Inet-3.0 generates realistic Autonomous System level network topologies using a transit-stub model.

ModelNet was deployed in a cluster of 5 workstations connected by switched 100Mbps Ethernet. Each workstation has a 2.4GHz Intel Celeron CPU, 512MB RAM, and a RealTek Ethernet controller. When hosting virtual nodes, they run Linux kernel 2.6.14 and IBM Java2 1.5 runtime. When running as an emulator, FreeBSD 4.11 is used.

The network model is generated using Inet-3.0 default of 3037 network nodes. Link latency is assigned by ModelNet according to pseudo-geographical distance. Client nodes are assigned to distinct stub nodes, also with the default 1 ms client-stub latency. A typical network graph has the following properties: average hop distance between client nodes is 5.54, with 74.28% of nodes within 5 and 6 hops; average end-to-end latency of 49.83 ms, with 50% of nodes within 39 ms and 60 ms.

4.4.2 Implementation and Configuration

The protocol was implemented by modifying an open source and lightweight implementation of the NeEM protocol [7] that uses the java.nio API for scalability and performance [42]. Briefly, NeEM uses TCP/IP connections between nodes to avoid network congestion. When a connection blocks, messages are buffered in user space, which then uses a custom purging strategy to improve reliability. The result is a virtual connection-less layer that provides improved guarantees for gossiping.

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

The protocol was configured with a gossip fanout of 11 and an overlay degree of 15. With 200 nodes, these correspond to a probability 0.995 of atomic delivery with 1% messages dropped, and a probability of 0.999 of keeping the overlay connected when 15% of nodes fail [13]. A retransmission period of 400 ms was used, which is the minimal value that results in approximately 1 payload received by each destination when using a fully lazy push strategy.

4.4.3 Traffic and Measurements

During each experiment, 400 messages are multicast, each carrying 256 bytes of application level payload. To each of them, a NeEM header of 24 bytes is added, besides TCP/IP overhead. Messages are multicast by virtual nodes in a round-robin fashion, with an uniform random interval with 500 ms average. All messages multicast and delivered are logged for later processing. Namely, end-to-end latency can be measured when source and destination share the same physical node, and thus a common clock[5]. Payload transmissions on each link are also recorded separately.

Results presented in the following section used 25 virtual nodes on each workstation, thus obtaining 100 virtual nodes. The reason for this limitation is that an epidemic multicast protocol produces a bursty load, in particular when using eager push gossip: Network and CPU load occurs only instants after each message is multicast. Using a larger number of virtual nodes was observed to induce additional latency which would falsify results. The configurations that result in lower bandwidth consumption, which are the key results and goals of this approach, were also simulated with 200 virtual nodes.

[5]We do not present such results in this chapter however, the interested reader can refer to [43] for more information.

4.4.4 Statistics

Consider the following statistics of each experiment with 100 virtual nodes using an eager push strategy: 40000 messages delivered, 440000 individual packets transmitted. This amounts to 2200 Kpkts/s and thus approximately 6 MBps. As the overlay evolves, TCP/IP connections are created and tear down. During each run, approximately 550 simultaneous and 15000 different connections are used. The experiments presented in the next section, when automated, take almost 7 hours to run. The total amount of resulting logs is 1Gb, that has then to be processed and rendered in plots.

Special care was taken to consider variance of each measure taken. When in the following section we affirm that a performance difference is relevant, this was confirmed by checking that confidence intervals with 95% certainty do not intersect. In fact, the large number of samples used are sufficient to make such intervals very narrow.

4.4.5 Experimental Results

To strengthen the intuition behind the approach, we depict the impact of the described strategies with the Oracle that considers pseudo-geographical position of nodes, as generated by Inet-3.0. Although this cannot be used to assess the performance of the protocol, as geometrical distance does not directly map to end-to-end distance, it allows the resulting emergent structure to be plotted and understood.

Figures 8-10 shows the result of running 100 node configurations with different strategies and then selecting the top 5% connections with highest throughput. The size of each red circle is proportional to the amount of payload transmitted by the node. Note that each connection is used for a brief period of time, as the membership management algorithm periodically shuffles peers with neighbors. This means that connections shown may have not existed simultaneously.

As a baseline, Fig. 8 shows an eager push configuration, where no structure is apparent. A confirmation of this is given by the fact that the top 5% connections account for only 7% of all traffic, i.e. payload transmissions are evenly spread across all connections. In sharp contrast, Fig. 9 shows an obvious emergent mesh structure as a result of the Radius strategy, in which the 5% connections account for 37% of all payload transmissions. Finally, Fig. 10 shows a sub-set of nodes emerging as super-nodes, accounting for a large share of links and also transmitting a higher number of payloads. Again, the emergent structure is confirmed by the fact that 5% of the connections account for 30% of total payloads transmitted.

5 Discussion and Future Directions

In this chapter we discussed how to add some structure to unstructured overlay networks. We have identified two distinct techniques that can be used for this end, namely: a technique based on optimizing the overlay and; a technique based on optimizing gossip itself. These techniques can be applied together as each of them operates at distinct levels in the typical architecture of gossip protocols. We have described in some detail one protocol for each of these techniques. The first protocol illustrates how to control the communication patterns to adapt the unstructured topology of the overlay, into a more structured and efficient infrastructure for peer-to-peer applications and protocols. The second protocol illustrates how the control of communication modes among peers can be used to create emergent communication structures in unstructured overlay networks. Moreover, we showed that by exploiting these emergent patterns, one can lower the typical communication overhead of gossip protocols, by applying according to different strategies, a combination of eager push and lazy push when gossiping with neighbors.

As future directions for research, we identify the following open issues that are raised when adding structure to unstructured overlay networks:

Unstructured overlay networks are a promising approach to develop high scale and highly resilient monitoring systems. Notice that one can map on top of the overlay links, monitoring relations. Therefore, one ensures that

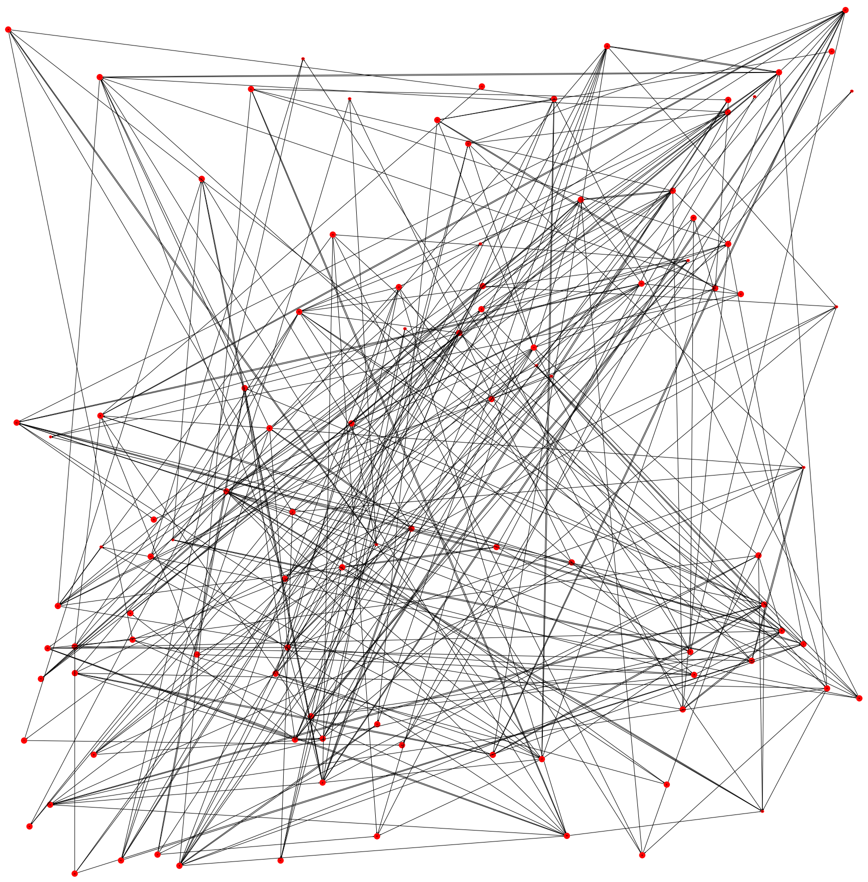


Figure 8. Flat (7% of traffic)

all components in the system (which are mapped on nodes) are constantly monitored by a given number of other components, and that every component shares the monitoring load. Moreover, adapting the overlay structure can be useful to promote efficient methods to disseminate monitoring information.

Location algorithms for unstructured overlay networks present a high overhead, specially compared with location algorithms for structured approaches such as DHTs. Usually, these algorithms degenerate in some kind of limited flooding in the overlay which presents a high communication overhead. On the other hand, unlike structured approaches, unstructured overlay networks can support more complex search semantics (as they do not operate based on hash keys) and can better tolerate churn scenarios. One can try to bias the topology of unstructured overlay networks to support more efficient, reliable and rich semantics resource location algorithms and protocols.

Finally, given that the methodologies illustrated in this chapter operate at distinct architectural levels, a promising direction in the peer-to-peer research field is to find efficient ways of combine both methodologies.

Acknowledgments

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

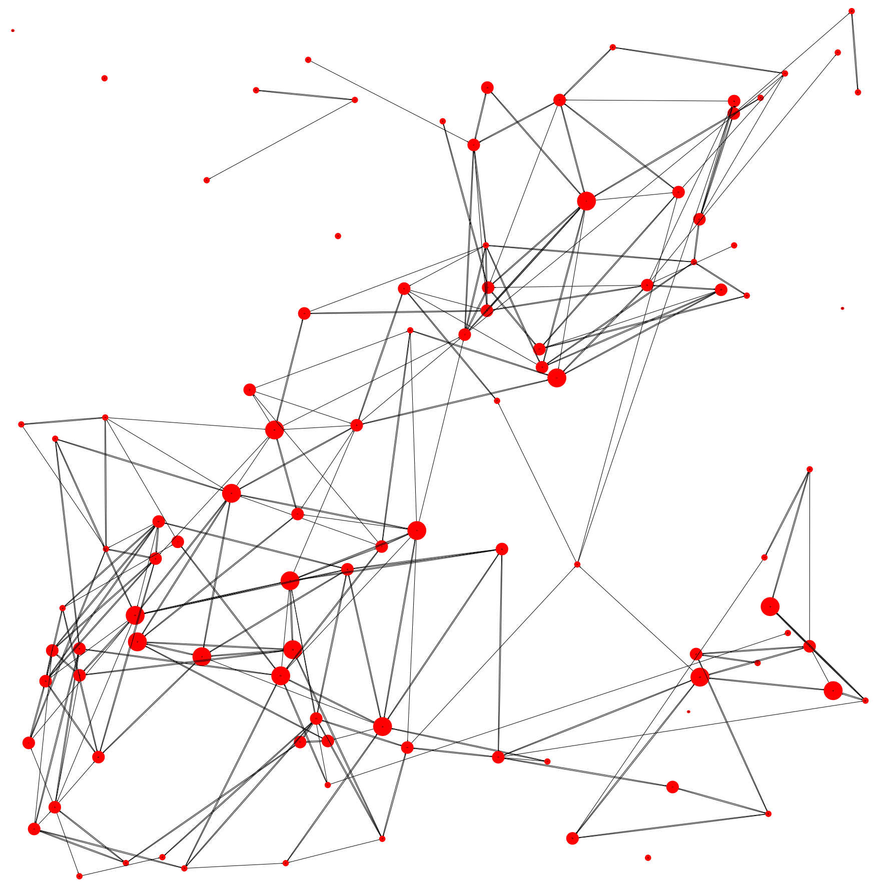


Figure 9. Radius (37% of traffic)

References

[1] Birman, K., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., Minsky, Y.: Bimodal multicast. ACM Transactions on Computer Systems **17**(2) (1999)

[2] Kermarrec, A.M., Massouli, L., Ganesh, A.: Probabilistic reliable dissemination in large-scale systems. IEEE Trans. Parallel Distrib. Syst. **14**(3) (2003) 248–258

[3] Eugster, P.T., Guerraoui, R., Handurukande, S.B., Kouznetsov, P., Kermarrec, A.M.: Lightweight probabilistic broadcast. ACM Trans. Comput. Syst. **21**(4) (2003) 341–374

[4] Hayden, M., Birman, K.: Probabilistic broadcast. Technical report, Ithaca, NY, USA (1996)

[5] Leito, J., Pereira, J., Rodrigues, L.: HyParView: A membership protocol for reliable gossip-based broadcast. In: DSN '07: Proc. of the 37th Annual IEEE/IFIP Intl. Conf. on Dependable Systems and Networks, Edinburgh, UK, IEEE Computer Society (2007) 419–429

[6] Lin, M.J., Marzullo, K.: Directional gossip: Gossip in a wide area network. In: European Dependable Computing Conference. (1999) 364–379

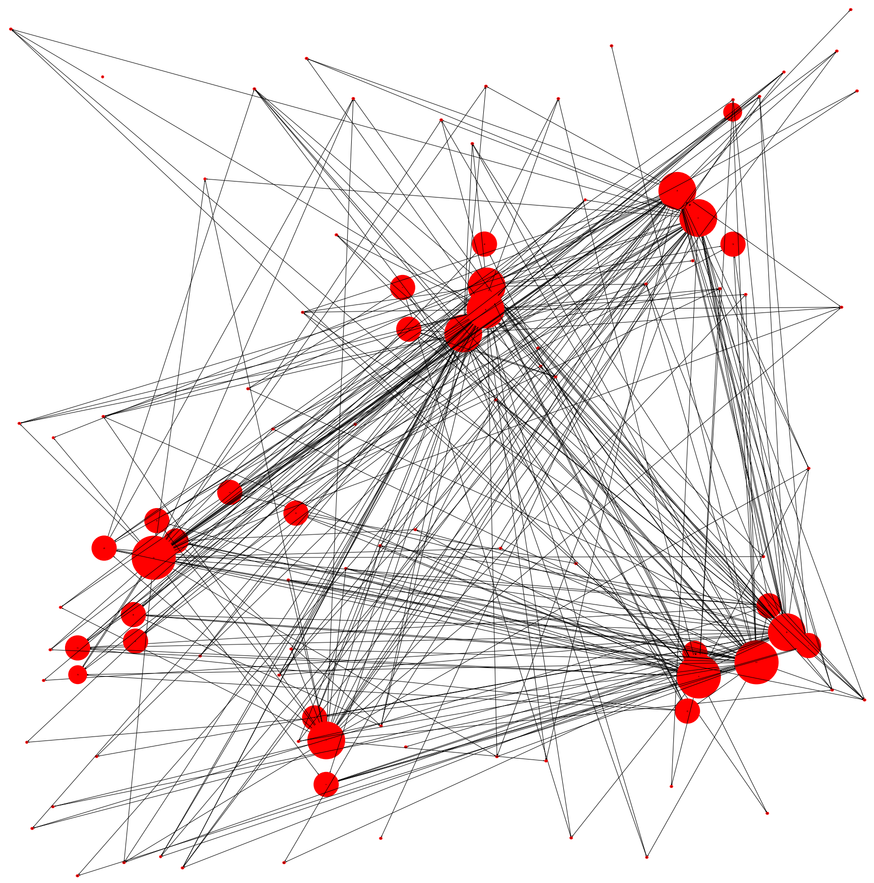


Figure 10. Ranked (30% of traffic)

[7] Pereira, J., Rodrigues, L., Monteiro, M.J., Oliveira, R., Kermarrec, A.M.: NeEM: Network-friendly epidemic multicast. In: Proceedings of the 22th IEEE Symposium on Reliable Distributed Systems (SRDS'03), Florence,Italy (2003) 15–24

[8] Jelasity, M., Montresor, A.: Epidemic-style proactive aggregation in large overlay networks. In: Proceedings of The 24th International Conference on Distributed Computing Systems (ICDCS 2004), Tokyo, Japan, IEEE Computer Society (2004) 102–109

[9] Eugster, P.T., Guerraoui, R.: Probabilistic multicast. In: DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks, Washington, DC, USA, IEEE Computer Society (2002) 313–324

[10] van Renesse, R., Minsky, Y., Hayden, M.: A gossip-style failure detection service. Technical report, Ithaca, NY, USA (1998)

[11] Li, H., Clement, A., Wong, E., Napper, J., Roy, I., Alvisi, L., Dahlin, M.: BAR gossip. In: Proceedings of the 2006 USENIX Operating Systems Design and Implementation (OSDI). (2006)

[12] Koldehofe, B.: Buffer management in probabilistic peer-to-peer communication protocols. In: Proceedings of the 22th IEEE Symposium on Reliable Distributed Systems (SRDS'03), Florence,Italy (2003) 76–87

[13] Eugster, P., Guerraoui, R., Kermarrec, A.M., Massouli, L.: From Epidemics to Distributed Computing. IEEE Computer **37**(5) (2004) 60–67

[14] 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 (1987) 1–12

[15] Jelasity, M., Guerraoui, R., Kermarrec, A.M., van Steen, M.: The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In: Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware, New York, NY, USA, Springer-Verlag New York, Inc. (2004) 79–98

[16] Ganesh, A., Kermarrec, A.M., Massouli, L.: SCAMP: Peer-to-peer lightweight membership service for large-scale group communication. In: Networked Group Communication. (2001) 44–55

[17] Voulgaris, S., Gavidia, D., Steen, M.: Cyclon: Inexpensive membership management for unstructured p2p overlays. Journal of Network and Systems Management **13**(2) (2005) 197–217

[18] Liu, Y., Xiao, L., Ni, L., Liu, Y.: Building efficient overlays. J. Grid Comput. **2**(2) (2004) 183–192

[19] Pereira, J., Oliveira, R., Rodrigues, L.: Efficient epidemic multicast in heterogeneous networks. In: Proceedings of the International Workshop on Reliability in Decentralized Distributed Systems, part of the OTM Federated Conferences and Workshops, Montpellier, France (2006)

[20] Floyd, S., Jacobson, V., Liu, C.G., McCanne, S., Zhang, L.: A reliable multicast framework for light-weight sessions and application level framing. IEEE/ACM Trans. Netw. **5**(6) (1997) 784–803

[21] Rowstron, A.I.T., Kermarrec, A.M., Castro, M., Druschel, P.: Scribe: The design of a large-scale event notification infrastructure. In: NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication, London, UK, Springer-Verlag (2001) 30–43

[22] Ratnasamy, S., Handley, M., Karp, R., Shenker, S.: Application-level multicast using content-addressable networks. In: NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication, London, UK, Springer-Verlag (2001) 14–29

[23] Zhuang, S., Zhao, B., Joseph, A., Katz, R., Kubiatowicz, J.: Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In: Proceedings of NOSSDAV. (2001)

[24] hua Chu, Y., Rao, S.G., Zhang, H.: A case for end system multicast (keynote address). In: SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, New York, NY, USA, ACM (2000) 1–12

[25] Leito, J., Pereira, J., Rodrigues, L.: Epidemic broadcast trees. In: Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS'2007), Beijing, China (2007) 301 – 310

[26] Massoulié, L., Kermarrec, A.M., Ganesh, A.J.: Network awareness and failure resilience in self-organising overlays networks. In: Synmposium on Reliable Distributed Systems (SRDS), Florence, Italy (2003)

[27] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equations of state calculations by fast computing machine. J. Chem. Phys. **21** (1953) 1087–1091

[28] Melamed, R., Keidar, I.: Araneola: A scalable reliable multicast system for dynamic environments. In: NCA '04: Proceedings of the Network Computing and Applications, Third IEEE International Symposium, Washington, DC, USA, IEEE Computer Society (2004) 5–14

[29] Tang, C., Ward, C.: GoCast: Gossip-enhanced overlay multicast for fast and dependable group communication. In: DSN '05: Proc. of the 2005 Intl. Conf. on Dependable Systems and Networks (DSN'05), Washington, DC, USA, IEEE Computer Society (2005) 140–149

[30] Jelasity, M., Babaoglu, O.: T-man: Gossip-based overlay topology management. In: The Fourth International Workshop on Engineering Self-Organizing Applications (ESOA'06), Hakodate, Japan (2006)

[31] Leito, J.: Gossip-based broadcast protocols. Master's thesis, University of Lisbon (2007)

[32] Karwaczynski, P.: Fabric: Synergistic proximity neighbour selection method. In: P2P '07: Proceedings of the Seventh IEEE International Conference on Peer-to-Peer Computing (P2P 2007), Washington, DC, USA, IEEE Computer Society (2007) 229–230

[33] Karwaczyński, P., Konieczny, D., Moçnik, J., Novak, M.: Dual proximity neighbour selection method for peer-to-peer-based discovery service. In: SAC '07: Proceedings of the 2007 ACM symposium on Applied computing, New York, NY, USA, ACM (2007) 590–591

[34] Leito, J., Pereira, J., Rodrigues, L.: Topology aware gossip overlays. Technical Report 36, INESC-ID (2008)

[35] Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: Planetlab: an overlay testbed for broad-coverage services. SIGCOMM Comput. Commun. Rev. **33**(3) (2003) 3–12

[36] Jelasity, M., Montresor, A., Jesi, G.P., Voulgaris, S.: (The Peersim simulator) `http://peersim.sf.net`.

[37] Winick, J., Jamin, S.: Inet-3.0: Internet topology generator. Technical Report UM-CSE-TR-456-02, EECS, University of Michigan (2002)

[38] Jelasity, M., Babaoglu, O.: T-man: Fast gossip-based construction of large-scale overlay topologies. Technical report, University of Bologna (2004)

[39] Jelasity, M.: A case study on gossip beyond gossip: Sorting. Ws. on Gossip Based Computer Networking, Lorent Center, Leiden (2006)

[40] Floyd, S., Fall, K.: Promoting the use of end-to-end congestion control in the Internet. IEEE/ACM Trans. Networking **7**(4) (1999)

[41] Vahdat, A., Yocum, K., Walsh, K., Mahadevan, P., Kostic, D., Chase, J., Becker, D.: Scalability and accuracy in a large-scale network emulator. SIGOPS Oper. Syst. Rev. **36**(SI) (2002) 271–284

[42] Santos, P., Pereira, J.: NeEM version 0.5. http://neem.sf.net (2006)

[43] Carvalho, N., Pereira, J., Oliveira, R., Rodrigues, L.: Emergent structure in unstructured epidemic multicast. In: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Edinburgh, UK (2007) (to appear)