X-BOT: A Protocol for Resilient Optimization of Unstructured Overlays*

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

João Pedro Marques
INESC-ID / IST
jmarques@gsd.inesc-id.pt

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

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

Abstract

Gossip, or epidemic, protocols have emerged as a highly scalable and resilient approach to implement several application level services such as reliable multicast, data aggregation, publish-subscribe, among others. All these protocols organize nodes in an unstructured random overlay network. In many cases, it is interesting to bias the random overlay in order to optimize some efficiency criteria, for instance, to reduce the stretch of the overlay routing. In this paper we propose X-BOT, a new protocol that allows to bias the topology of an unstructured gossip overlay network. X-BOT is completely decentralized and, unlike previous approaches, preserves several key properties of the original (non-biased) overlay (most notably, the node degree and consequently, the overlay connectivity). Experimental results show that X-BOT can generate more efficient overlays than previous approaches.

1. Introduction

Gossip, or epidemic, protocols have emerged as a highly scalable and resilient approach to implement several application level services such as reliable multicast [2, 14, 16, 17], data aggregation [11], publish-subscribe [5], among others. Typically, a gossip-based broadcast protocol operates as follows: in order 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.

This approach 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 system only grows logarithmically with the number of participants in order to ensure reliable broadcast with a high probability [2, 6], and finally, *iii)* it's inherent redundancy makes gossip protocols highly resilient to node and link failures.

Some gossip protocols have been designed to operate with full membership information [4, 2], by maintaining locally at each node a list with every other node identifier[1] participating in the system. However, such approach is not scalable, not only due to the large size of the membership list but mainly due to the cost of maintaining such large amount of information up-to-date in dynamic environments. For scalability, nodes may rely on a *peer sampling service* [10], which is 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 with whom they exchange messages.

*This work was partially supported by project "Redico" (PTDC/EIA/71752/2006). Selected portions of this Technical Report were published in the Proceedings of the 28th IEEE International Symposium on Reliable Distributed Systems.

[1] Typically, an identifier is a tuple $(ip, port)$.

Partial views establish *neighboring* associations among nodes that define an overlay network which can be used for data gossip. Typically, a peer sampling service aims at maintaining a random partial view of the system [7, 21, 16] which should ensure that a selection of peers from local partial views is equivalent to a random selection of peers across the full membership. Therefore, the resulting overlay has a random unstructured topology. Although this randomness is desirable, it prevents the peer sampling service to take into consideration any efficiency criteria, which usually leads to scenarios where many of the overlay links are suboptimal for instance, with regard to a given efficiency criteria, such as network bandwidth or latency. Unfortunately, the inefficiency of the overlay has a direct negative impact in the performance of applications that operate on top of the overlay (such as application level reliable broadcast services). Overlay efficiency has been recognized as a key research topic for gossip-based protocols [1].

In this paper, we present X-BOT, a new protocol to **B**ias the **O**verlay **T**opology according to some target efficiency criteria **X**, for instance, to better match the topology of the underlying network. X-BOT is completely decentralized and, unlike prior works, it presents the following set of characteristics: *i)* it operates only with local information and it does not require nodes to have a priori knowledge about their target location on the final topology; *ii)* it employs a new coordinated 4-node optimization technique that allows to achieve better overlay configurations; *iii)* the protocol strives to preserve the degree of the nodes that participate in an optimization step, which is fundamental to preserve the connectivity of the overlay; *iv)* every modification performed to the overlay increases its efficiency; this is feasible due to the dynamic nature of our model, which ensures that the overlay does not stabilize in some local minima; *v)* the optimization performed by X-BOT preserves several key properties of the overlay such as a low clustering coefficient and low overlay diameter; *vi)* our scheme is highly flexible, as we rely on a companion *oracle* to estimate the link cost and, therefore, our algorithm can bias the network according to different cost metrics.

The rest of this paper is organized as follows. In Section 2 we introduce the most relevant aspects of unstructured overlay networks which are at the core of our proposal, and provide a brief overview of some building blocks that we use as the basis for our protocol. Section 3 addresses the related work. Section 4 describes the X-BOT protocol, explaining the rationale for our architecture and the proposed algorithm. An experimental evaluation of X-BOT is provided in Section 5. Finally, Section 6 concludes the paper, presenting some directions for future work.

2. Unstructured Overlay Networks

In this section we enumerate several aspects of unstructured overlay networks and discuss their importance for reliable broadcast. For self containment, we also present an overview of the peer sampling service that we use as a building block for the development of X-BOT.

2.1. Structural Requirements

In order to support fast message dissemination and high level of resilience to node failures, the overlay networks defined by the partial views must own several properties[2]. Some of the most important properties are listed below. The interested reader can find a more detailed discussion of these and other properties of random overlays in [16].

Connectivity The overlay is connected if there is at least one path that allows every node to reach every other node in the overlay.

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. When partial views are asymmetric, the degree has two distinct components: in-degree and out-degree. The in-degree of a node a is the number of nodes in the system that contain a's identifier

[2]The reader should note that some of these properties are intrinsically related with graph properties. An overlay network can be seen as a graph, where nodes are represented by vertices, and links, or neighboring relations, are represented by edges. Depending on the nature of these relations, graphs can be directed or undirected.

in their partial views; it is a measure of the reachability of a. The out-degree of a node a is the number of distinct node identifiers present in a's partial view and can be seen as 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 in-degree will become more easily disconnected from the overlay as the number of faults increases, and the failure of nodes with high out-degree may have an undesired impact in the overall connectivity of the overlay.

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. This metric indicates a density of neighboring 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 a graph should be as small as possible, and failure to meet this requirements also has several 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 increases the overall latency of broadcast processes, and finally *iii)* it decreases the fault resilience of the overlay, as areas of the overlay which exhibit a high values of clustering can more easily became disconnected.

2.2. Performance Metrics

Several performance metrics can be used to measure the performance of a gossip-based broadcast protocol operating on top of a random overlay network. In this paper we are mainly concerned with the dependability of gossip-based broadcast protocols and associated applications. Therefore, we focus on biasing the overlay topology to minimize the message dissemination cost while preserving reliability.

Overlay Cost We assume that each link of the overlay may be tagged with a *cost*. The overlay cost is the sum of cost for all links that form the overlay. X-BOT is independent of the cost function and only requires costs to be comparable and totally ordered (in fact, the algorithm does not use the absolute values of link cost, only their relative values). Costs may be associated to a concrete (underlay) network metric such as link latency. However, the link cost may also capture higher level utility metrics; for instance, in a file sharing peer-to-peer system, it could measure the semantic similarity among data stored at the link edges. More generally, a link cost is inversely proportional to the "utility" of a link. The cost value is provided by an *oracle* which is locally available at each edge node.

The goal of our protocol is to reduce, as much as possible, the overlay cost without hampering relevant properties of that same overlay. For that purpose, when optimizing the overlay topology, each node tries to select better neighbors. This notion of "better" is obtained by comparing the cost value of each possible neighbor, or in other words, the cost of maintaining, or using, an overlay link for each peer.

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. Reliability of 100% is impossible to achieve if the support overlay network becomes disconnected.

2.3. Building Blocks

As noted before, most unstructured overlays use some form of peer sampling service. As a building block to construct X-BOT, we rely on the *Hy*brid *Par*tial *View* peer sampling service [16] (HyParView). Unlike previous membership protocols, HyParView relies on two distinct partial views which are maintained using different strategies and for different purposes (in fact, the protocol is said to be Hybrid because it combines two different strategies).

A small symmetric *active view* with (fanout+1) size, which is used mainly to disseminate broadcast messages, is maintained using a reactive strategy which means that these partial views only change in response to some

external event that affects the overlay (e.g. a node joining or leaving). A TCP connection is maintained to each neighbor in these partial views, which allows the selection of smaller fanout by assuming that the links do not omit messages. Moreover, TCP is used as an unreliable failure detector, which facilitates the implementation of the reactive maintenance strategy. The symmetry of these views facilitates to maintain the connectivity of the overlay, by providing each node with some degree of control over its in-degree.

Each node also maintains a larger *passive view* usually k times larger than the active view, whereas k is a constant that is related with the desired fault tolerance level of the protocol. The passive view is maintained by a cyclic strategy (periodically, each node performs a shuffle operation with one random node in the overlay that results in a update of both nodes passive views). This partial view is used for fault tolerance, as it works like a backup list of nodes that are used to ensure a constant out-degree for nodes.

3. Related Work

Narada [3] includes self-organizing protocols to construct and maintain overlay networks. Their approach is based in a utility function that is applied periodically to (some) peers; the output of the utility function is used to take local decisions concerning the addition and removal of links to the overlay. However, because Narada is targeted at small and medium scale systems, it operates using full membership information and, therefore, scales poorly.

A work by Gupta et. al. [8] also aims at increasing gossip efficiency by eliminating overlay links that transverse a given physical link multiple times. However, it relies on the fact that peers can be organized in a hierarchical manner, for instance, by grouping peers by network domains (*e.g.* Local Area Networks, Subnets or even Autonomous Systems). Our approach does not require knowledge concerning the physical location of nodes nor the maintenance of any hierarchy among peers, being therefore more generic.

The Localiser algorithm [18] is an algorithm that aims both at optimizing unstructured overlays according to a proximity criterion and to promote the balancing of node degrees. Localiser is based on a Metropolis scheme where nodes, iteratively, strive to minimize an energy function, by swapping connections among peers. Although this algorithm strives to balance the degree of nodes in the system, unlike our approach, it does not ensure a constant degree in nodes that participate in the optimization. Furthermore, the Localiser algorithm incurs in the risk of falling into local minima of the energy function, due to the fact the the pool of peers available to generate overlay optimization is limited. Therefore, it is required to sometimes increase the energy function which might compromise the stability of the overlay. Moreover, the localiser does not attempt to preserve low clustering and small diameter.

GoCast [20] and Araneola [19] also include mechanisms to bias the topology of overlay networks maintaining symmetric partial views supported by TCP connections. GoCast builds an overlay which is optimized to maintain both random (distant) neighbors and close neighbors while balancing node degree in such a way that degree of nodes converge to a given pre-established value D and varies only between $D-2$ and $D+2$. Araneola is similar to GoCast in the sense that it also builds an unstructured overlay network that presents several properties of k-regular graphs. Like in GoCast, Araneola controls its topology to take into consideration some network metric, ensuring that better links are kept with a larger probability. However, these protocols rely on mechanisms to bias the overlay that are more complex and where each node makes independent decisions. In sharp contrast X-BOT uses a coordinated 4-node optimization technique that is simpler and allows to improve the overlay topology while maintaining a better node degree distribution. In Section 5 we compare these protocols with our own.

T-Man [9] is a generic topology management scheme for overlay networks that is able to evolve a given overlay topology to a desired target topology (such as a torus, ring or some user defined topology). This is achieved by having neighbors periodically exchanging their partial views. Both nodes update their partial views by merging these views and selecting the best c nodes, where c is the size of a partial view. The selection is based on a single ranking function which captures the designed topology, in the sense that it enables each node in the system to

extract clues on its optimal position and desired neighbors in the overlay network. Unlike X-BOT, T-Man does not aim at protecting relevant properties of the original overlay, nor it ensures the stability of in-degree of nodes during the optimization of the overlay, as we will show in Section 5.

The reader should be aware that a more detailed description of existing overlay optimization algorithms can be found in [15], a survey of several optimization approaches (including X-BOT). The remainder of the paper provides a detailed description of an improved X-BOT and an extensive performance evaluation against the most relevant competing algorithms. These are original contributions that have not been published elsewhere.

4. X-BOT

4.1. Oracles

We assume that all nodes have access to a local *oracle*. Oracles are components that export a `getLinkCost(Peer` p`)` method, 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 the paper we use interchangeably link cost or node cost when referring to the output of the oracle). The implementation of the oracle is outside the scope of this paper. However, for completeness, we provide a brief description of two simple oracles: one based on a network metric and the second based on a more high level metric.

Latency Oracle One of the most simple oracles that can be devised is a latency oracle. This oracle operates by making measures of round trip times (RTT) between peers, using some specific probe messages exchanged between oracles. The oracle must be aware of the peers which are known at the local host, and it measures the RTT for each known node storing the last reading (or some weighted average), which can be directly used as the link cost value. Moreover, if TCP connections are maintained among peers, this oracle can use the estimated RTT calculated by TCP as the link cost.

Internet Service Provider Oracle In a setting where sending messages via different ISPs has an increased monetary cost, it might be useful to keep as many neighbors as possible from the local ISP. An Oracle to this end could be built by maintaining information concerning the local ISP and a table of costs for each known ISP (this table could simply store a low value for the local ISP and a high value for all others). When the oracle becomes aware of a new peer, it simply exchanges a message with the remote oracle to identify local ISP's, and assert the cost for the link using its local cost table.

Our protocol could also leverage in previous work [13] which addresses the use of inexpensive oracles for calculating neighbor proximity based on IP-based clustering (for instance, using a match of common IP prefixes to calculate a measure of proximity between two peers).

4.2. Rationale

The rationale of our approach is as follows. Like in HyParView, we maintain a small active view and a larger passive view. However, unlike HyParView, where we strive to ensure the stability of the overlay denoted by the active views, X-BOT relaxes stability in order to continuously attempt to improve the overlay according to some efficiency metric embedded in the companion oracle. This allows the topology of the unstructured overlay to self adapt in order to better match the requirements of the application, or services, executed on top of it. Periodically, each node starts *optimization rounds* in which it attempts to switch one member of its active view with one (better) neighbor of its passive view. While executing the optimization algorithm, a node uses its local oracle to obtain an estimate of the link cost to some randomly selected peers of its passive view. The number of nodes scanned in each set of optimization rounds is a protocol parameter called *Passive Scan Length* and simply denoted π. This parameter limits the maximum number of optimization rounds started by each node each time it runs the optimization procedure.

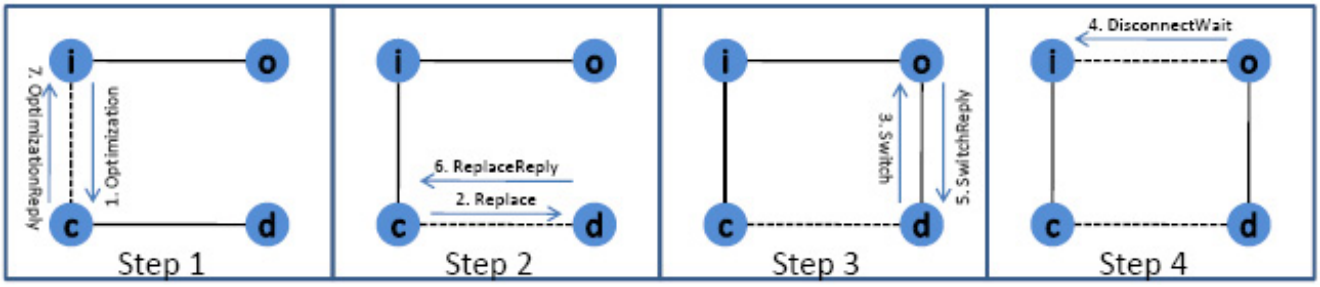


Figure 1. X-BOT steps

The passive view is not biased. However, the reader should notice that the passive view should be continuously updated during the system operation, so that it reflects the changes in the global membership (*e.g.* nodes that leave the system, are eventually purged from all passive views, and nodes that join the system eventually appear in some of the passive views). Therefore, passive views can be used as a continuous source of potential nodes that can be upgraded to the active view, in order to bias the overlay to a better topology. This intuitively, also prevents our algorithm from falling into local minima configuration.

X-BOT strives to preserve the connectivity of the overlay. This has two implications in our scheme: *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 an optimization procedure. Moreover, we ensure that optimizations preserve the symmetry of the active view, which is essential to ensure a good distribution of in-degree, which in turn has a significant impact on the connectivity of the overlay. Typically, each optimization round involves 4 nodes in the system as we will detail later in the text.

4.2.1 Unbiased Neighbors

By blindly imposing a bias on the topology of the overlay, one may easily break some of the key desirable properties of a random overlay, such as the low clustering coefficient, low average path length, or connectivity [20]. The negative effect of such bias can be even more notorious in an architecture such as ours, that relies on small active views. To avoid this flaw, we do not apply the bias to all members of the active view. Instead, each node should maintain both "high-cost" (unbiased) and "low-cost" (biased) neighbors. The number of "high-cost" 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 required to have global knowledge of the system, not only regarding membership information but also regarding global metrics, such as the overlay cost. To circumvent this problem, we maintain the active views sorted by link cost (the first element of each active view is the neighbor with the largest link cost). A node never attempts to apply the bias to the first μ members of its active view.

4.3. Algorithm

The X-BOT algorithm is depicted in *Algorithm* 1 and also in Figure 1. The reader should notice that the algorithm presented 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. A typical 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. We use the following definitions to identify each of the participating nodes. **Node** i **(*initiator*):** is the

Algorithm 1: X-BOT: Improving Procedure

```
1:  every Δ T do
2:      if isFull(activeView) then
3:          candidates⟵ randomSample(passiveView, PSL)
4:          for i := UN ; i < sizeof(activeView) ; i :=i + 1
5:              o ⟵ activeView[i]
6:              while candidates ≠ {} do
7:                  c ⟵ removeFirst(candidates)
8:                  if isBetter(o,c) then
9:                      Send(OPTIMIZATION, c, o, myself)
10:                     break

11: upon Receive(OPTIMIZATIONREPLY,answer,o,d,c) do
12:     if answer then
13:         if o ∈ activeView do
14:             if d ≠null then
15:                 Send(DISCONNECTWAIT, o, myself)
16:             else
17:                 Send(DISCONNECT, o, myself)
18:             activeView ⟵ activeView \{o}
19:         passiveView ⟵ passiveView \{c}
20:         activeView ⟵ activeView ∪{c}

21: upon Receive(OPTIMIZATION, o, peer) do
22:     if !isFull(activeView) then
23:         activeView ⟵ activeView ∪ {peer}
24:         Send(OPTIMIZATIONREPLY, true, o, null, myself)
25:     else
26:         d ⟵ activeView[UNOPT]
27:         Send(REPLACE, d, o, peer, myself)

28: upon Receive(REPLACEREPLY,answer,i,o,d) do
29:     if answer then
30:         activeView ⟵ activeView \{d}
31:         activeView ⟵ activeView ∪{i}
32:     Send(OPTIMIZATIONREPLY,answer,o,d,myself)

33: upon Receive(REPLACE, o, i, c) do
34:     if ! isBetter(peer,o) then
35:         Send(REPLACEREPLY, c, false, i, o, myself)
36:     else
37:         Send(SWITCH, o, i, c, myself)

38: upon Receive(SWITCHREPLY,answer,i,c,o) do
39:     if answer then
40:         activeView ⟵ activeView \{c}
41:         activeView ⟵ activeView ∪{o}
42:     Send(REPLACEREPLY,answer,i,myself)

43: upon Receive(SWITCH,i,c,d) do
44:     if i ∈ activeView or received(DISCONNECTWAIT from i) then
45:         Send(DISCONNECTWAIT,i,myself)
46:         activeView ⟵ãctiveView \{i}
47:         activeView ⟵ãctiveView ∪{d}
48:     Send(SWITCHREPLY,answer,di,c,myself)

49: isBetter(old,new)
50:     return Oracle.getCost(old) > Oracle.getCost(new)
```

node that starts the optimization round. **Node** o **(*old*):** is a node from i's active view which is replaced during the optimization process. **Node** c **(*candidate*):** is a node from i's passive view which is upgraded to the active view. **Node** d **(*disconnected*):** is the node to be removed from the candidate's active view in order to accept i.

Step 1 Step 1 is executed at node i (Algorithm 1, lines $1-10$) and its purpose is to contact one, or more, potential candidates to participate in a set of optimization rounds[3].

This step starts with the random selection of, at most, π nodes from the i's passive view. This random sample is a set of candidates for executing the optimization round. To check if a target node is a suitable candidate, i iterates over its active view, consulting the oracle to compare the cost of its neighbors with the cost of the target (Algorithm 1, lines $49-50$). When a suitable candidate c is found, which presents a possibility for improving a given neighbor o, i sends to c an OPTIMIZATION message, stating its interest in exchanging o for c in its active view. The reception of that message will trigger the execution of Step 2 in node c.

This step ends with the reception of an OPTIMIZATIONREPLY message from node c (Algorithm 1, lines $11-20$), or with the suspicion of failure of node c. If node c accepts the exchange, then i will add c to the active view. If o is still in its active view[4], i will send a DISCONNECTWAIT or DISCONNECT message to o. The difference between these messages is simple: DISCONNECT, only removes the sender from the active view (as described in [16]) while DISCONNECTWAIT also notifies the node that it should maintain (until an internal timeout expires) that free slot in the active view, which will be used in step 4. Node i chooses which message to send, based on information received from c, specifically, if c had to remove some node from its active view in order to insert i in its active view.

[3]More than an optimization round might be triggered in the context of this step.

[4]Note that o might have already disconnected from i as a result of the execution of step 4.

Step 2 Step 2 is initiated at node c with the reception of an OPTIMIZATION message from node i (Algorithm 1, lines $21 - 27$) and ends when c replies to i with a OPTIMIZATIONREPLY message.

If c does not have a full active view it immediately replies to i by sending an OPTIMIZATIONREPLY message accepting the exchange, and notifying i that no other node was involved in the optimization. In this case i will disconnect itself from o and insert c in its active view. Note that in this particular scenario, our algorithm does not preserve the degree of node o, although it preserves the number of links in the overlay. However, this is a uncommon scenario given that, according to our experiments, usually more than 97% of nodes in the system have full active views. On the other hand, if c has a full active view, c has to select some neighbor d from its active view to exchange for i. Therefore c sends to d a REPLACE message, stating its desire to remove d from its active view; this message also indicates to d that it can connect to o in exchange. The REPLACE message also carries information concerning the identification of the initiator of the optimization procedure. In order to promote the decrease of average link cost, the selection of d is deterministic, in such a way that d is the neighbor of c with the higher cost (excluding, naturally, the first μ protected members).

In the latter case, to conclude this step, c has to receive a REPLACEREPLY message from node d (Algorithm 1, lines $28 - 32$) or suspect that d has failed (in which case, node c acts as if it had a free slot in the active view from the beginning of this step). If d accepts the exchange, c will remove d from its active view and replace it with i. If d declines the exchange, c does not change its own views. In either case, c will notify i of d's answer using the OPTIMIZATIONREPLY message.

Step 3 This step begins with the reception at node d of a REPLACE message (Algorithm 1, lines $33 - 37$) and ends when a REPLACEREPLY message is sent back to node c.

A REPLACE message explicitly requests node d to exchange node c with node o in its active view. Node d consults the oracle to assess if o has a lower link cost than c. Note that our algorithm only requires 2 of the 4 nodes involved in an optimization round to consult the oracle in order to assess the merit of the proposed peer exchange. This is enough as we assume that link costs are approximately symmetric (in Section 5 we evaluate the performance of the protocol in a realistic scenario), and effectively, we are exchanging 2 existing links in the overlay, for other 2. We exchange the link between i and o for a link between i and c and the link between d and c with the the link between d and o. Therefore, only nodes i and d need to assess the gain resulting from these exchanges. The conservative use of the oracle provides a benefit if the implementation of the oracle adds some overhead (*e.g.* messages sent) whenever the oracle is consulted by a node.

Naturally, if node d verifies that there is no gain in the exchange of c for o, d aborts the exchange by notifying c. Otherwise, it will send a SWITCH message to node o notifying him to switch node i in its active view for himself. Moreover, it notifies node o. Finally, the answer received from o in a SWITCHREPLY (Algorithm 1, lines $38 - 42$) is forwarded to c.

Step 4 This step is executed by node o upon the reception of a SWITCH message (Algorithm 1, lines $43 - 48$) and ends when o sends a SWITCHREPLY to d. This step is required only to ensure the symmetry of active views. The behavior of node o in this step is deterministic. For clarity, in Algorithm 1 we only depict 2 of the constraints that are checked before accepting the exchange. The complete list of constraints that have to be checked can be found in [16]. After checking all constraints, node o sends a DISCONNECTWAIT message to i and adds d to its active view. This concludes an optimization round.

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

5. Evaluation

5.1. Experimental Setting

We conducted an extensive experimental evaluation of X-BOT against GoCast, Araneola, and T-Man using the PeerSim simulator [12]. To that purpose, all protocols were implemented in the simulator using the cycle-based engine of PeerSim. Furthermore, all protocols use the same Oracles to allow a fair comparison. Also, to assess that our implementations of GoCast, Araneola, and T-Man are accurate, we did compare their performance with the results that have been published in the corresponding papers. In the experiments reported here, we use the following two scenarios, that allow us to assess the benefits of X-BOT in environments with different characteristics:

Cartesian Scenario: This scenario uses a network of 10.000 nodes organized in a cartesian plan (a 100x100 square), where two direct neighbors are at a distance of 1. We model the cost of a link between two nodes, as being equal to the distance between those nodes in the cartesian space. This scenario is interesting as it offers a high potential to optimize a random overlay. Moreover the link costs in this scenario are symmetric and have a gaussian distribution.

Planet-lab Scenario: This scenario is composed of 341 nodes in which the cost of a link is defined according to the *all pair pings* trace[5] that contains ping times measured among a set of Planet-Lab[6] nodes. Each simulated node represents a Planet-Lab node and the cost between any two nodes, n_i and n_j, is set as half of the round trip time between these two nodes as measured from n_i according to the Planet-Lab traces. Notice that in this scenario, link costs are not necessarily symmetric. This scenario allows us to observe the performance of X-BOT in a realistic setting.

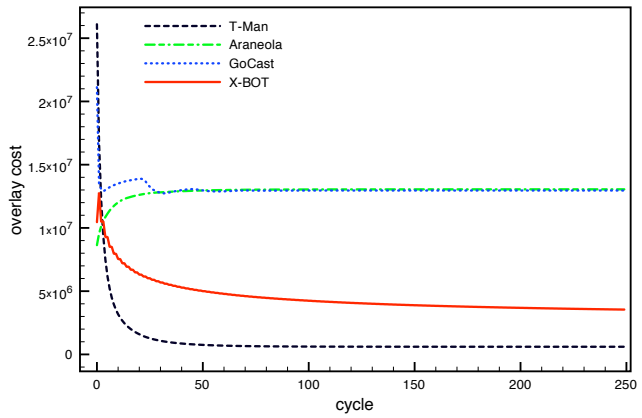
Due to lack of space we only provide experimental results for configurations where protocols attempt to maintain a single random/unbiased neighbor neighbor (both our experiments and previous work[20] suggest that this is the most useful configuration). The size of the passive views maintained by X-BOT was set to 30. The remaining protocols benefit from a (similar) random partial view maintained by a companion membership protocol. To ensure a fair comparison, we also set the size of these views to 30. Moreover, we initialized these views with contents extracted from the passive views of HyParView after 250 simulation cycles. Simple maintenance routines for these views, similar to the ones employed by X-BOT, were also added to each protocol. Furthermore, because T-Man lacks a join procedure, we initialized its views with contents extracted from the active views of HyParView in similar conditions.

For all simulations presented in this paper, X-BOT was configured as follows: **Period between optimizations** was set to 2 simulation cycles. This increases the probability of having new peers in the passive view of a node in each optimization step. As described in [16], passive views are updated each cycle; **Passive scan length (π)** was set to 2, so each time a node executes the step 1 of the optimization algorithm, it tests, at most, 2 nodes from its passive view. This also limits to 2 the number of nodes which are exchanged in a single round for a node. 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 and, *ii)* it lowers the cost of the overall optimization process. We will show that such conservative configuration allows to achieve fast convergence and a good level of optimization for the overlay.

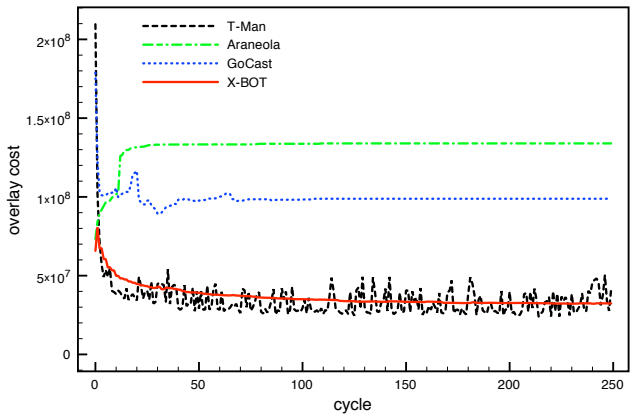
Finally, the initial (external) partial views provided to Araneola and GoCast were sorted by link cost. This is the most favorable configuration for these protocols. However it requires additional uses of Oracles, which might incur in additional overhead if the implementation of the Oracle requires the exchange of messages.

[5] A repository with these traces can be found in: `http://pdos.csail.mit.edu/~strib/pl_app/` .

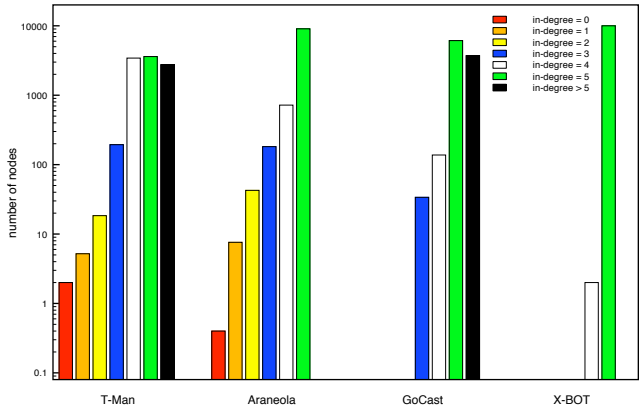
[6] `http://www.planet-lab.org/`



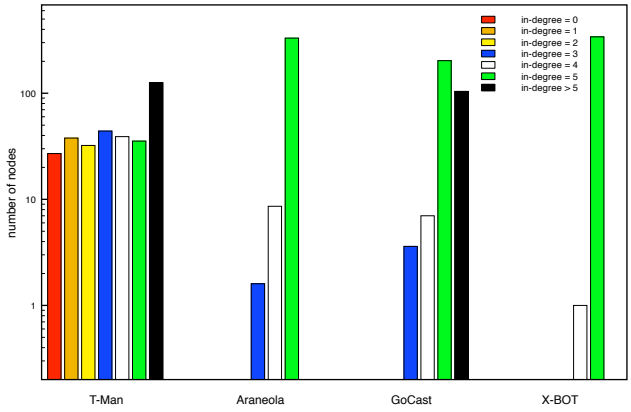
(a) Overlay cost in the cartesian scenario



(b) Overlay cost in the planet-lab scenario



(c) In-degree in the cartesian scenario



(d) In-degree in the planet-lab scenario

Figure 2. Resulting overlay networks.

5.2. Overlay Properties

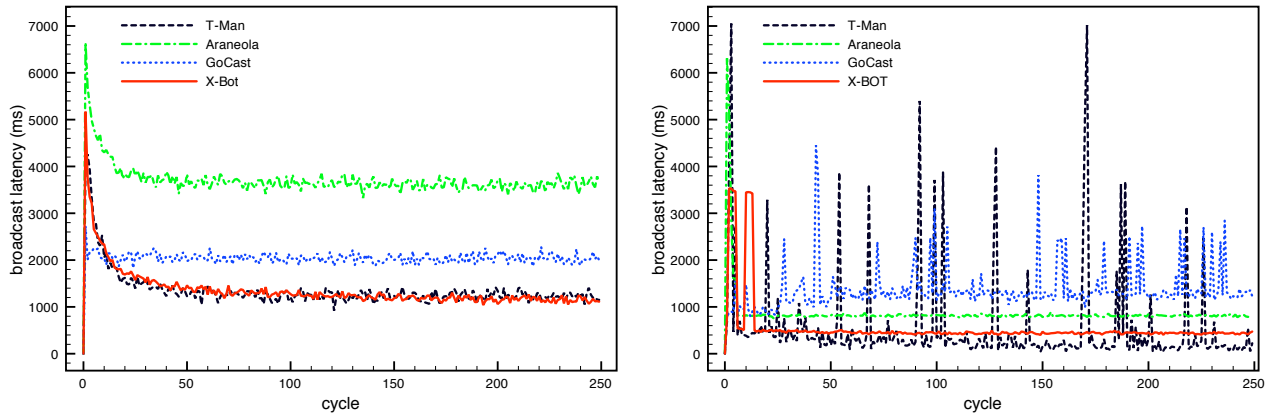
In this section we evaluate a set of relevant metrics concerning the overlay topology resulting from the operation of each protocol. Experiments were conducted by executing each protocol for 250 simulation cycles, and observing the evolution of the overlay and its final state. Values presented here are an average of several runs, and address the two simulation scenarios described earlier.

The most relevant metric is the overlay cost. Figures 2(a) and 2(b) depict, respectively, the overlay cost for the cartesian and planet-lab scenarios for all protocols. Compared with both Araneola and GoCast, X-BOT presents a lower overlay cost. This can be explained as follows: Araneola is a reactive protocol, in the sense that once the partial view of a node stabilizes (*i.e.* by matching all protocol requirements) it will never be updated again until some external event happens (*e.g.* a neighbor fails). Therefore Araneola does not explore the full optimization potential of the environment. GoCast, on the other hand, is able to iteratively improve the overlay topology, unfortunately the protocol does not ensure a constant degree of nodes (see Figures 2(c) and 2(d)), which results in the creation of additional links that increase the overlay cost. Moreover, considering that the average cost of each link maintained by GoCast is higher than the cost of the links maintained by X-BOT, it is possible to observe that our 4-node coordination optimization strategy offer better results than the GoCast strategy.

T-Man achieves a performance that is similar to X-BOT in the planet-lab scenario and can even achieve a

	Cartesian Scenario		Planet-Lab Scenario	
	CC	ASP	CC	ASP
T-Man	0.274	–	0.574	–
Araneola	0.196	9.904	0.101	4.300
GoCast	0.001	6.017	0.027	3.862
X-BOT	0.021	6.825	0.117	4.446

Table 1. Overlay properties.



(a) Cartesian scenario

(b) Planet-Lab scenario

Figure 3. Message dissemination latency.

more efficient overlay than X-BOT in the cartesian scenario. This is due to its aggressive optimization strategy. Unfortunately, this aggressive strategy has severe drawbacks, namely it has a negative impact in the overlay connectivity. Figures 2(c) and 2(d) depict the in-degree distribution obtained with each protocol. T-Man generates overlays where several nodes exhibit an in-degree of 0 while other nodes have a very a high in-degree (as high as 23). As a result, T-Man is unable to preserve the connectivity of the overlay, which has a negative impact on applications and services running on top of the overlay (such as gossip-based broadcast protocols).

Table 1 shows the resulting clustering coefficient (CC) and average shortest path (ASP) for all protocols in both scenarios. Notice that T-Man does not present a value for average shortest path, as none of the executions of the protocol was able to construct a connected overlay. X-BOT and GoCast offer the best results, although GoCast achieves these results at the expense of maintaining several nodes with a node degree much higher than the target value.

5.3. Message Dissemination

In this section we evaluate the performance of the gossip-based broadcast protocol described in [16] operating on top of each overlay. Considering that the target node degree in the overlay is 5 we set the fanout value of the gossip protocol to 4 (i.e., the largest fanout that prevents a message from being sent more than once on any given link). As before, the values presented are an average of several independent experiments. Link latency is captured by the link cost. The event-based engine of Peersim was used to implement the broadcast protocol.

Figures 3(a) and 3(b) depict the broadcast latency (*i.e.* the amount of time required to deliver a message to the maximum of participants) for each protocol. Only T-Man is able to provide better latency than X-BOT. This only

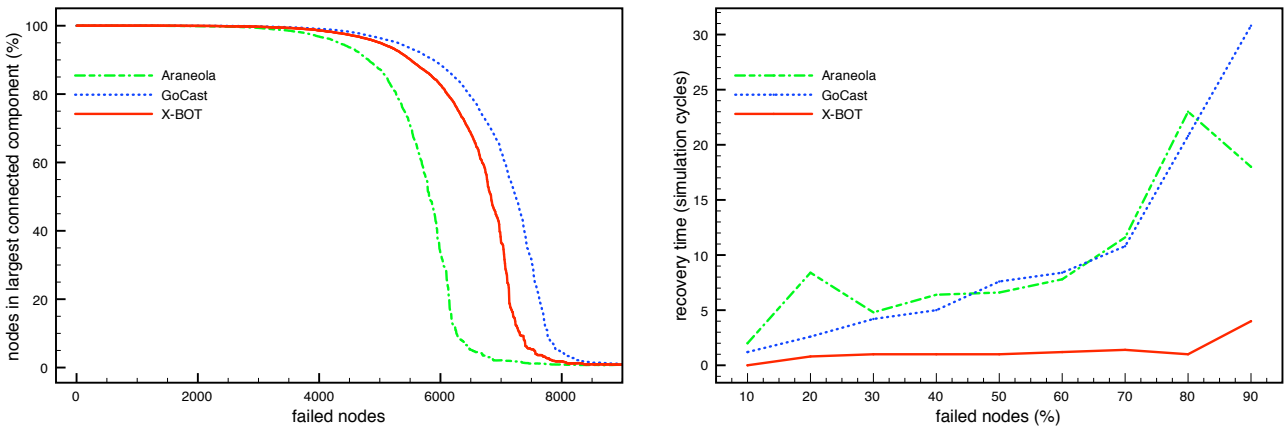
happens because T-Man is not able to provide a broadcast reliability of 100% as its overlay becomes disconnected. This is notorious in the planet-lab scenario, which has a non gaussian link cost distribution. The good performance of X-BOT its due to its capacity to improve the overlay efficiency while preserving the in-degree distribution and connectivity (as shown in the previous Section). Notice that in the planet-lab scenario, GoCast and T-Man latency exhibits several spikes. This is due to the fact that adaptations of the overlay affect node degrees, which can consequently, affect the overlay diameter resulting in additional latency.

To provide a better comparison among protocols, Table 2 shows both the latency and the reliability values of the message dissemination runs. Notice that, in both scenarios, X-BOT offers better reliability with a lower latency. This clearly shows that X-BOT, when equipped with a latency oracle, offers the best support to implement gossip-based broadcast protocols.

	Cartesian Scenario	
	Latency (ms)	Reliability (%)
T-Man	1149.4	99.420
Araneola	3659.4	99.996
GoCast	1995.8	99.998
X-BOT	1111.8	100.000
	Planet-Lab Scenario	
	Latency (ms)	Reliability (%)
T-Man	222.8	16.3655
Araneola	807.2	100.0000
GoCast	1228.0	100.0000
X-BOT	473.2	100.0000

Table 2. Overlay properties.

5.4. Fault Tolerance



(a) Connectivity

(b) Healing Time

Figure 4. Overlay resilience.

In this section we evaluate the resilience and healing capabilities exhibited by each protocol in face of node

failures. We assume that nodes can fail by crashing, and that TCP enables a node to detect these failures after a small amount of time (the following simulation cycle). Due to space limitations, we only present results for the cartesian scenario that is the one with a larger number of nodes. Moreover, we omit T-Man given that the overlays obtained with this protocol are not connected, even without faults.

Figure 4(a) plots the percentage of correct nodes in the largest connected overlay component as nodes fail one by one. In these simulations protocols were not allowed to take any corrective measures. X-BOT offers better connectivity in face of failures when compared with Araneola. However, GoCast connectivity surpasses that of X-BOT. This is not surprising, giving that GoCast maintains a significant number of nodes with degree above 5 (unfortunately, as discussed previously, this feature has a negative impact on the performance of gossip-based broadcast protocols).

We then evaluated the time required by each protocol to recover from massive failures that range from 10% to 90% of simultaneous nodes crashes. To this end, we measured the number of simulation cycles required, in average, for the broadcast protocol to regain its previous (or maximum) reliability values after the induction of failures[7]. Results are depicted in Figure 4(b). Notice that X-BOT is able to recover from failures much faster. This is due to the design of X-BOT which, unlike GoCast or Araneola, promotes connectivity, by avoiding to improve the overlay topology when nodes do not have a full active view.

6. Conclusions and Future Work

In this paper we proposed X-BOT, a new protocol that allows an unstructured overlay network to bias its topology according to a target efficiency criteria. The challenge addressed in the present paper was to improve the overlay topology by selecting better links without loosing the good properties of the original overlay (such as the low clustering coefficient, in-degree distribution, and high failure resiliency and recovery).

Experimental results show that X-BOT is able to improve the overlay topology to more efficient configurations than previous approaches. A significant feature X-BOT is its ability to promote the overlay connectivity, by preserving node degrees. As a result, X-BOT is able to support efficient and resilient gossip-based broadcast solutions when equipped with an appropriate Oracle (*e.g.* a latency oracle). Moreover, X-BOT is able to recover from failures faster than previous proposed solutions.

As future work we plan to experiment with other interesting oracles, such as oracles that reflect the similarity of the content stored by peers. These oracles could be used to build highly efficient resource location protocols on top of (biased) unstructured overlays.

References

[1] K. Birman. The promise, and limitations, of gossip protocols. *SIGOPS Oper. Syst. Rev.*, 41(5):8–13, 2007.

[2] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM TOCS*, 17(2), May 1999.

[3] Y.-H. Chu, S. Rao, S. Seshan, and H. Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1456–1471, Oct 2002.

[4] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proc. of the 6th PODC*, pages 1–12, 1987.

[5] P. Eugster and R. Guerraoui. Probabilistic Multicast. In *Proc. of the DSN*, 2002.

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

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

[8] I. Gupta, A.-M. Kermarrec, and A. Ganesh. Efficient and adaptive epidemic-style protocols for reliable and scalable multicast. *IEEE Trans. Parallel Distrib. Syst.*, 17(7):593–605, 2006.

[7] Notice that whereas in general X-BOT always regains a reliability of 100%, the same is not true for other protocols.

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

[10] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In *Proc. of the 5th Middleware*, pages 79–98, 2004.

[11] M. Jelasity and A. Montresor. Epidemic-style proactive aggregation in large overlay networks. In *Proc. of the 24th ICDCS*, pages 102–109, Tokyo, Japan, 2004.

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

[13] P. Karwaczyński, D. Konieczny, J. Moçnik, and M. Novak. Dual proximity neighbour selection method for peer-to-peer-based discovery service. In *Proc. of the 22nd ACM SAC*, pages 590–591, 2007.

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

[15] J. Leitao, N. A. Carvalho, J. Pereira, R. Oliveira, and L. Rodrigues. *Handbook of Peer-to-Peer Networking*, chapter 10. Springer, 2009, (to appear).

[16] J. Leitao, J. Pereira, and L. Rodrigues. HyParView: A membership protocol for reliable gossip-based broadcast. In *Proc. of the 37th DSN*, pages 419–429, Edinburgh, UK, 2007.

[17] M.-J. Lin and K. Marzullo. Directional gossip: Gossip in a wide area network. In *Proc. of the 3rd EDCC*, pages 364–379, 1999.

[18] L. Massoulie, A.-M. Kermarrec, and A. J. Ganesh. Network awareness and failure resilience in self-organising overlay networks. In *Proc. of the 22nd SRDS*, pages 47–55, 2003.

[19] R. Melamed and I. Keidar. Araneola: A scalable reliable multicast system for dynamic environments. In *Proc. of the 3rd NCA*, pages 5–14, Washington, DC, USA, 2004.

[20] C. Tang and C. Ward. GoCast: Gossip-enhanced overlay multicast for fast and dependable group communication. In *Proc. of the DSN*, pages 140–149, Washington, DC, USA, 2005.

[21] S. Voulgaris, D. Gavidia, and M. Steen. Cyclon: Inexpensive membership management for unstructured P2P overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.