Lightweight, Efficient, Robust Epidemic Dissemination

Miguel Matos[a], Valerio Schiavoni[b], Pascal Felber[b], Rui Oliveira[a], Etienne Riviere[b]

[a] *HASLab - High-Assurance Software Lab, INESC TEC & U. Minho, Portugal.*
[b] *University of Neuchâtel, Switzerland.*

Abstract

Today's intensive demand for data such as live broadcast or news feeds requires efficient and robust dissemination systems. Traditionally, designs focus on extremes of the efficiency/robustness spectrum by either using structures, such as trees for efficiency or by using loosely-coupled epidemic protocols for robustness.

We present Brisa, a hybrid approach combining the robustness of epidemics with the efficiency of structured approaches.

Brisa implicitly emerges embedded dissemination structures from an underlying epidemic substrate. The structures' links are chosen with local knowledge only, but still ensuring connectivity. Failures can be promptly compensated and repaired thanks to the epidemic substrate, and their impact on dissemination delays masked by the use of multiple independent structures.

Besides presenting the protocol design, we conduct an extensive evaluation in real environments, analyzing the effectiveness of the structure creation mechanism and its robustness under dynamic conditions. Results confirm Brisa as an efficient and robust approach to data dissemination in large dynamic environments.

Keywords: Data dissemination, Epidemic protocols, Gossip-based protocols, Peer-to-peer, Distributed systems

Email addresses: `miguelmatos@di.uminho.pt` (Miguel Matos), `valerio.schiavoni@unine.ch` (Valerio Schiavoni)

1. Introduction

We live in a digital era whose foundations rely on the production, dissemination, and consumption of data. The rate at which content is produced is constantly increasing [15], putting pressure on dissemination systems able to deliver the data to its intended consumers. Examples include the distribution of digital media (e.g., music, news feeds) on the Internet [13] or software updates in a datacenter infrastructure [34].

On account of its importance, significant research has been dedicated to conceiving efficient and robust data dissemination systems [5, 6, 25, 3, 10]. Unfortunately, both design vectors, efficiency and robustness, are often addressed disjointly: either by a highly efficient structure based on trees or by a highly robust unstructured epidemic or gossip-based approach.

Disseminating data using trees is attractive because, once formed, trees enable an efficient data delivery to all participants by avoiding duplicate message transmissions [39, 7]. However, under churn and faults, the rigid structure that makes the tree efficient must be rebuilt constantly, hindering robust dissemination and continuity of service, and significantly increasing delays for all nodes that lie in the subtree rooted at a failed node. These reconstruction delays moreover accumulate along the path to leaves, when multiple faults occur during a dissemination.

On the other hand, epidemic-based dissemination systems rely on redundancy instead of structure to offer guarantees on the delivery of data to all participants [3, 10]. Gossip-based dissemination was initially proposed in the context of database replica synchronization in the ClearingHouse project [9]. The transmission of several copies of the same message to random nodes enables epidemic-based systems to be oblivious to faults and churn, as the same message will be received through different paths. The cost is increased bandwidth and processor usage due to the transmission and processing of duplicates.

Epidemic principles have also been used elsewhere to build robust and scalable distributed systems components such as membership [17, 14, 21] and failure detection [31] services, or indexing mechanisms [29, 8]. As long as (1) the graph induced by the (partial) views offered by the membership service is connected and (2) all nodes have at least one incoming link, dissemination can trivially be achieved by flooding.

Contributions. In this paper we present BRISA, an efficient, robust and scalable data dissemination system. BRISA leverages the robustness and scalability of an epidemic substrate to build efficient dissemination structures that are correct, i.e., cover all nodes, by construction. Such structures are built in a distributed fashion with local knowledge only and with minimal overhead. Brisa has been designed in a way that upon failures or churn, trees are easily and rapidly repaired thanks to the underlying epidemic substrate that acts as a safety net. As dissemination structures we consider trees, directed acyclic graphs (DAGs) and forests of trees. We evaluated Brisa on PlanetLab [1] and on a local cluster comparing it with state-of-the-art data dissemination systems from the literature. An initial version of this work was presented in [27].

Roadmap. The remaining of this paper is organized as follows. Section 2 describes the design of Brisa and Section 3 presents the experimental evaluation. In Section 4 we discuss related work and finally Section 5 concludes the paper.

2. Brisa

In this section, we describe the design of Brisa. Brisa relies on an underlying peer sampling service (PSS), and thus we first discuss its requirements and the guarantees it provides. Then, we introduce the key design principles of the Brisa protocol and how the dissemination structures are constructed. Finally, we show how Brisa deals with dynamism, generalize the construction of dissemination structures with desirable efficiency/robustness criteria and discuss the creation of multiple dissemination structures.

2.1. Peer Sampling Service Layer

We assume an underlying PSS [17] that provides each node with a *view*, i.e., a set of non-faulty nodes chosen at random from the entire network. The PSS creates views at each node, that is, a set of neighbors to which the node is connected. One of the major objectives of the PSS is to ensure that the overlay composed by the views at all nodes is *connected*. This implies that every node is able to transitively reach every other node in the network even under high rates of churn and failures [21, 14]. Moreover, the PSS is expected to rapidly replace failed nodes from the views and provide similarly sized views and evenly distribute the number of incoming links at each node for load balancing and robustness purposes.

The update of the views can be either continuous (*proactive* peer sampling) or happen only when a node fails or a new one joins the system (*reactive*

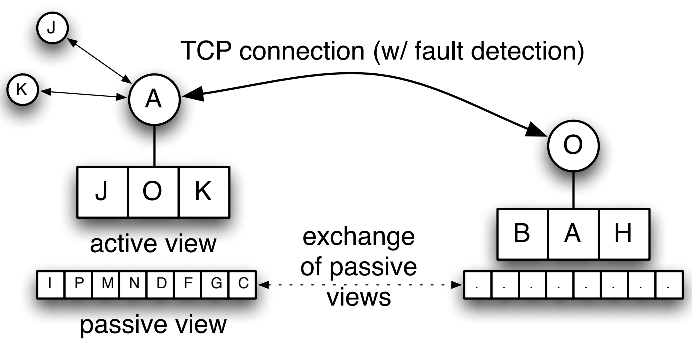
TCP connection (w/ fault detection)
J
K
A
O
J O K
active view
B A H
exchange of passive views
I P M N D F G C
passive view

Figure 1: HyParView [21]: views maintenance.

peer sampling). In the *proactive* case, nodes periodically share their views with their neighbors regardless of the actual need to replace failed entries, resulting in each view being a continuous stream of node samples from the network. Examples of proactive PSSs include Cyclon [36] and Newscast [37]. In the *reactive* case, the view is kept unchanged unless some of its entries need to be updated, i.e., for replacing a failed node or for accommodating a node joining the system. Typical examples include Scamp [14], Araneola [28] and HyParView [21].

In this paper we rely on a reactive PSS and more specifically on HyParView [21]. The motivation for this choice comes from the additional stability of reactive approaches, which simplifies the process of creating efficient and correct dissemination structures. In short, HyParView maintains two views at each node: a larger *passive* view and a smaller *active* view (see Figure 1). Only the active view containing the node's neighbors is exposed to the application and in particular to Brisa. The passive view is maintained in a proactive manner by periodic exchanges and shuffling of passive views with randomly selected neighbors, that are also selected from the passive view itself. The entries in the active view are managed in a reactive manner: a neighbor in this view only changes upon failures, or for accommodating a newly joined node. An opened TCP connection is maintained with each of the nodes in the active view for communication efficiency, in particular, latency. Due to the limited size of the active view, efficient heartbeat-based fault detection can be used for all of its members. Upon detection of a failed neighbor, a replacement node is selected from the passive view and moved

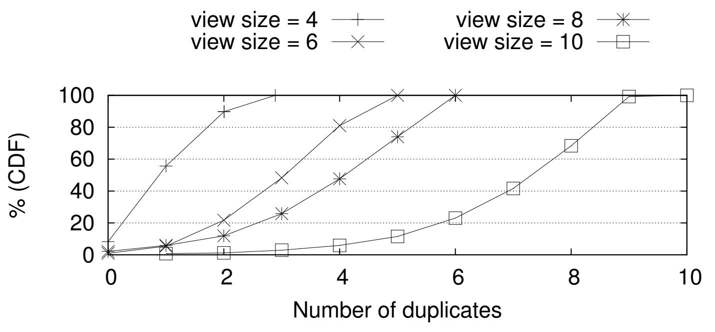

Figure 2: Distribution of duplicates per message for each node for 500 messages in a 512 nodes HyParView network for various active view sizes.

to the active view. When the active view is full and a new node attempts to join, a random node is removed from the active view to accommodate the joiner. In order to avoid chain reactions due to the massive number of joins when bootstrapping the system (node A's view size is full so it removes node B, B also removes A from its view and promotes a node C from its active view, C must add B to its view and thus remove an existing one as its active view is already full, removing D and so on and so forth), we allow the active view size to grow past the configured value by a given *expansion factor*. Nodes evictions do not result in replacements when the view size is between the target view size and this size times the expansion factor. We used an *expansion factor* of 2 throughout the evaluation. The impact on the actual view sizes is limited as shown later in the analysis of the degree distribution (Section 3.1, Figure 7).

An important aspect of HyParView is that links with neighbors are *bidirectional.* If node A has node B in its active view, then B also has A as its neighbor. In a connected overlay, using bidirectional links allows us to ensure that messages disseminated by flooding will reach all the nodes in the system without requiring anti-entropy mechanisms where nodes periodically poll other nodes for the content they might have missed [9]. A node receiving a message *for the first time* from a neighbor simply propagates it to all its other neighbors.

Flooding is ensured to reach all nodes as long as no node in the system has an active view with only failed nodes. The larger the active views the smaller the chances for this to occur. However, the larger the view, the larger the number of relayed messages and consequently the number of duplicate receptions. As a concrete example, Figure 2 presents the cumulative distribution function (CDF) of the number of duplicates during the dissemination of 500 messages over a 512 nodes HyParView network for different view sizes. We observe that as the size of the view grows, nodes quickly receive large amounts of duplicate messages. For instance, half of the nodes receive more than one duplicate with a view size of 4, while they receive more than 7 duplicates with a view size of 10.

Brisa develops on top of HyParView. It takes advantage of the connectivity guarantee that can tolerate up to 80% node failures [21] to emerge efficient dissemination structures that eliminate (or considerably reduce) the number of duplicates, while keeping the robustness offered by the underlying PSS.

2.2. Rationale

The objective of Brisa is to support the efficient, robust and scalable dissemination of a stream of messages from one or several sources to the entire network. Efficiency relates primarily to the limitation of duplicate message transmissions that waste bandwidth and processor resources. On top of that, Brisa can consider additional efficiency criteria, namely: the reduction of the end-to-end delay (dissemination time from the source to the last receiver) and network efficiency (ratio between the delay for receiving a message through Brisa as compared to a hypothetical direct communication from the source). Robustness relates to fault tolerance: dissemination should progress despite the inactivity of some nodes (failure or disconnection) and the system should be able to rapidly detect and mask such faults. Finally, Brisa scales to very large networks, because the view size is kept small and under strict control by the PSS thus preventing the load at any node to grow linearly with the system size.

The main idea behind Brisa stems from the observation that it is the *possibility* of receiving messages through multiple paths that makes epidemic-based approaches robust, not necessarily the actual data transmission. Our goal is to therefore to limit or even eliminate duplicate transmissions while maintaining the *possibility* of receiving the messages through multiple paths. Such possibility is given by the view provided by the PSS which contains a set of potential senders. From this set, Brisa selects one or more to perform the actual data transmission thus materializing the possibility into a concrete delivery.

Based on this selection, Brisa automatically derivates dissemination structures on top of the undirected HyParView overlay. Such structures are oriented and can be either trees, by restricting the inbound neighbors of every node to a single node (parent), or directed acyclic graphs (DAG) by allowing multiple parents for each node. The creation of a structure is performed by local and unilateral decisions made by the nodes about the set of neighbors that should be *active* and actually relay inbound traffic and those that should be *inactive*. In the case of a tree the reception of duplicates is effectively eliminated; in a DAG, it is selectively reduced.

The resulting dissemination structure must ensure complete disseminations, i.e. that all nodes receive all messages. To that end, we must ensure that it does not contain a non-connected sub-graph that would not receive the message from the other components of the structure. This property is ensured by enforcing the absence of *cycles*. In fact avoiding cycles is the main concern when determining the set of active and inactive neighbors of a node. In the following sections, we first describe how the emergence of a single tree is achieved in Brisa, then generalize the approach to DAGs, and finally delineate the use of forest of trees.

2.3. Emergence of a Dissemination Structure

The emergence of Brisa's dissemination structures is part of the natural operation of the system and is based on the reception of duplicates. Nodes start with all the links active and thus the initial dissemination structure exactly matches the HyParView overlay. These links form a graph that serves as the basis for the construction of a Brisa dissemination structure. Initially, a source node sends the first message of the stream to all its neighbors. Nodes receiving the message for the first time simply forward it to all the nodes in their view because all links are active, effectively flooding the network.

This flooding operation reaches all nodes, given the connected and bidirectional nature of the overlay provided by HyParView. During the initial flood, nodes receive the message from a number of different neighbors. Out of these sources, each node autonomously selects one as its parent in the dissemination structure and sends a deactivation message to all the others. Future messages in the stream will then be received only from the selected parent node. The selection is achieved by the use of a *link deactivation* mechanism and follows one of the selection strategies presented in Section 2.5. To emerge a tree each node simply needs to prune out *all but one* of its inbound links.

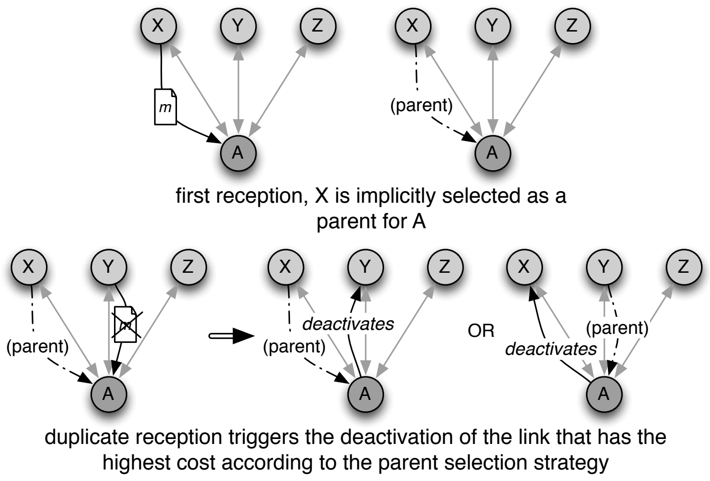
X
Y
Z
m
A
(parent)
first reception, X is implicitly selected as a parent for A
deactivates
OR
duplicate reception triggers the deactivation of the link that has the highest cost according to the parent selection strategy

Figure 3: Reception of a duplicate and deactivation of one link, for a tree Brisa structure. Depending on the parent selection strategy, the deactivated link can be the previous parent or the node sending the duplicate.

Note that the bootstrap can also be done by injecting an empty message (without payload) in the system if the initial flood of an application message poses bandwidth concerns.

It is important to note that deactivating a link does not imply removing the corresponding entry from the HyParView active view. The overlay constructed by the PSS remains available and is used both as a provision of nodes for reparations upon failure, or as a fallback for dissemination when reparation is temporarily not possible.

Figure 3 presents the principle of the link deactivation mechanism for constructing a tree. Initially, links from nodes X, Y, and Z belonging to node A's view and are all *active*. The first reception of a message from node X results in node A considering X as its parent. A subsequent reception of a duplicate from node Y triggers the link deactivation mechanism. As only one inbound link should be active, node A needs to deactivate either the link from node X or the link from node Y.

There are three guiding principles for deciding which link to deactivate. First, the dissemination structure must not contain cycles. Second, it must seek to meet the target number of parents for each node (one for the tree structures, more when generalizing to DAGs). Finally, when both condi-

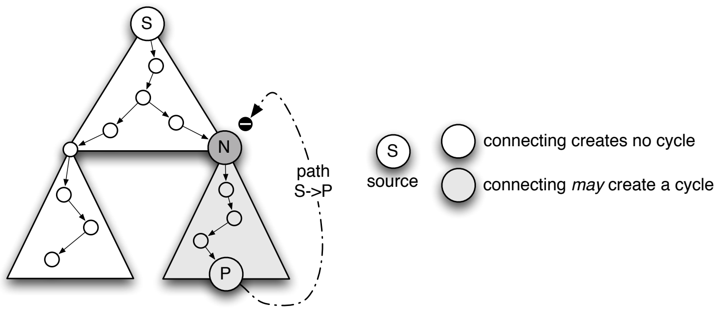

Figure 4: Avoiding creating a cycle for a tree, by checking that the node N is not in the dissemination path to the potential parent.

tions are met, the parent selection strategy chooses the new parent based on different criteria for shaping the dissemination structure (Section 2.5).

2.4. Preventing Cycles

A mandatory condition for selecting a parent node is that it does not yield a cycle in the dissemination structure. This means that the potential parent of a node N does not receive the stream directly or indirectly from N itself. For a tree this implies that the parent of N must not appear in the sub-tree rooted at N.

To verify this condition each node piggybacks on the application messages the node identifiers in the path from the source to itself. When selecting its parent, a node N rejects those candidates whose message path to the source includes N itself. This is illustrated in Figure 4, where grey nodes are not eligible as parents of node N. It is important to note that the overhead of path embedding is minimal and very attractive when compared to probabilistic inclusion structure such as Bloom filters [4]. As a matter of fact, the size of the embedded path is bounded by the tree height, which is expected to be $O(log_b(N))$ where N is the system size and b the active view size. For instance, in a system with 1×10^6 nodes with an active view size of 8, the average tree height is $log_8(1 \times 10^6) \approx 7$. This bounds the maximum metadata size a message needs to carry which, assuming a 48 bit (ip,port) pair as unique identifier, is only 336 $(7 * 48)$ bits. A bloom filter, to ensure a reasonable false positive probability to avoid detecting cycles where there is none, would require about the same number, or more, bits.

Taking into account the metadata size required, the fact that path embedding is exact (false positive probability is zero) and the computational overhead associated with Bloom filters (which requires computing several hashes), path embedding presents many advantages over Bloom filters.

The detection of cycles is not only done during the initial flooding phase: a node that detects a cycle from a parent simply makes the link from that parent inactive and selects a new parent using the regular selection mechanism or the fallback to flooding as we describe later in Section 2.6.

2.5. Parent Selection Strategies

From N's eligible parents (that is, those not having N in the path followed by the messages from the source), Brisa selects one according to the following strategies:

1. First-come first-picked. The node sending the first received message is selected as parent, all subsequent duplicates received trigger the deactivation of the incoming link.

2. Delay-aware. This strategy considers the round-trip time between N and the candidate nodes. The one with the lowest delay is selected as parent. We leverage the periodic keep-alive messages that are exchanged by the nodes in the active views at the HyParView level to measure round-trip times.

A simple optimization is available when building a dissemination tree using the first-come first-picked strategy: the deactivation of links can be symmetric. Supposing node A receives a message first from node B and then from node C, A will pick the link from B and send a deactivate message to C. But it can further mark its outgoing link to C as inactive as A knows it will not be not eligible as parent for C, as C already received the message first.

2.6. Dynamism

The insertion and removal of nodes in the system is handled by the underlying PSS. A new node joins by contacting a node already in the system. The new node is provided with an active view with the size of that of its contact point, and is inserted in the active views of the associated nodes. Brisa automatically marks links to new nodes as active. As a result, the joining node will have all its inbound links marked as active and will receive its first message multiple times. All that remains is to select its parent(s) according to the mechanism discussed previously.

The detection of node failures is performed at the level of the active view, by exchanging periodic keep-alive messages over the established TCP connections, or when a node fails to acknowledge the reception of a transmission (as detected by the TCP flow control for that link). When a node notices that one of its neighbors is removed from the active view (due to a failure), it first checks if that neighbor was a parent. If that is not the case, the removal can be ignored. Otherwise, the node needs to find a replacement parent using one of two strategies. It first attempts a *soft repair* by trying to select as parent one of the remaining neighbors. A simple approach is to reactivate all its inbound links and proceed with the normal parent selection process. This can however be optimized by leveraging the keep-alive messages used for monitoring the active view at the PSS level and piggyback up-to-date information required by the parent selection procedure. If a suitable parent is found then its inbound link is directly re-activated. Note that this mechanism uses local knowledge only and requires a single message exchange being thus very fast and efficient. Furthermore, as shown later in the evaluation, almost all repairs can be done using the soft repair strategy resulting in minimal disruptions and very fast recovery of the dissemination structure (Section 3.3).

If no replacement parent exists in the active view, we resort to a *hard repair* that uses the underlying flooding approach for rebuilding part of the dissemination structure. The orphan node first re-activates all its incoming links and considers itself a fresh node by forgetting its position in the cycle detection mechanism. This allows the orphan node to take any of its neighbors as a parent. To ensure the tree remains connected, it is necessary to rebuild the incoming links for a part of the structure rooted at that orphan node. The need to repair a portion of the tree is detected by the children of the orphan node when they receive an activation request from their (former) parent. Those nodes proceed then with the local repair attempting first a soft repair and if not possible resorting to a hard repair. We note that the effects of the hard repair are limited to a small portion of the tree and in practice stop as soon as a node can find a suitable parent in its active view. Besides, the former parent will receive subsequent messages from the children (remember the parent activated that link) and may effectively exchange roles.

The number of nodes affected by a hard repair is independent of the position of the original orphan node in the tree: it only depends on nodes in the sub-tree finding a suitable replacement parent, which is independent of

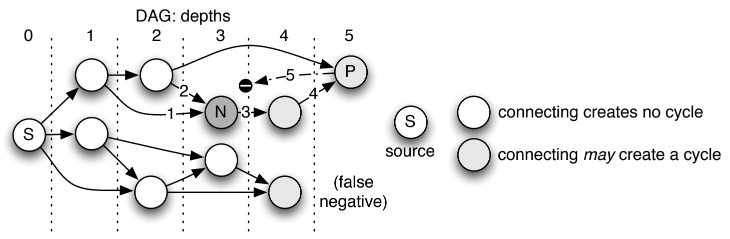

Figure 5: Avoiding creating a cycle for a DAG, by checking that the level of the potential parent it less than or equal to the level of the node.

the position of the original orphaned node.

Finally, nodes can compensate message loss during recovery by directly asking its new found parent to send the missing ones. Since parent recovery is quick (Section 3.3) the number of messages each parent needs to buffer is small. Nonetheless more complex approaches such as [18] could still be used to ensure nodes buffer messages for long enough to allow recovery.

2.7. Generalized Dissemination Structures

To enhance service continuity under failures and churn, Brisa can generalize the tree structure to directed acyclic graphics (DAGs) by having each node being served by several parents instead of only one. In this way, a node that sees one of its parents fail can seamlessly keep receiving the flow of messages without the need to first undergo through the parent recovery process. This is attained at the cost of handling a controlled level of duplicate messages.

The establishment of a DAG basically involves making a number $p > 1$ of inbound links active in such a way that cycles are avoided. The technique to prevent cycles we used for trees is however unfeasible in the case of DAGs due to the amount of control information required to be exchanged. Indeed, a node in the n^{th} level of the tree requires a set of n node identifiers to define the path from the stream source to itself, while for a DAG with p parents per node this set at level n could reach $p^{n+1} - 1$ should all paths be non-overlapping.

Conversely, for DAGs, we use an approximate quantitative approach that does not include the nodes identifiers but just the *depth* each node is in the DAG as illustrated by Figure 5.

The source node is at depth 0 and every message carries its sender's depth encoded by a single integer. Initially, the depth of a node N is undefined and, upon reception of its first message from a node with depth $i-1$, N places itself at depth i. From then on, N can select parents, and thus receive messages, from nodes at any depth not greater than i. Should N receive a message from a node at depth i (its current depth) then N moves to depth $i+1$ and immediately updates its downstream children nodes accordingly.

Similar to the technique we used for trees, it is clear that any node M served directly or transitively by node N will be at a depth strictly greater than N. Therefore, M cannot become a parent of N and yield a cycle.

As mentioned, the technique is however approximate because it can yield false negatives by discarding valid potential parents, as illustrated in Figure 5. Any two paths (rooted at S) are likely to be labeled similarly with respect to depths. Since the tagging is purely quantitative, a node from one path may be dismissed as a potential parent of a node in another path despite the paths being causally unrelated. An alternative is to rely on Bloom filters to maintain the set of nodes that need to be excluded for the parent selection process. However, as for trees this a costly technique when compared to the simplicity and efficiency of depth encoding. In our experiments, nodes are able to obtain the desired number of parents, thus we consider this approach an attractive alternative when compared to the cost of both an exact predictor (path embedding) and of a probabilistic one (Bloom filters).

After determining the set of potential parents with the above strategy all that remains is selecting the *best* ones by using the parent selection strategies presented in Section 2.5.

2.8. Multiple Dissemination Structures

So far we discussed the creation of a single dissemination structure, be it a tree or a DAG. In the remainder of this section we motivate and describe the support for multiple dissemination structures. For clarity of explanation, we focus on trees but the same principles apply to DAGs.

There are several cases where it is interesting to support more than one tree, for instance if the source needs to split the content across several trees as in SplitStream [5] or to apply network coding techniques, or simply if there are several sources in the system. Moreover, the use of multiple trees enables a better use of system resources as more nodes can contribute to the dissemination effort. This is because when using a single tree, the leaf nodes, which are a big portion of the system, do not upload data and thus their capacity is not used. Supporting several sources can be done by building a single tree rooted at a *rendezvous* node that acts on behalf of all sources as in Scribe [6]. This design suffers however from a bottleneck in the *rendezvous* node and fails to take advantage of the upload bandwidth available at leaf nodes.

Therefore, we consider instead the creation of several independent trees. In Brisa, a tree is simply given by the set of active and inactive links that each node locally maintains. Consequently, all that is required to maintain multiple trees is to locally maintain multiple such sets, one for each tree in the system. Each tree is uniquely identified by a *flowId* generated by the tree source at construction time. Note that as, by assumption, each node has a unique id, it is straightforward to generate unique *flowIds*, for instance by concatenating the node id with a local sequence number. The source then tags all application messages with its *flowId*, enabling other nodes to uniquely assign the messages to the appropriate tree. Upon reception of a message from an unknown *flowId*, a node locally creates a new set of active and inactive links dedicated to managing that tree and proceeds as detailed in Section 2.3.

This approach is very lightweight as it requires the maintenance of a small local state, yet due to the inherent randomness in tree creation enables a much more efficient use of the overall upload bandwidth as few nodes are leaf in all trees (as we show in Section 3.4, Figure 12).

Nonetheless, from a design point of view, we observe that the state each node needs to maintain grows linearly with the number of trees in the system. To mitigate this, we designed a tree reusing strategy that can be used when the number of trees grows. The base idea is very simple: instead of creating a new tree, a node simply reuses one it already knows to disseminate its messages. To this end, the node analyses the trees it knows and if it is close enough to the root of any tree according to *reuseDepth*, a protocol parameter, it uses that tree's *flowId* instead of creating a new one. Note that, due to path embedding, a nodes always knows its position in all trees it belongs to, so computing the distance to any root is inexpensive and requires only local knowledge. By reusing an existing *flowId*, the messages created by that node will simply be relayed through the existing tree with no further overhead. However, as the source node is not located at the root of the tree anymore, it is necessary to relay message upward in the tree, to ensure completeness. This is easily achieved by adding an *upward* flag to the message, implying that nodes need to relay those messages not only to their children but also to their parents. Another option would be to directly send the message to the root of the tree which would act as a *rendezvous* node. We note that the latter shows less bottlenecks problems than Scribe as it considers several *rendezvous* nodes, one for each existing tree, instead of just one. While simple, this strategy is very effective at reducing the number of total trees and the associated overhead. Obviously, reusing trees can have contradictory goals with the creation of multiple disjoint trees, e.g., as in SplitStream [5] or as shown in our evaluation Section 3. In these cases, the goal is to create multiple disjoint trees from a single source, in order for leaves in a tree to act as interior nodes in the other, and reversely, in order to balance the load of the dissemination of a stream that is split among the trees. Tree reusing can limit the benefit of this approach, leaves remaining leaves in multiple trees and keeping the dissemination load unbalanced. Nonetheless, tree reusing can still be beneficial, between trees that are used for different streams. Tree reusing shall only be prevented for the trees of a given stream. In this case, each such tree is marked with the identity of the other trees from the stream, and reusing is disabled for those trees in the reusing decision process.

3. Evaluation

In this section we evaluate Brisa on two different testbeds: (1) a local cluster of 15 computers equipped each with 2.2 GHz Core 2 Duo CPU and 2 GB of RAM and connected by a 1 Gbps switched network, supporting up to 512 Brisa nodes and (2) a slice of up to 200 nodes on the global-scale PlanetLab [1] testbed. The prototype leverages Splay [22], an integrated system for the development, deployment and evaluation of distributed applications.

The evaluation is focused on the aspects that drove Brisa's design: efficiency and robustness. For each experiment and unless otherwise stated, we bootstrap the system with the specified number of nodes using the first-come first-picked strategy with an expansion factor of two, randomly choose a node to be the source across all the experiment and then have it inject 500 messages at a rate of 5 per second, taking measurements as appropriate. The message payload is an opaque random bit string with the specified size.

We start with a preliminary study, in Section 3.1, on the structural properties of the dissemination structures created by Brisa as those properties impose well-known bounds in resource usage and dissemination time. Then, in Section 3.2 we inspect the network properties of Brisa, namely bandwidth consumption and routing delays and analyze the results according to

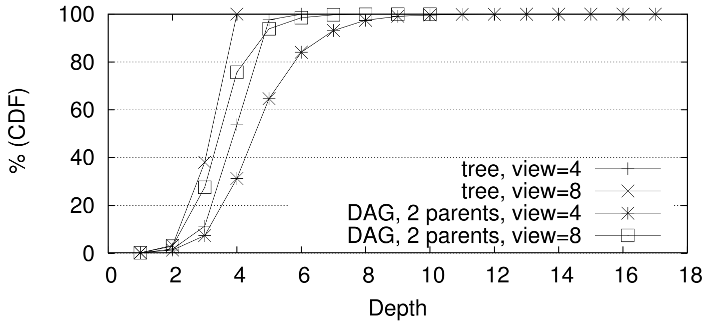

Figure 6: Depth distribution for 512 nodes (first-come first-picked strategy).

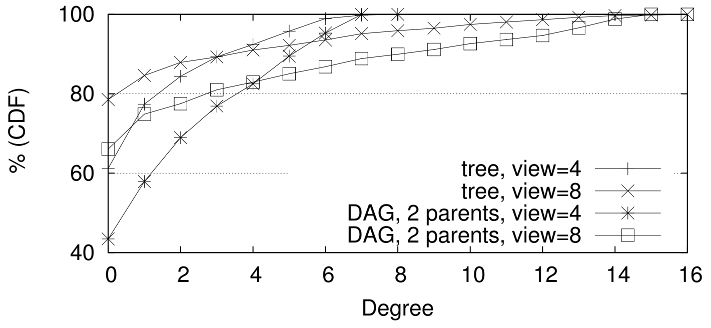

Figure 7: Degree distribution for 512 nodes (first-come first-picked strategy).

the structural properties. Next we evaluate the behavior of Brisa under churn in Section 3.3, and with multiple trees in Section 3.4. Finally, in Section 3.5, we compare Brisa with other approaches.

3.1. Structural properties

We first study the shape of the structures generated by Brisa, namely trees and DAGs with 2 parents. The shape (depth and degree), imposes constraints on latency and on the distribution of the dissemination effort. Results for each configuration are obtained after building the respective structure and letting it stabilize completely. The reason for using this basic strategy is twofold: i) a naive strategy helps to better understand the basic behavior of Brisa thus serving as a baseline for more elaborate strategies and ii) the limited number of physical nodes hides significative differences on the observation of structural properties changes that are better observed at larger

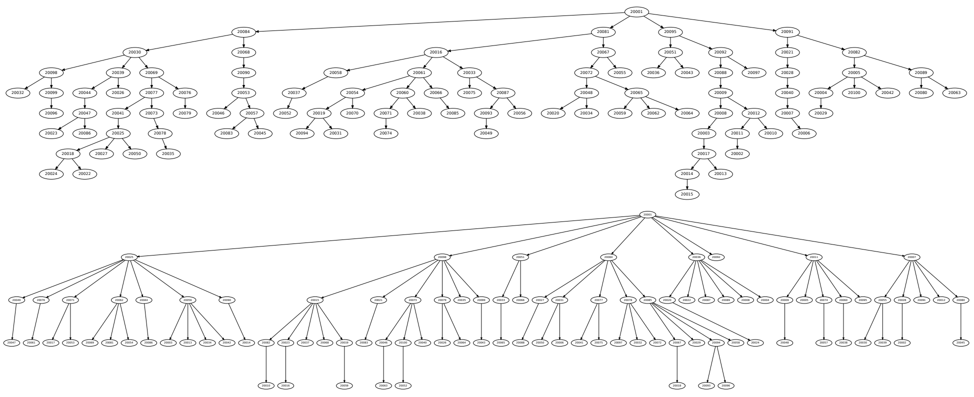

Figure 8: Sample tree shape for 100 nodes with a HyParView active view size of 4 on the top and 8 on the bottom. Expansion factor is 1.

scales. Depth places a lower bound on the dissemination time due to the cost of traversing several intermediate nodes and thus should be kept as low as possible. Figure 6 presents the depth distribution in a universe with 512 nodes. As expected, larger views allow nodes to have more children thus reducing maximum depth. The larger depths in DAGs are because depth measures the maximum distance, i.e. the longest path from the root to the node, which increases with the extra number of links. The steep curves hint that the structures built by Brisa are fairly balanced, i.e., do not degenerate into long chain even with a simplistic strategy thus preserving desirable properties for dissemination. An analysis of the degree distribution confirms this observation.

The degree of a node in Brisa is given by the number of outgoing links and thus bounds the message copies a node sends. This is directly related to the dissemination effort and as such degree distribution should be as narrow as possible indicating an evenly distributed load. When analyzing the degree distribution presented in Figure 7 three main observations arise. First DAGs are more effective than trees in having a greater share of the nodes contribute to the dissemination effort (nodes with degree zero are leaves). This is due to the additional number of parents that reduces the chance of having all outgoing links deactivated. Secondly, degree distribution is also highly affected by the view size provided by the PSS: higher values lead to shallower trees thus resulting in more leaves, while lower values lead to deeper trees due to the limitation imposed by the view sizes. Such relation between degree and depth can be observed in Figure 8, which depicts sample trees obtained by Brisa. As a matter of fact, despite using a simple strategy, the resulting trees are fairly balanced which is essential for efficient dissemination. Finally, despite using an *expansion factor* of 2 the number of nodes with degree higher than the configured value remains small as hinted in Section 2.1.

3.2. Network properties

In this section we focus on the network properties of the dissemination structures obtained by Brisa.

First, we analyze the routing delay of dissemination. To this end, we use the cumulative round trip times, taken at each hop, from the root to a given node. When compared against the round trip time of direct communication between the root and that node, it indicates the effectiveness of Brisa in building dissemination structures with low end-to-end delays, an essential property for a dissemination system. The ratio between the first and second measurements gives the stretch factor. However, due to PlanetLab asymmetries that deter direct communication between some nodes, we instead present the cumulative distribution of the raw results in Figure 9. Not surprisingly, the flooding strategy yields the worst results due to the heavy load imposed on the network. In this non-structural metric, the effects of a delay-aware strategy become clear when compared to the simplistic first-come first-pick: for instance, 40% of the nodes reduce the routing delays to half.

Next, we focus on the bandwidth usage. This measures the node's dissemination effort and is directly influenced by depth and degree distribution. Figure 10 and Figure 11 depict download and upload bandwidth usage, respectively, for payloads of 1, 10, 50 and 100 KB. We used stacked bars with decaying shades of grey for representing a distribution using a set of percentiles. For instance, the medium shade of grey gives the median value (half of the node below that value, the other half above), while the lighter shade gives the 90^{th} percentile: 90% of the nodes are associated with a lower bandwidth.

As expected, trees are more frugal with respect to download as nodes receive exactly one copy of each message whereas in DAGs nodes receive two copies (one for each parent). For each structure, the increase in bandwidth usage for the different view sizes is due to the PSS. The small difference, negligible when compared to application messages, hints at a low overhead service. The differences in the percentiles for the DAG are related to the

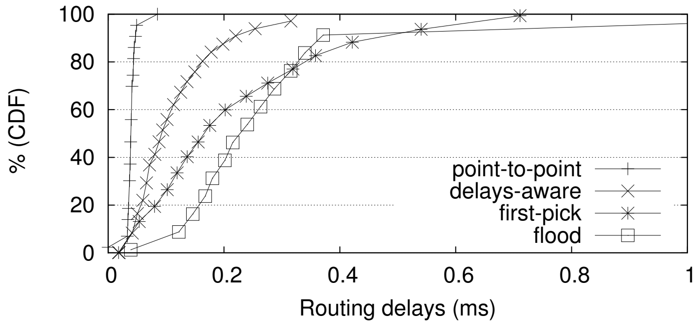

Figure 9: Routing delays distribution on PlanetLab for a 150 nodes network. Structure is a tree with view size 4. Message size is 1KB×200 messages.

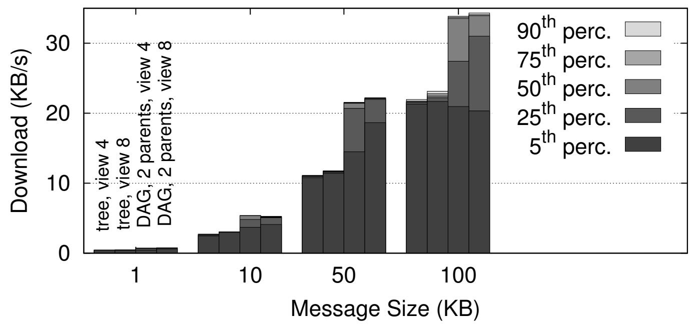

Figure 10: Bandwidth usage for a 512 nodes network, download

depth of nodes (Figure 6) as nodes at lower depths may not be able to find additional parents and thus receive messages only from a single parent.

For upload, results are naturally similar. DAGs require more links and consequently nodes will have to relay messages to more neighbors, increasing upload bandwidth usage. The differences between percentiles for a given configuration are explained by the degree distribution (Figure 7) as nodes with higher degrees need to upload more.

3.3. Robustness

We now focus on the behavior of Brisa under continuous churn in order to assess its robustness. Each experiment is associated with a synthetic churn trace based on the churn support module of Splay. The synthetic description is given in Listing 1 and proceeds as follows: first we bootstrap the system and let it stabilize. After, we induce churn at rate X by having X percent nodes fail at random and X percent new nodes join the system during each minute.

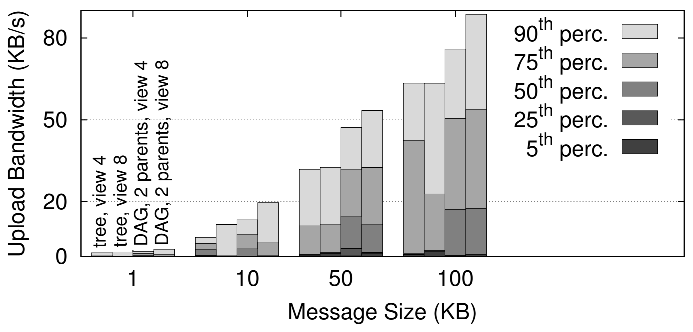

Figure 11: Bandwidth usage for a 512 nodes network, upload.

```
from 1s to N s join N
at 1000s set replacement ratio to 100%
from 1000s to 1600s const churn X% each 60s
at 1600s stop
```

Listing 1: Churn trace generation script

Table 1 depicts the results obtained for networks with 128 and 512 nodes. For simplicity we ensure that the source node does not fail. However, we note that the failure of source node would only produce a negligible impact in the presented results. In fact only the direct children of the source (a small number limited by the view size) would experience the effect of a parent failure.

We defined the following metrics:

- **Parents lost per minute:** rate at which nodes loose any of their parents;
- **Orphans per minute:** rate at which nodes loose all parents (implying disconnections);
- **Percentage of soft repairs:** upon disconnections, how many nodes successfully repair their incoming links using the *soft repair* mechanism;
- **Percentage of hard repairs:** upon disconnections, how many nodes required using the *hard repair* mechanism.

As expected the rate at which parents are lost is higher for DAGs than trees due to the larger number of parents of the former. Nonetheless DAGs

Churn conditions		**Parents lost/min.**	**Orphans/ min.**	**%Soft repairs**	**%Hard repairs**
		128 Nodes			
Churn rate:	Tree	2.3	2.3	87.0	13.0
X=3%	DAG, 2 parents	4.0	0.2	92.5	7.5
Churn rate:	Tree	3.4	3.4	79.4	20.6
X=5%	DAG, 2 parents	7.0	0.3	90.0	10.0
		512 Nodes			
Churn rate:	Tree	22.2	22.2	88.2	11.8
X=3%	DAG, 2 parents	36.8	2.3	94	6
Churn rate:	Tree	22.2	22.2	87.7	12.3
X=5%	DAG, 2 parents	32.3	1.7	94.1	5.9

Table 1: Impact of churn for a 128 and 512 node networks with active view size 4.

are much more robust with nodes being seldom fully disconnected. For instance, with a churn rate of 5% per minute, which implies half of the nodes leaving the system within the ten minutes of the experiment, only 17 nodes on an universe of 512 get disconnected (1.7 per minute * 10). Of those, all but one were able to recover using the soft repair, which simply implies activating a link to a new parent. Moreover, the time required for hard repairs, studied in the next section, is very low meaning that despite disconnections nodes are able to promptly repair connectivity with minimal effort. Finally, quick parent recovery also allows nodes to quickly recover lost messages thus ensuring that all application messages are effectively delivered. Such recovery capabilities under high churn, combined with efficient dissemination structures that are correct by design made Brisa a promising substrate for efficient and robust dissemination in the large scale.

3.4. Support for multiple trees

In this section, we analyze Brisa's support for multiple trees regarding load balancing and performance. The network size is 512 and the active view size is 8 as for the previous experiments. Unless otherwise stated, the multiple tree experiments below do not use the tree reusing strategy; the goal is instead to create multiple, independent and disjoint trees.

We first analyze Brisa's multiple trees effectiveness in balancing the dissemination effort among all nodes. Figure 12 depicts the number of trees where nodes are leaves. For instance, with 2 trees, 40% of the nodes are leaves in one tree. Results confirm our expectations that as the number of trees increases, the chance of nodes being a leaf in all trees becomes dismayingly

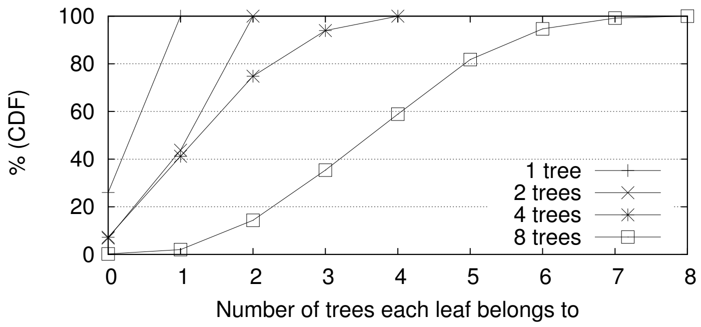

Figure 12: Distribution of the number of trees where nodes are leaves for a 512 nodes network with active view size of 8.

small, for instance for the 8 trees experiment only less than 5% of the nodes are leaves in more than 6 trees. As leaf-only nodes do not contribute to the dissemination effort, these results indicate that the use of multiple trees is essential to promote load balancing among nodes.

This is confirmed in Figure 13 which presents the number of children of each node across all trees. As is it possible to observe, the number of nodes that do not contribute to the dissemination, i.e. have zero children, diminishes dramatically with the number of trees in the system. In fact, with a single tree, almost 80% of the nodes do not upload whereas for 8 trees this value is very close to zero. These results confirm our motivation to use multiple trees as a mechanism to balance the dissemination effort among all the nodes (Section 2.8). We note that this is achieved without explicit coordination among nodes or by using more complex mechanism as in SplitStream [5]. In fact, Brisa just relies on the inherent randomness of the underlying PSS to build disjoint trees.

In the next experiment, we study the evolution of Brisa's performance with respect to the number of trees. This allows to access the impact deploying multiple trees has on the reception delay of the individual trees. The reception delay is defined as the time elapsed, at the source, since the message was published until the reception at nodes, and gives the compounded effect of: a) the routing delays inherent to the dissemination structure, and b) eventual delays due to the dissemination overhead (reception, processing and relaying of messages). Note that this measurement does not require synchronization among nodes: upon reception of a message, a node notifies the source which replies back with the time elapsed since the message was

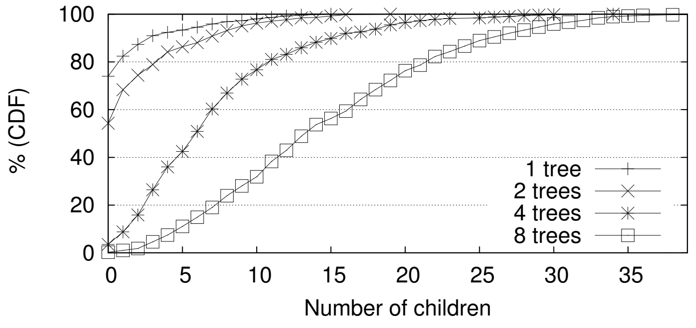

Figure 13: Distribution of the number of children across all trees for a 512 nodes network with active view size of 8.

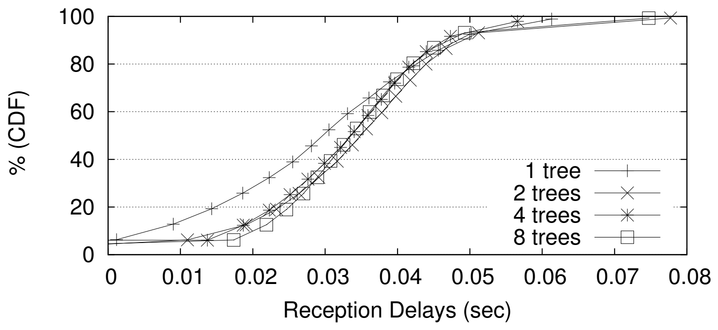

Figure 14: Reception delays per message when using multiple trees for a 512 nodes network with active view size of 8. The number of messages is 500.

published. The resulting valued is then weighted with the time elapsed since the node first sent the notification to minimize the network impact in the measurement. Results are depicted in Figure 14 and show that the reception delay is very similar regardless of the number of trees. This demonstrate that not only Brisa's multiple trees are effective in promoting load balancing among nodes but also the individual performance of multiple trees is similar to that of a single tree. We account this behavior precisely to the randomness in the tree creation process. As a matter of fact, as more trees are added, previously unused resources (leaf-only nodes' upload capacity), start being used enabling the performance of the system to remain stable despite the increased overall load.

Finally, we consider a scenario where multiple trees are used to split content and improve not only resource usage but also dissemination time.

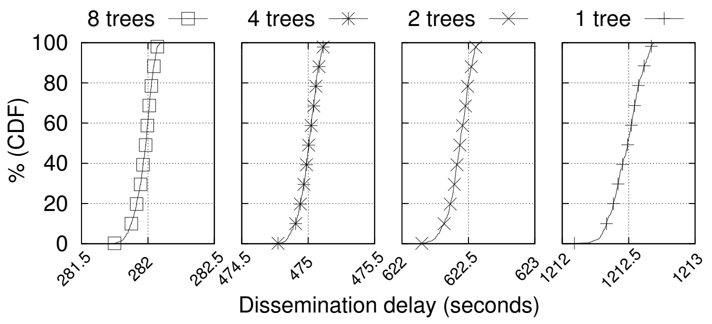

Figure 15: Dissemination delay when splitting the stream of messages across multiple trees for a 512 nodes network with active view size of 8. The number of messages is 500.

We note that this scenario is close to the one proposed in SplitStream where several disjoint trees are used to stream content.

In this experiment, we inject 500 messages on the system, evenly split across the given number of trees, and measure the dissemination delay. The dissemination delay is defined as the local time elapsed between the reception of the first message and the reception of all messages. Note that, while the reception delay measures the time elapsed since a message is published until it arrives at nodes, the dissemination delay measures the time it takes for a node to receive all messages. Results are shown in Figure 15. To improve readability, we show only the portion of the plot where the measurements lie. As expected, the dissemination delay is considerably reduced when increasing the number of trees. This is because more messages can be sent in parallel in each tree but also because the reception delay when using multiple trees does not increase. The cost is a naturally increased bandwidth usage due to parallelization. Such cost can however be observed in the distribution of children of each node, which essentially gives the upload requirement, enabling an application designer to choose the right amount of trees tolerated by the underlying physical network.

3.5. Comparison with existing approaches

In this section we compare Brisa's bandwidth usage, structure construction time, dissemination latency and parent recovery delays with several approaches. Those protocols were chosen as representatives of different points in the efficiency/robustness design spectrum. The comparison is done against a Brisa tree with a HyparView active view size of 4. In order to assess the inherent overhead of each approach, and for fairness reasons, the other approaches were implemented and evaluated in the same environment as BRISA and configured with equivalent settings. We considered the following:

SimpleGossip. This approach lies on the robustness end of the spectrum. We use Cyclon [36] as the PSS. Due to its proactive nature we use a combination of rumor mongering (push) to infect most of the nodes and anti-entropy (pull) to ensure completeness [9]. Rumor mongering follows an infect and die strategy with a fanout of $ln(N)$, where N is the system size and anti-entropy exchanges updates with a single random neighbor with a frequency that is the double of the message creation ratio.

SimpleTree. Oppositely, this approach lies on the efficiency side of the design spectrum. We consider a tree created randomly with the help of a centralized node. The only criteria for a node joining the tree is to connect to a parent that joined earlier in the past, which avoids creating a cycle in a similar manner to the one used in TAG. This parent is provided by the centralized node that randomly picks any of the previously joined nodes as a parent for a newly joined node. Dissemination is done by pushing the messages immediately through tree links thus minimizing latency.

TAG. For this approach which tries to achieve both robustness and efficiency we use TAG [26]. As BRISA, TAG maintains a tree and an epidemic-based overlay to combine the efficiency of trees and robustness of epidemics. Nodes are further organized in a linked list sorted by joining time, with nodes maintaining information about their predecessors/successors up to two hops away. New nodes traverse this list backwards until an application specific condition is met. In the traversal, nodes pick k random peers to form the gossip overlay and join the tree by choosing a suitable parent. Upon parent failures, nodes update the linked list and traverse it to find a new parent and thus restore the tree. Regarding dissemination, TAG uses a pull-based approach with nodes pulling content both from the tree and from overlay neighbors. Because TAG relies on pull we expect increased dissemination latency due to the additional roundtrips and pull period. We chose to compare BRISA against TAG due to its proximity in terms of goals and general approach (combining tree efficiency and gossip robustness) and the differences in its design choices (e.g., tree construction mechanism and a pull-based approach). We believe this choice allows a better assessment of the merits of each approach in the following evaluation scenarios.

Bandwidth usage We first focus on the bandwidth usage of each protocol by considering two metrics: *stabilization bandwidth* and *dissemination bandwidth*.

Stabilization bandwidth is the bandwidth used to bootstrap the protocol including the construction of the overlay and tree structures and is measured until stabilization. After stabilization we consider the *dissemination bandwidth* as the bandwidth associated with message disseminations and subsequent management overhead. Once the structure stabilizes, we inject messages with payload sizes from 0 to 20 KB in a network of 512 nodes. This differentiation allows us to clearly observe the overhead imposed in each phase. As SimpleGossip does not uses any structure we represent all the bandwidth consumed under *dissemination bandwidth*.

Figure 16 presents bandwidth consumption averaged over all nodes. As expected, Tag and Brisa are comparable and the actual cost is dominated by the sending of data among peers rather than the management cost of bootstrapping the dissemination structures. The smaller management overhead of SimpleTree is due to the fact that only a single communication step with the centralized node is needed while the other protocols require internode communications. The small extra bandwidth cost for Tag and Brisa when compared to SimpleTree is from the maintenance of the PSS layer and dissemination structures that are key to the performance in terms of delays and robustness as we explore later. For the smaller message sizes, SimpleGossip is comparable with both Brisa and Tag due to the absence of structure management and because Cyclon does not uses explicit fault detection mechanisms. However, this is quickly offset for larger message sizes due to the excessive number of duplicates SimpleGossip relays resulting in high bandwidth consumption.

Structure Construction Time In this experiment we measure the time necessary to bootstrap the dissemination structures both on the cluster and on PlanetLab. Due to the absence of structure of SimpleGossip and the construction simplicity of SimpleTree, they are not considered in this experiment. For Brisa we consider the time elapsed since a node sends the first deactivation message until all its inbound links except one are deactivated. In the case of Tag we use the time since a node joins the list until it settles its position on that list. Results are presented in Figure 17. It is interesting to observe that in absolute terms (note that the x scale is logarithmic) Tag is marginally faster than Brisa on the cluster but much slower on Planet-

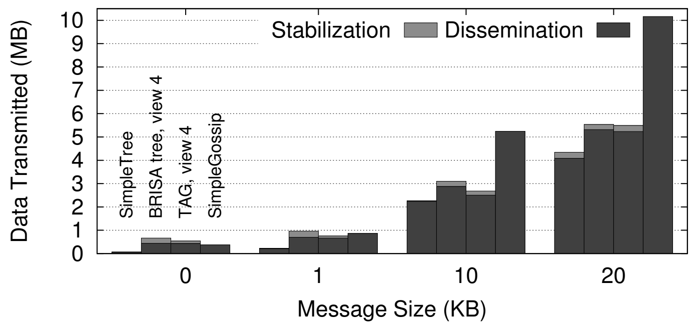

Figure 16: Bandwidth usage for a 512 nodes network.

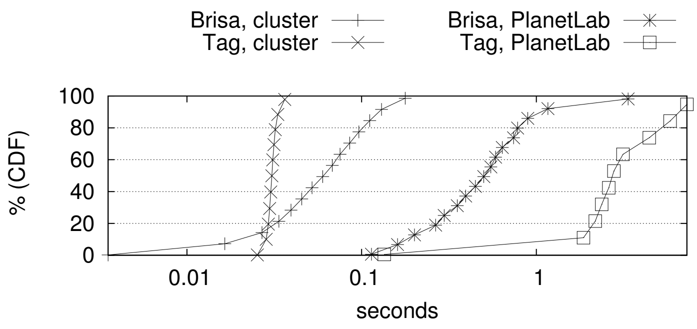

Figure 17: Construction time for 512 (on cluster) and 200 (PlanetLab) nodes.

Lab. This is because the construction mechanism happens at once in Tag by traversing the list, whereas in Brisa it is triggered by the reception of messages. As Brisa keeps the connection to its neighbors open, in the adverse environment of PlanetLab, the traversal cost of Tag (i.e. creating a connection to a node, exchanging messages, tearing it down and proceeding to the next node) easily outweighs the time Brisa needs to wait for the reception of the messages from all its neighbors.

Dissemination Latency We consider dissemination latency as the time elapsed between the reception of the first and last message among the set of all messages. When studied along with bandwidth usage, it highlights the tradeoffs of each approach. The message payload is 1 KB and the the ideal dissemination latency is 100 seconds (500 messages at 5 per second). Table 2 presents the results averaged over all nodes. As SimpleTree is very close to the ideal value we use it as a baseline of comparison for the other approaches.

Protocol	**Latency (seconds)**	**Overhead**
SimpleTree	100,025	-
Brisa	106,587	+6%
SimpleGossip	128,23	+28%
TAG	200,476	+100%

Table 2: Dissemination latency for a 512 nodes network for 500 messages of 1KB.

Latency for TAG is significantly higher than the other approaches. This is mainly because TAG uses a pull-based approach to get updates, while the others rely on push. We note however that this is a characteristic that pertains to pull approaches in general and not TAG in particular. The delays for BRISA are similar to the ones for SimpleTree, with a small variation that we account for the extra context switching and physical machines sharing on our cluster. Differences in practice are expected to be minimal with a SimpleTree, and largely in favor of BRISA when using a delay-aware selection strategy as previously illustrated in Figure 9. Somehow surprisingly, SimpleGossip performs worse than BRISA and SimpleTree. This is due to the overhead of dealing with duplicates and eventual omissions that need to be compensated by the slower anti-entropy mechanism.

Parent recovery delay Our last comparison considers the robustness of BRISA and TAG. As SimpleTree does not consider dynamic scenarios, and SimpleGossip does not maintain any structure both approaches are ignored in this experiment. We apply for both protocols the same churn conditions as described in Section 3.3, with a churn rate of 3% and focus on the parent recovery delay for hard repairs in both cases. In BRISA this corresponds to the case where no immediate replacement neighbor is available and the underlying gossip layer is used. In TAG this corresponds to the case where the linked list is broken (i.e., two consecutive simultaneous node failures) and the node needs to be re-inserted into the structure. Figure 18 depicts the results in a 128 nodes network. We note that BRISA, while yielding a similar bandwidth cost, and better dissemination delays, also outperforms TAG regarding robustness in two ways: i) the number of hard repairs almost doubles with TAG (not shown) in the same churn conditions and ii) the delay for recovery is twice as fast for BRISA. This means that both the disruption of dissemination happens less often with BRISA, and that the effect of such disruptions is less than what is experienced with TAG.

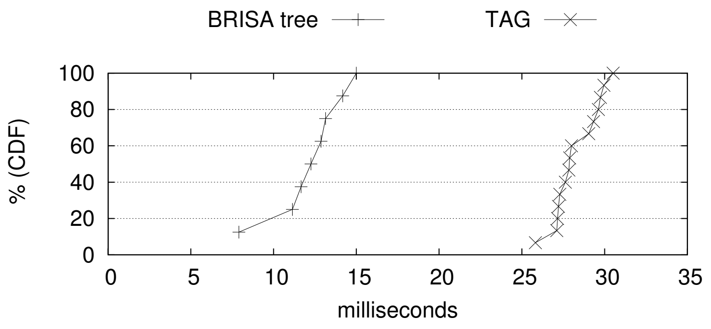

Figure 18: Parent recovery delays for a 128 nodes network with active view size 4, 3% continuous churn conditions.

4. Related Work

Existing approaches to large-scale data dissemination cover two main design domains: overlay management and application-level multicast. In the following we present existing work in this design space and compare it to our approach.

Scribe [6] is an application-level multicast layer that builds dissemination trees by aggregating reverse paths to a rendezvous node in the Pastry [32] distributed hash table (DHT). Unlike Brisa, where we assume that all nodes are interested in all messages, Scribe supports group membership management by having each node subscribe to group(s) it is interested in. Yet, the load of dissemination is shared by non-members of the groups that must act as interior (forwarding) nodes in the dissemination trees. Unlike epidemic-based dissemination, where the failure of a node has little impact on the system, Scribe's *rendezvous* nodes are single points of failure and bottlenecks in the system. Brisa also constructs a dissemination structure from an existing overlay, but can leverage the epidemic dissemination layer as a fallback for robustness. We note that group membership can be implemented in Brisa by maintaining on each node separate views for its subscribed groups, as done in the TERA publish/subscribe system [2]. These group specific views can themselves be constructed by the means of an epidemic-based clustering protocol [16].

SplitStream [5] is a high-bandwidth dissemination layer built on top of Scribe [6] and Pastry [32]. In order to balance the load of dissemination, it constructs multiple Scribe trees that are used for sending alternate pieces of a stream; nodes that participate as a leaf in one tree participate as an interior node in the other(s), thus balancing the in- and out-degrees of nodes. The same is achieved probabilistic by Brisa due to the inherent randomness of the PSS where the multiple Brisa trees are embedded.

Chunkyspread [35] also builds multiple dissemination trees, rooted at a single source node. These trees are built on top of an unstructured overlay and not on a DHT. They are used to parallelize the dissemination process by pushing different parts of the data in each tree. Cycles in the trees are avoided by using a technique derived from Bloom filters, whereas Brisa relies on simpler mechanism based on the path or the number of hops from the source. Chunkyspread trees can be constructed by taking into account latency and load metrics that can also be considered with Brisa's parent selection strategies.

In Bullet [19], a stream of data is also pushed through a tree structure. Different data blocks are intentionally disseminated to different branches of the tree, taking into account the bandwidth limits of participating nodes. Bullet complements this tree with an epidemic-like layer that allows the recovery of missed messages. This mechanism takes the form of a mesh that is used to locate peers with missing items, in a way similar to a PSS. In this sense, Bullet is based on a design choice that is opposite to ours: Brisa complements a robust dissemination layer (the PSS) with an efficient but failure-prone structure (tree/DAG), while Bullet complements a tree with an epidemic-style dissemination to support failures. Rappel [30] is another example of a dissemination service that combines a tree structure for dissemination with an epidemic-based service for optimization. In the case of Rappel, the epidemic-based layer is used to locate suitable peers based on interest-affinity and network distances, and not as a fallback mechanism for dissemination.

MON [25] relies on a mechanism similar to Brisa to construct spanning trees and DAGs on top of an unstructured overlay. The goal of MON is to manage large-scale infrastructures such as PlanetLab, by using the resulting trees/DAGs to disseminate management commands. Therefore, sessions in MON are intended to be short-lived and the protocol does not provide any support for dynamism in the population of peers. To disseminate data, MON relies on a pull strategy, where nodes can download content simultaneously from multiple parents, if available. This approach eliminates duplicates, as it is the receiver that decides which pieces to receive. However it requires nodes to maintain knowledge of the data blocks/messages present at each parent.

The work presented in [38] stems from an observation similar to ours that even though epidemic-based dissemination is attractive due to robustness, achieving completeness requires large fanouts resulting in high overhead. The authors thus propose a hybrid approach that uses an epidemic-based dissemination with fanouts lower enough to infect most of the population, and ensures completeness by relying on a ring structure that encompasses all nodes. Epidemics are used for the bulk dissemination of data, still resulting in many duplicates, as opposed to Brisa, where most of the dissemination happens on the dissemination structure with a controlled number of duplicates. Similarly, in [23, 24] a Chord-like ring overlay is combined with a push mechanism to disseminate messages over a spanning tree optimized for minimal latency. Brisa instead builds on top of an unstructured overlay, and it offers a wider set of options for the tree construction.

In [11] the authors propose an alternative approach to tree repair based on proactive principles. Each node computes alternative parents for its children that can be used upon failures. This minimizes disruptions as nodes known beforehand the new parent they need to connect to. Further it can cope to some extent with multiple concurrent failures and strictly control node degrees, a major goal of the authors. Due to this restriction, tree shape tends to degenerate to a chain overtime penalizing end-to-end delay. Brisa uses a notion similar to the alternative parents without however having the tree degenerate into a chain. This is because [11] only considers potential parents in the failed node subtree while Brisa can consider any node as long as it passes the cycle detection mechanism.

GoCast [33] builds a dissemination tree embedded on an epidemic-based overlay that takes into account network proximity to improve end-to-end latency. The tree is built using a traditional Distance Vector Multicast Routing Protocol (DVMRP) and used to push messages as in Brisa. Message identifiers are advertised through the overlay links as in PlumTree [20] and used to recover missing messages due to tree disruptions that, contrary to Brisa, imposes additional network overhead. Most strikingly this recovery information is not used to repair the tree, which relies solely on DVMRP and thus presents scalability problems due to the overhead of periodic floods to rebuild the tree. Furthermore, Brisa is able to adjust to different performance criteria but could nonetheless take advantage of the network-proximity offered by Gocast's overlay. TAG, the protocol we use in the direct comparison with Brisa also falls into this class due to the use of a tree and an epidemic-based overlay. More details can be found in Section 3.5. PlumTree [20] also relies on the detection of duplicates and subsequent deactivation of links to build an embedded spanning tree on an underlying unstructured overlay. However, inactive links are still used in a "lazy push" approach, by announcing the message identifier instead of the full payload. These announcements are used to repair the tree: when an announcement for an unknown message is received, the protocol starts a timer. If the timer expires before the reception of the payload the tree repair mechanism is triggered. This approach is highly sensitive to variations in network latency, which lead to unnecessary message recoveries as observed in [12]. Brisa does not separate the dissemination of the header and payload, the dissemination is deterministic, and faults are detected thanks to the underlying PSS layer, which avoids sending periodic probe messages at the level of the dissemination layer. Further, the generic construction mechanism can build trees and DAGS according to different criteria, which is not possible in PlumTree. Due to the use of message advertisements to manage faults both PlumTree and Gocast fall in an undesirable tradeoff: either advertisements are sent sparingly to conserve bandwidth with an impact on recovery time, or advertisements are eagerly sent imposing a constant management overhead in the system.

Thicket [12] uses the same principles of PlumTree to build multiple dissemination trees on top of an unstructured overlay. The goal is to provide similar functionality to SplitStream by balancing the number of trees where a node is interior and also by splitting the content among trees to improve fault-tolerance. The mechanism used to build trees imposes several constrains that do not ensure the resulting tree is connected by design. This is addressed with a tree repair mechanism based on missing messages, as in PlumTree, that requires periodic exchanges of received messages among neighbors which is also used to handle joins and leaves. The support for multiple trees in Thicket is based on the premise of load balancing and fault-tolerance by leveraging on network coding techniques. The cost however is a linear growth in the number of links with respect to the number of trees, which poses scalability concerns. In contrast, Brisa builds connected trees by design, despite controlled fanouts, and deals with joins and failures with a simple and lightweight mechanism that is triggered only when failures happen. Multiple trees are a natural extension of the system and therefore do not require additional maintenance mechanisms.

5. Conclusions

In this paper, we presented the design and evaluation of Brisa, a data dissemination system that combines the robustness of epidemic-based protocols and the efficiency of structured overlays. Brisa automatically emerges efficient dissemination structures from the flooding-based distribution of the first message in a stream. The construction of efficient dissemination structures exploits the path diversity that naturally exists in epidemic- and flooding-based dissemination, while avoiding the high level of duplicate reception these mechanisms typically yield. Robustness comes from the ability of the underlying epidemic layer to rapidly provide replacement nodes upon failures, and by acting as a dissemination fallback. Therefore, Brisa bridges the gap between robust but costly epidemic or gossip-based dissemination and efficient but failure prone structured approaches. We evaluated Brisa with a prototype deployed on a cluster and on PlanetLab. The experiments and comparisons to related work confirmed Brisa as a robust and efficient system for data-intensive applications.

Acknowledgments

This work was supported in part by the Swiss National Foundation under agreement number 200021-127271/1 and funded by ERDF - European Regional Development Fund through the COMPETE Programme (Operational Programme for Competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within grant SFRH/BD/62380/2009 and PTDC/EIA-CCO/115570/2009.

References

[1] Planetlab, http://www.planet-lab.org, Last accessed: September, 2012.

[2] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, S. Tucci-Piergiovanni, Tera: topic-based event routing for peer-to-peer architectures, in: Proceedings of the International Conference on Distributed Event-based Systems, DEBS, ACM, New York, NY, USA, 2007, pp. 2–13.

[3] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, Y. Minsky, Bimodal Multicast, ACM Transactions on Computer Systems 17 (1999) 41–88.

[4] B. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM 13 (1970) 422–426.

[5] M. Castro, P. Druschel, A.M. Kermarrec, A. Nandi, A. Rowstron, A. Singh, Splitstream: high-bandwidth multicast in cooperative environments, in: Proceedings of the 19th ACM symposium on Operating systems principles, SOSP, ACM, New York, NY, USA, 2003, pp. 298–313.

[6] M. Castro, P. Druschel, A.M. Kermarrec, A. Rowstron, Scribe: A large-scale and decentralized application-level multicast infrastructure, IEEE Journal on Selected Areas in Communications 20 (2002) 1489–1499.

[7] Y. Chu, S. Rao, S. Seshan, H. Zhang, A case for end system multicast, IEEE Journal on Selected Areas in Communications 20 (2002) 1456–1471.

[8] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, Dynamo: amazon's highly available key-value store, SIGOPS Operating Systems Review 41 (2007) 205–220.

[9] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, D. Terry, Epidemic algorithms for replicated database maintenance, in: Proceedings of the 6th ACM Symposium on Principles of distributed computing, PODC, ACM, New York, NY, USA, 1987, pp. 1–12.

[10] P. Eugster, R. Guerraoui, S. Handurukande, P. Kouznetsov, A.M. Kermarrec, Lightweight probabilistic broadcast, ACM Transactions on Computer Systems 21 (2003) 341–374.

[11] Z. Fei, M. Yang, A proactive tree recovery mechanism for resilient overlay multicast, IEEE/ACM Transactions on Networking 15 (2007) 173–186.

[12] M. Ferreira, J. Leitao, L. Rodrigues, Thicket: A Protocol for Building and Maintaining Multiple Trees in a P2P Overlay, in: Proceedings of the 29th IEEE International Symposium on Reliable Distributed Systems, SRDS, IEEE Computer, New Delhi, India, 2010, pp. 293–302.

[13] D. Frey, R. Guerraoui, A.M. Kermarrec, M. Monod, V. Quema, Stretching gossip with live streaming, in: Proceedings of the 39th IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, IEEE Computer Society, Budapest, Hungary, 2009, pp. 259–264.

[14] A. Ganesh, A.M. Kermarrec, L. Massoulié, Scamp: Peer-to-Peer Lightweight Membership Service for Large-Scale Group Communication, in: Networked Group Communication, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2001, pp. 44–55.

[15] J. Gantz, The Diverse and Exploding Digital Universe, Technical Report, IDC White Paper - sponsored by EMC, 2008.

[16] M. Jelasity, A. Montresor, O. Babaoglu, T-man: Gossip-based fast overlay topology construction, Computer Networks: The International Journal of Computer and Telecommunications Networking 53 (2009) 2321–2339.

[17] M. Jelasity, S. Voulgaris, R. Guerraoui, A.M. Kermarrec, M. van Steen, Gossip-based peer sampling, ACM Transactions on Computer Systems 25 (2007).

[18] B. Kaldehofe, Buffer management in probabilistic peer-to-peer communication protocols, in: Proceedings of the 22nd IEEE International Symposium on Reliable Distributed Systems, SRDS, IEEE Computer, Florence, Italy, 2003, pp. 76–85.

[19] D. Kostic, A. Rodriguez, J. Albrecht, A. Vahdat, Bullet: High bandwidth data dissemination using an overlay mesh, in: Proceedings of the nineteenth ACM symposium on Operating systems principles, SOSP, ACM, New York, NY, USA, 2003, pp. 282–297.

[20] J. Leitão, J. Pereira, L. Rodrigues, Epidemic Broadcast Trees, in: Proceedings of the 22nd IEEE International Symposium on Reliable Distributed Systems, SRDS, IEEE Computer, Beijing, China, 2007, pp. 301–310.

[21] J. Leitão, J. Pereira, L. Rodrigues, HyParView: A membership protocol for reliable gossip-based broadcast, in: Proceedings of the 37th IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, IEEE Computer Society, Edinburgh, Scotland, 2007, pp. 419–429.

[22] L. Leonini, E. Rivière, P. Felber, SPLAY: Distributed systems evaluation made simple (or how to turn ideas into live systems in a breeze), in: Proceedings of the 6th USENIX symposium on Networked systems design and implementation, NSDI, Usenix Association, Berkely, CA, USA, 2009, pp. 185–198.

[23] Z. Li, G. Xie, K. Hwang, Z. Li, Churn-resilient protocol for massive data dissemination in p2p networks, IEEE Transactions on Parallel and Distributed Systems 22 (2011) 1342–1349.

[24] Z. Li, G. Xie, Z. Li, Towards reliable and efficient data dissemination in heterogeneous peer-to-peer systems, in: Proceedings of the 22th IEEE International Parallel and Distributed Processing Symposium, IPDPS, IEEE Computer Society, Miami, FL, USA, 2008, pp. 1–12.

[25] J. Liang, S. Ko, I. Gupta, K. Nahrstedt, MON : On-demand Overlays for Distributed System Management, in: Proceedings of the 2nd conference on Real, Large Distributed Systems, WORLDS, Usenix Association, Berkely, CA, USA, 2005, pp. 13–18.

[26] J. Liu, M. Zhou, Tree-assisted gossiping for overlay video distribution, Multimedia Tools and Applications 29 (2006) 211–232.

[27] M. Matos, V. Schiavoni, P. Felber, R. Oliveira, E. Riviere, Brisa: Combining efficiency and reliability in epidemic data dissemination, in: Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium, IPDPS, IEEE Computer Society, Shangai, China, 2012, pp. 983–994.

[28] R. Melamed, I. Keidar, Araneola: A scalable reliable multicast system for dynamic environments, Journal of Parallel and Distributed Computing 68 (2008) 1539–1560.

[29] A. Montresor, M. Jelasity, O. Babaoglu, Chord on demand, in: Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing, P2P, IEEE Computer Society, Washington, DC, USA, 2005, pp. 87–94.

[30] J.A. Patel, E. Rivière, I. Gupta, A.M. Kermarrec, Rappel: Exploiting interest and network locality to improve fairness in publish-subscribe

systems, Computer Networks: The International Journal of Computer and Telecommunications Networking 53 (2009) 2304–2320.

[31] R.V. Renesse, Y. Minsky, M. Hayden, A gossip-style failure detection service, in: Proceedings of the ACM/IFIP/USENIX International Conference on Middleware, Middleware, Springer-Verlag, New York, Inc. New York, NY, USA, 2007, pp. 389–409.

[32] A. Rowstron, P. Druschel, Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems, in: Middleware, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2001, pp. 329–350.

[33] C. Tang, R.N. Chang, C. Ward, Gocast: Gossip-enhanced overlay multicast for fast and dependable group communication, in: Proceedings of the 35th IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, IEEE Computer Society, Washington, DC, USA, 2005, pp. 140–149.

[34] Twitter Engineering, Murder: Fast datacenter code deploys using BitTorrent, `http://t.co/uo5rEN4`, Last accessed: September, 2012.

[35] V. Venkataraman, K. Yoshida, P. Francis, Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast, in: Proceedings of the 14th IEEE International Conference on Network Protocols, ICNP, IEEE Computer Society, 2006, pp. 2–11.

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

[37] S. Voulgaris, M. Jelasity, M.V. Steen, A Robust and Scalable Peer-to-Peer Gossiping Protocol, in: Agents and Peer-to-Peer Computing, volume 2872 of *Lecture Notes in Computer Science*, Springer-Verlag Berlin, Heidelberg, 2005, pp. 47–58.

[38] S. Voulgaris, M. van Steen, Hybrid dissemination: Adding determinism to probabilistic multicasting in large-scale p2p systems, in: Proceedings of the ACM/IFIP/USENIX International Conference on Middleware, Middleware, Springer-Verlag, New York, Inc. New York, NY, USA, 2007, pp. 389–409.

[39] S. Zhuang, B. Zhao, A. Joseph, R. Katz, J. Kubiatowicz, Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination, in: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video, NOSSDAV, ACM, New York, NY, USA, 2001, pp. 11–20.