HyParView: a membership protocol for reliable gossip-based broadcast*

João Leitão
University of Lisbon
jleitao@lasige.di.fc.ul.pt

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

Luís Rodrigues
University of Lisbon
ler@di.fc.ul.pt

May 2007

Abstract

Gossip, or epidemic, protocols have emerged as a powerful strategy to implement highly scalable and resilient reliable broadcast primitives. Due to scalability reasons, each participant in a gossip protocol maintains a partial view of the system. The reliability of the gossip protocol depends upon some critical properties of these views, such as degree distribution and clustering coefficient.

Several algorithms have been proposed to maintain partial views for gossip protocols. In this paper, we show that under a high number of faults, these algorithms take a long time to restore the desirable view properties. To address this problem, we present HyParView, a new membership protocol to support gossip-based broadcast that ensures high levels of reliability even in the presence of high rates of node failure. The HyParView protocol is based on a novel approach that relies in the use of two distinct partial views, which are maintained with different goals by different strategies.

1 Introduction

Gossip, or epidemic, protocols have emerged as a powerful strategy to implement highly scalable and resilient reliable broadcast primitives [8, 3, 6, 1]. In a gossip protocol, when a node wants to broadcast a message, it selects t nodes from the system at random (this is a configuration parameter called *fanout*) and sends the message to them; upon receiving a message for the first time, each node repeats this procedure [8]. Gossip protocols are an interesting approach because they are highly resilient (these protocols have an intrinsic level of redundancy that allows them to mask node and network failures) and distribute the load among all nodes in the system.

As described above, the protocol requires each node to know the entire system membership, in order to select the target nodes for each gossip round. Clearly, this solution is not scalable, not only due to the large number of nodes that may constitute the view but also due to the cost of maintaining the complete membership up-to-date. To overcome this problem, several gossip protocols rely on *partial views* [11, 2, 3] instead of the complete membership information. A partial view is a small subset of the entire system membership. When a node performs a gossip step it selects t nodes at random from its partial view. The aim of a membership service (also called a peer sampling service [7]) is to maintain these partial views satisfying a number of good properties. Intuitively, selecting gossip peers from the partial view should provide the same resiliency as selecting them at random from the entire membership.

Unfortunately, if a node only has a partial view of the system, it becomes more vulnerable to the effect of node failures. In particular, if a large number of nodes fail, the partial view of each

*This work was partially supported by project "P-SON: Probabilistically Structured Overlay Networks" (POS_C/EIA/60941/2004). Parts of this report have been published in the Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Edinburgh, UK, June, 2007.

node may be severely damaged, and the network may become disconnected. Also, the membership service may take several membership rounds to restore the target properties of partial views, with a negative impact on the reliability of all messages disseminated meanwhile.

This paper proposes a novel approach to implement gossip-based broadcast protocols and describes a membership protocol that allows this approach to be used successfully. The key ideas of the paper are as follows:

- We propose a gossip strategy that is based on the use of a reliable transport protocol, such as TCP, to gossip between peers. In this way, the gossip does not need to be configured to mask network omissions.
- Each node maintains a small symmetric *active view* the size of the fanout+1. Note that the fanout may be selected assuming that the links do not omit messages; the strategy allows to use smaller fanouts than protocols that use unreliable transport to support gossip exchanges. Broadcast is performed by flooding the graph defined by the active views. While this graph is generated at random (using our membership service), gossip is deterministic as long as the graph remains unchanged.
- TCP is also used as a failure detector, and since all members of the active view are tested at each gossip step, failure of nodes in the active view are detected quickly.
- Each node maintains a *passive view* of backup nodes that can be promoted to the active view, when one of the nodes in the active view fails (*i.e.* disconnects, crash or blocks).
- A membership protocol is in charge of maintaining the passive view and selecting which members of the passive view should be promoted to the active view. In fact, two partial views are maintained by the protocol.

We named our protocol *Hy*brid *Par*tial *View* membership protocol, or simply *HyParView*[1]. We show that our approach not only allows the use of a smaller fanout (therefore, it is less resource consuming than other approaches) but also offers a strong resilience to node failures, even in the presence of extremely large numbers of crashes in the system. As we will show, our protocol recovers from percentages of node failures as high as 90% in as few as 4 membership rounds. This is significantly better than previous approaches. High resiliency to node failures is important to face occurrences, such as natural disasters (*e.g.* earthquakes) or informatic worms and virus that may take down all machines running a specific OS version (that may represent a significant portion of the system). For instance, a worm could affect 10.000.000 nodes in the space of days [13]; also, these worms can spread in a first phase and take down nodes simultaneously at a predetermined time.

The rest of the paper is structured as follows. Section 2 offers an overview of related work. A motivation for our work, namely an analysis of the impact of high percentage of node failures in protocols that use partial views is given in Section 3. HyParView is introduced in Section 4 and its performance evaluated in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

We start this section by defining more precisely the notion of partial view. Then we introduce the two main approaches to maintain partial views. Later, we enumerate the main properties that partial views must own. Finally, we give some examples of concrete membership protocols.

2.1 Partial Views

A *partial view* is a set of node identifiers maintained locally at each node that is a small subset of the identifiers of all nodes in the system(ideally, of logarithmic size with the number of processes

[1]The protocol is said to be hybrid because it combines both strategies described in section 2.2

in the system). Typically, an identifier is a tuple $(ip, port)$ that allows a node to be reached. A membership protocol is in charge of initializing and maintaining the partial views at each node in face of dynamic changes in the system. For instance, when a new node joins the system, its identifier should be added to the partial view of (some) other nodes and it has to create its own partial view, including identifiers of nodes already in the system. Also, if a node fails or leaves the system, its identifier should be removed from all partial views as soon as possible.

Partial views establish *neighboring* associations among nodes. Therefore, partial views define an overlay network, in other words, partial views establish an directed graph that captures the neighbor relation between all nodes executing the protocol. In this graph nodes are represented by a vertex while a neighbor relation is represented by an arc from the node who contains the target node in his partial view.

2.2 Maintaining the Partial View

There are two main strategies that can be used to maintain partial views, namely:

Reactive strategy: In this type of approach, a partial view only changes in response to some external event that affects the overlay (e.g. a node joining or leaving). In stable conditions, partial view remains unaltered. Scamp [5, 4] is an example of such an algorithm[2].

Cyclic strategy: In this type of approach, a partial view is updated every ΔT time units, as a result of some periodic process that usually involves the exchange of information with one or more neighbors. Therefore, a partial view may be updated even if the global system membership is stable. Cyclon is an example of such an algorithm [15, 14].

Reactive strategies rely on some failure detection mechanism to trigger the update of partial views when a node leaves the system. If the failure detection mechanism is fast and accurate, reactive mechanisms can provide faster response to failures than cyclic approaches.

2.3 Partial View Properties

In order to support fast message dissemination and high level of fault tolerance to node failures, partial views must own a number of important properties. These properties are intrinsically related with graph properties of the overlay defined by the partial view of all nodes. We list some of the most important properties here:

Connectivity. The overlay defined by the partial views should be connected. If this property is not meet, isolated nodes will not receive broadcast messages.

Degree Distribution. In an undirected graph, the degree of a node is the number of edges leaving that node. Given that partial views define a directed graph, we distinguish the *in-degree* from the *out-degree* of a node. The in-degree of a node n is the number of nodes that have n's identifier in their partial view; it provides a measure of the reachability of a node in the overlay. The out-degree of a node n is the number of nodes in n's partial view; it is a measure of the node contribution to the membership protocol and consequently a measure of the importance of that node to maintain the overlay. If the probability of failure is uniformly distributed in the node space, for improved fault-tolerance both the in-degree and out-degree should be evenly distributed across all nodes.

Average Path Length. A path between two nodes in the overlay is the set of edges that a message has to cross to move from one node to the other. The average path length is the average of all shortest paths between all pair of nodes in the overlay. This property is closely related to the overlay diameter. To ensure the efficiency of the overlay for information dissemination, it is essential to enforce low values of the average path length, as this value is related to the time a message will take to reach all nodes.

[2]To be precise, Scamp is not purely reactive as it includes a *lease* mechanisms that forces nodes to periodically rejoin.

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 neighbor relations across the neighbors of a node, having it's value between 0 and 1. The clustering coefficient of a graph is the average of clustering coefficients across all nodes. This property has a high impact on the number of redundant messages received by nodes when disseminating data, where a high value to clustering coefficient will produce more redundant messages. It also has an impact in the fault-tolerant properties of the graph, given that areas of the graph that exhibit high values of clustering will more easily be isolated from the rest of the graph.

Accuracy. We define accuracy of a node as the number of neighbors of that node that have not failed divided by the total number of neighbors of that node. The accuracy of a graph is the average of the accuracy of all correct nodes. Accuracy has high impact in the overall reliability of any dissemination protocol using a underlying membership protocol to select its gossip targets. If the graph accuracy values are low, the number of failed nodes selected as gossip targets will be higher, and higher fanouts must be used to mask these failures.

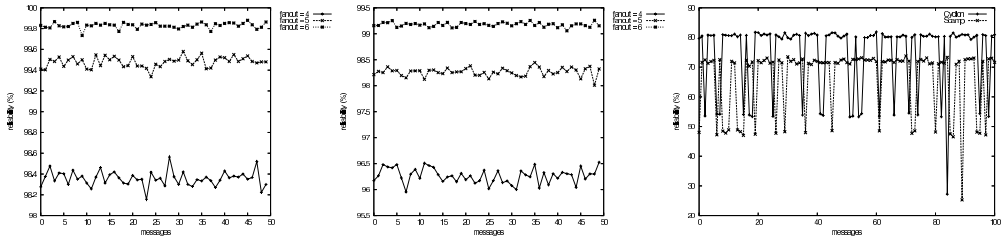
2.4 Membership and Gossip Protocols

Scamp [5, 4] is a reactive membership protocol that maintains two separate views, a *PartialView* from which nodes select their targets to gossip messages, and a *InView* with nodes from which they receive gossip messages. One interesting aspect of this protocol is that the *PartialView* does not have a fixed size, it grows to values that are distributed around $\log n$, where n is the total number of nodes executing the protocol, and without n being known by any node. The main mechanism to update the *PartialView* is a subscription protocol, executed when new processes jojn the system. However, in order to recover from isolation, nodes periodically send heartbeat messages to all nodes present in their *PartialView*. If a node does not receive a heartbeat for a long time, it assumes that it has become isolated and rejoins the overlay.

Cyclon [15] is a cyclic membership protocol where nodes maintain a fixed length *partial view*. This protocol relies in a operation that is executed periodically every ΔT by all nodes which is called *shuffle*. Basically, in a shuffle operation, a node selects the "oldest" node in its partial view and perform an exchange with that node. In the exchange, the node provides to its peer a sample of its partial view and, symmetrically, collects a sample of its peer's partial view. The join operation is based on fixed length random walks on the overlay. The join process ensures that, if there are no message losses or node failures, the in-degree of all nodes will remain unchanged.

NeEM [10], Network Friendly Epidemic Multicast, is a gossip protocol that relies on the use of TCP to disseminate information across the overlay. In NeEM, the use of TCP is motivated by the desire to eliminate correlated message losses due to network congestion. The authors show that better gossip reliability can be achieved by leveraging on the flow control mechanisms of TCP. In this paper we rely on TCP to mask network omissions and to detect failures. Therefore, our work is complementary of NeEM.

CREW [2] is a gossip protocol for flash dissemination, *i.e.* fast simultaneous download of files by a large number of destinations using a combination of pull and push gossip. It uses TCP connections to implicitly estimate available bandwidth thus optimizing the fanout of the gossip procedure. The emphasis of CREW is on optimizing latency, mainly by improving concurrent pulling from multiple sources. A key feature is to maintain a cache of open connections to peers discovered using a random walk protocol, to avoid the latency of opening a TCP connection when a new peer is required. The same optimization can be applied in HyParView, by pre-opening connections to some of the members of the passive view. CREW does not however explicitly manage such cache to improve the overlay, namely, regarding resilience when a large number of nodes fail.



(a) Cyclon (b) Scamp (c) 50% node failure

Figure 1: Fanout x Reliability and Effect of Failures

2.5 Reliability

We define reliability as the percentage of active nodes that deliver a gossip broadcast. A reliability of 100% means that the gossip message reached all active nodes, or in other words, the message resulted in an atomic broadcast [8].

3 Motivation

Our work is motivated by the following two observations:

- The fanout of a gossip protocol is constrained by the target reliability level and the desired fault tolerance of the protocol. When partial views are used, the quality of these views has also an impact on the fanout required to achieve high reliability. By using "better" views (according to the metric of Section 2) and a reliable transport such as TCP it should be possible to use smaller fanouts, and thus more cost-effective gossip protocols.
- High failure rates may have a strong impact on the quality of partial views. Even if the membership protocol has healing properties, the reliability of message broadcasts after heavy failures may be seriously affected. Therefore, gossip would strongly benefit from membership protocols with fast healing properties, which can be achieved by also using TCP as a failure detector.

In the following paragraphs, we show some figures that illustrate these facts.

3.1 On the Fanout Value

The first two plots in Figure 1 show simulation results where we depict the reliability of 50 messages sent by a gossip protocol that uses Cyclon or Scamp as the underlying membership protocol. The simulations were run with a network of 10.000 nodes (we describe our simulation model in detail later in Section 5). As it can be observed, in order to obtain reliability figures above 99%, Cyclon requires a fanout of 5; it requires a fanout of 6 to achieve values near $99,9\%$. Scamp requires a fanout of 6 to reach values of reliability above the 99%. In this run, with a fanout of 6, there are potentially 20.000 extra messages exchanged than in a scenario that uses a fanout of 4 ($\log(10.000) = 4$). More than 99% of these 20.000 extra messages are redundant, which means that less than 200 of these messages will, in fact, contribute to actual deliveries. We will later show that our approach allows to achieve higher reliability with a fanout value close to $\log(n)$.

3.2 Effect of Failures

The last plot in Figure 1 depicts reliability figures for the 100 messages exchanged after heavy node failure. In this scenario, we have failed 50% of the system nodes, and measured the effect on

a network of 10.000 nodes using Cyclon and Scamp as the membership protocol. These messages are sent before Cyclon has the opportunity to execute a cycle of shuffle (note that the Cyclon period is typically large enough to exchange several thousands of messages), or before the lease time of Scamp expires. As it can be observed, reliability is lost (as no message is ever delivered to more than 85% of the nodes, and many messages are delivered to much smaller numbers of nodes). This long period of unstable behavior may be unacceptable in applications exhibiting high reliability requirements and high throughput.

4 The HyParView Protocol

4.1 Overview

The HyParView protocol maintains two distinct views at each node: a small *active view*, of size $log(n) + c$, and a larger *passive view*, of size $k(log(n) + c)$. In all our experiments we have used $c = 1$ and $k = 6$. These parameters imply that, for a network of 10.000 nodes, the active views are of size 5 and the passive views of size 30.

The active views of all nodes create an overlay that is used for message dissemination. Links in the overlay are symmetric. This means that if node q is in the active view of node p then node p is also in the active view of node q. As we have stated before, our architecture assumes that nodes use a reliable transport protocol to broadcast messages in the overlay. In practice, this means that each node keeps an open TCP connection to every other node in its active view. This is feasible because the active view is very small. When a node receives a message for the first time, it broadcasts the message to all nodes of its active view (except, obviously, to the node that has sent the message). Therefore, the gossip target selection is deterministic in the overlay. However, the overlay itself is created at random, using the gossip membership protocol described in this section.

A reactive strategy is used to maintain the active view. Nodes can be added to the active view when they join the system. Also, nodes are removed from the active view when they fail. The reader should notice that each node tests its entire active view every time it forwards a message. Therefore, the entire broadcast overlay is implicitly tested at every broadcast, which allows a very fast failure detection.

In addition to the active view, each node maintains a larger passive view. The passive view is not used for message dissemination. Instead, the goal of the passive view is to maintain a list of nodes that can be used to replace failed members of the active view. The passive view is maintained using a cyclic strategy. Periodically, each node performs a shuffle operation with one of its neighbors in order to update its passive view.

One interesting aspect of our shuffle mechanism is that the identifiers that are exchanged in a shuffle operation are not only from the passive view: a node also sends its own identifier and some nodes collected from its active view to its neighbor. This increases the probability of having nodes that are active in the passive views and ensures that failed nodes are eventually expunged from all passive views.

4.2 Join Mechanism

Algorithm 1 depicts the pseudo-code for the join operation. When a node wishes to join the overlay, it must know another node that already belongs to the overlay. We call that node the *contact node*. There are several ways to learn about the contact node, for instance, members of the overlay could be announced through a set of well known servers.

In order to join the overlay, a new node n establishes a TCP connection to the contact node c and sends to c a JOIN request. A node that receives a join request will start by adding the new node to its active view, even if it has to drop a random node from it. In this case a DISCONNECT notification is sent to the node that has been dropped from the active view. The effect of the DISCONNECT message is described later in the section.

Algorithm 1: Membership Operations

```
upon init do
    Send(JOIN, contactNode, myself);

upon Receive(JOIN, newNode) do
    if isfull(activeView) then
        trigger dropRandomElementFromActiveView
    activeView ⟵ activeView ∪ newNode
        foreach n ∈ activeView and n ≠ newNode do
            Send(FORWARDJOIN, n, newNode, ARWL, myself)

upon Receive(FORWARDJOIN, newNode, timeToLive, sender) do
    if timeToLive== 0||#activeView== 0 then
        trigger addNodeActiveView(newNode)
    else
        if timeToLive==PRWL then
            trigger addNodePassiveView(newNode)
        n ⟵ n ∈ activeView and n ≠ sender
        Send(FORWARDJOIN, n, newNode, timeToLive-1, myself)

upon dropRandomElementFromActiveView do
    n ⟵ n ∈ activeView
    Send(DISCONNECT, n, myself)
    activeView ⟵ activeView \{n}
    passiveView ⟵ passiveView ∪{n}

upon addNodeActiveView(node) do
    if node ≠ myself and node ∉ activeView then
        if isfull(activeView) then
            trigger dropRandomElementFromActiveView
        activeView ⟵ activeView ∪ node

upon addNodePassiveView(node) do
    if node ≠ myself and node ∉ activeView and node ∉ passiveView then
        if isfull(passiveView) then
            n ⟵ n ∈ passiveView
            passiveView ⟵ passiveView \{n}
        passiveView ⟵ passiveView ∪ node

upon Receive(DISCONNECT, peer) do
    if peer ∈ activeView then
        activeView ⟵ activeView \ {peer}
        addNodePassiveView(peer)
```

The contact node c will then send to all other nodes in its active view a FORWARDJOIN request containing the new node identifier. The FORWARDJOIN request will be propagated in the overlay using a random walk. Associated to the join procedure, there are two configuration parameters, named *Active Random Walk Length* (ARWL), that specifies the maximum number of hops a FORWARDJOIN request is propagated, and *Passive Random Walk Length* (PRWL), that specifies at which point in the walk the node is inserted in a passive view. To use these parameters, the FORWARDJOIN request carries a "time to live" field that is initially set to ARWL and decreased at every hop.

When a node p receives a FORWARDJOIN, it performs the following steps in sequence: *i)* If the time to live is equal to zero *or* if the number of nodes in p's active view is equal to one, it will add the new node to its active view. This step is performed even if a random node must be dropped from the active view. In the later case, the node being ejected from the active view receives a DISCONNECT notification. *ii)* If the time to live is equal to PRWL, p will insert the new node into its passive view. *iii)* The time to live field is decremented. *iv)* If, at this point, n has not been inserted in p's active view, p will forward the request to a random node in its active view (different from the one from which the request was received).

4.3 Active View Management

The active view is managed using a reactive strategy. When a node p suspects that one of the nodes present in its active view has failed (by either disconnecting or blocking), it selects a random

node q from its passive view and attempts to establish a TCP connection with q. If the connection fails to establish, node q is considered failed and removed from p's passive view; another node q' is selected at random and a new attempt is made. The procedure is repeated until a connection is established with success.

When the connection is established with success, p sends to q a NEIGHBOR request with its own identifier and a priority level. The priority level of the request may take two values, depending on the number of nodes present in the active view of p: if p has no elements in its active view the priority is *high*; the priority is *low* otherwise.

A node q that receives a high priority NEIGHBOR request will always accept the request, even if it has to drop a random member from its active view (again, the member that is dropped will receive a DISCONNECT notification). If a node q receives a low priority NEIGHBOR request, it will only accept the request if it has a free slot in its active view, otherwise it will refuse the request.

If the node q accepts the NEIGHBOR request, p will remove q's identifier from its passive view and add it to the active view. If q rejects the NEIGHBOR request, the initiator will select another node from its passive view and repeat the whole procedure (without removing q from its passive view).

4.4 Passive View Management

The passive view is maintained using a cyclic strategy. Periodically, each node perform a shuffle operation with one of its peers at random. The purpose of the shuffle operation is to update the passive views of the nodes involved in the exchange. The node p that initiates the exchange creates an exchange list with the following contents: p's own identifier, k_a nodes from its active view and k_p nodes from its passive view (where k_a and k_p are protocol parameters). It then sends the list in a SHUFFLE request to a random neighbor of its active view. SHUFFLE requests are propagated using a random walk and have an associated "time to live", just like the FORWARDJOIN requests.

A node q that receives a SHUFFLE request will first decrease its time to live. If the time to live of the message is greater than zero and the number of nodes in q's active view is greater than 1, the node will select a random node from its active view, different from the one he received this shuffle message from, and simply forwards the SHUFFLE request. Otherwise, node q accepts the SHUFFLE request and send back, using a temporary TCP connection, a SHUFFLEREPLY message that includes a number of nodes selected at random from q's passive view equal to the number of nodes received in the SHUFFLE request.

Then, both nodes integrate the elements they received in the SHUFFLE/SHUFFLEREPLY message into their passive views (naturally, they exclude their own identifier and nodes that are part of the active or passive views). Because the passive view has a fixed length, it might get full; in that case, some identifiers will have to be removed in order to free space to include the new ones. A node will first attempt to remove identifiers sent to the peer. If no such identifiers remain in the passive view, it will remove identifiers at random.

4.5 View Update Procedures

Algorithm 1 depicts some basic manipulation primitives used to change contents of the passive and active views. The important aspect to retain from these primitives is that nodes can pass from the passive view to the active view in order to fill the active view (*e.g.* in reaction to node failures). Nodes can be moved from the active view to the passive view whenever a correct node has to be removed from the active view. Note that since links are symmetric, by removing a node p from the active view of node q, q creates a "free slot" in p's active view. By adding p to its passive view, node q increases the probability of shuffling q with other nodes and, subsequently, having p be target of NEIGHBOR requests.

5 Evaluation

We conducted simulations using the PeerSim Simulator [9]. We have implemented both HyParView, Cyclon and Scamp in this simulator in order to get comparative figures. In order to validate our implementation of Cyclon and Scamp, we have compared the results of our simulator with published results for these systems (we omit these simulations from the paper, as they do not add to assess the merit of our approach).

We have also implemented a version of Cyclon, to which we called CyclonAcked, that adds a failure detection system to Cyclon based on the exchange of explicitly acknowledgments during the message dissemination. Thus, CyclonAcked is able to detect a failed node when it attempts to gossip to it and, therefore, is able to remove failed members from partial views, increasing the accuracy of these views. We use this benchmark to show that the benefits of our approach do not come only from the use of TCP as a failure detector but also from the clever use of two separate partial views.

Finally, we have implemented on PeerSim a gossip broadcast protocol that can use any of the protocols above as a *peer sampling service*. In this protocol, a node forwards a message when it receives it for the first time (therefore, there is no *a priori* bound on the number of gossip rounds).

In all simulations, the overlay was created by having nodes join the network one by one, without running any membership rounds in between. Cyclon was initiated by having a single node to serve as contact point for all join requests. Scamp was initiated by using a random node already in the overlay as the contact point. These are the configurations that provide the best results with these protocols. HyParView achieves similar results with either method (we have used the same procedure as Cyclon).

5.1 Experimental Setting

All experiments were conducted in a network of 10.000 nodes and results show an aggregation from multiple runs of each experiment. Furthermore, each membership protocol was configured as follows: In HyParView, we set the active membership size to 5, and passive membership's size to 30. *Active Random Walk Length* parameter was set to 6 and the *Passive Random Walk Length* was set to 3. In each shuffle message, $k_p = 4$ elements (at most) were sent from the passive view, while $k_a = 3$ elements (at most) were sent from the active view. The total size of shuffle messages is 8, as nodes also send their own identifier in each shuffle message. Cyclon protocol was configured with partial views of 35 elements (this is the sum of HyParView's active and passive view sizes). Shuffle message lengths were set to 14 and the time to live to random walks in the overlay was configured to 5. Scamp was configured with parameter c - the parameter that is related with fault tolerance of the protocol - to 4. The reason behind the selected value to this parameter was because it generated partial views which size's where distributed around a middle point of 34, which is as near as we could be from the value used in other protocols. Our gossip based broadcast protocol was configured with a fanout of 4.

5.2 Effect of Failures

We first evaluate the impact of massive failures in the reliability of gossip when different membership protocols are used. In each experiment that we run, we make all nodes join the overlay, and execute 50 cycles of membership protocol to guarantee stabilization[3]. After the stabilization period, we induce failures at random in a percentage of all nodes in the system. We experimented with several values, ranging from 10% to 95%, of node failure. We then measure the reliability of 1.000 messages sent from random correct nodes. All these messages are sent before the execution of another cycle of the membership protocol. However, the membership protocols still execute all reactive steps; in particular, they can exclude a node from their partial views if the node is detected to be failed. The rationale for this setting is that the interval of the periodic behavior of

[3]In fact, this stabilization time is not required by Scamp, as it stabilizes immediately after the join period.

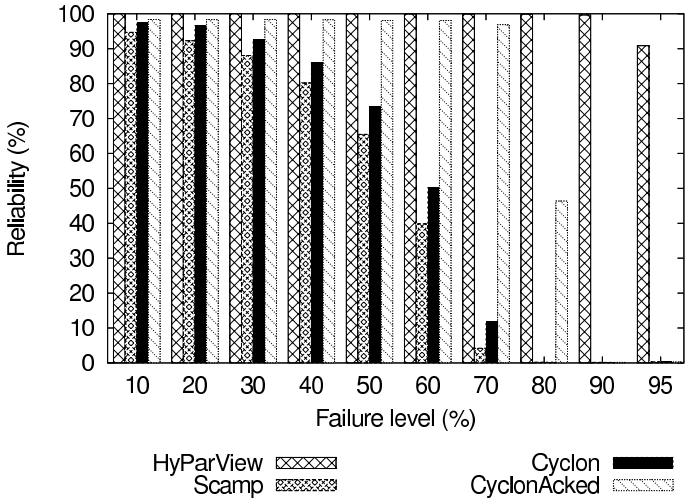


Figure 2: Reliability for 1000 messages

the membership protocols is often long enough to allow thousands of messages to be exchanged, and we are looking for the impact of failures in the reliability of these broadcasts.

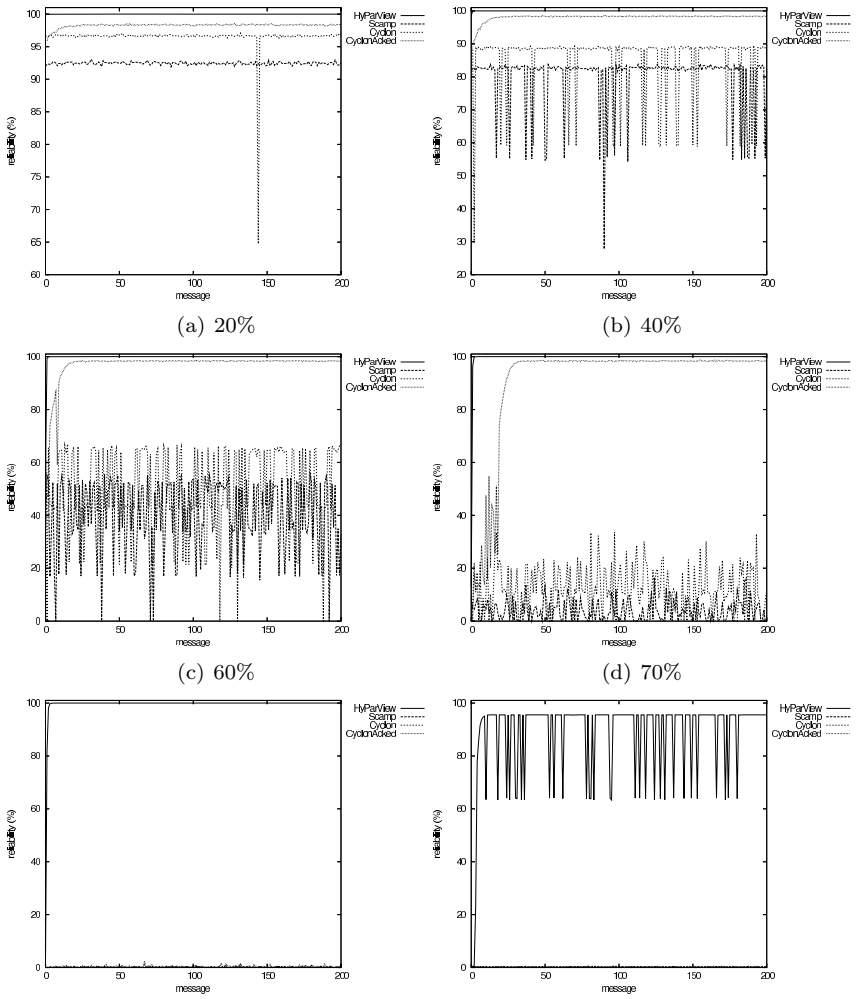
The average reliability for these runs of 1.000 messages is depicted in Figure 2. As it can be seen, massive percentage of failures have almost no visible impact on HyParView below the threshold of 90%. Even for failure rates as high as 95%, HyParView still manages to maintain a reliability value in the order of deliveries to 90% of the active processes. Both Scamp and Cyclon exhibit a constant reliability[4] for failure percentages as low as 10%, and their performance is significantly hampered with failure percentages above 40% (with reliabilities below 50% of nodes). On the other hand, CyclonAcked manages to offer a competitive performance. Although the reliability is not as high as with HyParView, it manages to keep high reliabilities for percentage of failures up to 70%. This behaviour highlights the importance of fast failure detection in gossip protocols.

The reader should also notice that HyParView has a better reliability even when failure rates are not as high as 40%, this happens because HyParView uses a deterministic selection of nodes to whom forward gossip messages, this combined with a symmetric view, ensures that in a stable environment HyParView, unlike other protocols, has 100% reliability, as long as the overlay remains connected.

Figures 5a-5f shows the evolution of reliability with each message sent, after the failures, for different failure percentages. In all figures, HyParView is the line that offers better and faster recovery usually near the 100%. Next appear CyclonAcked, Cyclon and Scamp in this order for all failure levels. Above 80% failures all these lines appear close to the value of 0%.

¿From the figures, it is clear that HyParView recovers almost immediately from the failures. This is due to the fact that all members of the active views are tested in a single broadcast. Basic Cyclon/ Scamp membership protocols, as they do not use a failure detector, are unable to recover until the membership protocol is executed again. In order to maintain reliability under massive percentage of failures, they would have to be configured with very high fanouts (which is a cost inefficient strategy in steady state). The figures also show that by adding acknowledgments to the Cyclon based gossip protocol, CyclonAcked recovers a high reliability after a small number of message exchanges (approximately 25). Note that, in Cyclon, a node is only tested when it is selected (at random) as a gossip target. However, for percentage of failures in the order of 80%, CyclonAcked is unable to regain the reliability levels as HyParView. This is due to the following phenomenon: given that the Cyclon overlay is asymmetric, some nodes may have outgoing links

[4]Although their reliability is unable to reach 100% with a fanout of 4.



(a) 20% (b) 40%

(c) 60% (d) 70%

(e) 80% (f) 95%

Figure 3: Reliability after failures

and no incoming link; therefore, some nodes are still able to broadcast messages but unable to receive any message. On the other hand, in HyParView, the active membership is symmetric, which means that if a node is able to reach another correct node in the overlay, it is necessarily reachable by messages sent by other nodes. This feature and a very low clustering coefficient (see Section 5.4) explains the high resilience of HyParView.

5.3 Healing Time

Figure 4 shows how many membership cycles are required to achieve the same reliability in the message dissemination after a massive node failure (for different percentage of node failures). These results were obtained as follows: in each simulation, after the stabilization period, failures are induced. Subsequently, multiple membership protocol cycles are executed. In each cycle, 10 random nodes are selected to execute a broadcast. We then calculate the average reliability of these messages, and count the cycles required for each protocol to regain a reliability equal or greater than the one exhibit by that same protocol before the induction of the failure.

As expected, after the results presented before, HyParView recovers in few rounds (only 1 or 2) for all percentages below 80%. Cyclon requires a significant number of membership cycles, that grows almost linearly with the percentage of failed nodes to achieve this goal. We do not present

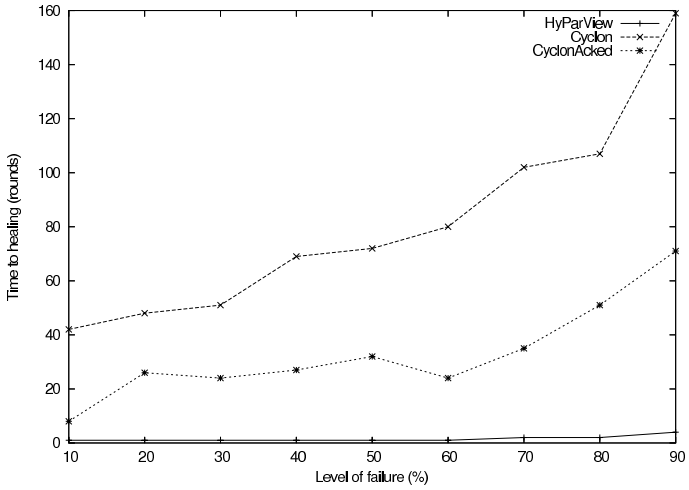


Figure 4: Healing time

	Average clustering coefficient	Average shortest path	Maximum hops to delivery
Cyclon	0,006836	2,60426	10,6
Scamp	0,022476	3,35398	14,1
HyParView	0,00092	6,38542	9,0

Table 1: Graph properties after stabilization

values for Scamp, because the total time for Scamp to regain it's levels of reliability depends on the Lease Time, which is typically high to preserve some stability in the membership.

5.4 Graph Properties

As noted in Section 2.3, the overlays produced by the membership protocol must exhibit some good properties such as low clustering coefficient, small average shortest path and balanced in-degree distribution. We now show how the different protocols perform regarding these metrics. Table 1 shows values to average clustering coefficient and average shortest path for all protocols[5] after a period of stabilization of 50 membership cycles. It can be seen that in terms of average clustering coefficient, HyParView achieves significantly lower values than Scamp or Cyclon, which is normal considering that HyParView's active view is much smaller than other protocols partial views. This is an important factor to explain the high resilience that HyParView exhibits to node failures.

In terms of average shortest path, we see that HyParView falls behind Scamp and Cyclon. This is no surprise, as we only maintain a smaller active view, which limits the number of distinct paths that exist across all nodes. Fortunately, this has no impact on the latency of the gossip protocol. The short level of global clustering and the fact that we use all existing paths between nodes to disseminate every message, makes our protocol deliver gossip within a smaller number of hops than the other protocols, as it is depicted in Table 1.

Figure 5 shows the in-degree distribution of all nodes in the overlay after the same period of stabilization. Cyclon and Scamp have distribution of in-degree across a wide range of values, which means that some nodes are extremely popular on the overlay, while other nodes are almost totally unknown. As stated before, because of this distribution some nodes on the overlay have

[5]results for HyParView concern its active view

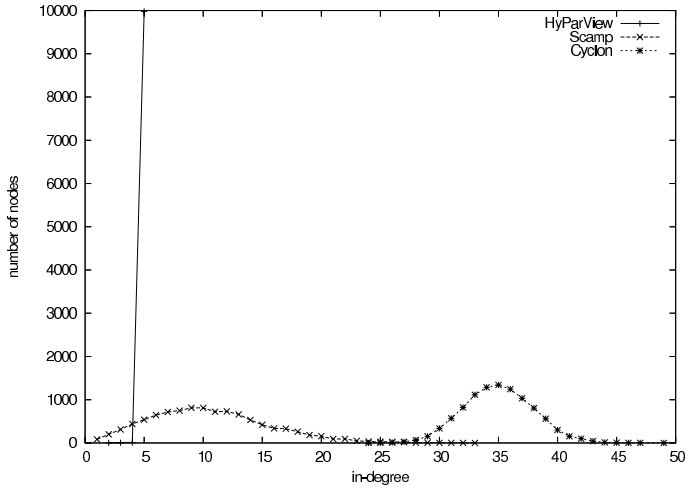


Figure 5: In-degree distribution

greater probability to receive redundant messages, while other nodes have a very small probability to see messages once. This is specially obvious in Scamp, where some nodes are only known by one other node.

Due to HyParView's symmetric active view, almost all nodes in the overlay are known by the maximum amount of nodes possible, which is the active view length (5). This means that all nodes, with high probability, will receive each message exactly the same amount of times, and also that there is little probability for any node not to receive a message at least once.

5.5 Discussion

It is possible to extract the following lessons from our results. To start with, the speed of failure detection is of paramount importance to sustain high reliability in the presence of massive percentage of faults. A gossip strategy that relies on the use of a reliable transport that also serves as a failure detector, over a fixed overlay (built using a probabilistic membership protocol) offers the best performance possible in this regard. Also, by using all the links of the overlay, it is possible to aim at 100% reliability as long as the overlay remains connected. Furthermore, it allows to use smaller fanouts than protocols that have to mask failures and network omissions with the redundancy of gossip. The use of small fanouts is what makes possible to use all the links of the overlay with small overhead. Additionally, the maintenance of a passive view, with candidates to replace failed nodes in the active view, offers high resilience to massive failures. Therefore, the use of an hybrid approach that contains a small active view and a larger (low cost) passive view, maintained by different strategies, offers a better resilience and better resource usage than using a single (large) view with a higher fanout.

The use of TCP might could cause a blockage in the overlay in the presence of slow nodes that do not consume messages from their reception buffers: TCP flow control would make the neighbors of that node block while trying to send messages to it and, in turn, also stop receiving messages. Eventually this effect would spread to all the overlay in a epidemic manner. This is however handled by the protocol using the techniques proposed in [10], which simply considers slow nodes as having failed, and expels them from all active views. A special notification should be sent to nodes, to assure that it will not try to reconnect to nodes from it's passive view repeating the process.

6 Conclusions and Future Work

Gossip protocols are appealing because they work on overlays that have very small maintenance cost. Therefore, they seem obvious candidates to support applications that require extremely high resilience to failures of large percentage of nodes. Such massive failures can happen due to attacks (for instance, a worm that shuts down all the machines of a particular make) or in catastrophic natural disasters (such as earthquakes). To the best of our knowledge, this is the first paper that studied the effect on the reliability of gossip under massive percentage of failures, when different approaches are used to maintain distinct partial membership information.

We defend a gossip strategy that consists of flooding overlay topology that is created by a probabilistic (partial) membership protocol. Furthermore, we have proposed a novel hybrid membership protocol for that purpose. The protocol maintains a small active view and a larger passive view for fault-tolerance. We have shown that our protocol is able to preserve very high values of reliability, with a small fanout, in faulty scenarios where the percentage of failed nodes can be as high as 80%.

As future work we would like to experiment, to better define, the relation between the passive view size and the resilience level of the protocol (*i.e.* how many failures are supported without the overlay became disconnected). A implementation of HyParView will be tested in the PlanetLab platform [12] in order to measure the packet overhead of our approach due to the use of TCP.

Finally, we would also like to experiment our approach with adaptive fanouts, by taking into account the heterogeneity of nodes, in order to maximize the use of available resources, like bandwidth. To do this and still maintain our deterministic selection of gossip targets, nodes would be required to adapt their degree (and in-degree), which might prove an interesting approach in order to obtain optimized emergent overlays.

References

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

[2] Mayur Deshpande, Bo Xing, Iosif Lazardis, Bijit Hore, Nalini Venkatasubramanian, and Sharad Mehrotra. Crew: A gossip-based flash-dissemination system. In *Proc. of the 26th ICDCS*, Washington, DC, USA, 2006.

[3] P. Th. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM TOCS*, 21(4):341–374, 2003.

[4] A. Ganesh, A. Kermarrec, and L. Massoulie. Peer-to-peer membership management for gossip-based protocols, 2003.

[5] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massoulie. SCAMP: Peer-to-peer lightweight membership service for large-scale group communication. In *Networked Group Communication*, pages 44–55, 2001.

[6] Mark Hayden and Kenneth Birman. Probabilistic broadcast. Technical report, Ithaca, NY, USA, 1996.

[7] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In *Proc. of Middleware '04*, pages 79–98, 2004.

[8] A. Kermarrec, L. Massoulie, and A. Ganesh. Probabilistic reliable dissemination in large-scale systems, 2001.

[9] Peersim p2p simulator. http://peersim.sourceforge.net/.

[10] J. Pereira, L. Rodrigues, M. J. Monteiro, R. Oliveira, and A.-M. Kermarrec. Neem: Network-friendly epidemic multicast. In *Proc. of the 22th SRDS*, pages 15–24, Florence,Italy, October 2003.

[11] J. Pereira, L. Rodrigues, A. Pinto, and R. Oliveira. Low-latency probabilistic broadcast in wide area networks. In *Proc. of the 23th SRDS*, pages 299–308, Florianopolis, Brazil, October 2004.

[12] Planetlab: Home. http://planet-lab.org/.

[13] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to own the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, Berkeley, CA, USA, 2002. USENIX Association.

[14] Angelos Stavrou, Dan Rubenstein, and Sambit Sahu. A lightweight, robust p2p system to handle flash crowds. Technical Report EE020321-1, Columbia University, New York, NY, February 2002.

[15] Spyros Voulgaris, Daniela Gavidia, and Maarten Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.