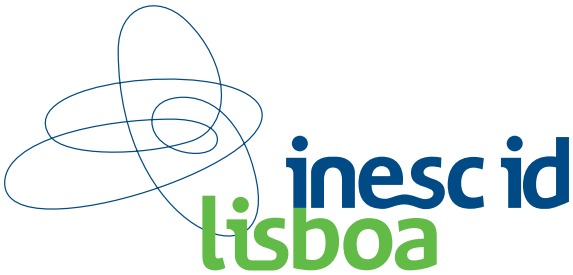


Technical Report RT/52/2009

RASM: A Reliable Algorithm for Scalable Multicast

Mouna Allani
University of Lausanne
mouna.allani@unil.ch

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

Benoit Garbinato
University of Lausanne
benoit.garbinato@unil.ch

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

May 2009

Abstract

Recently there has been an effort to build scalable and reliable application-level multicast solutions that combine the resilience of pure gossip-based with the efficiency of tree-based schemes. However, such solutions assume that participants have unlimited resources, for instance, that they can send an unbounded number of messages to mask network omissions. Such scenario is not realistic, specially for streaming protocols, where messages can be transmitted at a very high rate and have a small temporal validity.

In this paper, we propose RASM, a scalable distributed protocol for application-level multicast. Our protocol is based on the combination of gossip-based and tree-based multicast schemes. Unlike previous approaches, which strive to combine gossip-based and tree-based schemes, our solution takes into consideration the reliability of components: nodes and communication links can fail, unexpectedly, ceasing their operation and dropping messages, respectively. Experimental results show that our scheme offers better reliability than previous solutions with low overhead.
Keywords: Peer-to-Peer overlays, Streaming, Reliability

RASM: A Reliable Algorithm for Scalable Multicast *

Mouna Allani
University of Lausanne
mouna.allani@unil.ch

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

Benoît Garbinato
University of Lausanne
benoit.garbinato@unil.ch

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

Abstract

Recently there has been an effort to build scalable and reliable application-level multicast solutions that combine the resilience of pure gossip-based with the efficiency of tree-based schemes. However, such solutions assume that participants have unlimited resources, for instance, that they can send an unbounded number of messages to mask network omissions. Such scenario is not realistic, specially for streaming protocols, where messages can be transmitted at a very high rate and have a small temporal validity.

In this paper, we propose RASM, a scalable distributed protocol for application-level multicast. Our protocol is based on the combination of gossip-based and tree-based multicast schemes. Unlike previous approaches, which strive to combine gossip-based and tree-based schemes, our solution takes into consideration the reliability of components: nodes and communication links can fail, unexpectedly, ceasing their operation and dropping messages, respectively. Experimental results show that our scheme offers better reliability than previous solutions with low overhead.

1 Introduction

Application-level multicast solutions, that operate on top of peer-to-peer overlay networks, have gained significant attention from the research community in the past years. These solutions appear as an alternative to IP multicast that faces several technical and commercial deployment issues. These solutions have been used to support the operation of several Internet services, such as file sharing, resource location, audio/video streaming, among others.

There have been two main approaches to develop such multicast solutions. The first relies on epidemic gossip-based protocols, which disseminate messages to random peers [1, 2, 3]. The second relies on some sort of spanning tree structure formed by peers, which is used to route messages deterministically [4, 5, 6, 7, 8, 9, 10, 11]. Gossip-based protocols are an interesting approach because they are simple and can distribute the load among all participants in the system. Their natural redundancy is able to mask both node and link failures, making these approaches highly resilient. Unfortunately, the natural redundancy also induces a high message overhead.

On the other hand, tree-based protocols are more efficient, as they rely on a spanning tree structure to deterministically route messages among peers, without the transmission of redundant messages. Moreover, this approach makes it possible to take into consideration the heterogeneity of participants (for instance, by putting more reliable nodes closer to the root of the spanning tree). Unfortunately, tree-based approaches introduce additional complexity, as they require the construction and maintenance of a spanning tree.

*This research was jointly funded by project "Redico" (PTDC/EIA/71752/2006), the Swiss National Science Foundation, in the context of Project number 200021-108191, and the European Science Foundation (ESF) under the "Middleware for Network Eccentric and Mobile Applications" (MiNEMA) activity.

In summary, the tradeoff between these two approaches is basically scalability and resiliency *vs.* adaptiveness and efficiency.

1.1 Combining Approaches

Recently, there has been several efforts to combine both tree-based and gossip-based approaches [12, 13, 14]. With such an approach, one can build and maintain a low-cost spanning tree for efficient multicast, and rely on the remaining links of the network for fault-tolerance. Unfortunately, existing solutions, which combine tree-based and gossip-based approaches, do not take into account the reliability of nodes or links. Moreover, these solutions typically assume that participants have unlimited resources, for instance, that they can send an unbounded number of messages to mask network omissions.

This paper proposes a *Reliable Algorithm for Scalable Multicast* (or simply **RASM**), a multicast solution to serve peer-to-peer streaming applications. Our solution maximizes the multicast reliability in an unreliable environment by combining tree-based and gossip-based approaches. In this context, we define *reliability* as the probability that all nodes deliver a streamed packet. To do so, some determinism is added to traditional gossip solutions, in order to select, at each node, a subset of its direct neighbors to gossip with. The closure of all neighbors, and links, selected for gossip should, implicitly, define a tree-overlay that includes the most reliable paths in the system.

Our protocol is completely decentralized and aims at ensuring both low message and low computation overhead. The low message overhead derives from the use of a tree overlay whereas the low computation overhead derives from the use of a gossip based strategy. Furthermore, our solution can scale to a large number of simultaneous participants, as it does not require global knowledge concerning the entire system to be maintained by each participant. We do this by leveraging on a low cost unstructured overlay network, which can be built on top of a peer sampling service based on partial views of the system [15, 3].

1.2 Roadmap

The rest of this paper is organized as follows. In Section 2, we introduce the system model and define the problem that motivates our work. Section 3 describes the RASM protocol. Our solution is evaluated through simulations in Section 4. Section 5 covers the related work and finally, Section 6 concludes the paper.

2 Model & Problem Statement

We consider an asynchronous distributed system composed of processes (nodes) that communicate by message passing. Our model is probabilistic in the sense that nodes can crash and links can lose messages with a certain probability. More formally, we model the system topology as a connected graph $G = (\Pi, \Lambda)$, where $\Pi = \{p_1, p_2, ..., p_n\}$ is a set of n processes and $\Lambda = \{l_1, l_2, ...\} \subseteq \Pi \times \Pi$ is a set of bidirectional communication links[1].

We assume that nodes can crash and later recover and links can suffer omission faults. Both process crash and link message loss probabilities are modeled as a *failure configuration* $C = (P_1, P_2, ..., P_n, L_1, L_2, ..., L_{|\Lambda|})$, where P_i is the probability that process p_i crashes during a computation step, and L_j is the probability that link l_j loses a message during a communication step.

For scalability, we take into account the *limited memory* of each node p_i, by limiting the knowledge it keeps about the system to knowledge about its direct neighbors, denoted N_i and to the links connecting it to these neighbors, denoted LN_i. We assume that knowing about an environment component (link or process) includes knowing all its properties. That is, if a process p_i knows a neighbor n_k then, p_i knows n_k crash probability, noted P_k. It, also, knows l_k, the link relying p_i to n_k and its message loss probability L_k. Notice that this assumption is realistic as each node is only required to maintain a small number of neighbors.

Intuitively, the main question addressed in this paper is: *how to maximize the message multicast reliability, in spite of unreliable processes and links and in spite of the limited knowledge about the system?* More formally, using our system model presented earlier, we can restate the problem we address in this paper as follows: *given the set of neighbors N_i and the set links connecting it to these neighbors LN_i, how should process p_i propagate*

[1] This system view can be approximated by each process using, for instance, the results presented in [15].

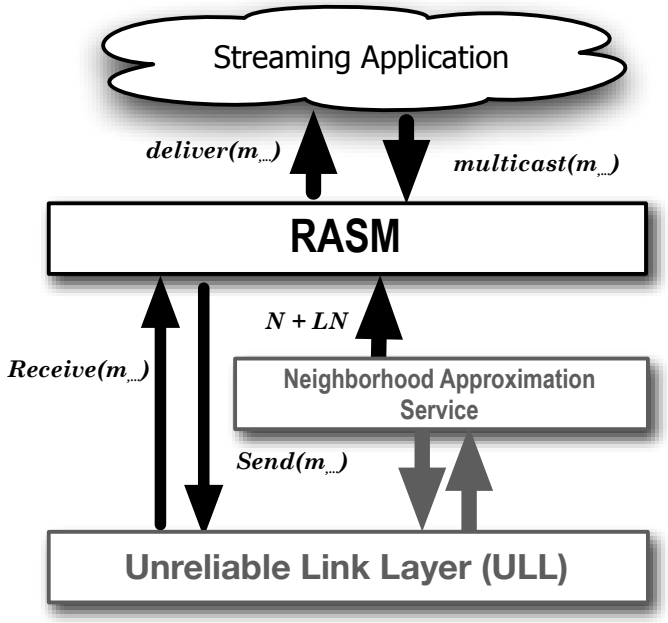


Figure 1. Streaming node architecture

a multicast packet in order to maximize the probability of having this multicast packet reach all processes, in an efficient manner?

3 RASM

In this section, we describe RASM our novel multicast solution to support reliable streaming in large-scale systems. We start by providing an overview of the system architecture in which RASM operates. Then we follow with a brief description of the rationale for RASM. Finally, we detail the most relevant aspects of the protocol.

3.1 Overview

As shown in Figure 1, RASM might serve a top *Streaming Application*. In this case, the top *Streaming Application* would be responsible for breaking the outgoing stream into a sequence of messages on the producer side, and for assembling these messages back into an incoming stream on the consumer side. The *Streaming Application* then relies on *RASM* to reliably multicast each stream packet. Our multicast solution is defined by two primitives: *multicast(m)* to multicast a packet m and *deliver(m)* to deliver a packet m to a top streaming application. Each RASM node maintains an up-to-date knowledge about its direct neighbors: N, and the links connecting it to these neighbors: LN. This knowledge includes also the reliability of these components, which is provided by an underlying layer: *Neighborhood Approximation Service*. The latter relies on Bayesian inference to approximate the neighborhood of each process p_i. Explaining about how neighborhood approximation actually works goes beyond the scope of this paper and can be found in [15]. Finally, the *Unreliable Link Layer* allows each process p_i to send messages to its direct neighbors in a probabilistically best-effort manner.

Algorithm 1 Tree Construction Algorithm executed by p_i.

1: **initialization:**
2: provider $\longleftarrow \emptyset$
3: receivedMsgs $\longleftarrow \emptyset$
4: eagerPushPeers $\longleftarrow$ N_i

5: **procedure** $multicast(m)$
6: mID $\longleftarrow$ hash(m+myself)
7: **call** EagerPush (m, mID, 1, myself)
8: deliver(m)
9: receivedMsgs $\longleftarrow$ receivedMsgs $\cup$ {(mID, myself,1)}

10: **procedure** $EagerPush(m, mID, proba, Sender)$
11: **for all** $p_j \in$ eagerPushPeers: $p_j \neq$Sender **do**
12: $proba \leftarrow proba \times [(1 - P_i) \times (1 - L_{i,j}) \times (1 - P_j)]$
13: Send(GOSSIP, m, mID, proba, p_i) to p_j

14: **upon** Receive(GOSSIP,m, $mID, proba, Sender$) **do**
15: **if** $\nexists\ e \in receivedMsgs : e.mID = mID$ **then**
16: deliver(m)
17: receivedMsgs $\longleftarrow$ receivedMsgs $\cup$ {(mID, Sender, proba)}
18: provider $\longleftarrow$ Sender
19: eagerPushPeers $\longleftarrow$ eagerPushPeers $\cup$ {Sender}
20: **call** EagerPush (m, mID, proba, Sender)
21: **else**
22: **let** (oldmID, oldSender, oldProba) $\in$ receivedMsgs / oldmID = mID
23: **if** $proba > oldProba$ **then**
24: receivedMsgs $\longleftarrow$ receivedMsgs $\setminus$ {(oldmID, oldSender, oldProba)}
25: receivedMsgs $\longleftarrow$ receivedMsgs $\cup$ {(mID, Sender, proba)}
26: eagerPushPeers $\longleftarrow$ eagerPushPeers $\cup$ {Sender}
27: eagerPushPeers $\longleftarrow$ eagerPushPeers $\setminus$ {oldSender}
28: provider $\longleftarrow$ Sender
29: Send(PRUNE, p_i) to oldSender
30: **else**
31: Send(PRUNE, p_i) to Sender
32: eagerPushPeers $\longleftarrow$ eagerPushPeers $\setminus$ {Sender}

33: **upon** Receive(PRUNE, $Sender$) **do**
34: **if** provider.ID = Sender.ID **then**
35: Send(REPAIR, p_i) to Sender
36: **else**
37: eagerPushPeers $\longleftarrow$
eagerPushPeers $\setminus$ {Sender}

38: **upon** Receive(REPAIR, $Sender$) **do**
39: eagerPushPeers $\longleftarrow$ eagerPushPeers $\cup$ {Sender}

3.2 Rationale

Our protocol operates as any pure gossip protocol, in the sense that, in order to multicast a message, each node gossips with a set of nodes. Our strategy follows a similar architecture to that of the Plumtree protocol [12] where nodes gossip deterministically using a combination of eager and lazy push gossip. By doing so, Plumtree aims to implicitly create a spanning tree overlay with the minimum latency through which multicast packets are routed.

Similarly to Plumtree, each node executing the algorithm, maintains a sub-set of its neighbors, named *eagerPushPeers*, with which it uses an eager gossip. Unlike Plumtree, the selection of neighbors to be added to the *eagerPushPeers* set ensures that the closure of links among those peers form the *most reliable* tree covering the whole system.

Hereafter, we give an overview of the most relevant aspects of our protocol. That is, the following section focuses on the definition of the eagerPushPeers set at each node. The reader can also refer to Algorithm 1 for additional details.

3.3 Protocol Description

We describe our decentralized mechanism to build a spanning tree that attempts to use the most reliable paths available in the underlying overlay network. Algorithm 1 depicts the RASM protocol. Before describing how RASM builds a tree with the most reliable paths, we first need to introduce the notions of *reachability probability* and *reachability function*. These notions are borrowed from a previous paper [15].

Reachability Probability The *reachability function*, noted $R()$, computes the probability to reach all processes in some propagation tree T when a given message is propagated through T. We then define the probability returned by $R()$ as T's *reachability probability*. Equation 1 below presents the reachability function borrowed from [15]. In this function, R is measuring the *Reachability Probability* of a tree T with n-1 links, where P_j denotes the crash probability of node p_j, $P_{pred(j)}$ denotes the crash probability of the node preceding p_j in T (denoted by $pred(j)$), and L_j denotes the message loss probability of link l_j connecting $pred(j)$ to p_j.

$$R(T) \;=\; \prod_{j=1}^{n-1}[(1 - P_{pred(j)}) \times (1 - L_j) \times (1 - P_j)] \tag{1}$$

As already mentioned, at each node, the *eagerPushPeers* set of a node represents a subset of its direct neighbors, with which that node gossips stream packets. Therefore, the goal is to ensure that neighbors in this set are the most reliable ones (both in terms of link and process failure probability).

The multicast of a packet m starts by having the stream source sending m to all its neighbors in its *eagerPushPeers* set by calling the $EagerPush()$ procedure (line 7) Initially, each *eagerPushPeers* set of a process p_i, is initialized with the set of its direct neighbors N_i (line 4). Thus, the multicast of the first stream packet m is performed by a flooding mechanism, where each node sends m to all neighbors in its *eagerPushPeers* set.

Obviously, some nodes will receive duplicates of m. When a duplicate of a packet is received by a node p_i (line 21), this is an indication that several existing paths allow the node to receive stream packets sent from the same stream source. In such a scenario, the goal is to select the path with the highest reachability probability and to prune the remaining paths to avoid packets duplication. Implicitly this will generate a tree with the maximum reachability probability. The information about the reachability probability of a path crossed by a packet m is piggybacked to the stream packet itself. Indeed, when a node sends a message m to a neighbor is also sends additional information representing the reachability probability of the path crossed by m so far.

This probability is computed iteratively at each overlay hop performed by the stream packet, using the product depicted in the reachability function R. That is, each node p_i sending message m to a node p_j, computes the reachability probability taking into account the reachability probability of the upstream path, its own crash probability P_i, the crash probability of the target neighbor P_j, and the message loss probability of the link to that neighbor L_j (line 12).

Thus, when a node p_i receives a duplicate of the packet m from its neighbor p_j, it compares the reachability probability of the path crossed by m via p_j and the reachability probability of other paths that also allow p_i to

receive m from a given predecessor node p_k (line 23). When the reachability probability via p_j is higher, p_i keeps p_j as its stream packet provider, removing p_k from its *eagerPushPeers* set and sending a PRUNE message to p_k to remove the path through this previous sender from the emergent spanning tree (line 29). Upon receiving a PRUNE message, p_k also removes p_i from its *eagerPushPeers* set, hence p_k will not propagate the following stream packets to p_i. This mechanism eventually and implicitly defines a spanning tree overlay that includes the most reliable paths of the original peer-to-peer overlay network while avoiding message redundancy, i.e., imposing the minimum message overhead.

Execution Example To illustrate our approach, Figure 2 depicts the operation of our algorithm in a simple scenario. In this example, we assume that only links may be unreliable, while processes are assumed to be reliable. Labels associated with each link in Figure 2 represent the link unreliability L_i (the probability to lose a propagated message). That is, links $l_{1,2}$, $l_{1,3}$ and $l_{3,4}$ have a probability to lose a message routed through them of 0.2, whereas link $l_{1,4}$ has a probability to lose a message of 0.6 being therefore less reliable.

The multicast starts by having the source node (node p_1) sending a multicast message m to all nodes in its *eagerPushPeers* set (Figure 2 (a)). Initially, the *eagerPushPeers* set of p_1 includes all its direct neighbors ($\{p_2$, p_3, $p_4\}$). To each neighbor, p_1 sends with m the reliability of the path followed by m to reach that neighbor. For example, to reach p_4 directly from p_1, m crosses a path of reliability $1 - 0.6 = 0.4$ considering the reliability of the link between p_1 and p_4. Upon receiving m, process p_3 will forward the message to all nodes in its *eagerPushPeers* set except p_1, i.e., p_3 forwards m only to p_4 (Figure 2 (b)). In addition, process p_3 sends an adjusted reliability of the path followed by m to reach p_4 computed using function R. When receiving a duplicate of m from p_3, p_4 compares the reliability of the path crossed by m from p_3: 0.64[2]; and the reliability of the path through which m was previously sent: 0.4. To select the path through p_3, p_4 sends a PRUNE message to the previous sender of m: p_1 (Figure 2 (c)). When receiving this PRUNE message, p_1 removes p_4 from its *eagerPushPeers* set. The resulting tree defined by the *eagerPushPeers* sets is depicted in Figure 2 (d); as the reader can notice, the resulting spanning tree includes the most reliable links.

Disconnection Risk Note that in the previous example, p_4 also sends m to p_3 upon receiving it for the first time. However, p_3 is not interested by this alternative path to receive multicast packets[3]. This scenario represents a disconnection risk as p_3 may discard p_4 from its *eagerPushPeers* set while this latter counts on p_3 to receive multicast messages. To overcome this problem, we define an efficient and simple mechanism based on a type of messages named `Repair`. For this mechanism, we assume that each process p_i defines as *provider* the neighbor in its *eagerPushPeers* set that is supposed to provide p_i with multicast packets (its upstream neighbor). When p_i receives a multicast message m for the first time it defines the neighbor that sent m as provider. When p_i receives a duplicate of m through a more reliable path, it then switches the provider to the neighbor that sent the duplicate of m. The idea of this simple mechanism is that when p_i receives a PRUNE message from its provider, p_i simply replies with a REPAIR message to reconnect itself to that provider. In its turn, p_i's provider, upon receiving the REPAIR message from p_i, adds it to its *eagerPushPeers* set again. Preliminary experimental results have shown that this simple mechanism improves the spanning tree reliability imposing a low overhead.

Resource Optimization The solution described before may be further improved by a simple well-known mechanism: message redundancy. However, applying a strategy based on a blind message redundancy to our tree will contradict one of the main goals of this paper: efficiency. In addition, P2P networks are by nature limited in resources. Hence applying a blind message redundancy technique under these constraints is undesirable. As proposed in [4], resources constrains (e.g., CPU, memory and bandwidth) could be addressed by relying in a fixed quota of messages at the disposal of each node to diffuse a single multicast message. That quota expresses the capacity of each node to process messages and also its available outgoing bandwidth. Formally, the *available resource* is modeled as $Q = (q_1, q_2, ..., q_n)$, the set of quotas associated to nodes in the system; q_i is the *individual quota of messages* at the disposal of process p_i to forward a single multicast message. For the correctness of our algorithm, we assume that q_i is at least equal to N_i, the direct neighbors of process p_i. We argue that this assumption is realistic. Indeed, in a peer-to-peer system, we can consider that a link between two nodes is an overlay link formed

[2]This value is given by $(1 - 0.2) * (1 - 0.2) = 0.64$.

[3]Notice that p_3 is not interested in receiving multicast messages via p_4 because that path does not offers the better available reliability.

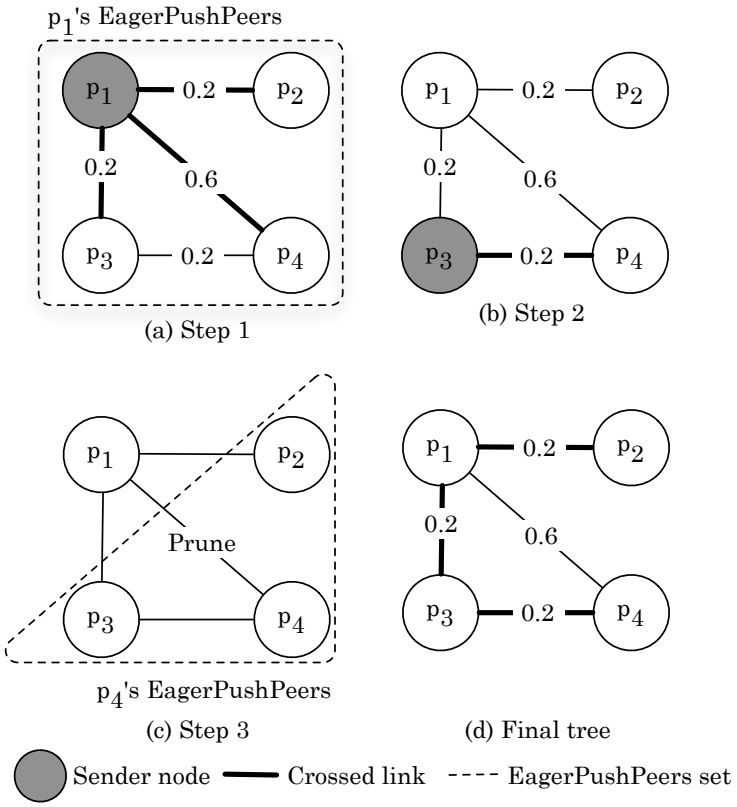


Figure 2. Execution example

a as a result of a peering relationship between those nodes. Such a peering relationship is established using a bootstrapping mechanism performed by an arrival node trying to select existing nodes with whom such relationships can be established. In our case, such a capability would reflect the available quota of messages. Moreover, by leveraging on the work proposed in [3] one can easily limit the maximum number of neighbors maintained by each node, while ensuring the overall connectivity of the resulting peer-to-peer unstructured overlay network.

In this section, we propose a resource optimization mechanism which enabled us to improve the reliability of each packet multicast, optimizing the use of available message quota at each node, by transmitting more than a single message to a single neighbor in the *eagerPushPeers* set, allowing to mask potential link omissions.

To take maximum advantage of the available message quota at each process, we slightly change our solution to compute a quota distribution before gossiping each stream packet. The idea is that. before forwarding a new message m, a process p_i will first optimize the distribution of its quota q_i. Such a distribution defines how many retransmissions should be sent through each link in order to maximize the probability of reaching all neighbors in the *eagerPushPeers* set. To measure the benefit of the quota optimization, we define a new version of function $R()$ (Equation 1) named R' (Equation 2) that measures the impact of message retransmissions assigned to each link of a tree on that *Reachability Probability*.

$$R'(T,\vec{m}) \;=\; \prod_{j=1}^{|\vec{m}|} 1-[1-(1-P_{pred(j)})\times(1-L_j)\times(1-P_j)]^{\vec{m}[j]} \tag{2}$$

Based on the message redundancy definition at links of a tree T, function R' (Equation 2) returns the reachability probability of T, where the number of times a multicast message is forwarded in each link of the tree is represented

by vector $\vec{m}$.[4]

The quota optimization works iteratively starting with a minimal distribution allocating one message to each receiver of the *eagerPushPeers* set. The remaining quota is then allocated one by one until it is exhausted.

Each additional message transmission is allocated, by our optimization mechanism, to the outgoing link in order to maximize the gain in probability to reach all receivers computed using the R' function (equation 2). Considering the tree T_i composed by a sender process p_i and a set of receiver nodes ($\subseteq$ p_i *eagerPushPeers* set) and a vector $\vec{m}_i$ defining the distribution q_i (p_i's quota). The *reachability probability* given by $R(T_i, \vec{m}_i)$ corresponds to the maximum probability to reach all the receivers of p_i given the crash probability P_i, the receivers crash probability, loss probability for the relevant links, and available quota q_i.

4 Evaluation

In order to evaluate the benefits of RASM, we have implemented a prototype of the protocol in the Sinalgo simulator.[5] RASM is compared to Plumtree [12], which is also implemented in Sinalgo and executed in the same scenarios as RASM. Plumtree [12] also combines eager and lazy push gossip strategies on top of an overlay network, in such a way that links where eager push is used, implicitly define a spanning tree with the minimum latency. The lazy push gossip approach is used on the remaining links of the overlay to handle failures and to provide support for fast tree repair. The goal of the protocol is to achieve reliable broadcast with a lower message overhead without hampering the latency of the system.

To compare RASM with Plumtree, we performed experiments in a network composed of 100 nodes organized in random topologies. We have varied the node degree in experiments from a degree of 2 (equivalent to a ring topology) to 20. Overlay links are symmetric. To avoid regular network configurations, we introduced 20% of nodes to which we call *hubs*. A *hub* is a reliable node (where P_{hub}= 0). Additionally, links that connect a hub to any other node are almost fully reliable, *i.e.*, we set the message loss probability for these links to 10^{-4}.

We have performed experiments in two distinct scenarios. In the first scenario, no resource optimization is employed and only the spanning tree is constructed by taking into consideration the reliability of nodes and links. In the second scenario, we introduce the resource optimization mechanism, to optimize the use of available quotas of messages in both protocols. This allows to show the benefits extracted from each of the two complementary techniques and also from their combination.

4.1 RASM Benefit

Without Resource Optimization

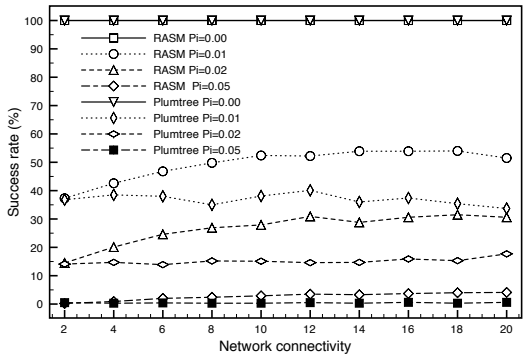
Our goal is to evaluate the reliability of spanning trees implicitly built by Plumtree and RASM. We express the reliability of these spanning trees using the reachability probability as defined in Equation 1 (Section 3.3). Therefore, we have disabled the resource optimization described in Section 3. In Figure 3 (a), (b) and (c), we respectively assign an approximative message loss probability of link L_i to 0 (reliable links), 2% and 5%. Each curve represents an approximative process crash probability P_i: 0, 1%, 2%, and 5% respectively.

As shown, in Figure 3 (a) in a reliable environment (L_i=0 and P_i=0), there is no difference between RASM and Plumtree since both of them provide a 100% reliability. As soon as we inject unreliability to the environment configuration RASM achieves better reliability than Plumtree. Such a reliability difference vary with the environment configuration. For the same process crash probability P_i, this difference increases as the message loss probability L_i increases. Contrary, for the same message loss probability L_i, this difference decreases as the crash probability P_i increases. Regarding the network connectivity, while Plumtree reliability remains almost constant, whereas RASM reliability is able to increase as the network connectivity increases.

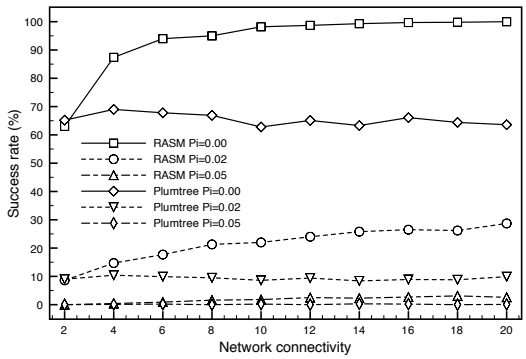
Indeed, as the node degree increases, the reliability of the spanning tree also increases. As the number of links in the overlay increases, more links associated to hubs are available. Therefore, RASM, unlike Plumtree, is able to leverage in such links to build a more resilient spanning tree.

[4]Consider that a tree T is composed of a set of links $\{l_1, l_2, ...\}$. The number of times a message m is retransmitted over link l_i is denoted by $\vec{m}[i]$. Intuitively, this value should be at least 1.

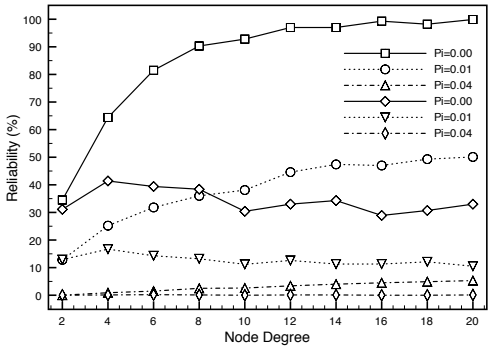
[5]Available in: `http://dcg.ethz.ch/projects/sinalgo`.



(a) $L_i = 0$



(b) $L_i \in [0, 2\%]$



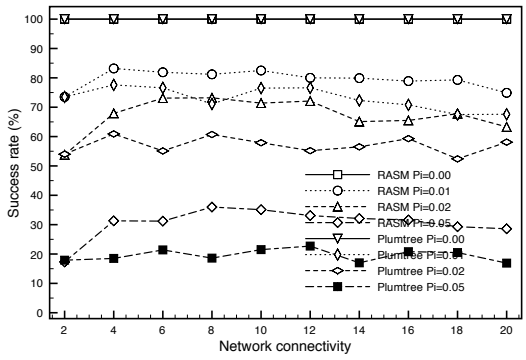
(c) $L_i \in [0, 5\%]$

Figure 3. Plumtree vs RASM reliability without Resource Optimization

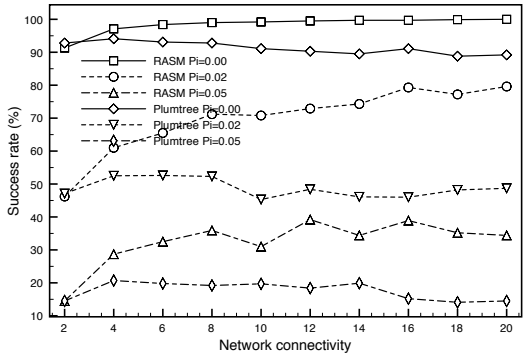
With Resource Optimization

Figure 4, shows the reliability provided by both Plumtree and RASM after the optimization of quota of messages at the disposal of each process. Here we vary the environment configuration as in the previous experiments. As explained in Section 3.3, we assume that the quota of messages for one packet diffusion at each process is equal to its connectivity. This ensures that each process is able to send at least one message per stream packet to each direct neighbor. By varying the network connectivity, we consequently vary the quota of messages. As we simulate a regular topology such a connectivity is the same for all processes which allows us to interpret our results and

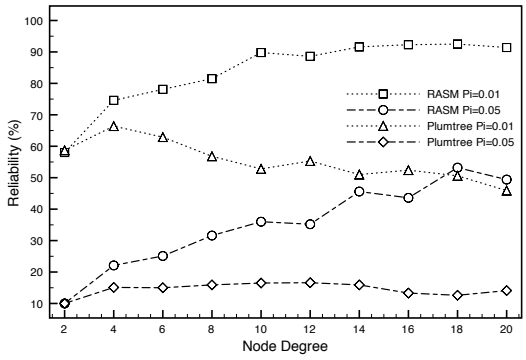
better understand the behavior of our algorithm[6].



(a) $L_i = 0$



(b) $L_i \in [0, 2\%]$



(c) $L_i \in [0, 5\%]$

Figure 4. Plumtree vs RASM reliability after quota optimization

Figure 4 (a) shows the reliability provided by both Plumtree and RASM while assuming a reliable links, i.e., L_i=0. In this configuration RASM and Plumtree offer almost the same reachability probability. Here, the number of switches performed by RASM in order to improve the reliability is small since whenever a duplicated message is received from a neighbor, the alternative new path used by this duplicate includes at least one additional node. Consequently the reachability probability provided by a duplicated message is (most of the time) lower than the one provided by the path used to disseminate the first received copy. Indeed, the reachability probability of

[6]Thus at each execution, all processes have the same quotas of messages to propagate a packet.

the path used by the duplicated message takes into account at least one more process crash probability which makes the decision of switch less frequent. Unless the previous path is fairly unreliable even counting only crash probabilities (P_i) which explains the minor advantage of RASM over Plumtree. Regarding the network connectivity, the reliability provided by both Plumtree and RASM, in this configuration decreases as the network connectivity increases. When this latter is equals to 2 (the topology is a ring), the overlay trees of both approaches are similar to a line in which a maximum of two individual quotas is unused (those of leaves). In this topology, almost all individual quotas are used to send a packet through one link which improves considerably the reliability.

As shown in Figure 4 (c) & (b), as soon as we inject some measure of unreliability to links, RASM and Plumtree act differently. This is due to their different overlay tree structures, that are built by taking into consideration different aspects of the network configuration. In Plumtree, the behavior remains almost the same: the generated tree has high degree in internal nodes dividing their quotas in several portions while losing the leaves quotas. This effect increases as the network connectivity increases. Contrary, in RASM, the tree structure selects the most reliable paths which are improved as the choice of links is larger, i.e., as the network connectivity increases.

4.2 RASM Cost

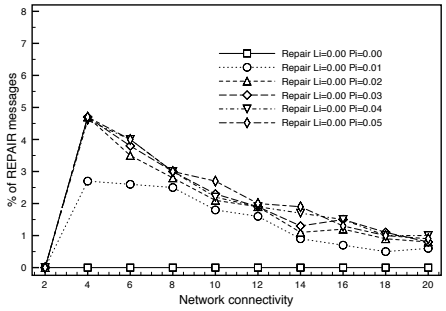
We now provide some results concerning the overhead imposed by our additional repair mechanism that ensures the connectivity of the spanning tree, despite the need to perform adaptations to its topology, in order to increase its reliability.

Figure 5 shows the percentage of REPAIR messages produced by our protocol. By percentage we refer to the ratio of REPAIR messages in relation to all messages sent during the simulation. In a reliable environment (L_i=0 and P_i=0; Figure 5 (a)) no switch are performed to select the most reliable paths to the source, thus no REPAIR messages are required. As the ranges of the loss messages and crash probabilities get larger as the percentage of REPAIR messages increases.

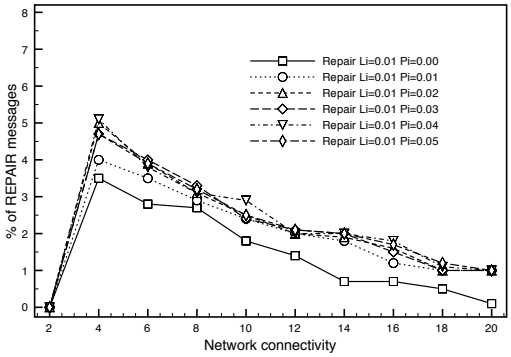
When the network connectivity is equal to two, few links are connecting the system nodes, thus the tree includes almost all links in the system. In this case no switch is performed since no alternatives paths are available. As soon as additional paths become available (e.g. when the network connectivity is above two), switches start to be performed and consequently some REPAIR messages are sent. As the average network connectivity increases the percentage of sent REPAIR messages decreases. Indeed, with a small network connectivity (e.g., network connectivity=4), packets are initially diffused through a deep tree with most of the nodes connected to source through a long path. In this case any component reliability can make the difference in the overall tree. As the network connectivity increases the tree becomes shorter and closer to a star topology where most of the nodes are directly connected to the source. Thus most of the nodes are already likely to rely on the most reliable path to the source. Hence few switches allow to improve the tree reliability, which results in less REPAIR messages being sent.

4.3 Rounds to build the tree

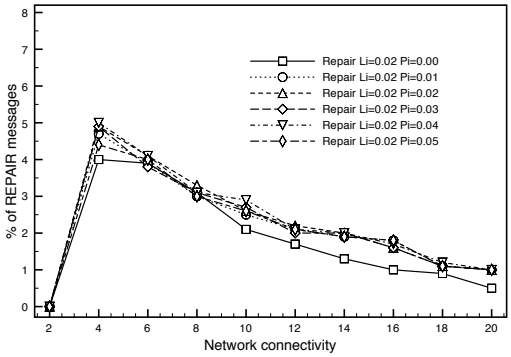
Figure 6 shows the number of rounds needed by both RASM and Plumtree to stabilize their trees. As described before, our simulations are performed in rounds. In each round, each node forwards packets received for the first time in the previous round to nodes in its *eagerPushPeers* (except to the node from which it received those packets). New packets are diffused by the source each 50 rounds starting from the round 1. We then measure the number of rounds required to ensure the convergence of our protocol. By convergence, we mean that the tree does not change during a certain number of consecutive rounds. In our simulation, if the tree does not change during 100 consecutive rounds we assume that it has became stable and we capture the round in which it was changed for the last time, being this the number of rounds required by our protocol to converge. Indeed, the number of rounds needed to build a tree reflects the number of forward steps needed for this. As shown in Figure 6 the number of rounds depends on the network connectivity, hence the number of forward steps required to reach all nodes. When the network connectivity is low the number of required rounds is higher, which reflects the number of forward steps needed to reach every node in the system. As the network connectivity increases the number of rounds needed to converge decreases. This is the reason that explains the small number of steps that are required that for all nodes to be reached and become part of the spanning tree. RASM is therefore able to build the spanning tree in only a



(a) $L_i = 0$



(b) $L_i \in [0, 1\%]$



(c) $L_i \in [0, 2\%]$

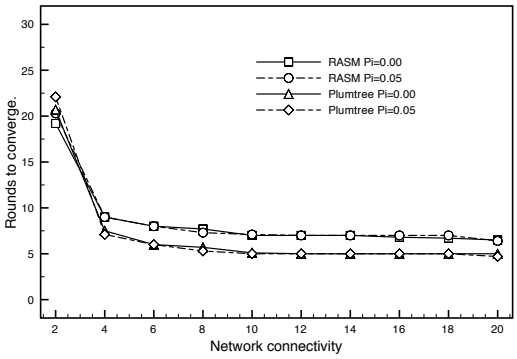
Figure 5. The percentage of REPAIR messages imposed by RASM

few additional rounds when compared with Plumtree. This is due to the switch mechanism of RASM that ensures the higher reliability of the resulting spanning tree.

5 Related Work

There exist several works that propose multicast solutions on top of overlay networks.

FloodTrail [16] is a resource location technique that, similar to our approach, builds a spanning tree on top of an unstructured overlay. MON [17] also enables the embedding of a short-lived spanning tree on top of an unstructured overlay. however, none of these works addresses the problem of fault tolerance nor the reliability of the resulting spanning tree.



$L_i \in 5\%$

Figure 6. Plumtree vs RASM rounds to build tree

Bayeux [11] and Scribe [18] are examples of protocols for application-level multicast that leverage in the routing infrastructures of DHT's. However, Bayeux required the nodes at the root of spanning trees to maintain global information concerning all receivers which limits its scalability. On the other hand, although Scribe is fault-tolerant and it provides a mechanism to handle root failures, it only provides best-effort guarantees and does not address the issue of offering strong reliability.

GoCast [13], is a protocol that also embeds a spanning tree in an unstructured overlay network for efficient message dissemination. GoCast can be seen as a complementary work to our own, as it main focus is on the maintenance of a proximity-aware random overlay, while our work main focus is how to efficiently create and maintain a highly reliable embedded spanning tree on top of the random overlay.

6 Conclusion and Future Work

In this paper we proposed RASM an algorithm for large-scale streaming that operates by implicitly defining a spanning tree on top of a low cost peer-to-peer overlay network. Unlike previous works that propose efficient multicast solutions on top of overlay networks, RASM is able to take into consideration constraints in the execution environment namely in terms of physical properties of the nodes (processing capacity, memory, and available bandwidth) but also the reliability of nodes and links that form the peer-to-peer network.

Experimental results presented in the paper show that the reliability of spanning trees produced by a prototype of RASM is higher than spanning trees built without taking into consideration such environmental constraints. Moreover, we have shown that the overhead imposed by our solution is not significant. RASM presents itself as a promising way to develop reliable and scalable multicast and streaming solutions for the Internet.

As future work we will evaluate RASM in additional and richer settings, taking into consideration additional metrics such as the global latency and reliability of the multicast protocol. Furthermore, we will explore additional ways to leverage on the additional links that are available at the overlay network level to avoid scenarios in which nodes, that own highly reliable links with their neighbors, become somewhat overloaded as a result of their links having a higher probability of becoming part of the spanning tree constructed by RASM.

References

[1] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," *ACM Transactions on Computer Systems*, vol. 17, no. 2, May 1999.

[2] A.-M. Kermarrec, L. Massoulié;, and A. J. Ganesh, "Probabilistic reliable dissemination in large-scale systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 3, pp. 248–258, 2003.

[3] J. Leitão, J. Pereira, and L. Rodrigues, "Hyparview: a membership protocol for reliable gossip-based broadcast," in *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Edinburgh, UK, Jun. 2007, pp. 419–429.

[4] M. Allani, B. Garbinato, F. Pedone, and M. Stamenkovic, "Scalable and reliable stream diffusion: A gambling resource-aware approach," in *Proceedings of 26th IEEE Symposium on Reliable Distributed Systems (SRDS'2007)*, Beijing, CHINA, Oct. 2007, pp. 288 – 297.

[5] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in cooperative environments," in *Proceedings of ACM SOP'03, October 2003*, October 2003.

[6] L. Mathy, R. Canonico, and D. Hutchison, "An overlay tree building control protocol," in *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication*. London, UK: Springer-Verlag, 2001.

[7] Y. Cui, B. Li, and K. Nahrstedt, "ostream: asynchronous streaming multicast in application-layer overlay networks," *Selected Areas in Communications, IEEE Journal on*, vol. 22, no. 1, pp. 91–106, 2004. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1258118

[8] F. Liu, X. Lu, Y. Peng, and J. Huang, "An efficient distributed algorithm for constructing delay and degree-bounded application-level multicast tree," in *ISPAN '05: Proceedings of the 8th International Symposium on Parallel Architectures,Algorithms and Networks*. Washington, DC, USA: IEEE Computer Society, 2005.

[9] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. James W. O'Toole, "Overcast: Reliable multicasting with an overlay network," in *Proceedings of OSDI, October 2000*, October 2000.

[10] Y. Chu, S. Rao, and H. Zhang, "A case for end system multicast," in *Proceedings of ACM Sigmetrics, June 2000*, June 2000, pp. 1–12.

[11] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *Proceedings of NOSSDAV*, June 2001. [Online]. Available: citeseer.ist.psu.edu/zhuang01bayeux.html

[12] J. Leitão, J. Pereira, and L. Rodrigues, "Epidemic broadcast trees," in *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS'2007)*, Beijing, China, Oct. 2007, pp. 301 – 310.

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

[14] N. Carvalho, J. Pereira, R. Oliveira, and L. Rodrigues, "Emergent structure in unstructured epidemic multicast," in *Proc. of the Internacional Conf. on Dependable Systems and Networks (DSN)*, Edinburgh, UK, June 2007.

[15] B. Garbinato, F. Pedone, and R. Schmidt, "An adaptive algorithm for efficient message diffusion in unreliable environments," in *Proceedings of IEEE DSN'04 , June 2004)*, June 2004, pp. 507–516.

[16] S. Jiang and X. Zhang, "FloodTrail: an efficient file search technique in unstructured peer-to-peer systems," in *IEEE Globecom'03*, San Francisco, California, December 2003.

[17] J. Liang, S. Y. Ko, I. Gupta, and K. Nahrstedt, "MON: On-demand overlays for distributed system management," in *2nd USENIX Workshop on Real, Large Distributed Systems (WORLDS'05)*, 2005. [Online]. Available: http://cairo.cs.uiuc.edu/publications/paper-files/worlds05-jinliang.pdf

[18] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, "SCRIBE: The design of a large-scale event notification infrastructure," in *Networked Group Communication*, 2001, pp. 30–43. [Online]. Available: citeseer.ist.psu.edu/rowstron01scribe.html