QuoCast: A Resource-Aware Algorithm for Reliable Peer-to-Peer Multicast*

Mouna Allani Benoît Garbinato
University of Lausanne, Switzerland
{mouna.allani, benoit.garbinato}@unil.ch

Amirhossein Malekpour Fernando Pedone
University of Lugano, Switzerland
{malekpoa, fernando.pedone}@unisi.ch

***Abstract*—This paper presents QuoCast, a resource-aware protocol for reliable stream diffusion in unreliable environments, where processes may crash and communication links may lose messages. QuoCast is resource-aware in the sense that it takes into account memory, CPU, and bandwidth constraints. Memory constraints are captured by the limited knowledge each process has of its neighborhood. CPU and bandwidth constraints are captured by a fixed quota on the number of messages that a process can use for streaming. Both incoming and outgoing traffic are accounted for. QuoCast maximizes the probability that each streamed packet reaches all consumers while respecting their incoming and outgoing quotas. The algorithm is based on a tree-construction technique that dynamically distributes the forwarding load among processes and links, based on their reliabilities and on their available quotas. The evaluation results show that the adaptiveness of QuoCast to several contraints provides better reliability when compared to other adaptive approaches.**

***Index Terms*—large-scale systems; reliable streaming; resource awareness;**

I. Introduction

Many internet services, such as audio and video streaming, and peer-to-peer file exchange systems, rely on multicast. Consequently, their performance depends on the performance of the underlying multicast mechanism. Since IP multicast is facing several technical and commercial deployment issues, these internet services are increasingly relying on application-level multicast solutions [1], [3], [5], [6], [7], [8], [10], [11], [13], [14], [17] as an alternative to IP multicast. In this paper, we are interested in application-level multicast in peer-to-peer (P2P) cooperative environments. In such contexts, peers contribute resources in exchange for their ability to use the multicast service. Peers may want to have control over the percentage of resources (i.e., CPU, memory and bandwidth) they dedicate to that service, which is especially true when the user has limited resources or has to pay for them—in some Internet subscriptions, clients pay only for what they consume or have access to a bounded bandwidth. For the peers, defining a percentage of the resources dedicated to multicast gives them more freedom to control their resources. For the multicast service, however, it represents an additional difficulty in terms of *resource constraints*. A further complication for the multicast service is that the environment might be unreliable, that is, nodes and communication links may fail, unexpectedly ceasing their operation or dropping messages. Given these constraints, an effective solution should use the available resources sparingly and account for node crashes and message losses.

In this paper, we introduce QuoCast, a resource-aware solution to the reliable multicast problem. QuoCast accounts for CPU, memory, and communication constraints. CPU and bandwidth resources constraints at each peer are modeled as a bounded capacity of receiving and sending data; the memory resource limit of each peer implies that each one has a partial view of the system, including a set of peers in its neighborhood. Each process has two quotas representing its reception and sending capabilities. The main goal of our approach is to maximize the probability that each packet reaches all consumers, given these quotas. Our multicast algorithm is based on a tree overlay that dynamically distributes the propagation load among processes and links, based on their reliabilities and available quotas. Our tree construction technique builds a routing overlay covering the whole system, but based only on the partial view each process has about its neighborhood. This tree construction technique is inspired from the one described in [1] taking into account components reliability and only resources-limitations related to sending messages. Thus, QuoCast can be seen as a generalization of the solution proposed in [1] in the sense that it adds the adaptiveness to the limitation a process may have to receive messages. As we will see from our evaluation, this generalization brings some interesting gain in reliability.

We quantify the advantages of QuoCast by comparing its performance with other adaptive approaches taking into account a subset of the constraints considered by QuoCast. We also show that QuoCast is not very sensitive to the scope of the partial view used by processes to build the global distribution tree, attesting the scalability of the approach.

This paper is organized as follows. Section II describes our model and the problem we aim to solve. Section III presents a preliminary version of QuoCast in a context where each node has global knowledge about the system; this allows us to focus on our tree construction technique. Section IV extends this result to environments where each node has only a partial view of the system. The results of our performance evaluation are presented in Section V while Section VI describes related work. Section VII concludes the paper.

* This research is funded by the Swiss National Science Foundation, in the context of Project number 200020-120188.

II. System model and 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 processes can crash and links can lose messages with a certain probability. More formally, we model the system's 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. Process crash probabilities and message loss probabilities are modeled as *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 one computation step and L_j as the probability that link l_j loses a message during one communication step.

Furthermore, we consider a set of physical constraints related to the limited hardware resources or to the dedicated percentage arbitrary fixed by the peer itself.[1] To capture these limitations, we define for each process p_i a fixed number of messages it can receive, its *in-quota*, and a fixed number of messages it can send, its *out-quota*, denoted respectively inq_i and $outq_i$. These quotas are a translation of both the percentage of CPU and memory a peer is willing to dedicate to forward a message and the download/upload limit of the ISP of the peer, which might be further limited by the percentage of that bandwidth the peer is willing to dedicate to the multicast. While $outq_i$ captures the percentage of CPU, memory and the upload bandwidth p_i dedicates to the multicast, the in-quota inq_i represents the dedicated CPU, memory and download bandwidth at process p_i. Associated to processes $p_i \in \Pi$, we then define $InQ = (inq_1,\ inq_2,\ ...,\ inq_n)$ as the set of individual in-quotas inq_i and $OutQ = (outq_1, outq_2, ..., outq_n)$ as the set of individual out-quotas $outq_i$. Based on these definitions, we then can say that the tuple $S = (\Pi,\ \Lambda,\ C,\ OutQ,\ InQ)$ completely defines the system considered in this paper.

In order to take into account the *limited memory constraints*, we further assume that each process has only a partial view of the system, meaning that its routing decisions can only be based on incomplete knowledge. Formally, the limited knowledge of process p_i is modeled as *distance* D_i, which defines the maximum number of links separating p_i from any other node in its known subgraph. That is, distance D_i implicitly defines the partial knowledge of p_i as $s_i = (\Pi_i,\ \Lambda_i,\ C_i,\ OutQ_i,\ InQ_i)$, with $\Pi_i \subseteq \Pi$, $\Lambda_i \subseteq \Lambda$, $C_i \subseteq C$, $InQ_i \subseteq InQ$, and $OutQ_i \subseteq OutQ$. In the remainder of this paper, any graph comprised of processes and links includes the corresponding configuration and quotas information.

Intuitively, the main question addressed in this paper is the following: *how to ensure a message multicast with the maximum of reliability, in spite of unreliable processes and links, and of the limited hardware resources and memory available at each process?*

[1]This captures the fact that the user behind a peer can voluntary limit the resources dedicated to the multicast.

III. Global QuoCast

For the sake of simplicity, we first describe the Global QuoCast algorithm, a variant of QuoCast relying on the assumption that each process knows the whole system S. This simplification allows us to focus, in this section, on our tree construction technique ensuring the maximum probability to reach all nodes. As we will detail later, such a probability is named the *reachability probability*. We assume that a view of S is given by an underlaying *Environment Modeling Layer* (EML) permitting to get an up-to-date view of the system as defined by our model. Explaining how the environment modeling actually works goes beyond the scope of this paper and can be found in [10]. Of course, assuming that each process has a global knowledge about the system does not ensure scalability, so we present a scalable version of QuoCast in Section IV.

A. Overview

QuoCast is defined by the primitives *multicast*(m) and *deliver*(m). To be adaptive to the environment changes, for each multicast QuoCast gets an up-to-date view of S by calling the EML primitive *getView()*. To diffuse a message m, by calling *multicast*(m) primitive, a source process p_i first builds a data delivery tree covering an up-to-date of view of the system S. Such a tree ensures the maximum success rate defined as the probability to reach all nodes, given in/out-quotas of messages at the disposal of each node. This is achieved by having p_i compute a *Maximum Probability Tree* (MPT) covering the system by calling the *mpt()* primitive (line 5). The MPT construction technique is presented in Section III-B. As we will see, MPT is at the heart of our approach as it ensures the reliability and the resources-awareness aspects of our solution.

```
1: uses: EML
   initialization:
2:    S ← ∅
3: procedure multicast(m)
4:    S ← EML.getView()
5:    T ← mpt(S, ({p_i}, ∅, {P_i}, {outq_i}, {inq_i}) )
6:    m⃗ ← optimize(T)
7:    propagate(m, T, m⃗)
8:    deliver m to p_i

9: upon receive (m, T, m⃗) for the first time do
10:   propagate(m, T, m⃗)
11:   deliver m to p_i

12: procedure propagate(m, T, m⃗)
13:   for all p_j such that link (p_i, p_j) ∈ E(T) do
14:      repeat m⃗[j] times : send(m, T_i, m⃗) to p_j
```

Algorithm 1. Global QuoCast

Since p_i is the source, the initial propagation tree passed as argument is simply composed of p_i and its associated information (failure probability P_i, its out-quota $outq_i$ and its in-quota inq_i). As discussed in Section III-B, the returned tree T maximizes the probability to reach everybody in S,

based on the available in/out-quotas. Process p_i then calls the $optimize()$ primitive, passing it T (line 6). This primitive is discussed in details in Section III-B. At this point, all we need to know is that it returns a propagation vector $\overrightarrow{m}$ indicating, for each link in T, the number of m retransmissions that should be sent through that link in order to maximize the probability to reach everybody in S. Finally, p_i calls the $propagate()$ primitive (line 7), which simply follows the forwarding instructions computed by $optimize()$. That is, it sends message m, together with the routing tree T and the propagation vector $\overrightarrow{m}$, to the p_i's children in T.

B. Maximum Probability Tree

The concept of *Maximum Probability Tree (MPT)* is at the heart of our approach, as it materializes the resource-awareness and the reliability aspect of our approach. Intuitively, an MPT maximizes the probability to reach all processes within a given scope by optimizing the in/out-quota use of these processes. Before describing how the $mpt()$ function given in Algorithm 2 builds up an MPT, we first need to introduce the notions of *reachability probability* and *reachability function*.

Reachability probability.: The *reachability function*, denoted by $R()$, computes the probability to reach all processes in some propagation tree T with configuration $C(T)$, given a vector $\overrightarrow{m}$ defining the number of retransmissions of a message that should transit through each link of T. We then define the probability returned by $R()$ as T's *reachability probability* given a vector $\overrightarrow{m}$. Equation 1 below proposes a reachability function that assumes that only links can fail by losing messages with a given probability, whereas processes are assumed to be reliable. Extending this function to process failures is straightforward [10].

$$R(T, \overrightarrow{m}) \;=\; \prod_{j=1}^{|\overrightarrow{m}|} 1 - L_j^{m[j]} \;\; with \;\; L_j \in C(T) \qquad (1)$$

Using $R()$, we then define the $maxR()$ function presented in Algorithm 2 (lines 8 to 10), which returns the maximum reachability probability for T. To achieve this, $maxR()$ first calls the $optimize()$ function in order to obtain a vector $\overrightarrow{m}$ that optimally uses the in/out-quotas available to processes in T. It then passes this vector, together with T, to $R()$ and returns the corresponding reachability probability.

The $optimize()$ function iterates through each process p_s in T and divides its out-quota $outq_s$ in a way that maximizes the probability to reach direct children of p_s (line 14 to 18). This function materializes the resource-awareness aspect as it sparingly divides the out-quota of each process p_s by taking into account the in-quotas of its children in T. For this, it allots messages of $outq_s$ one by one (line 16 to 18). That is, in each iteration step it chooses the outgoing link l_u from p_s that maximizes the gain in probability to reach all p_s's children in T, when sending one more message through l_u if the children p_u's in-quota inq_u permits it, i.e., $\overrightarrow{m}[u] < inq_u$ (line 17). Thus, the maximum number of messages assigned to a link l_u is the minimum of the $outq_s$, the out-quota of its start process p_s and the inq_u, the in-quota of its end process p_u. When all individual quotas have been allocated or all children in-quotas are exhausted, $optimize()$ returns a vector $\overrightarrow{m}$ that provides the maximum reachability probability when associated with T.

```
1: function mpt(S, T)
2:   while V(S) − V(T) ≠ ∅ do
3:     O ← {l_{j,k} | l_{j,k} ∈ E(S) ∧ p_j ∈ V(T) ∧ p_k ∈ V(S) − V(T)}
4:     let l_{u,v} ∈ O such that ∀l_{r,s} ∈ O :
5:     maxR( T ∪ p_v ) ≥ maxR( T ∪ p_s )
6:     T ← T ∪ p_v
7:   return T

8: function maxR(T)
9:   m⃗ ← optimize(T)
10:  return R(T, m⃗)

11: function optimize(T)
12:  let m⃗ : ∀l_j ∈ E(T), m⃗[j] is the number of messages to be sent through link l_j
13:  m⃗ ← (0, 0, ··· , 0)
14:  for all p_s ∈ V(T) do
15:    let Λ_s ⊂ E(T) : l_k ∈ Λ_s ⇒ (p_s, p_k) ∈ E(T)
16:    while (Σ_{l_k∈Λ_s} m⃗[k] < outq_s) ∧ (∃ l_v ∈ Λ_s | m⃗[v] < inq_v) do
17:      let m⃗_u : l_u ∈ Λ_s ∧ ∀_{t≠u} m⃗_u[t] = m⃗[t] ∧
         m⃗[u]+1 ≤ inq_u ∧ m⃗_u[u] = m⃗[u]+1 ∧ R(T, m⃗_u) − R(T, m⃗) is max
18:      m⃗ ← m⃗_u
19:  return m⃗
```

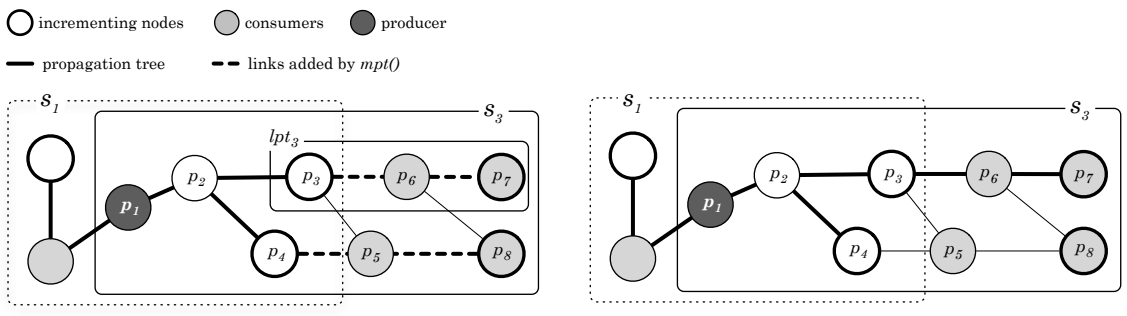
Algorithm 2. Maximum Propagation Tree

MPT building process.: We now have all the elements needed to present the MPT building technique carried out by the $mpt()$ function, given the system S and an initial propagation tree T. This function simply iterates until all processes in S but not in T have been linked to T, i.e., it only stops when T covers the whole system S (line 2 to 6). In each iteration step, the $mpt()$ function then adds the link that produces a new tree exhibiting the maximum reachability probability (line 5).

While being resource-aware by taking into account various resources limits captured into in/out-quotas, Global QuoCast does not take into account one of the main resources: memory. Thus, assuming that each process has a global knowledge about the system prevents our solution from being scalable. In addition, imposing to only one process the computation of the MPT covering the whole system represents a significant overhead. The next section precisely proposes a scalable variant of the QuoCast in which the MPT computation overhead is shared by nodes considering only a limited scope when calling the $mpt()$ primitive. Such an overhead is further reduced by a parameter marking nodes within a scope that will have to contribute to the tree computation. This parameter allows us to control the MPT computation overhead with respect to nodes available CPU resource.

IV. Scalable QuoCast

In this section, we assume that each process has a limited memory used to capture a partial knowledge about the system.



(a) before the return statement

(b) returned propagation tree

Fig. 1. Propagation tree incrementing.

The known view of each process p_i covers a certain width of its neighborhood. Such width is defined by a scope D_i, which defines the maximum number of links separating p_i from any other node in its known subgraph. To simplify the description, we assume that the scope D is the same for all processes, i.e., $\forall p_i, D_i = D$.

In the following we describe a solution, borrowed from [1], to make QuoCast scalable based on limited views about the system. To obtain these partial views, QuoCast relies also on EML. This layer defines the *getScope()* primitive, which returns to each process p_i an up-to-date view of its subgraph s_i defined by scope D. It also defines the *distance()* primitive returning the distance between any two nodes in s_i when called by p_i. By distance between two nodes, we mean the number of links in the shortest path separating these two nodes.

Contrary to global QuoCast our scalable solution consists in disseminating messages through a *propagation graph* generated in a decentralized manner by several processes rather than by only the source process. The idea is to define the propagation graph as a result of the spontaneous aggregation of several scope limited *propagation trees*. Each propagation tree is in turn the result of an incremental building carried out along the paths from the producer to the consumers. It is important to note that the resulting propagation graph itself might well not be a tree. This incremental building follows the MPT computation technique aiming to get a *propagation graph* close to the global MPT that we could build, if we had a global knowledge about the system. That is, processes contribute to the MPT computation based on their partial knowledge about the system. Those processes are defined by an *incrementing rate* parameter, noted r and representing a scope within D, i.e., $r \leq D$. When a source process p_i initiates a multicast, it builds the first *propagation tree* as the local MPT covering its known subgraph s_i defined by its scope D. This MPT is then incremented by each interior node p_r at a distance r from the source using the MPT construction technique. The *incrementing node* p_r is in charge to add to the just received *propagation tree* nodes in its view s_r that are not in the tree it received. This treatment is repeated by each interior node at a distance r from the last *incrementing node* ancestor in the received *propagation tree*.

Intuitively, r defines how often a propagation tree should be incremented as it travels through the propagation graph. The latter then spontaneously results from the concurrent and uncoordinated increments of propagation trees finding their ways to the consumers. While the scope D allows us to control the amount of memory at each process, r allows us to control the CPU resources. Indeed r defines the propagation graph computation overhead distribution. When r equals D such an overhead is carried by the minimum number of incrementing nodes. As r gets lower than D, the number of incrementing nodes becomes higher. This permits to distribute the computation load of the propagation graph among more processes. Such a ratio may reflect the available CPU at the incrementing nodes.

Figure 1 illustrates the propagation tree incrementing algorithm in a simple example. In this scenario, the distance defining the scope and the incrementing rate r are the same for all processes and equal to 2. Process p_1, the producer, builds a first *propagation tree* pt_1 covering its scope s_1; this tree is pictured in Figure 1 (a) using bold links. All nodes in pt_1 that are at a distance $r = 2$ from p_1 are *incrementing nodes*, which means they have to increment pt_1 when they receive it. Process p_3 being such a node, calls the $mpt()$ function, passing it pt_1 and its scope s_3. This function adds the dashed links pictured in Figure 1 (a) to pt_1 and returns the resulting Maximum Probability Tree (MPT); this MPT contains the local propagation tree rooted at p_3, i.e., lpt_3. This subtree routed at p_3 is then extracted from the MPT, merged with the initial propagation tree pt_1 and returned. Figure 1 (b) pictures the new propagation tree resulting from the above incrementing technique.

V. Performance evaluation

The performance of QuoCast was evaluated through a simulation model. In the evaluation that follows, each reported value reflects the success rate of the protocol after 1000 distinct executions. At each execution, a multicast of a stream packet is simulated. An execution is successful if a multicast packet reaches all nodes in the system. Note that a missed execution, i.e., where the stream packet does not reach all nodes, could be due either to the probabilistic behavior of links losing messages or due to our propagation graph not covering the whole system. As discussed in [1], this type of misses

can be repaired with a countermeasure inducing a negligible overflow of quotas. In this evaluation, we focus on misses due the probabilistic behavior of the links.

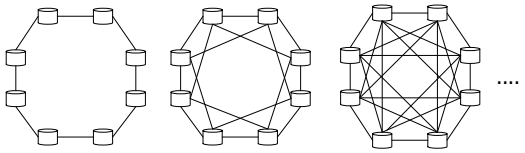
A. Evaluation setup

For simplicity, we consider link failures only, and assume that processes are reliable. We distinguish two types of processes in the experiments: regular processes and hubs. Hubs have twice the in-quota and the out-quota of regular processes and are connected to their neighbors through highly reliable links. Table 1 summarizes the range of values used in the experiments.

param	**process type**		**description**
	regular	**hub**	
–	80	20	number of processes
P_i	0	0	probability to crash
L_i	0.05..0.55	0.0001	probability to lose msg
D_i	D	D	scope
inq_i	inq	$2 \times inq$	in-quota
out_i	out	$2 \times out$	out-quota

TABLE I
SYSTEM PARAMETERS

Processes were organized in regular topologies of varying connectivity. We executed experiments starting with a ring (i.e., connectivity of 2, Figure 2(a)) and then incrementally augmented the connectivity, as shown in Figure 2, until we reached 20 neighbors per process. Notice that a ring provides a worst case analysis since messages have to traverse in the average half of the processes in the system to be delivered. In this topology, our tree construction technique provides its minimum advantage as the number of links in the system is the lowest making the choice of links to include in the tree limited.



(a) Connectivity=2 (b) Connectivity=4 (c) Connectivity=6

Fig. 2. Different topologies.

To facilitate the evaluation, we set the scope to be the same for all processes during an execution, i.e., $\forall p_i : D_i = D$. To avoid regular network configurations, we then defined 20% of processes to be hubs with the configuration described in Table 1.

B. Adaptiveness benefit

This section discusses the advantage of combining both resource and unreliability awareness when building the propagation tree, that is, it shows the benefit of our MPT construction technique. We compare our MPT to three relevant solutions. The first is inspired by the tree defined in Overcast [11]. Overcast is targeted at bandwidth-intensive applications. It defines a tree overlay that aims at maximizing the outgoing bandwidth by placing nodes as far as possible from the root (the source) without sacrificing bandwidth. The available bandwidth resource in Overcast is modeled as weights assigned to links. In order to adapt the Overcast tree construction technique to our model, we consider the link weight as the number of messages assigned to the link, calculated by dividing the node out-quota by the number of its outgoing links in the tree. Thus, when building the Overcast tree in our model, at each iteration we add the link through which we can assign the maximum number of outgoing messages without considering the in-quota of that link end-node.

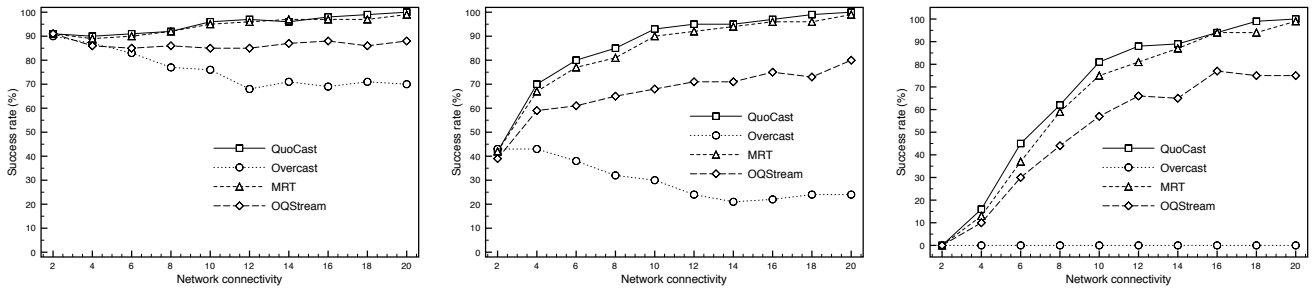
The second protocol is part of our previous work defined in [10] defining a reliable broadcast taking into account nodes failures probabilities (P_i) and links message loss probabilities (L_i). This broadcast solution is also based on a tree overlay named the *Maximum Reliability Tree* (MRT). This tree defines the most reliable tree of a known subgraph through which a message will be propagated. For fair comparison, we also apply our *Optimize()* function after defining this tree overlay. Thus, once the MRT is built all nodes out-quotas are distributed in way to maximize the advantage of this resource with respect to the reliability of components included in MRT.

The third one is the tree defined by the multicast protocol proposed in our previous work [1]. This protocol, which hereafter we refer to as *OQStream*, is also resource-aware but does not take into account the in-quota of processes, that is, when laying off a multicast tree, no limits are considered for the in-quota of processes; messages exceeding the in-quota of a process are simply lost. OQStream is a good baseline as experiments have shown that by itself it is better than traditional gossip techniques.

In what follows, we show the results provided by our solution when using these three tree structures: the Overcast tree, the MRT and the OQStream tree. When it comes to the limited knowledge each node has about the system, we assume that, in our strategy and the compared protocols, nodes has only a partial view. Based on this knowledge, we use our Gambling increment strategy in order to build a propagation graph covering the whole system while using the different tree build criteria.

In this comparison, we vary the network connectivity c, while fixing the incrementing rate r to 2 and the scope D defining the known subgraph of each process to 5. Figure 3 shows the benefit of QuoCast over the compared protocols when facing the same constraints for different L_i ranges with the $outq$ = 10 and inq = 2 messages. Hereafter, we interpret the most interesting observations we detected by comparing QuoCast based on MPT and QuoCast with each of its comparison tree.

QuoCast vs. Overcast: QuoCast provides a higher reliability when using MPT than when using the Overcast tree. Also, our approach has a different behavior when using the Overcast tree while varying the network connectivity. Indeed, as the connectivity increases more links in the system are created offering a larger choice of links to the MPT construction technique. While the MPT takes advantage to



(a) Li ∈ [0.05, 0.1] (b) Li ∈ [0.05, 0.25] (c) Li ∈ [0.05, 0.55]

Fig. 3. QuoCast *vs.* compared protocol reliability with $outq = 10$ and $inq = 2$

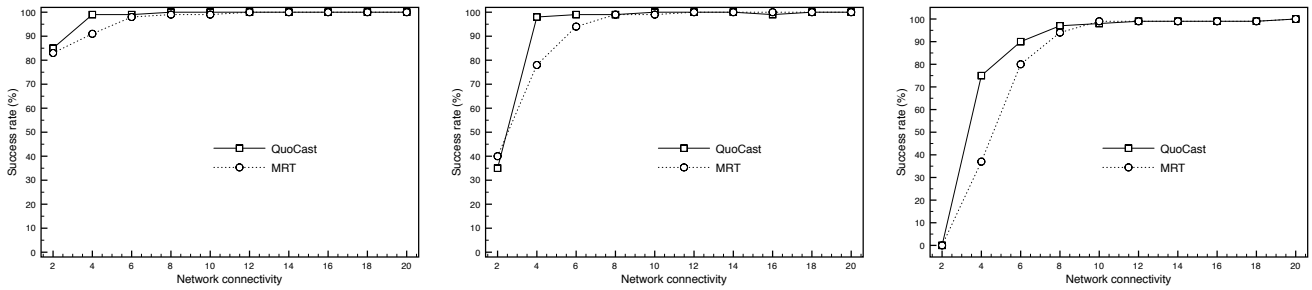
include a more reliable links, the Overcast tree moves away from the line structure which imposes more leaves and thus more lost out-quotas. These out-quotas would contribute to hide the environment unreliability if used. In addition, we noticed that varying the out-quota with the same in-quota (inq = 2) using Overcast tree does not improve the reliability. Indeed, aiming to use the maximum of outgoing bandwidth Overcast will try to assign the whole out-quota of one node to only one outgoing link. This allocation is however wasteful since the end-node of that link will not be able to deliver more than its defined in-quota, which lower than the sender out-quota. Thus the number of messages resulting from the difference between the sender out-quota and the receiver in-quota are simple lost.

QuoCast vs. MRT: As shown in Figures 3 and 4, the success rate of our approach is globally higher when using MPT than when using the MRT. In Figure 3, results using MPT (QuoCast) and using MRT are too close. Indeed, having the resulting MPT with an in-quota of 2 is too close to the tree built if we had to send only one message per link. Such a tree is the MRT. Thus, in case of limited resources, our MPT construction technique focusses on the components reliability as the MRT do. However, as shown in Figure 4, with a low connectivity (except the connectivity of 2 where the system is a ring) the difference between MPT and MRT is bigger when more in-quota are available at each node. For example, when the $L_i \in$ [0.05, 0.55], with the same out-quota $outq = 10$, the connectivity of 4 and the in-quota $inq = 4$, using MPT provides a success rate 40% better than when using MRT. Here, our MPT construction technique increases the overall reliability by taking advantage of available in/out resources and adapting the system reliability.

QuoCast vs. OQStream: In this section, we discuss the success rate of QuoCast and OQStream while varying the network connectivity and the reliability of links. For both algorithms, as the connectivity increases the success rate increases—as links are added to the network, the construction tree algorithm has more choices to select reliable paths. QuoCast has better success rate than OQStream, and its advantage increases as the range of L_i increases. Indeed, the improvement of QuoCast over OQStream relies on the profit gained by taking into account the in-quota, which avoids losing messages exceeding it by distributing the out-quota towards processes able to deliver the sent messages. The improvement of sending more messages through more links depends on those links reliability. That is, the higher the unreliability of one link, the higher the improvement when sending one more message through it. In terms of reachability probability computed using our *reachability function*, sending two messages instead of one through a link with 0.55 of unreliability, improves its reachability probability by approximately 25% ($(1-0.55^2)-(1-0.55) = 24.75\%$). On the other hand, sending two messages instead of one through a link with 15% of unreliability improves its reachability probability by 9% ($(1-0.1^2)-(1-0.1) = 9\%$). Thus, the improvement percentage of QuoCast on the final reachability probability depends on the links reliability. This improvement is approximately of 10% when $L_i \in [0.05 - 0.1]$ and approximately 30% when $L_i \in [0.05 - 0.55]$.

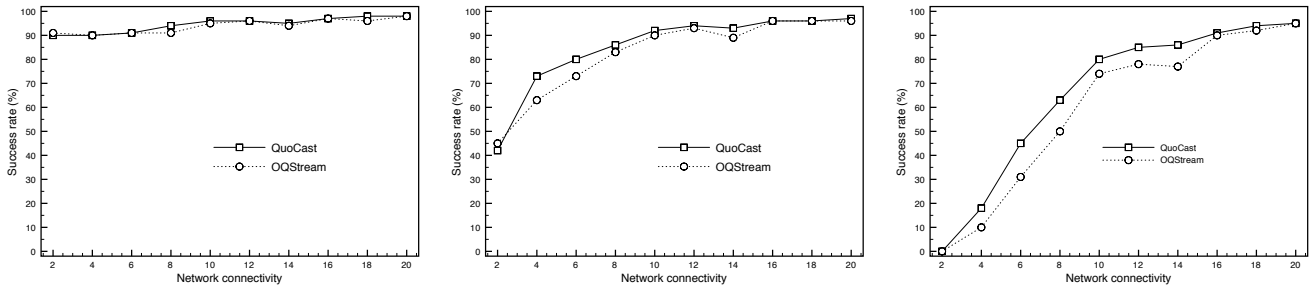
Figure 5, shows the same measurements comparing QuoCast and OQStream with $outq = 4$ and $inq = 2$ messages. Globally, QuoCast offers a higher success rate than OQStream. Note also that the difference between $outq$ and inq has a different impact on QuoCast and OQStream.

As shown in Figures 3 and 5, while QuoCast remains stable as the $outq$ increases, OQStream reacts negatively to large offer of this resource. In this case, QuoCast did all possible improvements starting from $outq = 4$ with respect to the in-quota constraint. Offering more outgoing messages, does not improve the global reliability as we remain limited by inq. OQStream, however, follows a different strategy when facing a large $outq$. Indeed, when building the tree in OQStream, the links reliability has no more impact when selecting a link to add to the tree being constructed. This impact is hidden by the big number of messages associated with any selected link. That is, whatever the link reliability, the reachability probability tends towards 1 from the first selected link thanks to the big number of messages associated to it. Thus, the reachability probability comparison that should exist between links candidate to increment the tree has no utility and stops at the first selected link. When facing the in-quota limit, the resulting tree reachability probability becomes a function of the reliability of the links included to the tree arbitrary. Obviously, this probability is improved as the reception quota inq gets bigger. When fixing the reception limit inq, as the available out-quota decreases, the selected links have more effect on the reachability probability when building the tree. Thus, the resulting tree includes the most reliable links, which



(a) $L_i \in [0.05, 0.1]$ (b) $L_i \in [0.05, 0.25]$ (c) $L_i \in [0.05, 0.55]$

Fig. 4. QuoCast *vs.* MRT protocol reliability with $outq = 10$ and $inq = 4$



(a) Li $\in$ [0.05, 0.1] (b) Li $\in$ [0.05, 0.25] (c) Li $\in$ [0.05, 0.55]

Fig. 5. QuoCast *vs.* OQStream reliability with $outq = 4$ and $inq = 2$

consequently increases the success rate.

C. QuoCast scalability

In this section, we discuss the impact of the partial knowledge each process has about the system, defined by the scope D. As shown in Figure 6, the scope has no relevant impact on the success rate of the protocol. This confirms the scalability aspect of our solution ensuring the same success rate even based on limited view of the system. This is guaranteed by the incrementing process of scalable QuoCast building a propagation graph covering the whole system that is very close to the tree built if we had a global knowledge at each process.

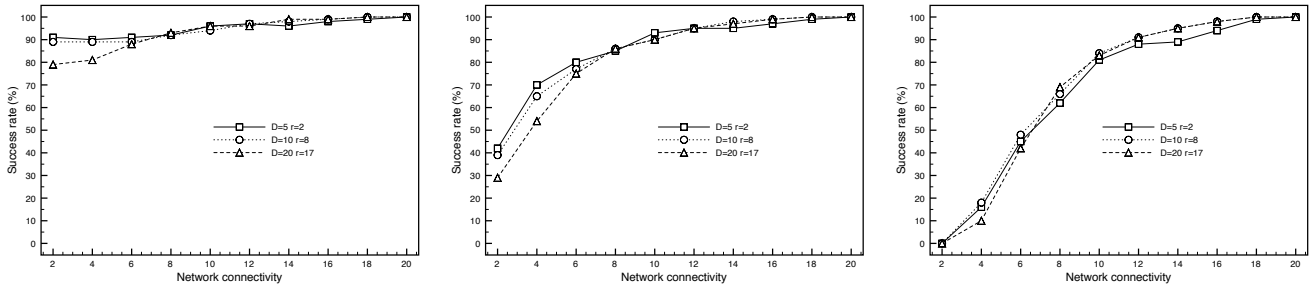
VI. Related Work

Several application-level multicast systems based on a tree have been proposed in the literature [5], [7], [11], [13]. Some of them define a multicast tree that balances the propagation load by optimizing the resources using, notably Narada [7] and Overcast [11]. Others, also deal with scalability by limiting the knowledge each process has about the system [5], [13]. Yet, other systems aim at increasing reliability with respect to packet loss [2], [18]. Our approach differs from these systems in that it targets the three goals simultaneously. Our propagation structure is built collaboratively by distributed processes using their respective partial views of system. Reliability is accounted for by each process when building its local tree. Finally, resources constraints are considered when defining how to forward packets along the propagation graph.

Narada [7] builds an adaptive $mesh$ that includes group members with low degrees and with the shortest path delay between any pair of members. A standard routing protocol then is run on the overlay mesh. This work differs from ours by considering latency as the main cost related to links. While using the probing to change links in order to optimize the mesh, Narada does not take into account the loss probability of added or retrieved links. Furthermore, Narada nodes maintain a global knowledge about all group participants. In comparison, we take process and link failure probabilities into account and maintain local information only.

In [9] and [12] the authors show how to implement a gossip-based reliable broadcast protocol in an environment in which processes have a partial view of the system membership. Our protocols as well do not require processes to know all the system members or the topology connecting them. In addition to [9] and [12], our approach takes reliability properties of processes and links into account in order to ensure reliable multicast. Reducing the number of gossip messages exchanged between processes by taking the network topology into account is discussed in [15] and [16]. Processes communicate according to a pre-determined graph with minimal connectivity to attain a desired level of reliability. Similarly to our approach, the idea is to define a directed spanning tree on the processes. Differently from ours, process and link reliabilities are not taken into account to build such trees.

As already mentioned, this paper extends an initial contribution [1] sharing with this paper some awareness about a subset of constraints. In [1], we focus on the sending capacity of a process as the main treatment consuming available resources. In this paper, we argue that both the reception and sending are resources consumers in a streaming service. Both this paper and the initial contribution ensure scalability by assuming that



(a) Li ∈ [0.05, 0.1] (b) Li ∈ [0.05, 0.25] (c) Li ∈ [0.05, 0.55]

Fig. 6. Varying scope D with $outq = 10$ and $inq = 2$

each process has a limited view about the system.

Finally, our strategy shares some design goals with broadcast protocols such as [10]. Both rely on the definition of a criteria for selecting the message diffusion graph. In our strategy, however, we strive to both decrease packet loss and balance the forwarding load. The notion of *reachability probability* of a tree is presented in [10] to define the *Maximum Reliability Tree* (MRT). This tree defines the most reliable tree of a known subgraph through which a message will be propagated. In this work, we define the reachabiliy probability of the streaming differently, by considering local knowledge only. The approaches illustrate a tradeoff in streaming algorithms: while the protocol in [10] can lead to the optimum propagation tree, it requires global topology knowledge. Our current algorithm relies on local knowledge but may result in the sub-optimum propagation tree.

VII. Conclusion

In this paper, we proposed resource-aware streaming solution to support multicast in a reliable and scalable way. Our contributions includes: (1) A *tree construction technique* to define a multicast tree overlay with the maximum reliability: the *Maximum Probability Tree* (MPT). We, formally, defined such a reliability as a function of the overlay included components reliability and the available resources to use. (2) To ensure scalability, we proposed an *incrementing algorithm* to built a propagation graph trying to cover the whole system while based on partial view about the system at each process. Such a graph is very close to the global MPT built if we had global knowledge about the system. This graph ensures the same streaming reliability whatever is the width of the partial knowledge about the system each process has. (3) By performance evaluation, we show that the reliability of our approach is very promising and out-performs an initial contribution [1] proved to outperforms gossip-based algorithm [4] when subject to similar environment constraints. The current variant of QuoCast is clearly less adequate to real-time applications. As an improvement, we aim to include to our performance goals the low-latency property so that to adapt QuoCast to applications where very low latency is important, e.g., video conferencing.

References

[1] 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)*, pages 288 – 297, Beijing, CHINA, October 2007.

[2] J. G. Apostolopoulos. Reliable video communication over lossy packet networks using multiple state encoding and path diversity. In *Visual Communications and Image Processing, 2001*, January 2001.

[3] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, volume 32, pages 205–217, New York, NY, USA, July 2000. ACM Press.

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

[5] M. Castro, P. Druschel, A.M. 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] Yatin Chawathe, Steven McCanne, and Eric A. Brewer. RMX: Reliable multicast for heterogeneous networks. In *INFOCOM*, pages 795–804, Tel Aviv, Israel, March 2000. IEEE.

[7] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM Sigmetrics, June 2000*, pages 1–12, June 2000.

[8] Helder D. A. and Jamin S. Banana tree protocol, an end-host multicast protocol. Technical report, University of Michigan, 2002.

[9] P. Eugster, R. Guerraoui, S. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. In *Proceedings of the 2001 International Conference on Dependable Systems and Networks (DSN '01)*, pages 443–452, July 2001.

[10] 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)*, pages 507–516, June 2004.

[11] J. Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and Jr James W. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proceedings of OSDI, October 2000*, October 2000.

[12] A.-M Kermarrec, L. Massoulie, and A.J. Ganesh. Probabilistic reliable dissemination in large-scale systems. Technical report, Microsoft Research, June 2001.

[13] D. Kostic, A. Rodriguez, J. Albrecht, A. Bhirud, and A. Vahdat. Using random subsets to build scalable network services. In *Proceedings of USITS, March 2003*, March 2003.

[14] Li Lao, Jun-Hong Cui, and Mario Gerla. Toma: A viable solution for large-scale multicast service support. In *NETWORKING*, pages 906–917, 2005.

[15] M.-J. Lin and K. Marzullo. Directional gossip: Gossip in a wide area network. Technical Report CS1999-0622, University of California, San Diego, June 1999.

[16] M.-J. Lin, K. Marzullo, and S. Masini. Gossip versus deterministic flooding: Low message overhead and high reliability for broadcasting on small networks. Technical Report CS1999-0637, University of California, San Diego, November 1999.

[17] Laurent Mathy, Roberto Canonico, and David Hutchison. An overlay tree building control protocol. In *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication*, pages 76–87, London, UK, 2001. Springer-Verlag.

[18] T. Nguyen and A. Zakhor. Distributed video streaming with forward error correction. In *Packet Video Workshop, 2002*, 2002.