Probabilistic Multicast

Patrick Th. Eugster* Rachid Guerraoui
Distributed Programming Laboratory
Swiss Federal Institute of Technology, CH-1015 Lausanne, Switzerland

Abstract

Gossip-based broadcast algorithms have been considered as a viable alternative to traditional deterministic reliable broadcast algorithms in large scale environments. However, these algorithms focus on broadcasting events inside a large group of processes, while the multicasting of events to a subset of processes in a group only, potentially varying for every event, has not been considered. We propose a scalable gossip-based multicast algorithm which ensures, with a high probability, that (1) a process interested in a multicast event delivers that event (just like in typical gossip-based broadcast algorithms), and that (2) a process not interested in that event does not receive it (unlike in broadcast algorithms).

1 Introduction

Context. Many *content-based publish/subscribe* (cf. [2]) applications require scalable and reliable dissemination of events inside a large process group (e.g., 10 000 processes). In such applications, processes *publish* events and *subscribe* to events. Each subscriber describes its individual interests through specific criteria based on properties of events (e.g., an attribute b of events must be of value b_0). The destination subset for a given event is hence defined individually and implicitly by the interests associated with the processes in the group. The motivation of our work is to devise algorithms to disseminate events in such applications with reasonable reliability guarantees despite crash failures of processes. To scale, such algorithms must clearly have a "selective" dissemination flavor, and can hardly assume any global *membership* or subscription knowledge.

Gossip-based algorithms. While "traditional" *Reliable Broadcast* (à la [6]) and also *network-level protocols* deriving from IP Multicast have turned out to scale insufficiently for many of the various applications requiring reliable dissemination of events [9], *gossip-based algorithms* (a class of *probabilistic algorithms*) seem more appealing for large scale settings, by trading the strong reliability guarantees offered by deterministic approaches against very good scalability properties, yet still achieving a "high degree of reliability". Following the example given by *Probabilistic Broadcast* (*pbcast*) [1], the design of many recent general-purpose broadcast algorithms has been based on gossips, e.g., *Reliable Probabilistic Broadcast* (*rpbcast*) [11], *Directional Gossip* [8], *Lightweight Probabilistic Broadcast* (*lpbcast*) [3].

From broadcast to multicast. To our knowledge, these gossip-based algorithms implicitly assume that *all* processes in a group are interested in *all* events. Of course, filtering (i.e., verifying the contents of these events for their conformance with the interests of processes) can be performed upon reception before a possible delivery. Though events might very well be delivered with a high probability to all interested processes, they are received with the same probability by, potentially numerous, non-interested processes. Clearly, such a flooding technique is not adequate when many events are only of interest for say half the processes in the overall group.

Alternatively, one can also modify an existing gossip-based broadcast algorithm to perform the filtering before gossiping, i.e., when gossiping, a process forwards an event only to interested processes. This leads indeed to a more scalable solution, where only concerned processes are involved in an algorithm run. However, such a *genuine multicast* [4], would clearly offer a limited reliability. Indeed, a crucial intermediate process might not be interested in an event, leading to the isolation of interested processes, unless we make the (rather unrealistic) assumption that every process knows every other process and also its precise interests.

A third alternative consists in using broadcast algorithms by mapping possible destination subsets of a large group to smaller, possibly overlapping, broadcast groups [7] (e.g., all processes interested in an event whose attribute b has

*Current affiliation: Distributed Computing and Systems Research Group, Chalmers University of Technology, S-412 96 Göteborg, Sweden value b_0 are contained in a broadcast group $G_b^{b_0}$). By doing so, and especially when supporting interests which are not restricted to discrete values (e.g., a floating point attribute $c > c_0$), one can however end up with a large number of groups (2^n at maximum in an application involving n processes in total). But, above all, establishing these individual broadcast groups requires a global knowledge of the interests of processes, and might have to be repeated every time the composition of the overall group (interests, processes) varies.

Contributions. We present here a gossip-based multicast algorithm, called *Probabilistic Multicast* (*pmcast*), which deals with the case of multicasting events only to *subsets* of the processes in a large group. Our *pmcast* algorithm relies on a specific orchestration of processes in the group, which can be viewed as a "superimposition" of spanning trees. The resulting compound "tree" exploits (1) the topology of the network, and at the same time (2) commonalities in the interests of processes, and adapts easily to variations in the composition of the group. The scalability in terms of network resources, inherent to gossip-based algorithms, is combined with (3) a notion of membership scalability, and further improved by (4) mainly involving those processes in the dissemination of an event that are effectively interested in that event.

We present an analysis which accounts for all four factors, and illustrate our claims of reliability and scalability through simulation results. We exhibit limits of *pmcast* in extreme cases of a multicast setting (very small destination subsets), and we discuss the compromises that were made when tuning *pmcast* to remedy possible performance losses in these extreme cases.

Roadmap. The remainder of this paper is organized as follows. Section 2 introduces the membership scheme underlying *pmcast*. Section 3 presents the main *pmcast* algorithm. Section 4 describes an analysis of our *pmcast* algorithm, and Section 5 presents simulation results and discusses tuning techniques. Section 6 concludes this paper with some final remarks.

2 Membership

This section first gives an intuitive description of the membership scheme underlying the tree-based multicasting in *pmcast*. Then we model the "tree" more rigorously for the analysis presented in Section 4.

2.1 Overview

A *pmcast* group is split into *subgroups* (e.g., *subnetworks*), and for each such subroup a set of processes, called *delegates*, is selected to represent that subgroup. The parent node of such a subgroup is populated by these delegates, and merged with the parent nodes of a set of neighbor subgroups. From this "compound node", a set of delegates is again chosen. This select/merge-procedure is performed recursively, leading to a form of compound spanning tree. Figure 1 illustrates a simple example.

In such a graph, which we henceforth simply refer to as "tree", scalability is improved by having every process only know processes corresponding to the nodes on its path up to the root, and reliability is improved by having each node encompass a set of processes, which remain all in their child nodes as well.[1] This is illustrated in Figure 1, where the "view" of the processes sharing prefix x_0 is limited to the shaded processes, while the complete tree can expand arbitrarily to the right.

Interactions between processes vary by their "distance", and processes have a complete knowledge of their respective immediate neighbors, but only a decreasing knowledge about more "distant" processes (e.g., processes in distant LANs). Delegates that appear closer at the root are known by more processes in the group than processes at an increased depth. Nevertheless, all processes have membership views of comparable sizes, since every process has a view of its subgroup for every tree depth.

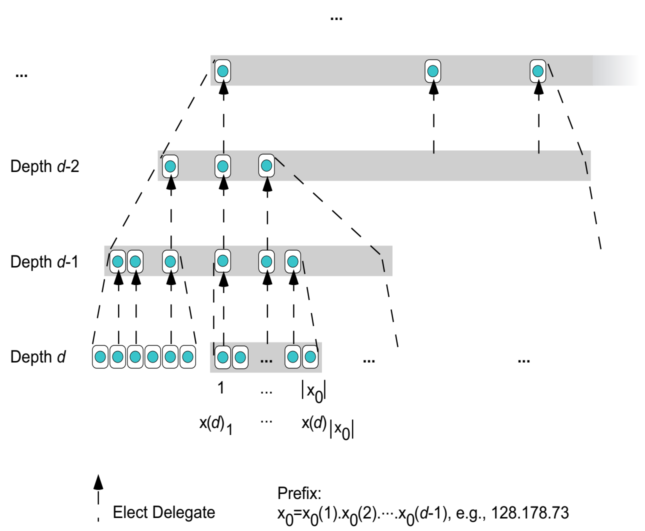

Figure 1. Electing Delegates; $R = 3$

2.2 Membership Model

The tree scheme underlying our approach relies strongly on a notion of "distance" between any two processes. In-

[1]In classic graph theory, one would rather use the term *pseudograph* for such a graph with *loops*, i.e., nodes connected to themselves.

tuitively, a distance approximates the communication delay between two processes.

Distances and addresses. An approach to measure the "distance" between two processes in an asynchronous system could consist in using an *average communication delay* between these processes. However, in unstable phases, such values are difficult to determine. We thus base the decision of which processes out of a subgroup are to be considered as neighbors, and also as prioritized (to become delegates), on the *addresses* of those processes, more precisely, on the distances between those addresses.

This notion of "distance" can be *approximated* by network addresses (since these do not always reflect the effective topology of the network), but can as well be simulated by associating *logical addresses* with processes. Irrespective of how these addresses are determined, we will in the following simply consider such addresses as sequences of values, of the following form:

$$x(1).\cdots.x(d), \\ \forall i \; 1 \leq i \leq d, 0 \leq x(i) \leq a_i - 1 \qquad (1)$$

This notation can represent different kinds of addresses, like IP or DNS addresses (in the latter case, the order would have to be inverted). The maximum number of different addresses, and thus the maximum number of processes, is given by $\prod_{1 \leq i \leq d} a_i$.

Though processes can be colocated on the same host, or even in the same process, we will consider here, for the sake of simplicity, that there is one process per host in a considered group. Indeed, the last components of an address could easily be used to express a port number. For instance, to cover all possible IP addresses, one could choose $d = 4$ and $a_i = 2^8 \; \forall i$; or $d = 11$ and $a_i = 2^4 \; \forall i$ to include 2^{12} port numbers (without considering IP addresses and port numbers reserved for special purposes, e.g., IP addresses 224.0.0.0 to 239.255.255.255 reserved for IP Multicast groups).

Prefixes. A "partial" address, like $x(1).\cdots.x(i-1)$ ($1 < i \leq d$; $\emptyset$ for $i = 1$) is called a *prefix* (following the terminology adopted for the Internet) of depth i. Such a prefix denotes a subgroup (e.g., a subnetwork). All processes sharing a prefix belong to the corresponding subgroup (see Figure 1). In other terms, there might be several addresses $x(i).\cdots.x(d)$ that can be appended to a same prefix to denote concrete processes in the group.

The distance between two processes is inverse proportional to the length of their longest common prefix: if the longest prefix that two processes share is of depth i, then their distance is given by $d - i + 1$. Also, they belong to the same subgroup of depth i of the tree. A distance of 0 would mean that the two processes share the same address and are thus equivalent.

Electing delegates. All processes which share a given prefix $x_0 = x_0(1).\cdots.x_0(d-1)$ form a group of depth d. The number of such processes, i.e., the *branch factor* for the corresponding node in the tree, at the moment of observation, is denoted $|x_0(1).\cdots.x_0(d-1)|$. For any given prefix $x(1).\cdots.x(d-1)$, $|x(1).\cdots.x(d-1)| \leq a_d$.

Of the $|x_0|$ processes, R delegates are chosen deterministically by all processes sharing x_0, e.g., by taking the R processes with the smallest addresses (we assume that $\forall \; x(1).\cdots.x(d-1)$, $|x(1).\cdots.x(d-1)| \geq R$, i.e., every populated group of depth d contains at least R processes.

R represents a *redundancy factor*, which has no relationship with the notion of redundancy observed in interests of processes. R represents the number of delegates that are elected to represent a subgroup, and is best chosen such that $R > 1$, in order to improve the reliability of the membership.

In general, $\forall x = x(1).\cdots.x(i-1), |x| \leq a_i$ represents the number of different $x(i)$ that can be appended to x to denote an existing prefix, or in other words, the number of populated subgroups of x.

Together with R delegates for each other of the $|x_0(1).\cdots.x_0(d-2)|$ neighbor trees, the R delegates of $|x_0|$ form a group of depth d-1. Recursively, any prefix $x_0(1).\cdots.x_0(i-1)$ $(1 \leq i \leq d)$ is shared by $|x_0(1).\cdots.x_0(i-1)| \leq a_i$ subgroups with a different $x(i)$, each represented by R delegates. Together, these form a group of depth i. At depth 1 of the tree, there are $|\emptyset| = |x_0(1).\cdots.x_0(i-1)|_{i=1}$ subgroups, each of these again being represented by R processes.

Individual knowledge. A process characterized by an address $x_0 = x_0(1).\cdots.x_0(d)$ knows for each of its prefixes $x_0(1).\cdots.x_0(i-1)$ $(1 \leq i \leq d)$ a number of subgroups equal to $|x_0(1).\cdots.x_0(i-1)|$, and for each of those subgroups, R delegates. Furthermore, at depth d, that process knows all $|x_0(1).\cdots.x_0(d-1)|$ immediate neighbor processes. Thus, the total number of processes known by a given process $x_0(1).\cdots.x_0(d)$ is

$$|x_0(1).\cdots.x_0(d-1)| + \sum_{i=1}^{d-1} R \, |x_0(1).\cdots.x_0(i-1)| \qquad (2)$$

where a delegate of a given depth i is also taken into account at any depth $i+1$ (since it appears in all successive depths). Also, the knowledge of the process itself is considered (depending on the depth of the considered process, between 1 and d occurrences). By promoting a process as delegate, the knowledge of that process does not increase, but it becomes known by more processes.

2.3 Membership Management

The membership views are loosely coordinated between processes. Every process periodically sends information about its view of the tree to a subset of the group.

Membership information. To that end, each process maintains a table for each depth, representing the view (mainly processes and their interests) of the process at that depth. Figure 2 gives a possible composition of the views of a set of processes represented by IP addresses sharing the prefix 128.178.73 (e.g., corresponding to the subgroup of depth d in Figure 1). Interests are described for a single type of events, encompassing attributes b (integer), c (float), e (string), and z (integer). The absence of a criterion for a given attribute is interpreted as a wildcard.

Membership information updating is based on *gossip pull*. Every line in every table has an associated timestamp (omitted in Figure 2 for presentation simplicity), representing the last time the corresponding line was updated. Periodically, a process randomly selects processes of a table and gossips to those processes. A gossip carries a list of tuples (line, timestamp) for every line in every table. The receiver compares all the timestamps to its own timestamps, and updates the gossiper for all lines in which the gossiper's timestamps are smaller. Membership information can be piggybacked when gossiping events, or in the absence of such events, can be propagated with dedicated gossips.

Joining. When a process decides to join a group, it needs to know at least one process that is already in that group. Latter process contacts the "lowest" delegates it knows that the joining process will have. This is made recursively, until the most immediate delegates of the new process have been contacted. Once these neighbors have been contacted, they transmit their views of the group to the new process. An a priori knowledge of a range for the expected size of the tree can be very useful to adjust parameters of the tree, such as its depth d.

Leaving and Failures. The same immediate neighbors are also involved when a process leaves. A process wishing to leave informs a subset of its closest neighbors. These remove the leaving process from their views, and this information successively propagates throughout the concerned subgroup through subsequent gossips.

For the purpose of detecting the failure of processes, every process keeps track of the last time it was contacted by its most immediate neighbor processes.

Interests. A line of a table of depth $i + 1$ is constructed by compacting the table of depth i corresponding to that

View of Depth 1

Infix	Interests	Postfixes
3		2.230.23, 18.2.78, 188.203.99
18	$z > 10000$	12.2.183, 12.34.24, 180.37.217
128	$b > 0$	3.2.230, 18.120.2, 56.12.234

View of Depth 2 (Prefix = 128)

Infix	Interests	Postfixes
3	$b > 3$, $10.0 < c < 220.0$	2.230, 18.2, 188.203
18	$b = 2$, $e =$"Bob" $\vee$ "Tom"	120.4, 122.2, 180.37
56	$b > 1$, $c > 155.6$	12.24, 18.220, 173.3
178	$b > 0$	41.21, 73.3, 88.10

View of Depth 3 (Prefix = 128.178)

Infix	Interests	Postfixes
41	$b = 3$, $z = 42000$	21, 23, 24
73	$b > 0$, $c > 20.0$,	3, 17, 19
88	$b > 5$, $e =$"Tom"	10, 13, 78
98	$b > 4$, $20.0 < c < 35.0$, $z < 23002$	15, 17, 128
110	$b > 6$, $z > 45320$	1, 6, 7

View of Depth 4 (Prefix = 128.178.73)

Infix	Interests
3	$b = 2$, $c > 40.0$, $z = 20000$
17	$b = 5$, $c > 53.5$
19	$b > 1$, $20.0 < c < 30.0$, $z \leq 50000$
116	$b > 0$, $c > 20.0$
119	$b = 4$, $2000 < z < 30000$
124	$b = 3$, $c \geq 35.997$
223	$b = 2$

Figure 2. View of a Tree of Depth 4

subgroup (see Figure 2), by performing the following operations:

Interest regrouping: To represent the interests of all processes in a table, the interests of the respective processes must be regrouped. This is done in a way which avoids redundancies, i.e., not just by simply forming a conjunction of the individual interests, but by reducing the complexity of the interests both in terms of memory space and in terms of evaluation time.

Process count: The total number of processes at any depth is very useful for several kinds of heuristics. In particular, the number of processes at a given depth enables the estimation of the number of gossip rounds necessary such that all concerned processes at that depth have received the considered event.

Delegate selection: Delegates have to be chosen based on a deterministic characteristic, since all processes in a subgroup of depth i must decide on the same set of delegates without explicit agreement. Currently, delegates with the smallest addresses are chosen. Alternatively, one could take into consideration other criteria associated with processes, like their resources in terms of computing power or memory, or also the nature of their interests, to reduce the amount of "pure forwarding" of delegates.

3 Probabilistic Multicast

This section presents *pmcast*, our gossip-based multicast algorithm which is based on the tree described in the previous section. We first overview below some characteristics of our *pmcast* algorithm before describing it in more detail.

3.1 Overview

As opposed to many gossip-based broadcast algorithms (e.g., *pbcast* [1], *rpbcast* [11]), in which events are first broadcast by a "conventional" deterministic best-effort algorithm and gossips are used to exchange digests of received events, *pmcast* uses gossips primarily to propagate the events themselves. This avoids the systematic sending of events to uninterested processes.

More precisely, when multicasting an event, *pmcast* follows the underlying tree, by gossiping *depth-wise*, starting at the root. By mapping tree depths to the network topology, the expensive crossing of boundaries between remote (sub)networks only occurs a "reasonable" number of times, and if necessary. Indeed, at each depth, a given event is only gossiped about among interested processes (or delegates of interested processes). Hence, at each depth of the tree, only interested subgroups are *infected* with that event, i.e., they receive the event.

Note that by considering delegates, which represent interested processes yet are themselves not interested, as *susceptible* (i.e., to be infected), *pmcast* is not a genuine multicast in the sense of [4]. In most cases however, the fraction of infected processes comes close to the fraction of interested processes.

3.2 Algorithm

Figure 3 presents our *pmcast* algorithm. Since events are first propagated at the root of the tree, from where they move down to increased depths, an effective gossip, besides conveying an event, also includes the depth at which the event is currently being multicast, as well as the computed *matching rate* (i.e., the rate of interested processes) at that depth with respect to the considered subgroup.

Receiving: Upon reception of a gossip, the information about the depth is used to place the event in the corresponding gossip buffer. The event is only delivered by the receiving process if it is effectively of interest for that process (reflected by $\lhd$). To ensure that the event passes from one depth to the next, it is crucial that a process at depth i gossips a received event in any depth $i' > i$, and thus also remains in the view of any of these subsequent depths.

Multicasting: A process would only be involved in the gossiping, and hence only require gossip buffers, from the "upmost" depth in the tree at which it appears down to its subgroup of depth d. However, when PMCAST-ing, a process takes part in the entire gossip procedure at all depths, especially at the root. This increases the probability that an event is well propagated in contrast to a simple scheme where a new event would simply be sent once to a subset of the delegates forming the root. Also, since the membership is dynamic, a process can be "bumped up" at any moment, or conversely, can be "dropped".

Note however, that if an event is only of more "local" interest, which manifests when the only processes at the root of the tree who are interested in that event are delegates of the same subtree, (possibly the PMCAST-ing process' own delegates), the event can then be immediately passed to the next depth, until a depth has been found where not only a single subtree is interested.

3.3 Parameters

Our *pmcast* algorithm relies on a set of parameters, in particular the number of processes effectively interested in a given gossiped event at a given depth of the tree. We below describe how the algorithm obtains these parameters.

Bound gossiping. Our *pmcast* algorithm strongly relies on a "passive" form of garbage collection, achieved by limiting the number of rounds that an event is gossiped about in a given subgroup at a considered depth of the tree. To that end, the expected number of rounds necessary to infect all interested processes in a given subgroup at any depth of the tree, represented by the function $T(\#processes, fanout)$ in the algorithm in Figure 3, is approximated to inherently limit the life-time of an event. The estimation of the number of rounds is necessary at every depth of the tree, since this number depends on the number of interested processes at the considered depth.

Expected number of rounds. According to Pittel [10], the total number of rounds necessary to infect an entire group of size n (large), in which every infected process tries to infect $F > 0$ other processes at every round, is given by the following expression:

$$T(n, F) = log\, n \left(\frac{1}{F} + \frac{1}{log\,(F+1)}\right) + c + O(1) \quad (3)$$

```
initialization
view[1-..$d$]
gossips[1..$d$] $\leftarrow \emptyset$

task GOSSIP                                   {every $P$ milliseconds}
    for all depth $\in$ [1..$d$] do
        for all (event, rate, round) $\in$ gossips[depth] do
            if round $< T(|\text{view[depth]}| \cdot R \cdot \text{rate}, F\cdot\text{rate})$ then
                round $\leftarrow$ round + 1
                dests[1..$F$] $\leftarrow \emptyset$
                repeat $F$ times              {choose potential destinations}
                    dest $\leftarrow$ RANDOM(view[depth]-dests)
                    dests $\leftarrow$ dests $\cup$ {dest}
                    if event $\lhd$ dest then
                        SEND(event, rate, round, depth) to dest
            else
                gossips[depth] $\leftarrow$ gossips[depth] $\setminus$ {(event, rate, round)}
                if depth $< d$ then
                    gossips[depth+1] $\leftarrow$ gossips[depth+1] $\cup$ {(event,
                    GETRATE(depth+1, event), 0)}

upon RECEIVE(event, rate, round, depth): do
    if $\nexists$ depth $\in$ [1..$d$] $\exists$ (event, ..., ...) $\in$ gossips[depth] then
        gossips[depth] $\leftarrow$ gossips[depth] $\cup$ {(event, rate, round)}
        if event $\lhd$ thisprocess then
            PMDELIVER(event)

upon PMCAST(event): do
    gossips[$d$] $\leftarrow$ gossips[$d$] $\cup$ {(event, GETRATE($d$, event), 0)}

function RANDOM(from)                         {choose random processes}
    return dest $\in$ from

function GETRATE(depth, event) {event matching rate at this depth}
    hits $\leftarrow$ 0
    for all dest $\in$ view[depth] do
        if event $\lhd$ dest then
            hits $\leftarrow$ hits + 1
    return $\frac{\text{hits}}{|\text{view[depth]}|\ R}$
```

Figure 3. Probabilistic Multicast Algorithm

In the case of *pmcast*, this formula is used to estimate the number of gossip rounds in every subgroup at every depth i. Hence, the number of processes and the fanout for the above expression are both conditioned by the matching rate, which can be obtained by a process by dividing (1) the number of processes interested in the considered event by (2) the total number of processes at that depth.

Number of interested processes: The number of processes interested in a particular event at a given depth i can be measured by matching that given event against all the interests of the processes for the respective subgroup at that given depth i. This is a costly operation, but is only performed by a maximum of R processes at each depth w.r.t. that subgroup, since this is the maximum number of processes infected initially in a subgroup (the matching rate is propagated with the event). The root represents an exception, in that only the process effectively PMCAST-ing the event will perform this matching.

Total number of processes: The size of a subgroup at a considered depth i is given by the number of delegates forming that depth (with respect to the subgroup of the considered process), which is given by multiplying the number of different subgroups of depth $i+1$ in the view of depth i, i.e., the number of lines in the corresponding view table (noted $|view[i]|$ in Figure 3), by the number of delegates for each subgroup, R.

Environmental parameters. Environmental parameters, such as the probability of message loss, or the probability of a crash failure of a process, are to be considered when computing the expected number of rounds necessary to spread an event. These are however more difficult to approximate. Like in most gossip-based algorithms, where simulations or analytical expressions enable the computing of "reasonable" values for parameters such as the number of times a gossiped event is forwarded, choosing conservative values is the best way of ensuring a good performance. For simplicity, these parameters have not been added (in Figure 3 or Expression 3), but will appear in the analysis presented in the next section.

4 Analysis

In this section, we give an analysis of *pmcast*, describing the *spreading of a multicast event* over gossip rounds. This analysis characterizes the reliability of *pmcast*.

4.1 Analysis Model

For our formal analysis, we consider a group composed of n processes, and we observe the propagation of a single event, in which every process in the group is interested with a probability of p_d. We assume that the composition of the group does not vary during the run (consequently n is constant).

Assumptions. The stochastic analysis presented below is based on the assumption that processes gossip in synchronous rounds, and there is an upper bound on the network latency which is smaller than a gossip period P. Note however that this analysis does not rely on the assumption that the underlying system is synchronous, nor does the algorithm force the system to behave so.

P is furthermore constant and identical for each process, just like the fanout F. We assume furthermore that failures are stochastically independent. The probability of a network message loss is $\epsilon > 0$, and the probability of a process crashing during a run is considered to be $\tau = f \ / \ n$, where f is the number of processes crashing during that run. We do not take into account the recovery of crashed processes, nor do we consider Byzantine (or arbitrary) failures.

Joint interest. The processes in the considered group are orchestrated in a tree as presented in Section 2. In such a tree, the total number of processes that a delegate at a particular depth i with address $x_0(1).\cdots.x_0(d)$ represents (provided that it is delegate at that level, of course) is given by

$$\| x_0(1).\cdots.x_0(i-1) \| = \sum_{x(i).\cdots.x(d-1)} |x_0(1).\cdots.x_0(i-1).x(i).\cdots.x(d-1)| \quad (4)$$

We consider an event as being significant only for a fraction p_d (matching rate) of processes in the entire group. In other terms, there is a probability p_d that a considered event (p_d varying potentially for each event) is of interest to any given process, and when considering a group of n processes, only $n\ p_d$ participants are expected to be interested in the observed event. Hence, a delegate of depth i, on behalf of the represented processes, is expected to be interested in a given event with the following probability:

$$p_i = 1 - (1-p_d)^{\|x_0(1).\cdots.x_0(i-1)\|} \quad (5)$$

It is easy to see that, besides the special case where $p_d = 1$, p_i is always strictly bigger than p_d for any depth below.

Regular tree. For our analysis, we presuppose a "regular" tree, i.e., for any process in the group, all prefixes derived from the address $x = x(1).\cdots.x(d)$ of that process have the same number of subgroups, which we denote by a. This does not presuppose that our algorithm only works for regular trees (any tree can however be captured by a bigger regular tree).

$$\begin{aligned} &\forall x, i\ x = x(1).\cdots.x(d), 1 \le i \le d \\ &|x(1).\cdots.x(i-1)| = a \ \le\ a_i \end{aligned} \quad (6)$$

The total number of processes in the group is straightforwardly given by $n = a^d$. Furthermore, we consider that, with respect to a given event, the processes interested in that event are uniformly distributed over the entire group (which is of course not required by the algorithm itself). Hence, the probability p_i that a process at depth i manifests interest in a given event, possibly on behalf of processes it represents, is given by:

$$p_i = 1 - (1-p_d)^{a^{d-i}} \quad (7)$$

4.2 Event Propagation in a "Flat" Group

We first analyze the spreading of an event in a "flat" group of processes, i.e., a group of processes in a tree of depth 1. We then show how the established Markov chain can be used to compute the expected number of infected processes in a subgroup of depth i ($1 \le i \le d$) after gossiping at that given depth.

Fraction of infected processes. The probability p that a considered gossiped event is received by a given process, is a conjunction of several conditions, namely that (1) the considered process is effectively chosen as target, (2) the gossiped event is not lost in transit, and (3) the target process does not crash. In the expression given below, according to the algorithm presented in Figure 3, the effective group size and the effective fanout are both conditioned by p_d:

$$p(n\ p_d, F\ p_d) = \left(\frac{F\ p_d}{n\ p_d - 1}\right)(1-\epsilon)(1-\tau) \quad (8)$$

Accordingly, $q(n\ p_d, F\ p_d) = 1 - p(n\ p_d, F\ p_d)$ represents the probability that a given process is *not* reached by a given infected process.

We denote the number of processes infected with a given event at round t by s_t, $1 \le s_t \le n\ p_d$. Note that when the event is injected into the group at round $t = 0$, we have $s_t|_{t=0} = 1$. Given a number j of currently infected processes, we are now able to define the probability that exactly k processes will be infected at the next round ($k-j$ susceptible processes are infected during the current round). The resulting homogenous Markov chain is characterized by the following probability p_{jk} of transiting from state j to state k ($1 \le j \le k \le n\ p_d$, 0 if $j > k$):

$$p_{jk}(n\ p_d, F\ p_d) = P(n\ p_d, F\ p_d)[s_{t+1} = k | s_t = j] = \binom{n\ p_d - j}{k - j}(1 - q(n\ p_d, F\ p_d)^j)^{k-j} q(n\ p_d, F\ p_d)^{j(n\ p_d - k)} \quad (9)$$

The distribution of s_t, $t > 0$ ($s_0 = 1$ if $k = 1$, 0 otherwise) can then be computed recursively ($1 \le k \le n\ p_d$):

$$P(n\ p_d, F\ p_d)[s_t = k] = \sum_{j=k/(1+F)}^{k} P(n\ p_d, F\ p_d)[s_{t-1} = j] p_{jk}(n\ p_d, F\ p_d) \quad (10)$$

Expected number of rounds. Pittel's model [10] does not consider the possibility of losing events between a gossiper and a (potential) destination. In our case, only $F\ p_d(1-\epsilon)(1-\tau)$ processes among $n\ p_d$ are expected to be infected at a given round by a gossiper, leading to the following expression:

$$\begin{aligned} &T_f(n\ p_d, F\ p_d) = \\ &\quad T(n\ p_d(1-\epsilon)(1-\tau), F\ p_d(1-\epsilon)(1-\tau)) \end{aligned} \quad (11)$$

Note however that since Pittel's formula is valid for large groups, it only offers useful results as long as $n\ p_d$ is still large (see Section 5.3).

4.3 Event Propagation in a Tree

The above result can be used to express at each depth, for an infected node, the probability of having a certain number of child nodes infected after gossiping at that depth.

View sizes. Every process must know the delegates of every depth along its path from the root, as shown in Figure 1. According to Equation 2, every process in a regular tree knows the same number of processes for the different depths of the tree. More precisely, a process characterized by $x(1).\cdots.x(d)$, knows:

$$m_i = \begin{cases} R\,|x(1).\cdots.x(i-1)| \;=\; R\,a & 1 \leq\; i \;<\; d \\ |x(1).\cdots.x(i-1)| \;=\; a & i \;=\; d \end{cases} \tag{12}$$

This adds up to $m \;=\; \sum_{i=1}^{d} m_i \;=\; R\;a\;(d-1)+a \;\in\; O(d\;R\;n^{1/d})\;\forall\;d \;\geq\; 2$. This value decreases as d increases, and finds a minimum in $d \;=\; log\;n$, which is however not reached as long as $R \;\geq\; 3$ [2].

Expected number of rounds. Based on Expression 7 and Expression 11, the expected number of rounds necesssary for *pmcast* to infect all interested processes in a regular tree can be approximated by the sum of the rounds spent at each depth of the tree:

$$T_{tot} \;=\; \sum_{i=1}^{d} T_i \;=\; \sum_{i=1}^{d} T_f(m_i\,p_i, F\,p_i) \tag{13}$$

This expression is pessimistic, by neglecting the very fact that every interested subgroup, except the topmost one, starts with an expected number of infected processes which is bigger than 1. In fact, in most cases, all delegates are already infected, and we can obtain a more precise expression by subtracting at each depth the rounds necessary to get from 1 to R infected processes. Based on this, the number of rounds necessary to infect an entire group can be shown to be the same without a tree, as in an arbitrary-depth tree [2]; namely $T_f(n, F)$. In terms of rounds necessary to multicast events, the tree does not have a considerable impact on the event dissemination procedure.

Fraction of infected processes. The expected number of infected processes after gossiping at depth i is hence given by the following, again pessimistic (we do not consider the possibility that the subgroup initially comprised more than one infected process) expression:

$$E[s_{T_i}] \;=\; \sum_{j=0}^{m_i\;p_i} P(m_i\,p_i, F\,p_i)[s_{T_i} = j]j \tag{14}$$

We are now able to compute the probability that a "node" (i.e., a set of processes, e.g., delegates representing the same subtree) of depth i is infected after gossiping at that depth (provided the corresponding subgroup of depth $i-1$ was initially infected):

$$r_i \;=\; 1 - \left(1 - \frac{E[s_{T_i}]}{m_i\,p_i}\right)^{\frac{m_i}{a}} \tag{15}$$

For all depths, except the lowest one, a "node" of depth i means a subgroup of depth i (that is, its R delegates for that depth). At the leaves of the tree, a "node" refers to a single process. $r_i|_{i=d}$ simply resumes to $E[s_{T_d}]/a\,p_d$, the expected fraction of processes infected when gossiping in a group of depth d.

Provided that $g_{i-1} = j \leq a^i\;p_{i-1}$ entities were infected at depth $i-1$ (in total), the probability of ending up with $g_i = k$ entities infected at depth i ($1 \leq i \leq d$) is given by recursion as follows (for $k \leq j\;a\,p_i$, 0 otherwise):

$$p_{ijk} = P[g_i = k|g_{i-1} = j] \;=\; \binom{j\;a\,p_i}{k} r_i^k (1-r_i)^{j\;a\;p_i - k} \tag{16}$$

Finally, we can compute the probability of having k infected entities at depth i ($0 < i \leq d$, $g_0 = 1$ if $k = 1$, 0 otherwise):

$$P[g_i = k] \;=\; \sum_{j=k/(a\;p_i)}^{a^i\;p_{i-1}} P[g_{i-1} = j]p_{ijk} \tag{17}$$

Expected number of infected processes. We are able to express the expected number of totally infected processes in the group based on the expected number of infected entities $E[g_i]$ after gossiping in a subgroup at depth i ($1 \leq i \leq d$):

$$\prod_{i=1}^{d} E[g_i] \;=\; \prod_{i=1}^{d} r_i\;a\,p_i \tag{18}$$

The expected reliability degree can be simply obtained by dividing the upper expression by the number of effectively interested processes, $n\;p_d$.

5 Simulation Results

We have simulated *pmcast* through successive rounds, and have observed the spreading of a single event. We give some of these results below to illustrate the good scalability and reliability properties of *pmcast*, before discussing its behavior in extreme cases, as well as a simple way of tuning the algorithm for these cases. More results are given in [2].

5.1 Reliability

Figure 4 conveys the high probability with which a process interested in a multicast event delivers that event. Figure 5 on the other hand, shows the low probability with which a process that is not interested in a multicast event receives that event.

Figure 4 however also illustrates the limitations of stochastic approaches underlying epidemiology, and hence gossip-based algorithms, which namely reflect the interest in very large populations. Indeed, Pittel's asymptote gives extremely good results for events which are of interest for a large fraction of processes in the group. On the other hand, "very small" values for p_d, are less well captured by this asymptote. In fact, the computed expected number of rounds increases first with a decreasing p_d, before becoming 0 for $p_d = \frac{1}{n}$. Accordingly, the reliability degree decreases with a decrasing p_d, since the asymptote gives less accurate information towards such "small" values. Even with an approximation which performs better for such small values, this problem can be observed. This loss in reliability was expected, and there are several ways of counteracting it (see Section 5.3).

Note that the interpretation of "very small" depends on the considered subgroup size, $m_i\, p_i$. Since p_i becomes bigger for a depth i closer to the root, and hence, the number of potentially interested processes increases, making smaller depths less prone to this type of undesired effect.

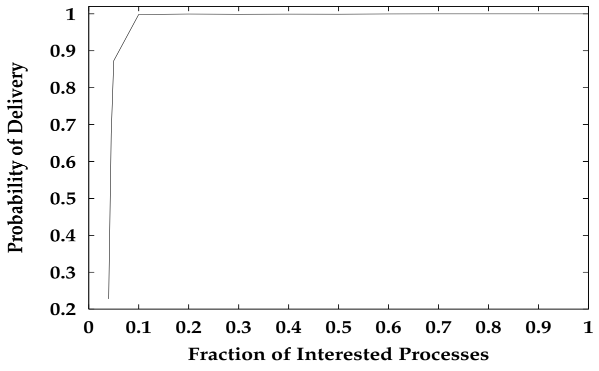

Figure 4. Infected Interested Processes; $n \approx 10000\ (a = 22), d = 3, R = 3, F = 2$

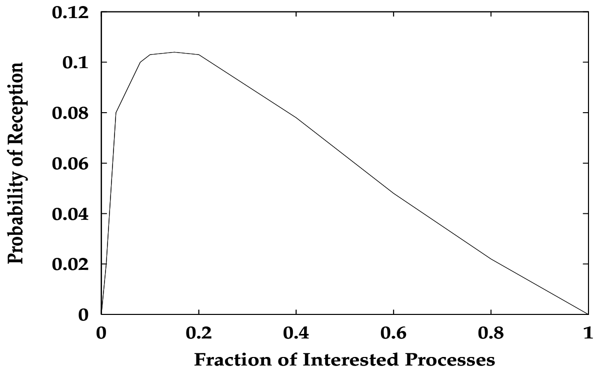

Figure 5. Infected Uninterested Processes; $n \approx 10000\ (a = 22), d = 3, R = 3, F = 2$

5.2 Scalability

It is not immediately visible from the analysis whether, and how, the performance of *pmcast* is impacted when increasing the scale of the group. As conveyed by Figure 6, *pmcast* has very good scalability properties when increasing a in a tree of fixed depth (note that the group size increases following a^d). The scalability however depends on the proportion of interested processes. With a small p_d, the above-mentioned problem manifests through a slightly stronger decreasing reliability despite the increased group size.

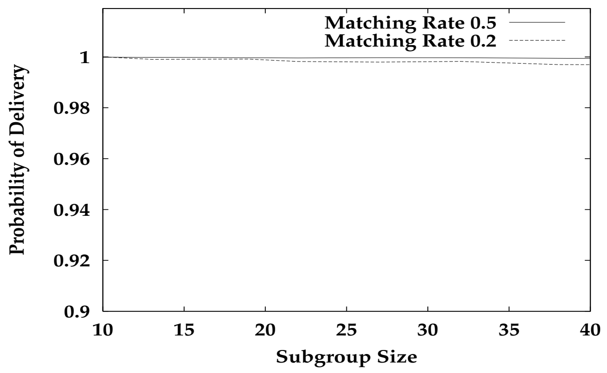

Figure 6. Scalability; $d = 3, R = 4, F = 3$

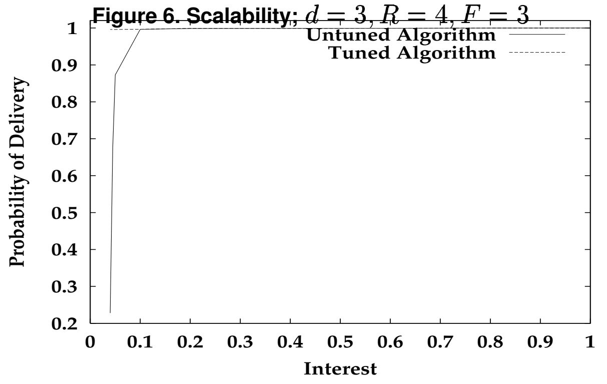

Figure 7. Tuned vs Untuned Algorithm; $n \approx 10000\ (a = 22), d = 3, R = 3, F = 2$

5.3 Tuning for Small Matching Rates

Basically, one can distinguish two ways of improving the performance of *pmcast* for small matching rates, i.e., with small values for p_d, namely (1) increasing artificially the number of interested processes to obtain a better approximation of the necessary number of rounds with Pittel's asymptote, and (2) applying another approximation for the number of necessary rounds. The second approach can for instance be achieved using a more rough approximation of the number of rounds for large matching rates, yielding however more precise values for small matching rates.

Aiming at applications in which critically small values for p_d are seldom (typically, half the processes are interested in a given multicast event), we have adopted the approach of artificially increasing the “audience” by adding processes to the set of interested processes. To that end, we have modified the algorithm presented in Figure 3 to gossip to non-interested processes if the number of interested processes in the group drops below a threshold h. In that case, every involved process decides that the h first processes in its view of the corresponding depth are interested, in addition to the remaining effectively interested processes outside of the first h processes in the corresponding view. By fixing a lower bound on the desired reliability degree, h can be obtained through analysis or simulation. The result of such a tuning is illustrated by Figure 7, which compares the original degree of reliability with the improved one. Note however, that this tuning leads to a compromise, in the sense that the rate of infected non-interested processes increases with respect to Figure 5.

6 Concluding Remarks

The strong scalability and reliability guarantees offered by our *Probabilistic Multicast* (*pmcast*) algorithm are a consequence of its underlying tree-like orchestration of processes.

The idea of arranging processes according to a specific tree-like scheme is not new. A typical example is *Capt'n Cook* [12], a gossip-based resource location algorithm for the Internet, which can in that sense be seen as a membership algorithm without broadcasting of events. The hierarchy underlying *Capt'n Cook* has been reused as the *Grid Box* [5] for the means of computing an aggregate function on outputs (e.g., sensor values) of all processes in a large group, or in *Astrolabe* [13], which, similarly to our approach, supports the multicasting of events to a subset of processes. The multicasting in *Astrolabe* is however not based on gossips but performed deterministically, with higher throughput than *pmcast* in “stable” phases of the system, yet a reduced robustness in “unstable” phases. The *Directional Gossip* algorithm [8] also describes a simple two-level hierarchy in which events are however again broadcast, i.e., *all* processes in the group are invariably addressed.

The tree underlying *pmcast* can be further exploited by making use of different mechanisms at different depths concerning (1) the spreading of events (e.g., flooding the leaf subgroups if there is a high density of interests), (2) the filtering of events (approximating the filters applied by delegates closer to the root to reduce computation), but also (3) the monitoring of processes (e.g., processes in leaf subgroups actively ping each other, and possibly even perform a form of agreement before excluding a suspected process from their views).

Acknowledgements

We are very grateful to Boris Pittel for his highly valuable comments and suggestions concerning the epidemiologic aspects of our algorithm. We would also like to thank Robbert van Renesse and Luis Rodrigues for commenting on an early version of this work.

References

[1] K. Birman, M. Hayden, O.Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal Multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, May 1999.

[2] P.Th. Eugster. *Type-Based Publish/Subscribe*. PhD thesis, Swiss Federal Institute of Technology, Lausanne, Dec. 2001.

[3] P.Th. Eugster, R. Guerraoui, S. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight Probabilistic Broadcast. In *Proceedings of the 2001 IEEE International Conference on Dependable Systems and Networks*, June 2001.

[4] R. Guerraoui and A. Schiper. Genuine Atomic Multicast in Asynchronous Distributed Systems. *Theoretical Computer Science*, 254(1–2):297–316, Mar. 2001.

[5] I. Gupta, R. van Renesse, and K. Birman. Scalable Fault-Tolerant Aggregation in Large Process Groups. In *Proceedings of the 2001 IEEE International Conference on Dependable Systems and Networks*, June 2001.

[6] V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. In S. Mullender, editor, *Distributed Systems*, chapter 5. Addison-Wesley, 1993.

[7] K. Jenkins, K. Hopkinson, and K. Birman. A Gossip Protocol for Subgroup Multicast. In *International Workshop on Applied Reliable Group Communication*, Apr. 2001.

[8] M.-J. Lin and K. Marzullo. Directional Gossip: Gossip in a Wide Area Network. In *Proceedings of the 3rd European Dependable Computing Conference*, Sept. 1999.

[9] R. Piantoni and C. Stancescu. Implementing the Swiss Exchange Trading System. In *Proceedings of the 27rd IEEE International Symposium on Fault-Tolerant Computing*, June 1997.

[10] B. Pittel. On Spreading of a Rumor. *SIAM Journal of Applied Mathematics*, 47:213–223, 1987.

[11] Q. Sun and D. Sturman. A Gossip-Based Reliable Multicast for Large-Scale High-Throughput Applications. In *Proceedings of the 2000 IEEE International Conference on Dependable Systems and Networks*, July 2000.

[12] R. van Renesse. Scalable and Secure Resource Location. In *Proceedings of the 33rd IEEE Hawaii International Conference on System Sciences*, January 2000.

[13] ”R. van Renesse and K. Birman” Scalable Management and Data Mining Using Astrolabe. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, March 2002.