# Hierarchical Probabilistic Multicast*

**Patrick Th. Eugster** **Rachid Guerraoui**
Swiss Federal Institute of Technology
CH-1015 Lausanne, Switzerland
{patrick.eugster, rachid.guerraoui}@epfl.ch

## Abstract

*In our DACE project [6], diverging requirements expressed through QoS are mainly explored by a variety of different delivery semantics implemented through different dissemination algorithms ranging from "classic" Reliable Broadcast [18], to new and original algorithms, like the broadcast algorithm we introduce in [7], and which ensures reliable delivery of events despite network failures.*

*While striving for strong scalability, we have invested considerable effort in exploring probabilistic (gossip-based) algorithms. These appear to be more adequate in the field of large scale event dissemination than traditional strongly reliable approaches like [18]. Basically, probabilistic algorithms trade the strong reliability guarantees against very good scalability properties, yet still achieve a "pretty good degree of reliability" [12].*

*Until now, most work on gossip-based algorithms considers broadcasting information to all participants in a system, paying little or no attention to individual and dynamic requirements, as typically encountered in content-based dissemination.*

*We present here Hierarchical Probabilistic Multicast (hpmcast [9]), a novel gossip-based algorithm which deals with the more complex case of multicasting an event to a subset of the system only. Requirements, such as limiting the consumption of local memory resources by view and message buffering, as well as exploiting locality (the proximity of participants) and redundancy (commonalities in interests of these participants), are all addressed.*

*Though hpmcast has been motivated by our specific context of TPS, it is general enough to be applied to any context in which a strongly scalable primitive for event, message, or information dissemination is required.*

## 1 Introduction: Probabilistic Broadcast Algorithms

The achievement of strong reliability guarantees (in the sense of [18]) in practical distributed systems requires expensive mechanisms to detect missing messages and initiate retransmissions.

### 1.1 Reliability vs Scalability

Due to the overhead of message loss detection and reparation, algorithms offering such strong guarantees do not scale over a couple of hundred participants [30].

#### 1.1.1 Network-Level Protocols

In [11], we describe a simple publish/subscribe architecture based on IP Multicast. Such network-level protocols however have turned out to be insufficient: IP Multicast lacks any reliability guarantees, and so-called reliable protocols do not scale well. The well-known *Reliable Multicast Transport Protocol* (RMTP) [29] for instance generates a flood of positive acknowledgements from receivers, loading both the network and the sender, where

*This work is partially supported by Agilent Laboratories and Lombard Odier & Co.

these acknowledgements converge.[1] Moreover, such protocols hide any form of *membership* [2, 24], making them difficultly exploitable with more dynamic dissemination (filtering).

### 1.1.2 Probabilistic Algorithms

Gossip, or *rumor mongering algorithms* [5], are a class of *epidemiologic algorithms*, which have been introduced as an alternative to such reliable network-level broadcast protocols. They have first been developed for replicated database consistency management, and have been mainly motivated by the desire of trading the strong reliability guarantees offered by costly deterministic algorithms against weaker reliability guarantees, but in return obtaining very good scalability properties.

The analysis of such algorithms is usually based on stochastics similar to the theory of epidemics [3], where the execution is broken down into steps. Probabilities are associated to these steps, and such algorithms are therefore sometimes also referred to as *probabilistic algorithms*.

### 1.1.3 Reliability Degree

The "degree of reliability" is typically expressed by a probability; like the probability 1-$\alpha$ of reaching *all* participants in the system for any given message, or by a probability 1-$\beta$ of reaching *any* given participant with any given message. Ideally, $\alpha$ and $\beta$ are precisely quantifiable. A more precise measure, called $\Delta$-*Reliability*, based on the distribution of the probability of reaching a fraction of participants, is given in [12].

## 1.2 Basic Concepts

Decentralization is the key concept underlying the scalability properties of gossip-based broadcast algorithms, i.e., the overall load of (re)transmissions is reduced by decentralizing the effort. Participants are viewed as *peers*, symmetric in role, which are all equally eligible to forward information.[2] This makes of gossip-based algorithms ideal candidates for systems with an underlying *peer-to-peer* model.

### 1.2.1 Parameters

More precisely, retransmissions are initiated in most gossip-based algorithms by having every participant periodically, i.e., every $P$ ms (*step interval*), send information to a randomly chosen subset of participants inside the system (*gossip subset*). The size $F$ of the subset is usually fixed, and is commonly called *fanout*. Gossip algorithms differ in the number of times the same information is gossiped. Every participant might gossip the same information the same number of times, meaning that the number of *repetitions* is fixed. Alternatively, the same information might be forwarded only once by a same participant, and the longest causal chain of message forwards can be limited by fixing the number of *hops* $H$ (or *forwards*). Also, the number of rounds $T$ (step intervals) that a message remains in the system can be limited.

### 1.2.2 Approaches

Gossiping techniques have been proposed in a broad spectrum of contexts. Consequently, these algorithms vary a greate deal of further characteristics.

**Messages.** Gossip-based algorithms differ in the kind of information that is shipped by gossiped messages (*gossips*). In early gossip algorithms, gossips reflect the sender's message buffer, including information about missing messages. Gossips have also been used to directly propagate the multicast payload (e.g., [10]), like events in the case of TPS.

[1] Similarly, the scalability offered by other reliable network-level protocols, like *Reliable Multicast Protocol* (RMP) [35], *Log-Based Receiver-Reliable Multicast* (LBRM) [19], or *Scalable Reliable Multicast* (SRM) [13] is not sufficient for many current application scenarios.

[2] Note that the SRM protocol also relies on a peer-based approach. A retransmitted message is however rebroadcast to the entire system.

**Interaction between peers.** With latter type of gossip-based algorithms, the interaction between peers invariably relies on *pushing* messages, e.g., events, from one participant to a set of neighbors.

Former type of gossip-based algorithms, i.e., aiming at propagating *digests*, vary in the way participants react to incoming gossip messages. With a *gossiper-pull*, a gossip receiver retransmits missing messages to the gossip sender. With *gossiper-push*, a gossip target replies with a retransmission request to the gossip sender. The term *anti-entropy* is sometimes used to denote a combined push/pull scheme, i.e., a bidirectional updating [14]. In database replication, gossiper-pull has been shown to converge faster [5]. The same observation is made in the context of gossip-based broadcast algorithms, when a majority of participants have a message [32].

### 1.2.3 Faces of Scalability

Gossip-based approaches are said to be inherently scalable, meaning that the consumption of network resources (the amount of network messages necessary for successfully disseminating an application message) increases only slightly with an increasing system size. Scalability appears however under a variety of other faces, which can be devoted different priorities, depending on the context.

**The case of membership.** Most importantly in the context of TPS, implementations of content-based publish/subscribe have revealed the inherent difficulty of mapping individual and strongly dynamic requirements to a set of static groups [26]: not *all* possible values for *all* attributes of events are known in advance, especially as new event types are added at runtime. When considering a broadcast group for every possible subset of participants of a system of size $n$, the views of an individual participant sum up to a total of $\sum_{m=1}^{n-1} \binom{n-1}{m} m = (n-1)2^{n-2}$ entries.[3] Since these membership views have to be managed explicitly, a further barrier to scalability is introduced. There have been indeed proposals on how to reduce the views of participants, however again without taking into account individual interests of these participants.

**Interferences.** Furthermore, the scalability of an algorithm can be limited by the size of message buffers, requiring subtle schemes for garbage collection. In general, the different faces of scalability are intermingled. As a first example, by highly loading the network, information can be spread quickly, reducing the size of buffers. As a second example, message buffers and data structures representing the system view compete for memory resources. Last but not least, when performing filtering to avoid sending events to participants which do not manifest any interests in these events, network resources are more wisely used, at the expense of processing power.

## 1.3 Related Gossip-Based Algorithms

Instead of presenting an exhaustive view of all work on gossip-based algorithms up to date, we overview approaches that are closest to ours. These approaches are discussed by pointing out the way they deal with the different aspects of scalability overviewed above.

### 1.3.1 Probabilistic Broadcast

With *Probabilistic Broadcast* (*pbcast* [4]), Birman et al. have triggered a resurrection of gossip-based algorithms. *pbcast* is also called *Bimodal Multicast*, due to its two phases: a "classic" best-effort multicast (e.g., IP Multicast) is used for a first rough dissemination of messages. A second phase assures reliability with a certain probability, by using a gossip-based retransmission: every participant in the system periodically gossips a digest of its received messages, and gossip receivers can solicit such messages from the sender if they have not received them previously.

**Membership scalability.** The membership problem is not dealt with in [4], but the authors refer to a paper by Renesse et al. which deals with failure detection based on gossips [34], while another paper describes *Capt'n Cook* [33], a gossip-based resource location algorithm for the Internet, which can in that sense be seen as a membership algorithm.

This also enables the reduction of the view of each individual participant: each participant has a precise view of its immediate neighbours, while the knowledge becomes less exhaustive at increasing "distance". The notion

[3] And this without counting the participant itself. Otherwise, the sum totals even to $n2^{n-1}$ entries.

of distance is expressed in terms of host addresses (names). Capt'n Cook only considers the propagation of membership information.

The *Grid Box* [17] describes a more recent approach to arranging participants in a system according to a hierarchy, for the means of computing an *aggregate function* on outputs of all participants (e.g., sensor values). The hierarchy used is in essence the same as the one we will discuss later on, yet is applied in a very specific manner, by aggregating values at every level of the hierarchy.

**Buffer scalability.** The significant work accomplished at Cornell around gossiping techniques also includes efforts on how to enforce scalability in message buffering, by limiting the number of participants which store a same message [28], or applying gossiping techniques to perform garbage collection [16].

These, as well as the membership and failure detection facets of scalability are all dealt with separately, proving their applicability to a wide range of algorithms (even algorithms which do not make use of gossiping techniques for the main spreading of the payload), yet only little to no information is given on the possibility and consequences of a cooperation in *pbcast*.

### 1.3.2 Reliable Probabilistic Broadcast

*Reliable Probabilistic Broadcast* (*rpbcast* [32]), an algorithm developed by IBM in the context of the Gryphon project, is strongly inspired by *pbcast*. There are two main differences. First, while *pbcast* has been originally described as using gossiper-push, *rpbcast* uses a pull scheme for its faster convergence. Second, and more important, *rpbcast* adds a third phase to achieve strong reliability. The system is instrumented with *loggers*, which log messages on stable storage. These are consulted whenever the two initial phases fail in providing some participant with a *relevant*[4] message. [32] does not give hints on membership management, nor on message buffering.

### 1.3.3 Directional Gossip

*Directional Gossip* [22] is an algorithm especially targeted at wide area networks, developed at the University of San Diego. By taking into account the topology of the network and the current participants, optimizations are performed. More precisely, a *weight* is computed for each neighbour node, representing the connectivity of that given node. The larger the weight of a node, the more possibilities exist for it to be infected by any node. The algorithm applies a simple heuristic, which consists in choosing nodes with higher weights with a smaller probability than nodes with smaller weights, reducing the number of redundant sends.

**Membership scalability.** The algorithm hence supposes that not all participants are connected, or rather, know each other. This implies partial views, and in practice, a single *gossip server* is assumed per LAN which acts as a bridge to other LANs. This however leads to a *static hierarchy*, in which the failure of a gossip server can isolate several participants from the remaining system.

**Determinism.** An approach to analyzing the performance achieved when every participant attempts to infect a deterministically determined subset of the system, involves the same authors [23]. Subsets are established through a graph connecting the participants, called *Harary* graph, leading to a flooding of the system over such a graph. The introduced determinism, as intuition suggests, reduces the number of message sends. However, the reliability of the gossip-based algorithm used for comparison appears to degrade slightly more gracefully with an increasing number of participant failures, and the establishment of the connections according to the Harary graph introduces an important overhead.

### 1.3.4 Lightweight Probabilistic Broadcast

*Lightweight Probabilistic Broadcast* (*lpbcast* [10]) is a probabilistic broadcast algorithm developed in our lab. *lpbcast* adds an inherent notion of memory consumption scalability to the notion of network consumption scalability primarily targeted by gossip-based algorithms.

[4] Similar to other algorithms, an upcall to the application determines how much effort is deemed suitable to recover a missed message [27].

**Probabilistic membership.** In contrast to the deterministic hierarchical membership approaches in Directional Gossip or Capt'n Cook, *lpbcast* represents a probabilistic approach to membership: each participant has a *random partial view* of the system. *lpbcast* is lightweight in the sense that it consumes little resources in terms of memory and requires no dedicated messages for membership management: gossips are used to disseminate the payload (i.e., events) and to propagate digests of received events, but also to propagate membership information. The analysis presented in [10] includes all of these aspects.

**Probabilistic buffering.** The basic *lpbcast* algorithm furthermore also buffers events in a fully probabilistic sense. To respect a maximum buffer size, every participant only buffers a random subset of the events gossiped in the system. This results in an effect similar to the one targeted by [28], namely consisting in buffering individual messages only on a subset of the system

Optimizations for *lpbcast*, such as trying to "force" a more uniform distribution of the individual views, or prioritizing the buffering of more recent events, have been proposed in [21].

## 2 From Broadcast to Multicast

A broadcast algorithm can be obviously used to multicast events. We depict two gossip-based broadcast algorithms used to achieve multicasting of events, and outline the respective limitations of these approaches, leading to a more consolidated algorithm presented in the following sections.

### 2.1 Broadcast with Receiver Filtering

A pragmatic way of multicasting information inside a system subset consists in broadcasting within the entire system and filtering upon reception of events, i.e., an event is delivered to the application iff that participant is interested in that given event.

#### 2.1.1 *rfpbcast* Algorithm

Figure 1 outlines a modified probabilistic broadcast algorithm called here *Receiver Filtering Probabilistic Broadcast* (*rfpbcast*). Every participant, similarly to a probabilistic broadcast, periodically (every $P$ milliseconds) gossips to a randomly chosen subset of the system. In our context, a gossiper forwards every buffered event to a randomly chosen subset of size $F$ of the system.

When receiving an event, a participant only delivers that event if it effectively is of interest for it.[5] This is represented at Line 16 of Figure 1 through the $\triangleleft$ operator indicating whether a given event is of interest for a certain participant. This can be viewed as evaluating the participant's subscription pattern for the considered event: $event \triangleleft participant$ returns *true* iff *participant* is interested in *event*.

#### 2.1.2 Analysis

For our formal analysis we consider a system composed of $n$ participants, and we observe the propagation of a single event notification. We assume that the composition of the system does not vary during the run (consequently $n$ is constant). According to the terminology applied in epidemiology, a participant which has delivered a given notification will be termed *infected*, otherwise *susceptible*.

**Assumptions.** The stochastic analysis presented below is based on the assumption that participants gossip in synchronous rounds, and there is an upper bound on the network latency which is smaller than a gossip period $P$.[6] $P$ is furthermore constant and identical for each participant, just like the fanout $F < n$. We assume furthermore that failures are stochastically independent. The probability of a network message loss does not exceed a predefined $\epsilon > 0$, and the number of participant crashes in a run does not exceed $f < n$. The probability of a participant crash during a run is thus bounded by $\tau = f / n$. We do not take into account the recovery of crashed participants,

[5] This is equivalent to performing the filtering in a higher layer, possibly in the application itself.

[6] This analysis does not rely on the assumption that the underlying system is synchronous, nor does the algorithm force the system to behave so.

```
Executed by participant_i
 1: initialization
 2: view
 3: gossips ← ∅

 4: task GOSSIP                                                    {every P milliseconds}
 5:   for all (event, rounds) ∈ gossips do
 6:     if rounds < T then                                         {not too many rounds}
 7:       rounds ← rounds + 1
 8:       dests ← ∅
 9:       repeat F times                                           {choose potential destinations}
10:         dest ← RANDOM(dests)
11:         dests ← dests ∪ {dest}
12:         SEND(event, rounds) to dest

13: upon RECEIVE(event, rounds): do
14:   if ∄ (event, ...) ∈ gossips then                             {buffer the gossip and deliver it}
15:     gossips ← gossips ∪ {(event, rounds)}
16:     if event ◁ participant_i then
17:       RFPDELIVER(event)

18: upon RFPBCAST(event): do
19:   gossips ← gossips ∪ {(event, 0)}

20: function RANDOM(previous)                                      {choose random participants}
21:   return dest ∈ view | dest ∉ previous
```

**Figure 1. Receiver Filtering Broadcast Algorithm**

nor do we consider Byzantine (or arbitrary) failures. At each round, we suppose that each participant has a complete view of the system.

**Number of infected participants.** The analysis presented resembles the analysis applied to *pbcast* in [4] and *lpbcast* in [10]. The probability $p$ that a given gossiped event is received by a given participant, is given as a conjunction of several conditions, namely that (1) the considered participant is effectively chosen as target, (2) the gossiped event is not lost in transit, and (3), the target participant does not crash.

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

We denote the number of participants infected with a given event at round $t$ as $s_t$, $1 \leq s_t \leq n$. Note that when the event is injected into the system at round $t = 0$, we have $s_t = 1$.

Accordingly, $q(n, F) = 1 - p(n, F)$ represents the probability that a given participant is *not* reached by a given infected participant. Given a number $j$ of currently infected participants, we are now able to define the probability that exactly $k$ participants will be infected at the next round ($k - j$ susceptible participants 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 $k$ ($j > 1$, $1 \leq k \leq n$):

$$\begin{aligned} p_{jk}(n, F) &= P(n, F)[s_{t+1} = k | s_t = j] \\ &= \begin{cases} \binom{n-j}{k-j}(1 - q(n, F)^j)^{k-j} q(n, F)^{j(n-k)} & j \leq k \\ 0 & j > k \end{cases} \end{aligned} \tag{2}$$

The distribution of $s_t$ can then be computed recursively ($1 \leq k \leq n$). In summary:

$$P(n, F)[s_t = k] = \begin{cases} 1 & t = 0, k = 1 \\ 0 & t = 0, k > 1 \\ \sum_{j=k/(1+F)}^{k} P(n, F)[s_{t-1} = j] p_{jk}(n, F) & t \geq 1 \end{cases} \tag{3}$$

**Expected number of rounds.** In the *rfpbcast* algorithm, there are still two undefined parameters, which are the fanout $F$, and the number of rounds $T$. According to Pittel [31], the total number of rounds necessary to infect an entire system of size $n$ (large), in which every infected participant tries to infect $F > 0$ other participants, is given by the following expression:

$$\begin{aligned} & log_{F+1} n + \frac{1}{F} log\ n + c + O(1) \ = \\ & log\ n \left( \frac{1}{F} + \frac{1}{log\ (F+1)} \right) + c + O(1) \end{aligned} \tag{4}$$

By fixing either $F$ or $T$, the other value can be computed based on this expression, or if more detailed information is required, through the above Markov chain.

However, the model in [31] does not consider the possibility of losing events between a gossiper and a (potential) destination. In our case, only $F(1-\epsilon)(1-\tau)$ participants are expected to be infected at a given round by a gossiper, leading to the following expression:

$$T(n,F) \ = \ log\ n \left( \frac{1}{F(1-\epsilon)(1-\tau)} + \frac{1}{log\ (F(1-\epsilon)(1-\tau)+1)} \right) + c + O(1) \tag{5}$$

Note that it has been shown in [20] that, when limiting the number of repetitions to 1 (every participant forwards a given event at most once), choosing the natural logarithm of the system size as value for $F$ brings the probability of reaching all participants very close to 1. Though in our model the number of gossips is not limited through the number of repetitions, but through the maximum number of rounds that an event can spend in the system, a logarithmic value could reflect a reasonable fanout.

## 2.2 Sender Filtering

A first, very strong limitation of the above *rfpcast* algorithm appears immediately. As reflected through the analysis, a gossiped event is sent to every participant, regardless of whether it is effectively interested in that event. Accordingly, especially with events which are of interest for only a small fraction of the system, there is a high waste of network bandwidth and also local memory for buffering.

To avoid sending an event to a participant for which that event is irrelevant, the following modified broadcast algorithm (*pmcast*) performs the filtering before sending. It can be seen as a *genuine multicast* ([15]) algorithm in the sense that events are only received by interested participants, and only these participants are involved in the algorithm.

### 2.2.1 *pmcast* Algorithm

Similarly to the previous *rfpbcast* algorithm, every participant periodically gossips every event in its buffer to a subset of participants in the system. In this *Probabilistic Multicast* (*pmcast*) algorithm (Figure 2) however, after picking a random subset of size $F$ of the system, the set of destinations is further restricted to the subset of participants effectively interested in the considered event.

Note that no filtering is necessary at reception, since no spurious event is sent to any participant.

### 2.2.2 Analysis

The fraction of the system which is effectively interested in an observed event is of primary importance for the analysis. We can represent the size of the interested subset as $np_1$, where $p_1$ is the probability that a given participant is interested in the event. In other terms, when considering that $n_1$ participants among $n$ are interested in a particular obvent, then $p_1 = \frac{n_1}{n}$.

```
Executed by participant_i
 1: initialization
 2: view
 3: gossips ← ∅

 4: task GOSSIP                                                    {every P milliseconds}
 5:   for all (event, rounds) ∈ gossips do
 6:     if rounds < T then                                         {limit the time an event spends in the system}
 7:       rounds ← rounds + 1
 8:       dests ← ∅
 9:       repeat F times                                           {choose potential destinations}
10:         dest ← RANDOM(dests)
11:         dests ← dests ∪ {dest}
12:         if event ◁ dest then
13:           SEND(event, rounds) to dest

14: upon RECEIVE(event, rounds): do
15:   if ∄ (event, ...) ∈ gossips then                             {buffer the gossip and deliver it}
16:     gossips ← gossips ∪ {(event, rounds)}
17:     PDELIVER(event)

18: upon PMCAST(event): do
19:   gossips ← gossips ∪ {(event, 0)}

20: function RANDOM(previous)                                     {choose random participants}
21:   return dest ∈ view | dest ∉ previous
```

**Figure 2. Probabilistic Multicast Algorithm**

**Number of infected participants.** The effective expected number of participants that are gossiped to at each round by an infected participant is $Fp_1$ (without considering network message loss and participant failures). The expression for $p$ (the probability that a gossip reaches a given participant), is hence given in this case as follows:

$$\begin{aligned} p(np_1, Fp_1) &= \left(\frac{Fp_1}{np_1 - 1}\right)(1-\epsilon)(1-\tau) \\ q(np_1, Fp_1) &= 1 - p(np_1, Fp_1) \end{aligned} \tag{6}$$

This is similar to gossiping in an effective system of size $p_1 n$, however with a fanout of only $p_1 F$. The resulting Markov chain is characterized by the probability $p_{jk}$ of transiting from state $j$ to state $k$, as follows:

$$\begin{aligned} p_{jk}(np_1, Fp_1) &= P(np_1, Fp_1)[s_{t+1} = k | s_t = j] \\ &= \begin{cases} \binom{np_1 - j}{k-j}(1 - q(np_1, Fp_1)^j)^{k-j} q(np_1, Fp_1)^{j(np_1-k)} & j \le k \\ 0 & j > k \end{cases} \end{aligned} \tag{7}$$

And the distribution of $s_t$ can then again be computed recursively. In summary:

$$P(np_1, Fp_1)[s_t = k] = \begin{cases} 1 & t = 0, k = 1 \\ 0 & t = 0, k > 1 \\ \sum_{j=k/(1+F)}^{k} P(np_1, Fp_1)[s_{t-1} = j] p_{jk}(np_1, Fp_1) & t \ge 1 \end{cases} \tag{8}$$

**Expected number of rounds.** Similarly, the expression for the expected number of rounds above (Equation 5) can be adapted to reflect the sender filtering.

$$T(np_1, Fp_1) = \\ log\ np_1 \left( \frac{1}{Fp_1(1-\epsilon)(1-\tau)} + \frac{1}{log\ (Fp_1(1-\epsilon)(1-\tau)+1)} \right) + c + O(1) \quad (9)$$

Yet, since Pittel's formula is valid for large systems, this formula only offers useful results as long as $np_1$ is still large.

# 3 Overview of Hierarchical Probabilistic Multicast

The above *pmcast* algorithm is indeed smarter than the *rfpbcast* algorithm, yet still presents an important number of sensible limitations. We discuss these and overview how our Hierarchical Probabilistic Multicast (*hpmcast*) repairs these lacks.

## 3.1 Membership Scalability

In both modified broadcast algorithms, every participant has a "full" view of the system, i.e., every participant knows every other participant. As already pointed out in [10], this can become a severe barrier for scalability as the system grows in size.

In order to reduce the amount of membership knowledge maintained at each participant, a participant should only know a subset of the system. The individual subsets known by the participants should nevertheless ensure two properties, namely (1) that every participant is known by several others (for reliability), and (2) that no participant knows all participants (for scalability).

### 3.1.1 Random Approach

A random subset as chosen in *lpbcast* also ensures membership scalability, and can be put to work in a way that, with a uniform distribution of knowledge, the propagation of information is virtually not impacted by the reduction of the amount of membership knowledge. This approach applies well to broadcast, but gives less good results in a practical multicast setting, especially if the exploiting of locality and redundancy is desired.

### 3.1.2 Hierarchical Approach

In contrast, *hpmcast* is based on a hierarchical disposition of participants bearing strong resemblances with the Capt'n Cook and the Grid Box approaches. The extent of interactions between participants depends on their "distance", but the hierarchical membership is used to multicast events inside the system. Participants have a complete knowledge of their respective immediate neighbours, but only a decreasing knowledge about more "distant" participants.

To that end, a participant can *represent* a *subnetwork*, or *subsystem*, for participants outside of the respective subsystem. In other terms, a participant outside of a subsystem, yet who knows that subsystem, will only know such representing participants, called *delegates*. Among the delegates for neighbor subsystems, again a set of delegates are chosen recursively, giving rise to a hierarchy of several ($l$) levels.

Delegates that appear at a high level in the hierarchy are known by more participants in the system than lower-level participants. Nevertheless, all participants have membership views of comparable sizes, since every participant has a view of its subsystem for every level. Also, a participant which is elected as delegate for a given hierarchy level still remains visible in views of its lower level subsystems.

## 3.2 Locality

Another limitation of the two previous algorithms (*rfpbcast* and *pmcast*) is that they do not take *locality* into account, i.e., a participant randomly picks destinations regardless of its "distance" to those participants. Far away participants are chosen with the same probability than close neighbors. It seems more adequate to aim first a, rough and wide distribution of events, before attempting a more complete and local dissemination.

Dissemination of events inside our hierarchy follows a *level-wise dissemination*, i.e., the highest level of the hierarchy is first infected, and then the following levels are successively infected in the order of their depth. This approach is opposed to the Grid Box, where however the goal is different; a global function has to be applied to values collected from all participants. In contrast, when multicasting inside the hierarchy, a value originates from a single participant and is propagated as such.[7]

Since higher levels regroup participants representing various distant subsystems, a level-wise gossiping ensures that events are in a first step spread coarsely, and that a finer coverage of the individual subsystems takes place successively. Only a "reasonable" number of sends between distant participants takes place, given by the number of gossip rounds expected to infect the corresponding level of the hierarchy. Once sparsely spread, an event is only more sent between more local participants.

### 3.3 Redundancy

Furthermore, in the *pmcast* algorithm, every participant stores every other participant's individual interests, and filtering is made independently for every chosen participant. Significant performance optimizations, like the exploiting of redundancies of individual subscription patterns, as proposed in [1], cannot be applied.

The higher the level at which a delegate can be found in the hierarchy, the more participants that delegate represents. Accordingly, the delegate manifests interest in any event that is of interest for any of the participants it represents. Or, in the terminology of TPS, its subscription pattern, from the perspective of another participant, is a compound pattern created from the subscription patterns of the participants it represents. Redundancy of these individual patterns can be exploited, by creating such condensed patterns which avoid redundancies between the individual patterns.

Observe however that redundancy competes to some extent with locality: geographically "close" neighbors do not necessarily present "close" interests, and any scheme relying on one of these two notions of proximity might rule out the exploiting of the other notion.

### 3.4 Garbage Collection

Most dissemination algorithms apply a combination of *acknowledgements* (*acks*) and *negative acknowledgements* (*nacks*) to verify the stability of a given message and to perform garbage collection. Such schemes, as well as more sophisticated schemes (cf. [28, 16] for *pbcast*, or [21] for *lpbcast*), rely on unique message identifiers. This again applies well to the case of broadcast, but is less straightforward to apply to a multicast setting, where a given multicast event is only significant for a subset of participants, and this significance can only be verified through the event itself, not through an identifier.

On the other hand, it is difficult to perform garbage collection by statically limiting the number of repetitions, forwards or hops. Indeed, as shown by Equation 9 above, parameters $F$ and $T$ are related, yet in a way depending on the fraction of interested participants (reflected by $p_1$), which is individual for every considered event. In fact, according to Equation 9, the number of rounds necessary to infect all interested participants increases as the number of these interested participants decreases. By fixing the number of rounds that an event can spend in the system, one ends up with considerably weaker reliability for events with a small "audience". While this can indeed make sense in specific contexts, we view this rather as an undesirable property.

We have hence chosen to integrate garbage collection into the multicast algorithm, by limiting the number of rounds that an event remains in the system. In fact, the expected number of rounds necessary to infect all participants in a subset of the system, as well as parameters of the system can be approximated. Inherently limiting this way the life-time of an event applies naturally, since gossips are used primarily to transport events, and not to exchange message identifiers aiming at detecting message stability. The feasibility of limiting the lifetime of gossiped events a priori, i.e., renouncing to any explicit garbage collection algorithm, has been illustrated by *lpbcast*.

[7] It would be very interesting to exploit this aggregate function to reflect *event correlation*.

# 4 Hierarchical Membership

This section describes a precise model of our hierarchy, while the corresponding membership management is informally described. We elucidate how this hierarchy, besides reducing the memory resource consumption of the membership view, also offers a nice compromise between locality and redundancy.

## 4.1 Model

As elucidated above, *hpmcast* is based on a hierarchical membership, where the knowledge that an individual participant has about other participants decreases with the "distance" from these participants.

### 4.1.1 Addresses

Before going further into the description of the membership, we require a definition of the notion of "distance" between two participants. An approach could consist in using an *average communication delay* between two participants as a measure for their distance. However, since we are considering asynchronous systems, such values are difficult to determine. We will thus base the decision of which participants out of a subsystem are to be considered as neighbors, and also as prioritized (to become delegates), on the *addresses* of those participants, more precisely, on the distances between them.

This notion of "distance" can be approximated by network addresses, but can as well be simulated by associating *logical* addresses with participants. Irrespective of how these addresses are determined, we will in the following simply consider such addresses as sequences of values, of the following form:

$$\begin{gathered} x(l-1).\cdots.x(0), \\ \forall i\ 0 \leq i \leq l-1, 0 \leq x(i) \leq a_i - 1 \end{gathered} \tag{10}$$

The total number of different addresses and thus the maximum number of participants is given by

$$\prod_{0 \leq i \leq l-1} a_i \tag{11}$$

Though participants in the sense of TPS 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 participant per host. Indeed, the last components of an address could easily be used to express a port number. To cover all possible IP addresses for instance, one could choose $l = 4$ and $a_i = 2^8 = 256\ \forall i$, or $l = 11$ and $a_i = 2^4 = 16\ \forall i$ to include $2^{12} = 4096$ port numbers.[8]

### 4.1.2 Branch Addresses

We call a "partial" address, like $x(l-1).\cdots.x(i)$ $(0 < i \leq l-1$; $\emptyset$ for $i = l)$ a *branch address* of level $i$. Such a branch address denotes a subsystem or subnetwork. All participants sharing a branch address belong to the corresponding subsystem. In other terms, there might be several addresses $x(i-1).\cdots.x(0)$ that can be appended to a same branch address to denote a concrete participant.

The distance between two addresses is expressed based on this notion. In fact, the distance between two participants is equal to the level of their longest common branch address, e.g., if two participants $p_1$ and $p_2$ share a branch address of level $i$, then they are said to be at a distance of $i$. Also, they belong to the same subsystem of level $i$ of the hierarchy. A distance of 0 would mean that the two participants share the same address and are thus equivalent.

[8] Note that certain IP addresses and port numbers are reserved for special purposes, e.g., IP addresses 224.0.0.0 to 239.255.255.255 are reserved for IP Multicast groups. Those exceptions will not be discussed here.

### 4.1.3 Electing Delegates

All participants which share a given branch address $x_0 = x_0(l-1).\cdots.x_0(1)$ form a group of level 1. The number of such participants (at the moment of observation) is denoted $|x_0(l-1).\cdots.x_0(1)|$. Quite obviously, for any given branch address $x(l-1).\cdots.x(1)$, $|x(l-1).\cdots.x(1)| \leq a_0$.

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

Figure 3 illustrates a simple example. $R$ represents a *redundancy factor*, which however has no relationship with the notion of redundancy observed in subscription patterns. $R$ represents the number of delegates that are elected to represent a subsystem, and is best chosen such that $R > 1$, in order to improve the reliability of the membership: with $R = 1$, the hierarchy is very sensitive to crash failures of individual participants, leading to an increased risk of a partitioned membership.

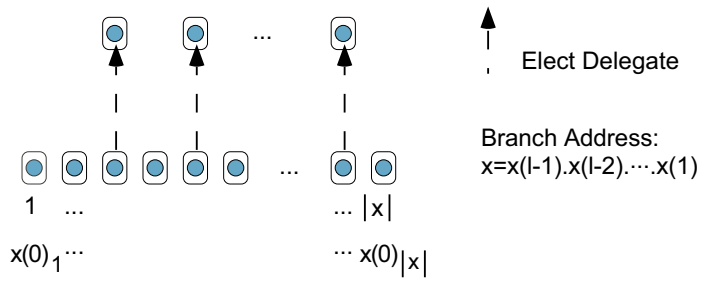


**Figure 3. Electing Delegates for a Group of Level 1**

In general, $\forall x = x(l-1).\cdots.x(i), |x|$ represents the number of different $x(i-1)$ that can be appended to $x$ to denote a legal branch address, or in other words, the number of populated subsystems of $x$. Quite obviously, for any such branch address $x = x(l-1).\cdots.x(i)$, $|x| \leq a_{i-1}$.

**Constructing a hierarchy.** Together with $R$ delegates for each other of the $|x_0(l-1).\cdots.x_0(2)|$ neighbor trees, the $R$ delegates of $|x_0|$ form a group of level 2.

Recursively, any branch address $x_0(l-1).\cdots.x_0(i)$ $(l > i > 0)$ is shared by altogether $|x_0(l-1).\cdots.x_0(i)| \leq a_{i-1}$ subtrees with a different $x(i-1)$, each represented by $R$ delegates. Together, these form a group of level $i$. (Figure 4). At the highest level $(l)$, there are $|\emptyset| = |x_0(l-1).\cdots.x_0(i)|_{i=l}$ subtrees.

Remember that by promoting a participant as delegate, that participant's knowledge does not increase. This merely means that it is known by more participants, not that itself will have to know more participants. Consequently, a participant which appears as delegate of level $i$ thus also appears as delegate of all levels $i$', such that $1 \leq i' < i$.

**Individual knowledge.** A given participant with address $x_0 = x_0(l-1).\cdots.x_0(0)$ knows for each of its branch addresses $x_0(l-1).\cdots.x_0(i)$ $(1 \leq i \leq l)$ all the $|x_0(l-1).\cdots.x_0(i)|$ different subtrees, and for each of those subtrees $R$ delegates. Furthermore, for level 1, it knows all $|x_0(l-1).\cdots.x_0(1)|$ participants.

Thus, the total number of participants known by participant $x_0(l-1).\cdots.x_0(0)$ is

$$|x_0(l-1).\cdots.x_0(1)| + \sum_{i=2}^{l} R\ |x_0(l-1).\cdots.x_0(i)| \tag{12}$$

where a delegate of level $i$ is also taken into account at any level below $i$. Also, the knowledge of the participant itself is considered (depending on the level of the considered participant, between 1 and $l$ occurrences.)

**Joint interest.** The total number of participants that a delegate at level $i$ with address $x_0(l-1).\cdots.x_0(0)$ represents is given by

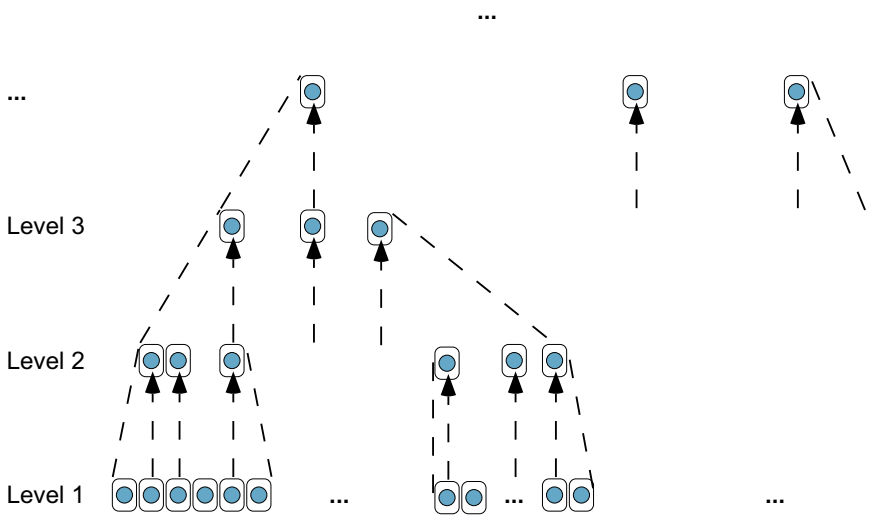


**Figure 4. Electing Delegates Recursively ($R = 3$)**

$$\| x_0(l-1).\cdots.x_0(i) \| = \sum_{x(i-1).\cdots.x(1)} |x_0(l-1).\cdots.x_0(i).x(i-1).\cdots.x(1)| \tag{13}$$

Hence, when considering that every such participant is expected to be interested in a given event with $p_1$, the delegate itself, on behalf of these participants, is interested in any event with a probability

$$p_i = 1 - (1 - p_1)^{\|x_0(l-1).\cdots.x_0(i)\|} \tag{14}$$

## 4.2 Membership Management

Due to the inherent complexity of the membership algorithm, we present the way the membership is managed more informally.

The membership views are exchanged between participants through gossips, i.e., they are piggybacked by event gossips, except in the absence of events (dedicated gossips are used in this case). Hence, every participant periodically sends information about a random level of its view of the hierarchy to a subset of the system.

### 4.2.1 Propagating Information

More precisely, each participant maintains a table for each level, representing the participant's view of that level. Membership information updating is based on gossip-pull. To that end, every line in every table has a timestamp associated. This represents the last time the corresponding line of the table was updated. Periodically, a participant randomly selects participants of a hierarchy level and gossips to those delegates. 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.

As explained above, such membership information can be piggybacked when gossiping events, or in the absence of such events, can be propagated with dedicated gossips. Similarly, as we will elucidate in Section 5.1.2, other gossips can be used to piggyback information. In some cases, this can even lead to a bidirectional updating as in anti-entropy (e.g., [14]).

**Joining.** When a participant decides to join, it needs to know at least one participant. That participant contacts the "lowest" delegates it knows that the new participant will have. This is made recursively, until the immediate

delegates of the new participant have been contacted. Hence, if the becoming participant contacts a "close" participant (if there are any), this procedure completes faster and induces less overhead.

Once neighbors (lowest-level neighbors of the becoming participant) have been contacted, these transmit their view of the system to the new participant.

The hierarchy is hence constructed stepwise, by adding one participant after another. An a priori knowledge of the approximate size of the hierarchy can in that sense be very useful to adjust parameters of the hierarchy, such as its depth: as we will show in the following sections, this parameter indeed has an impact on the performance of the system.

**Leaving and Failures.** The same lowest-level neighbors are also involved when a participant leaves. A participant wishing to leave will send a message to a subset of its lowest-level neighbors. These will remove the leaving participant from their views, and this information will successively propagate throughout the system through subsequent gossips.

For the purpose of detecting the failure of participants, every participant keeps track of the last time it was contacted by its lowest-level participants. This implements a simple form of failure detection, and makes sense at the lowest level, since such neighbors are supposed to be "close". The reduced average network latency makes failure detection more accurate. Section 6.1 discusses more refined ways of detecting failures.

#### 4.2.2 Subscription Patterns

The operation of compacting a table of level $i$ into a line of the table of level $i+1$ is called *condensation function* in Capt'n Cook. In our case, this function consists of the three following operations:

*Regroup patterns:* To represent the interests of all participants of the table, the subscription patterns of the respective participants must be regrouped. This is done in a way which avoids redundancies, i.e., not just by simply forming a conjunction of the individual patterns. A simple example of optimizing accessors representing simple method invocations is depicted in [8].

*Count participants:* The total number of participants at any level can be very useful for several kinds of heuristics. In particular, it enables the estimation of the number of gossip rounds necessary to complete the infection of all concerned participants.

*Select delegates:* Delegates have to be chosen based on a deterministic characteristic, since all participants in the same subsystem of level $i$ 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 participants, like their resources in terms of computing power or memory, or also the nature of their subscription patterns, to reduce the amount of "pure" forwarding of delegates, i.e., handling events as delegate on behalf of other participants, without being itself interested in these events. This optimization task is however not trivial: one can choose participants such that they *individually*, or *altogether*, cover as many interests of the represented participants as possible.

#### 4.2.3 Example Scenario

We depict the view of a small system in the case of multicasting based on type information. Subscriptions are made to unrelated types, e.g., $A$, $B$, and more fine-grained subscription patterns are not considered to keep this illustration intuitive.

We map IP addresses straightforwardly to our logical addresses ($l = 4, \forall 0 \leq i \leq l-1\ a_i = 256$). Every participant has a table representing its view of level 1, its view of level 2, a.s.o. Consider a possible configuration of the views for several participants sharing branch address 128.178.73, illustrated in Figure 5. The selected redundancy level $R$ is 3. 128.178.73.3 is delegate of level 3, which means that it is known by all participants with $x_3 = 128$.

## 5 Hierarchical Probabilistic Multicast (*hpmcast*)

In this section we present *hpmcast*, our gossip-based multicast algorithm which is based on the hierarchical membership outlined in the previous section.

**View of Level 4**

| $x_3$ | Types of interest | $x_2.x_1.x_0$ |
|---|---|---|
| 3 | A, C, D, E | 2.230.23, 18.2.78, 188.203.99 |
| 18 | B, C, E, F | 12.2.183, 12.34.24, 180.37.217 |
| 128 | A, B, C, D, F | 3.2.230, 18.120.2, 56.12.234 |

**View of Level 3** ($x_3 = 128$)

| $x_2$ | Types of interest | $x_1.x_0$ |
|---|---|---|
| 3 | A, B, C | 2.230, 18.2, 188.203 |
| 18 | B, C, D | 120.2, 122.34, 180.37 |
| 56 | C, F | 12.234, 18.220, 173.3 |
| 178 | A, B, C, F | 41.21, 73.3, 88.10 |

**View of Level 2** ($x_3.x_2 = 128.178$)

| $x_1$ | Types of interest | $x_0$ |
|---|---|---|
| 41 | A, C | 21, 23, 24 |
| 73 | A, B, C, F | 3, 17, 19 |
| 88 | B, F | 10, 13, 78 |
| 98 | A, B, C | 15, 17, 128 |
| 110 | C | 1, 6, 7 |

**View of Level 1** ($x_3.x_2.x_1 = 128.178.73$)

| $x_0$ | Types of interest |
|---|---|
| 3 | A, C |
| 17 | A |
| 19 | C, F |
| 115 | F |
| 116 | A, B, F |
| 119 | B, C |
| 124 | A, B |
| 223 | B |

**Figure 5. Hierarchical Membership View**

## 5.1 *hpmcast* Algorithm

The algorithm presented for *hpmcast* in Figure 6 differs from the *pmcast* algorithm presented in Figure 2 mainly by applying the above-mentioned hierarchical multicast scheme, and by furthermore making the inherent performing of garbage collection based on an estimation of the number of necessary rounds more explicit.

### 5.1.1 Level-Wise Multicasting

As we discussed earlier, the system is pictured as a hierarchy, and the multicasting procedure follows this structure. An event is first propagated in the highest level, from where it moves down level by level. As a consequence, the effective gossips, besides conveying events, also contain the level in which the event is currently being multicast.

Note here that this multicast algorithm does not comply with the notion of genuine multicast proposed in [15]: a genuine multicast differs from a *feigned multicast* by the *minimality property*: only participants which are interested in the considered event are effectively involved in the algorithm. Here, a delegate can be involved in the dissemination at a given level, though it is not itself interested. Just like *rfpbcast*, *hpmcast* is a feigned multicast according to [15], though one can expect *all* participants to be infected iff $p_1 = 1$. With *rfpbcast*, this is much more likely to occur, since it is the declared goal of that algorithm to treat participants regardless of their interests when disseminating, and to filter events only locally before possibly passing them to the application (layer).

Executed by participant$_i$

```
1: initialization
2: view[1..l]
3: gossips[1..l] ← ∅

4: task GOSSIP                                                          {every P milliseconds}
5:    for all level ∈ [l..1] do
6:       for all (event, prob, rounds) ∈ gossips[level] do
7:          if rounds < ROUNDS(level, prob) then                      {not too many rounds}
8:             rounds ← rounds + 1
9:             dests[1..F] ← ∅
10:            repeat F times                                         {choose potential destinations}
11:               dest ← RANDOM(level, dests)
12:               dests ← dests ∪ {dest}
13:               if event ◁ dest then
14:                  SEND(event, prob, rounds, level) to dest
15:         else
16:            if level > 1 then
17:               gossips[level] ← gossips[level] \ {(event, prob, rounds)}
18:               gossips[level-1] ← gossips[level-1] ∪ {(event, GETPROB(level-1, event), 0)}

19: upon RECEIVE(event, prob, round, level): do
20:    if forall level ∈ [1..l] ∄ (event, ..., ...) ∈ gossips[level] then   {buffer and deliver}
21:       gossips[level] ← gossips[level] ∪ {(event, prob, round)}
22:       if event ◁ participant_i then
23:          HPDELIVER(event)

24: upon HPMCAST(event): do
25:    gossips[l] ← gossips[l] ∪ {(event, GETPROB(l, event), 0)}

26: function ROUNDS(level, prob)                                        {expected number of rounds}
27:    return log(|view[level]| R prob)  ( 1/log(F prob+1) + 1/(F prob) )

28: function RANDOM(level, previous)                                    {choose random participants}
29:    return dest ∈ view[level] | dest ∉ previous

30: function GETPROB(level, event)                                      {probability of matching this event at this level}
31:    hits ← 0
32:    for all dest ∈ view[level] do
33:       if event ◁ dest then
34:          hits ← hits + 1
35:    return hits / (|view[level]| R)
```

**Figure 6. Hierarchical Probabilistic Multicast Algorithm**

**Receiving.** Upon reception of a gossip, the information about the level is used to place the event in the corresponding gossip buffer. To ensure that the event passes from one level to the next, it is crucial that a participant at level $i$ gossips a received event in any level $i' < i$, and thus also remains in the view of any of these subsequent levels.

**Multicasting.** A participant would only require gossip buffers from the highest level at which it appears, down to the bottom level, since it will not receive messages from higher levels. However, when HPMCAST-ing, it is reasonable that a participant takes part in the entire gossip procedure at all levels, especially at the topmost one. 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 upmost level. Also, since the membership is dynamic, a participant can be "bumped up" to a higher level at any moment, as well as it can be "dropped" to a lower level.

### 5.1.2 Parameters

As previously outlined, the expected number of rounds can be used to estimate the number of rounds necessary to disseminate a given event. Note here that this does not require participants to be synchronized, nor does it make any assumption on delivery delays. The computation of this expected number of rounds relies however on several parameters, like the fraction of interested participants, or the average message transmission loss (Equation 9).

**Fraction of interested participants.** The probability $p_1$ that a given participant is interested in a particular event at a given level can be simply measured by matching that given event against all the subscription patterns of all participants for that given level. This is a costly operation, but is only performed by a maximum of $R$ participants at each level except the topmost one, since this is the maximum number of processes infected initially in a subsystem. At the upmost level, only the participant effectively publishing the event will perform this matching.

Since delegates represent several participants, the measured probability can be expected to be higher than $p_1$ (except for the lowest level), according to 14.

**Expected number of rounds.** With the fraction of interest, the expected number of rounds for a given level can be computed. This requires the number of delegates forming the level (with respect to the subsystem of the considered participant), which is given by multiplying the number of different subsystems of level $i-1$ in the view of level $i$, i.e., the number of lines in the corresponding view table (noted $|view[i]|$ in Figure 6), by the number of delegates for each subsystem. If this number of delegates is not a system constant, one can alternatively simply use the total number of delegates in the view of level $i$.

**Environmental parameters.** Environmental parameters, such as the probability of message loss, or the probability of a crash failure of a participant, are to be considered when computing the expected number of rounds necessary to spread an event. These are however more difficult to approximate ([12]), especially the latter one. Like in most gossip-based algorithms, where simulations or analytical expressions enable the computing of "reasonable" values for parameters such as hops or forwards, choosing conservative values is the best way of ensuring a good performance. (For simplicity, these parameters have not been added in the algorithm.)

## 5.2 Analysis

For analysis, we presuppose a "regular" hierarchy, i.e., for any participant, all branch addresses derived from that participant's address $x = x(l-1).\cdots.x(0)$ have the same number of subsystems, which we denote by $a$.

$$\begin{gathered}\forall x, i \;\; x = x(l-1).\cdots.x(0), 1 \leq i \leq l \\ |x(l-1).\cdots.x(i)| = a \;\; \leq \;\; a_i\end{gathered} \tag{15}$$

Accordingly, the total number of participants in the system is simply given by

$$n = a^l \tag{16}$$

Also, we consider that, with respect to a given event, the participants interested in that event are uniformly distributed over the entire system.

### 5.2.1 View Size

Every participant must know the delegates of every level as shown in Figure 4 (for $l = 4$ and $R = 3$).

According to Equation 12, a participant knows the following number of participants for the different levels of a regular hierarchy:

$$m_i = \begin{cases} Ra & 1 < \ i \ \leq \ l \\ a & i \ = \ 1 \end{cases} \tag{17}$$

which adds up to a total of

$$\begin{aligned} m \ &= \ \sum_{i=1}^{l} m_i = Ra(l-1) + a \\ &\in \ \emptyset(lRn^{1/l}) \qquad l \geq 2 \end{aligned} \tag{18}$$

### 5.2.2 Expected Number of Rounds

Based on Equation 14, we can determine that, in a regular hierarchy,

$$p_i = 1 - (1 - p_1)^{a^{i-1}} \tag{19}$$

Furthermore, the number of expected rounds can be approximated by the sum of the rounds spent at each level of the hierarchy:

$$T_{tot} \ = \ \sum_{i=1}^{l} T_i \ = \ \sum_{i=1}^{l} T(m_i p_i, F p_i) \tag{20}$$

This expression is pessimistic, by neglecting the very fact that every interested subsystem, except the topmost one, starts with an expected number of infected participants which is bigger than 1, namely $R$.[9] These delegates already being infected, we can obtain a more precise expression by subtracting at each level the time necessary to get from 1 to $R$ infected participants:

$$T'_{tot} \ = \ \sum_{i=1}^{l} T(m_i p_i, F p_i) - (l-1)T(R, F) \tag{21}$$

The probability in the terms added in Equation 21 reflects that, at each level, every interested subsystem is represented by $R$ delegates, which are all (probability of 1) interested in the considered event.

[9] To be fully accurate, we would also have to consider that the number of participants to infect at the topmost level is in the general case given by $m_l p_l + 1$, since the HPMCAST-ing participant is the initially infected one. That participant would have to be furthermore considered in any of its own subsystems.

### 5.2.3 Number of Infected Participants

A precise analysis of the spreading of the event in time, yielding a distribution of the probability for the infection of a fraction of the system as in the above algorithms, introduces a high complexity in this case due to the decomposition of the entire system into subsystems. (At the lowest level, the number of different states is potentially $a^{(a^{l-1})}$ for a given number of rounds.) One can however reuse the same Markov chain introduced previously to compute the expected number of infected participants (or an approximation as in [3], or [12]) in a subsystem of level $i$ ($1 \leq i \leq l$) after gossiping at that given level:

$$E[s_{T_i}] \;=\; \sum_{j=0}^{m_i p_i} P(m_i p_i, F p_i)[s_{T_i} = j] j \tag{22}$$

This is again a pessimistic value, since we do not consider the possibility that the subsystem ($1 \leq i < l$) initially comprised more than one infected participant (cf. 21).

We are now able to compute the probability that an "entity" of level $i$ is infected after gossiping at that level (provided the corresponding subsystem of level $i+1$ was initially infected):

$$r_i \;=\; 1 - \left(1 - \frac{E[s_{T_i}]}{m_i p_i}\right)^{\frac{m_i}{a}} \tag{23}$$

For all levels, except the lowest one, an "entity" of level $i$ means a subsystem of level $i$ (that is, its $R$ delegates for that level). At the lowest level, an "entity" refers to a participant. $r_i|_{i=1}$ hence simply resumes to $\frac{E[s_{T_i}]}{a p_i}$, the expected fraction of participants infected when gossiping in a system of level $l$.

Provided that $g_{i+1} = j \leq a^{(l-i)} p_{i+1}$ entities were infected at level $i+1$ (in total), the probability of ending up with $g_i = k$ entities infected at level $i$ ($1 \leq i \leq l$) is given as follows:

$$\begin{aligned} p_{ijk} &= P[g_i = k | g_{i+1} = j] \\ &= \begin{cases} P[g_{i+1} = j] \binom{j a p_i}{k} r_i^k (1 - r_i)^{j a p_i - k} & k \leq j a p_i \\ 0 & k > j a p_i \end{cases} \end{aligned} \tag{24}$$

Finally, we can compute the probability of having $k$ entities infected at level $i$:

$$P[g_i = k] \;=\; \begin{cases} 1 & i = l+1, k = 1 \\ 0 & i = l+1, k \neq 1 \\ \sum_{j=k/(j a p_i)}^{a^{(l-i)} p_{i+1}} P[g_{i+1} = j] p_{ijk} & 1 \leq i \leq l \end{cases} \tag{25}$$

### 5.2.4 Expected Number of Infected Participants

Based on the upper equation, we are able to express the expected number of infected entities $g_i$ after gossiping in a subsystem at level $i$ ($\leq i \leq l$)

$$E[g_i] \;=\; r_i a p_i \tag{26}$$

Consequently, the expected number of totally infected participants is given by the following expression:

$$\prod_{i=1}^{l} E[g_i] \tag{27}$$

The expected reliability degree can be simply obtained by dividing the upper expression by the number of effectively interested participants, $n p_1$.

## 5.3 Simulation Results

We have simulated *hpmcast*, by simulating successive rounds, and observing the spread of a single obvent. Different parameters of the algorithm have been varied. We use these results to pinpoint the limitations of the basic variant of *hpmcast* presented throughout this section. Ways of counteracting these limitations, not considered here for simplicity of presentation, will be discussed in Section 5.4.

### 5.3.1 Interest

On the one hand, Pittel's formula gives extremely good results when the system grows in size. This is visible in Figure 7, which shows a very good overall degree of reliability. On the other hand, smaller systems, and hence "rather small" values for $p_1$, are not well captured by this asymptote. Figure 8 zooms in on the previous figure, to show the decrease of reliability with small values for $p_1$. Indeed, the computed expected number of rounds increases first with a decreasing $p_1$, before decreasing quickly and becoming 0 in $p_1 n = 1$ (Figure 9). Though this last value reflects reality, the asymptote gives less accurate information towards that value. Even with a better approximation this problem can be observed, since the stochastic approaches underlying epidemiology, and hence gossip-based algorithms, indeed reflect the interest of large populations. This loss in reliability was hence expected, and there are several ways of counteracting it (see Section 5.4).

More precisely, the interpretation of "rather small" depends on the considered level, and hence on $p_1 i$, but also on $m_i$. In fact, $p_i$ becomes bigger at every level $i$, and hence, the number of potentially interested participants increases, making higher levels less prone to this type of undesired effect. Lower levels, especially the lowest one, are most likely to reach critically low sizes, since their $p_i$ become smaller. At the level $i = 1$, $m_i$ is furthermore smaller than in all above layers.

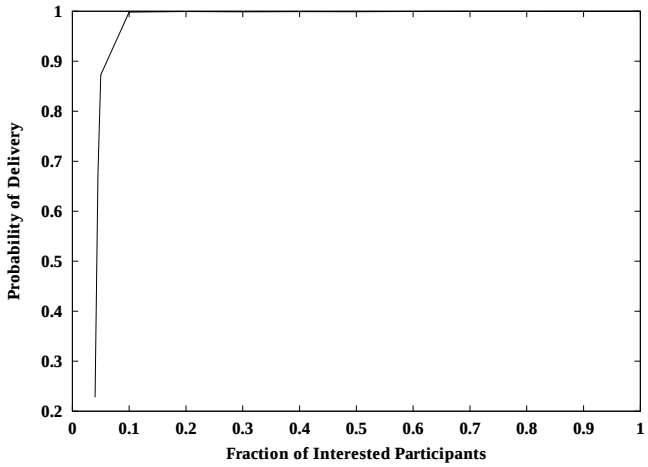


**Figure 7. Varying $p_1$; $n \approx 10000$ $(a = 22), l = 3, R = 3, F = 2$**

### 5.3.2 Fanout

As typical for gossip-based algorithms, increasing the fanout decreases the number of rounds necessary to infect the concerned participants. Since in our case we have not considered the gossiping of digests for mutual updates based on gossiper-pull, but limit the entire propagation of events by the number of expected rounds, the estimation of this latter value has a certain impact on reliability.

### 5.3.3 Scalability

It is however not clear whether, and how, the performance of *hpmcast* is impacted when increasing the scale of the system. For the above reasons, a slightly increased number of participants leads to a better approximation of the number of necessary rounds, and can even lead to a better reliability degree. On the other hand, one can

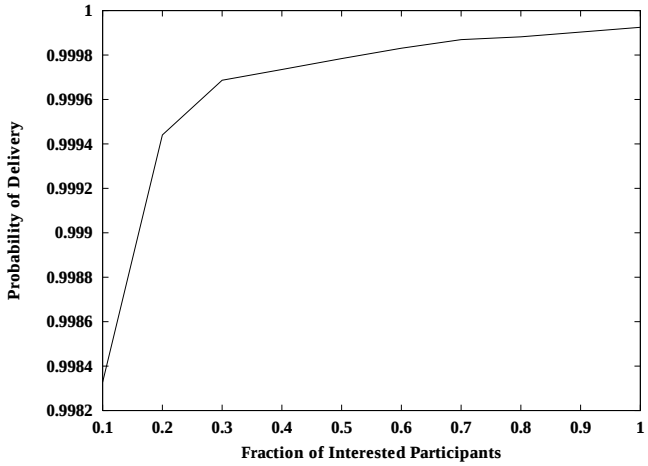


**Figure 8. Reliability Depending on $p_1$; $n \approx 10000\ (a = 22), l = 3R = 3, F = 2$**

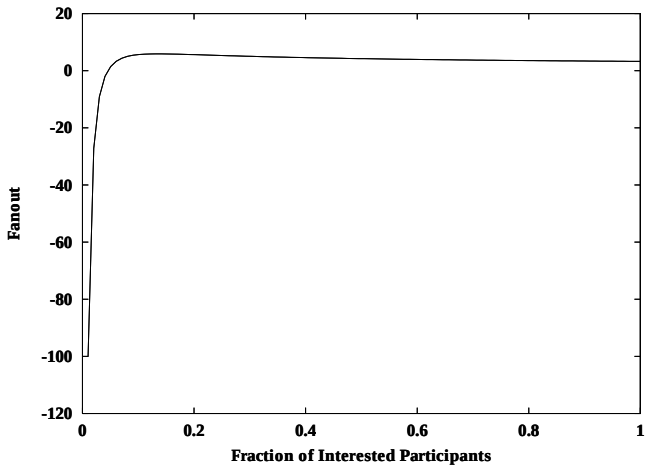


**Figure 9. Rounds Depending on $p_1$; $n = 100F = 2$**

also expect that increasing the size of the system, and hence the number of expected rounds, also increases the potential difference between latter value and the *effective number* of necessary rounds.

As conveyed by Figure 11 in any case, *hpmcast* shows very good scalability properties when increasing $a$ in a hierarchy of fixed depth (the system size increases following $a^l!$). The scalability however depends on the proportion of interested participants. With a small $p_1$, the above-mentioned problem manifests itself in a slightly stronger decreasing reliability with an increased system size. This effect is however counteracted by the better approximation of the number of necessary rounds with larger (sub)system sizes.

### 5.3.4 Hierarchy Depth

Similarly, one can expect that increasing the hierarchy depth has a similar impact on the reliability degree than decreasing $p_1$, since one can expect the size $m_i$ of the individual groups to decrease, and reach quickly a critically small size.

This can be indeed observed in certain cases, just like the opposite. The various intervening parameters seem to hinder the appearance of any clear trend.

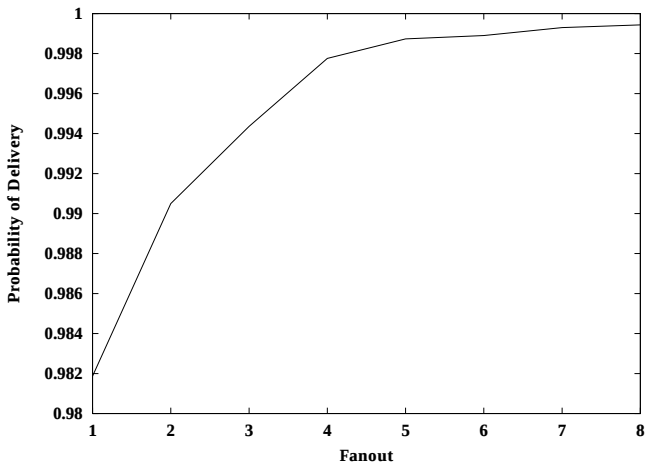


**Figure 10. Reliability Depending on $F$; $n \approx 10000\,(a = 22), l = 3, R = 3, p_1 = 0.2$**

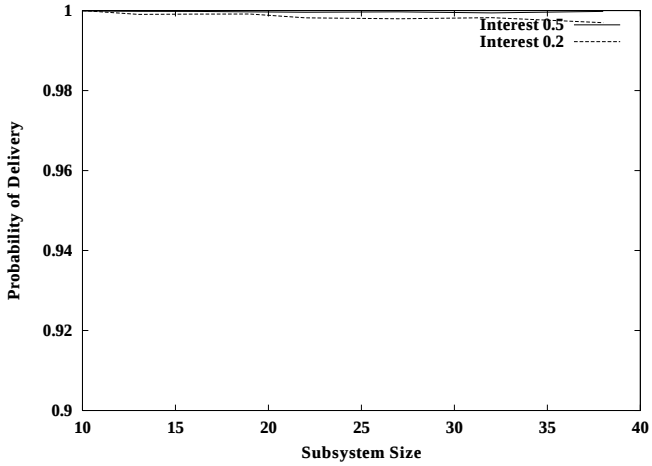


**Figure 11. Scalability when Increasing $a$; $l = 3, R = 4, F = 3$**

### 5.3.5 Redundancy

Increasing redundancy obviously increases reliability, since the probability that at any given level an interested subsystem becomes isolated decreases by increasing $R$. This redundancy however has less impact on the the main problem encountered with small $p_1$, since this problem appears first at the lowest level, and any redundancy factor $R > 1$ leads to multiplying the number of potentially interested participants by $R$. Hence, when further increasing $R$, only little more reliability is gained (see Figure 12).

## 5.4 Optimizations

We have considered several possibilities of improving the behavior of *hpmcast* for small $p_1$.

### 5.4.1 Possible Improvements

Besides adding a gossip-based exchange phase for digests of received events (periodically sending identifiers of buffered events to interested participants), one can basically distinguish two ways of improving the performance of *hpmcast*. These are namely, (1) using Pittel's asymptote, however increasing artificially the number of interested participants, and (2) by applying another approximation of the number of necessary rounds. The second approach

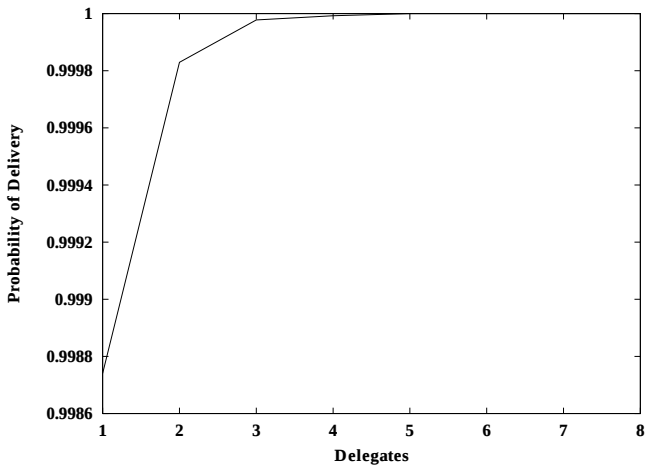


**Figure 12. Reliability Depending on $R$; $n \approx 10000\,(a = 22), l = 3, R = 3, p_1 = 0.2$**

can for instance be achieved by using a more rough approximation of $T(n, F)$, possibly associated with an expected reliability degree (e.g., [3]).[10] We have however been more attracted by the accuracy offered by Pittel's asymptote, and are more interested in very large systems (even exceeding the feasibility of a simulation), where even an absolutely small $p_1$ still leads to a large number of interested participants in a lowest-level group.

### 5.4.2 Increasing the Audience

We have adopted a more pragmatic approach, consisting in increasing the audience by adding participants to the set of interested participants. To that end, we have modified the above algorithm to include non-interested participants if the number of interested participants in the system drops below a threshold $D$. In that case, every involved participant decides that the $D$ first participants in the view of the corresponding level are interested, in addition to the remaining effectively interested participants. By fixing a lower bound on the desired reliability degree, $D$ can be obtained through analysis or simulation. The result of such an improvement is illustrated by Figure 13, which compares the original degree of reliability with the improved one.

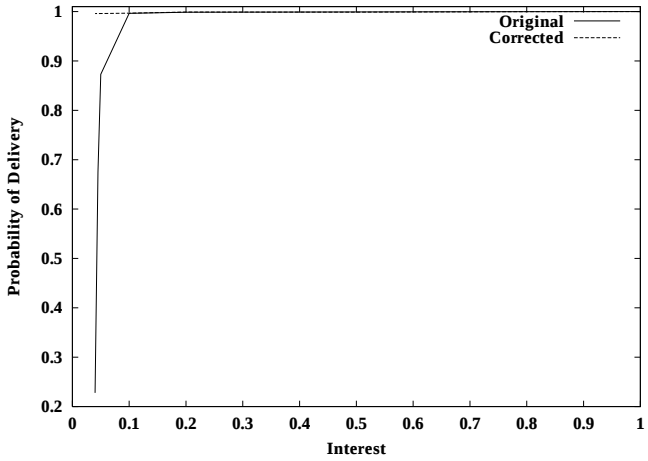


**Figure 13. Varying $p_1$; $n \approx 10000\,(a = 22), l = 3, R = 3, F = 2$ with Optimization**

[10] With such an approach, the term $\frac{E[s_{T_i}]}{m_i p_i}$ in 23 can be directly replaced by the expected reliability degree of gossiping in a system of size $m_i p_i$.

# 6 Discussion

We discuss several issues related to our hierarchical approach outlined above, including the effect of increasing the depth of the hierarchy.

## 6.1 Exploiting Hierarchies

The hierarchical organization of participants can be exploited for several aspects of reliable event delivery.

### 6.1.1 Filtering

Higher-level delegates receive more gossips than lower-level participants. One way of reducing the load on these delegates could consist in applying filters with a weaker accuracy, in order to speedup filtering.

**Filter coverage.** More precisely, when filtering at a high level, events might successfully pass this filtering, with respect to a set of participants, though these events are of interest for none of these participants, nor for any of the participants they might represent. This idea is based on the notion of *filter coverage* [25], where a first filter is said to be *covered* by a second filter if everything that passes the first filter also passes the second one (but not necessarily vice-versa). By applying such a weaker filter instead of the covered precise filter, filtering cost can be substantially reduced.

**Event coverage.** The difference becomes more important if the format of the filtered events follows this modified accuracy. For instance, (parts of) events can be reflected by more primitive representations, such as XML structures transferred with the effective event objects. This can avoid the deserialization of events every time they are filtered. At a very first level, an event can for instance even be viewed as "data plus a type identifier".

**Example.** A resulting mapping of filter and event coverage to hierarchy levels could be the following (by increasing hierarchy levels):

*Method invocations:* The original filter expressed on arbitrary method invocations, as in the model presented in [8], is applied in the subscriber's process, to ensure that the subscriber receives *exactly* what was specified through the filter.

*Attribute comparisons:* The next level already works with an XML-like representation of events, reflecting their attributes. Attribute comparisons, though expressed through access methods in the initial filter code, are mapped to reads of the corresponding entries of the XML structure with comparisons. Only such simple comparisons are applied at this level, which presupposes that the middleware has the possibility of identifying the attributes of arbitrary event objects.[11] This can be for instance achieved through conventions on the names of corresponding methods (`getxxx()`). Filters applied at this level represent a simple conjunction (without redundancies) of the filters regrouped from the different subscribers at the next hierarchy level.

*Selective comparisons:* At the next level, only more certain attributes are verified. For instance, attributes for which a subsystem gives different ranges of values of interest are not filtered anymore. Also, filters for related types, which are in a subtype relation, are replaced by a compound filter possibly only checking events for conformance with the most general type. In a hierarchy with more levels, one could imagine several degrees for this selectivity.

*Type conformance:* At the highest level, events are only filtered based on their event type. As suggested above, type information can even be attached as an identifier encoded in a set of bytes. This circumvents any deserialization of events, or parsing of XML descriptions.

[11] Note here that, on the one hand, directly accessing attributes in *filters*, in a way controlled by the middleware, can lead to a considerable benefit in terms of performance, and should hence not be precluded. Supporting attributes as the *only* properties of events for describing *subscription patterns*, on the other hand, is very compelling and should be prohibited.

When covering filters, the effective probability $p_1$ of interest for any given participant cannot be as easily computed based on the previous level. In contrast, without weakening filters, one could compute an estimation of the probability $p_1 i$ at a given level based on $p_1 i + 1$ and Equation 19, and then improve that approximation by successively matching the event against $F$ participants and adjusting the effective matching rate.

#### 6.1.2 Dissemination

The hierarchical disposition of participants can also be used to apply different dissemination media. As an example, a UDP Broadcast could be used at the lowest level instead of gossips, if for instance the lowest level is characterized by highly populated LANs.

One can also imagine using a more reliable network protocol, also for intermediate levels. Depending on the desired compromise between reliability and performance, a "more" reliable algorithm can be very advantageous at the upmost level. Indeed, an unsuccessfully completed attempt of infecting a higher-level "entity" induces a domino effect, by isolating the entire subsystem.

#### 6.1.3 Failure Detection

Last but not least, the membership can be implemented in a way that applies different failure detection mechanisms at different levels. For instance, delegates of a given level can decide to autonomously exclude other participants if they have not received any gossips from them for some time.

At a lower level, e.g., LAN, participants can apply a less "passive" style, by actively pinging participants from which no gossips have been received for a long time before effectively removing them from views.

### 6.2 Broadcasting with *hpmcast*

It might be interesting to apply our hierarchical membership to the broadcasting of gossips. We show here that the number of expected rounds necessary to infect the entire system does not depend on the use of a hierarchy, more precisely, on the number of levels in the hierarchy, in the case of a broadcast ($p_1 = 1$).

In fact, for $p_1 = 1$, the number of rounds necessary to multicast an event is similar to the number of rounds in the non-hierarchical algorithm (*pmcast*), or, in other terms, $T_{tot} = T(n, F)$.

Indeed, according to Equation 21 (and 17):

$$\begin{aligned} T'_{tot} &= (l-1)T(aR,F) + T(a,F) - (l-1)T(R,F) \\ &= \left[log\ (Ra)^{l-1} + loga - log\ (R^{l-1})\right]\left(\frac{1}{F(1-\epsilon)(1-\tau)} + ...\right) \\ &= log\ (n)\left(\frac{1}{F(1-\epsilon)(1-\tau)} + \frac{1}{log(F(1-\epsilon)(1-\tau)+1)}\right) \\ &= T(n,F) \end{aligned} \tag{28}$$

Consequently, the depth of the hierarchy has no impact on the time it takes to broadcast an event in the entire system.

This might seem surprising at first glance, since in this *hpmcast* algorithm, an event is *simultaneously* gossiped in $a^{l-i}$ distinct subsystems at level $i$. As however already pointed out in [31], a gossiped event spreads slowly at the beginning, since very little participants are infected and can hence infect susceptible participants, as well as towards the very end, since an increasing fraction of the system is already infected, and these "absorb" many gossips. In between, the spreading proceeds quickly. Increasing the size of the system only increases the number of necessary rounds logarithmically (4), which makes gossip-based approaches so attractive for large scale settings.

### 6.3 Levels

We analyze the effect of increasing the depth of the hierarchy on the performance of the system, through several aspects.

### 6.3.1 View Size

The first considered measure is the size of the view that every participant has for its corresponding subsystem.

We can show that the number of participants that a given participant knows is minimal with a hierarchy level $l = log\ n$. More precisely, $m$ decreases as $l$ increases, as long as $l\ < log\ n$.

Indeed, based on 18, and the fact that

$$\frac{d(n^{1/l})}{dl} = -\frac{log\ n}{l^2}\, n^{1/l} \tag{29}$$

we can see that $m$ decreases as long as $l \leq log\ n$:

$$\begin{aligned} \frac{dm}{dl} &= Rn^{1/l}\left(1 - \frac{log\ n}{l}\right) \\ &< 0 \qquad \forall l < log\ n \end{aligned} \tag{30}$$

In fact, $l = log\ n$ is a minimum:

$$\begin{aligned} \frac{d^2m}{dl^2} &= Rn^{1/l}\frac{log^2\ n}{l^3} \\ \left(\frac{d^2m}{dl^2}\right)_{l=logn} &= \frac{R\ e}{log\ n} \\ &> 0 \end{aligned} \tag{31}$$

In practice, this limit is probably never reached, since in most cases $R$ will be chosen such that $R > e$:

$$\begin{aligned} & R > e \\ \Rightarrow & a > e && (a \geq R) \\ \Rightarrow & l < log\ n && (a = n^{1/l}) \end{aligned} \tag{32}$$

### 6.3.2 Failure Sensitivity

Increasing the depth of the hierarchy however also has effects on the stability of the membership.

**Participants to be updated.** Increasing $R$ improves fault tolerance, by reducing the probability of partitioning, and the probability of isolation of an interested subsystem. This is however not the only measure of reliability. For instance, the average number of participants whose membership views have to be updated upon failure of any given participant decreases as $l$ increases.

For a given level $i$, we have $d_i$ participants:

$$d_i = \begin{cases} Ra & i = l \\ Ra^{l-i}(a-1) & 1 < \ i < l \\ a^{l-1}(a-R) & i = 1 \end{cases} \tag{33}$$

The number of participants which know a given participant of level $i$, i.e., the number of participants which have to be updated upon failure of a given participant of level $i$ is

$$u_i = a^i - 1 \quad 1 \leq i \leq l \tag{34}$$

The average number of participants which have to be updated after a failure is hence given by

$$\begin{aligned} u_{upd} &= \frac{1}{n}\sum_{i=1}^{l} d_i u_i \\ &= (a-1)[R(l-1)+1] \end{aligned} \tag{35}$$

which in fact corresponds to the number of participants known by every participant *without considering duplicates*, i.e., by counting a delegate at level $i$ only for that level, and not for lower levels. This is not really surprising, since every participant knows $u_{upd}$ distinct participants, and hence, in average, a participant is known by $u_{upd}$ participants. $u_{upd} \in \emptyset(n^{1/l})$ scales well with $n$, and decreases similarly to $m$.

The failure of a participant which is not delegate can be dealt with more easily than the failure of a delegate. Latter failure type involves more membership updates. By increasing the depth $l$ of the hierarchy, the number of participants which are delegates of some level increases. However, one can expect the number of participants that have to be updated to decrease: a hierarchy of 1 level will see the updating of all participants in the system upon the failure of a single participant, while in a hierarchy of 2 levels already, a failed participant is with a big probability only known by approximately a number of participants equal to the square root of the size of the system, while only $R$ times that same number of participants are known by all participants, etc.

**Average level of a participant.** This is however not the only measure of stability for the hierarchy. Indeed, if a participant of a higher level fails, the information about its failure will have to travel over more levels, which becomes increasingly important if such updates are piggybacked by event gossips, like in our case.

In turns out that the average number of levels of the hierarchy that an update will travel increases with the depth of the hierarchy, since the average level of a participant in the hierarchy increases as the number of levels increases.

This result can be intuitively easily explained, since in a hierarchy consisting of only 1 level, any participant can potentially only reach level 1, while in a hierarchy of level 2 already, several participants will reach level 2, etc.

Consider the average level $l_{upd}$ of all participants given by the following expression:

$$\begin{aligned} l_{upd} &= \frac{1}{n}\sum_{i=1}^{l} d_i i \\ &\approx \frac{1}{n}\sum_{i=1}^{l} R(a-1)a^{l-i}i \qquad (large\ l) \\ &\approx Ra\int_{i=1}^{l} a^{-i}i\ di \\ &= \frac{Ra}{loga}\left(ia^{-i} - \frac{a^{-i}}{loga}\right)_{i=1}^{l} \\ &= \frac{l^2 R n^{1/l}}{nlogn}\left(1 - \frac{1}{logn}\right) + \frac{lR}{logn}\left(\frac{l}{logn} - 1\right) \end{aligned} \tag{36}$$

Both of these remaining terms increase as $l$ increases.

Hence, increasing the number of levels in the hierarchy, as intuition suggests, cannot be done indefinitely without side effects. Indeed, as already illustrated by simulation results, similar tradeoffs can be expected concerning the dissemination of the events in the hierarchy. Parameters, such as the number of total rounds, the obtained reliability degree, but moreover also the number of infected participants which are not themselves interested in a given event, but only act as delegates, and the filtering load on an average participant might probably not be modified without affecting the others.

## 7 Conclusions

The Hierarchical Probabilistic Multicast (*hpmcast*) algorithm presented in this chapter is a novel gossip-based algorithm, which abides well to strongly dynamic and largely scaled settings, such as TPS. The principles adopted in *hpmcast* are nevertheless general, and do not only make sense in our precise context of TPS, but in any form of completely decentralized, peer-to-peer architecture.

*hpmcast* exploits locality and redundancy properties of the underlying system, and intrinsically reduces the view of each participant.

The presented analysis and simulations convey the strong scalability of *hpmcast*, and enable the trimming of parameters, e.g., based on a prediction of the upper bound on the number of participants in the system. As a result of the inherent compromises between certain faces of scalability, e.g., memory, as well as computing and network resources, it is difficult to define optimal values for parameters such as the depth of the hierarchy.

## References


[1] M.K. Aguilera, R.E. Strom, D.C. Sturman, M. Astley, and T.D. Chandra. Matching Events in a Content-Based Subscription System. In *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing (PODC '99)*, pages 53–62, November 1999.

[2] Y. Amir, D. Dolev, S. Kramer, and D. Mahlki. Membership Algorithms for Multicast Communication Groups. In *6th Intl. Workshop on Distributed Algorithms proceedings (WDAG)*, pages 292–312, November 1992.

[3] N.T.J. Bailey. *The Mathematical Theory of Infectious Diseases and its Applications (second edition).* Hafner Press, 1975.

[4] K.P. 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.

[5] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Maintenance. In *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing (PODC '87)*, pages 1–12, August 1987.

[6] DACE Distributed Asychronous Computing Environment. http://www.d-a-c-e.com.

[7] P.Th. Eugster, R. Boichat, R. Guerraoui, and J. Sventek. Effective Multicast Programming in Large Scale Distributed Systems. *Concurrency and Computation: Practice and Experience*, 13(6):421–447, May 2001.

[8] P.Th. Eugster and R. Guerraoui. Content-Based Publish/Subscribe with Structural Reflection. In *Proceedings of the 6th Usenix Conference on Object-Oriented Technologies and Systems (COOTS'01)*, pages 131–146, January 2001.

[9] P.Th. Eugster and R. Guerraoui. Hierarchical Probabilistic Multicast. Technical Report DSC/2001/051, Swiss Federal Institute of Technology, Lausanne, October 2001.

[10] P.Th. Eugster, R. Guerraoui, S. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight Probabilistic Broadcast. In *IEEE International Conference on Dependable Systems and Networks (DSN 2001)*, pages 443–452, June 2001.

[11] P.Th. Eugster, R. Guerraoui, and J. Sventek. Distributed Asynchronous Collections: Abstractions for Publish/Subscribe Interaction. In *Proceedings of the 14th European Conference on Object-Oriented Programming (ECOOP 2000)*, pages 252–276, June 2000.

[12] P.Th. Eugster, P. Kouznetsov, and R. Guerraoui. $\Delta-$Reliable Broadcast. Technical Report DSC/2001/010, Swiss Federal Institute of Technology, Lausanne, January 2001.

[13] S. Floyd, V. Jacobson, S. McCanne, C. G. Liu, and L. Zhang. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. *IEEE/ACM Transactions on Networking*, pages 784–803, November 1996.

[14] R. Golding. *Weak-Consistency Group Communication and Membership*. PhD thesis, University of California at Santa Cruz, December 1992.

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

[16] K. Guo, M. Hayden, R. van Renesse, W. Vogels, and K.P. Birman. GSGC: An Efficient Gossip-Style Garbage Collection Scheme for Scalable Reliable Multicast. Technical Report TR97-1656, Cornell University, Computer Science, December 1997.

[17] I. Gupta, R. van Renesse, and K.P. Birman. Scalable Fault-Tolerant Aggregation in Large Process Groups. In *IEEE International Conference on Dependable Systems and Networks (DSN 2001)*, pages 433–442, June 2001.

[18] V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. In S. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. Addison-Wesley, 2nd edition, 1993.

[19] H.W. Holbrook, S.K. Singhal, and D.R. Cheriton. Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation. In *Proceedings of the 1995 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '95)*, pages 328–341, August 1995.

[20] A.-M. Kermarrec, L. Massoulie, and A.J. Ganesh. Reliable Probabilistic Communication in Large-Scale Information Dissemination Systems. Technical Report MSR-TR-2000-105, Microsoft Research Cambridge, October 2000.

[21] P. Kouznetsov, R. Guerraoui, S.B. Handurukande, and A.-M. Kermarrec. Reducing Noise in Gossip-Based Reliable Broadcast. In *Proceedings of the 20th IEEE Symposium On Reliable Distributed Systems (SRDS'01)*, October 2001.

[22] M.-J. Lin and K. Marzullo. Directional Gossip: Gossip in a Wide Area Network. In *Proceedings of the 3rd European Dependable Computing Conference (EDCC-3)*, pages 364–379, September 1999.

[23] M.-J. Lin, K. Marzullo, and S. Masini. Gossip versus Deterministically Constrained Flooding on Small Networks. In *Proceedings of the 14th International Conference on Distributed Computing Distributed Computing (DISC 2000)*, pages 253–267, October 2000.

[24] L.E. Moser, P.M. Melliar-Smith, and V. Agrawala. Membership Algorithms for Asynchronous Distributed Systems. In *Proceedings of the 11th IEEE International Conference on Distributed Computing Systems (ICDCS '91)*, pages 480–489, May 1991.

[25] G. Mühl. Generic Constraints for Content-Based Publish/Subscribe Systems. In *Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS)*, September 2001.

[26] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R.E. Strom, and D.C. Sturman. Exploiting IP Multicast in Content-Based Publish-Subscribe Systems. In *Proceedings of the 3rd IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000)*, pages 185–207, April 2000.

[27] J. Orlando, L. Rodrigues, and R. Oliveira. Semantically Reliable Multicast Protocols. In *Proceedings of the 19th IEEE Symposium On Reliable Distributed Systems (SRDS'00)*, October 2000.

[28] O. Ozkasap, R. van Renesse, K.P. Birman, and Z. Xiao. Efficient Buffering in Reliable Multicast Protocols. In *Proceedings of the 1st International COST264 Workshop on Networked Group Communication (NGC '99)*, pages 188–203, November 1999.

[29] S. Paul, K.K. Sabnani, J.C. Lin, and S. Bhattacharyya. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, 15(3):407–421, April 1997.

[30] R. Piantoni and C. Stancescu. Implementing the Swiss Exchange Trading System. In *Proceedings of the 27rd IEEE International Symposium on Fault-Tolerant Computing (FTCS '97)*, pages 309–313, June 1997.

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

[32] Q. Sun and D.C. Sturman. A Gossip-Based Reliable Multicast for Large-Scale High-Throughput Applications. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN 2000)*, pages 347–358, July 2000.

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

[34] R. van Renesse, Y. Minsky, and M. Hayden. A Gossip-Style Failure Detection Service. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, September 1998.

[35] B. Whetten, T. Montgomery, and S. Kaplan. A High Performance Totally Ordered Multicast Protocol. In *Theory and Practice in Distributed Systems*, number 938 in LNCS, pages 33–54. Springer, 1995.