Data-Aware Multicast

S. Baehni P. Th. Eugster R. Guerraoui
Distributed Programming Laboratory
EPFL

Technical Report IC/2003/73

Abstract

This paper presents a multicast algorithm for peer-to-peer dissemination of events in a distributed topic-based publish-subscribe system, where processes publish events of certain topics, organized in a hierarchy, and expect events of topics they subscribed to. Our algorithm is "data-aware" in the sense that it exploits information about process subscriptions and topic inclusion relationships to build dynamic groups of processes and efficiently manage the flow of information within and between these process groups. This "data-awareness" helps limit the membership information that each process needs to maintain, preserves processes from receiving messages related to topics they have not subscribed to and provides the application a means to control, for each topic in a hierarchy, the trade-off between the message complexity and the reliability of event dissemination. We convey this trade-off through both analysis and simulation.

I. Introduction

Many distributed applications are best supported by publish/subscribe systems that ensure the dissemination of *events* from *publisher* processes to *subscriber* ones. The matching between publishers and subscribers is typically achieved through event *topics*. All topic-based publish/subscribe systems, including the very early ones, e.g., TIBCO [18] and Vitria [17], or more advanced ones, organize these events in a hierarchical manner. Ideally, we expect from a topic-based publish/subscribe system that all processes that subscribe to a given topic receive all events produced on that topic (i.e., *reliability*), and no process receives any event of a topic it is not *interested in*[1] (i.e., increasing *efficiency* by avoiding "parasite messages"). Furthermore, we would like to minimize the total number of messages sent in the system (i.e., *message complexity*), while reducing the size of the membership knowledge each process needs to maintain (i.e., *memory complexity*).

Gossip-based (or so called epidemic) information dissemination algorithms ([9], [1]) are appealing candidates to support some of these requirements. They indeed limit the total number of messages sent in the system and can be tuned to limit the memory complexity of every process [6], while ensuring good overall reliability. The basic idea is inspired by the way infectious diseases propagate. Each infected individual (i.e., process) randomly tries to infect (i.e., send events to) a small subset of the population, and so on until the entire population is infected (hence achieving highly "reliable" dissemination).

Yet, gossip-based algorithms are inherently best suited for *broadcasting* within a group of processes. When viewing the set of publisher and subscriber processes involved in an application as such a group, processes receive many parasite messages regarding events of topics they are not interested in. A breakdown of this group into smaller groups, corresponding each to a

The work presented in this paper was supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

[1]A process p_l is said to be *interested* in a topic T_i, if p_l either wants to publish an event of topic T_i, or if p_l has subscribed to T_i.

topic, is already a much better approach, but nicely illustrates the difficulty of minimizing the overhead of memory complexity when avoiding parasite messages. By mapping topics *arranged in a hierarchy* to groups, every group gathers either exactly (1) the *publishers* of a topic, or (2) the *subscribers* of a topic. With (1), a subscriber for a topic T_i has to become member not only of the group representing T_i, but also of every group representing a subtopic of T_i, and hence moreover has to constantly listen for new subtopics of T_i. With (2), a publisher of a topic T_i has to publish these events not only within the group representing T_i, but within every group representing a supertopic of T_i, which increases the load on the publishers and makes of these single points of failures. Performing selective gossiping based on message *contents* (and viewing topics as a particular instance thereof), as in [5], might look like a viable alternative at first glance. However, this approach makes it impossible to subdivide the overall set of publishers and subscribers to reduce memory complexity without introducing parasite messages, i.e., without forcing processes to participate in the dissemination of events beyond their own interests.

We present in this paper a completely decentralized multicast algorithm that is *data-aware* in the sense that it makes use of information about the hierarchical disposition of topics[2] to dynamically create groups of processes, according to their interests, and interconnect these groups based on inclusion relations between topics. Published events are propagated within every group in a gossip-based manner, and disseminated between groups following a bottom-up approach imposed by the topic hierarchy. This *data-awareness*, combined with an underlying gossip-based membership technique (we consider the one of [10]), ensures the following very interesting properties. (1) Each process which is interested in a topic T_i, must only deal with a memory complexity of $ln(S_{T_i})+c_{T_i}+z_{T_i}$, where S_{T_i} denotes the number of processes which are interested in the topic T_i, without caring about any super- or subtopics. (2) The two constants c_{T_i} and z_{T_i} make it possible for the application to trade, for every topic of the hierarchy, the message complexity of the dissemination with the reliability of this dissemination. (3) The number of messages required for the publication of an event (i.e., message complexity) of topic T_i grows only in $O(S_{T_{max}} \cdot ln(S_{T_{max}}))$, where T_{max} denotes the (super-)topic of T_i with most subscribers. (4) No parasite message is ever received (a process receives only events of topics it is interested in). (5) No central server is relied upon (avoiding single points of failure and bottlenecks). In the extreme case where no topic relationship information is available (or if there is only one topic of interest in the system) our *data-aware multicast* algorithm (*daMulticast* for short) suffers no degradation (in terms of reliability, memory complexity, and message complexity) with respect to a traditional gossip-based membership algorithm (e.g., [1], [6], [10]).

The rest of the paper is organized as follows: Section II discusses related work. Section III describes our model and the assumptions made in this paper. Section IV gives a brief overview of our algorithm. Section V describes our *daMulticast* algorithm. Section VI analysis our algorithm by comparing its behavior with alternative approaches, in terms of message and memory complexity, as well as reliability. Section VII gives some simulation results of our algorithm. Finally Section VIII draws several conclusions. Optionnal Appendix gives the full mathematical analysis presented in Section VI.

II. Related Work

A. The newsgroup propagation algorithm

NNTP (Network News Transfer Protocol, [8]) is the algorithm commonly used for disseminating events in newsgroups. This algorithm takes into account the topics of the events sent in the system to disseminate them to the right set of subscribers. However, in NNTP, each publisher/subscriber must choose a server that will collect its publications/subscriptions. This server ends up being a performance bottleneck and a single point of failure. In our *daMulticast*

[2]Which is anyway available in most publish/subscribe systems we know of.

algorithm, no single process is responsible for collecting subscriptions or publications for multiple different topics, and the propagation of events is done in a completely decentralized manner.

B. Gossip-based algorithms

Various *gossip-based* algorithms, (e.g., [1], [11], [6], [10]) have been proposed in the literature. As pointed in the introduction, these offer good reliability while requiring only a total number of messages in the order of $O(n \cdot ln(n))$ to disseminate an event in a group of n processes. They can thus be efficiently used to propagate events within subgroups representing topics. By not taking into account relationships between topics, they incur, however, large memory complexity overhead. The approach described in [7] exploits *overlaps* between the groups of processes (when processes are parts of several groups) to limit the participation of a process in gossiping events. This does not circumvent the issue of inclusion relations between topics that motivated our approach, but is useful when processes are interested in many distinct topics, and could hence be combined with *daMulticast*. In Section VI we precisely compare *daMulticast* with different variants of gossip-based algorithms using analytical and simulation results.

C. P2P multicast algorithms

Publish/subscribe interaction can also be built on "traditional" P2P unicast algorithms, e.g., Scribe [16] (on Pastry [15]) and HiCAN [14] (on CAN [13]). These algorithms are all based on spanning trees and are sensitive to failures of processes located at the nodes of those spanning trees. Even if those systems provide fault-tolerance mechanisms for replicating the node processes, these are resource-consuming and the respective node processes must have more bandwidth and processing power than "average" processes. Moreover, just like the above-mentioned gossip-based approaches, none of these algorithms considers the hierarchical disposition of topics, leading to bad memory complexity and/or parasite messages.

D. Content-based systems

SIENA [2] and Gryphon [12] are two examples of Internet-scale event notification services based on content-based routing mechanisms. None of them is really comparable with our algorithm as, in our case, we limit the content involved in filtering to a single topic. In addition, unlike in [2], [12], our algorithm does not rely on any network of dedicated application-level routers (so-called "brokers") used to achieve efficient content-based filtering. In [12], process subscriptions are matched to IP multicast groups, which limits the applicability of the dissemination to LANs. Furthermore, the maintenance and the creation of the matching between interests and IP multicast groups must involve all processes and is maintained by a central server (a single point of failure). In [2], the published events are routed from the more general filter to the most specific one, meaning that brokers responsible for a general filter have plenty of filtering to do. PMcast [5] aims at providing content-based publish/subscribe in a completely decentralized manner and thus does not rely on a set of "brokers". Processes are arranged into a hierarchy to (1) reduce the memory complexity of each process and (2) to perform efficient filtering without the help of brokers. However, the processes elected to accomplish the actual filtering receive parasite messages and must be capable of handling a large number of events.

III. MODEL

This section presents some basic elements underlying our algorithm.

A. Topics and processes

A process p_l is said to be *interested* in a topic T_i (e.g., *.dsn04.reviewers*) if p_l either wants to publish an event of topic T_i, or if p_l has subscribed to topic T_i. For presentation simplicity, we assume that a process is interested in one topic T_i in the topic hierarchy only (and as a consequence to all subtopics of T_i). A process p_l communicates with another process p_m via unreliable, i.e., best effort, channels and processes might crash and recover (a process that is not crashed is said to be alive). We denote by Π_{T_i} the group of all processes that are interested in topic T_i. The *root group* is the group of processes interested in the root topic (i.e., "."). By misuse of language, we also denote by T_i the group of processes interested in T_i. The number of processes in a group is denoted by S_{T_i} which represents the cardinality of Π_{T_i}. The direct supertopic of T_i is denoted by $super(T_i)$. For instance, in *.dsn04.reviewers*, *dns04* is the supertopic of *reviewers*. Only the root topic has no supertopic. The depth of a topic hierarchy is equal to t. In this paper, we talk about *inclusions* of topics when a topic T_a is a supertopic (direct or not) of T_b (in this case T_a includes T_b). Finally, we say that p_k ($\in \Pi_{T_j}$) is a *superprocess* of a process p_l ($\in \Pi_{T_i}$) if p_k is interested in a topic T_j that includes T_i (in which p_l is interested).

B. Notations

A published event of a specific topic T_i is denoted by e_{T_i}, and $\Psi^l_{T_i}$ denotes a group of processes interested in topic T_i *known* by a process p_l. By *known* we mean that process l can communicate with each of the processes in the group Ψ_{T_i}. The nearest set of reachable processes from a process p_l is denoted by *neighborhood(p_l)*. The *topic table* for a specific topic T_i of a process p_l, denoted by *Table*$^l_{T_i}$, contains information about processes interested in T_i.[3] The *supertopic table* for a specific topic T_i of a process l, denoted *sTable*$^l_{T_i}$, contains information about processes interested in $super(T_i)$ or, if no process is interested in $super(T_i)$ (i.e., no direct superprocess(es) exist(s)), information about processes interested in the next immediate supertopic of T_i, according to the topic hierarchy level, that induces T_i.

IV. Algorithm Overview

We present here an overview of the main concepts underlying *daMulticast* before describing it in more detail and analyzing its performance in subsequent sections.

A. Topic/group pattern

Consider the example of an event of topic T_b published by a process p_b, and another process p_a subscribing to topic T_a, where T_a is the supertopic of T_b. As sketched previously, there are two straightforward ways according to which an event of topic T_b can be transmitted to p_a: (1) a group is created for the *publishers of a topic* (this is done for each topic and corresponds to the dashed arrows in Figure 1); a subscriber (p_a) of topic T_a becomes a member of the group T_a and member of all the groups of the subtopics of T_a (in this case T_b). When an event of topic T_b is published, this event is only disseminated in the group T_b. (2) A group is created for the *subscribers of a topic* (this is also done for each topic and corresponds to the plain arrows scenario of Figure 1); the subscriber p_a for topic T_a becomes only a member of the group T_a and when an event of topic T_b is published, this event is disseminated in the group T_b and to all the groups of all the supertopics of T_b. The first solution has the disadvantage to overload the subscribers (they become members of many groups). The second solution has the disadvantage to overload the publishers (they must publish in several groups). In our algorithm, we consider an optimized variant of the second pattern to achieve a better load distribution (dotted arrows of Figure 1).

[3]The difference between $\Psi^l_{T_i}$ and *Table*$^l_{T_i}$ resides in the fact that $\Psi^l_{T_i}$ is built upon a weekly consistent overlay network and is used in the bootstrapping technique while *Table*$^l_{T_i}$ is built via the underlying membership algorithm and is used in the dissemination of events.

B. Process grouping and membership

Our algorithm takes into account the hierarchy of the topics to limit the membership information a process must maintain. In *daMulticast*, the processes are all put into groups representing the topics they are interested in. These groups are created and maintained dynamically when the processes join or leave the system. To join a group, a process goes through an initialization phase. This phase is responsible for initializing the topic and the super topic tables for that process. Those tables represent the only membership information a process must maintain for any number of topics it is interested in, if those topics include one another. The topic table of a process p_l contains information about processes interested in the same topic as p_l (let us say T_i). The super topic table contains information about processes interested in the direct super topic of T_i, $super(T_i)$. If no process is interested in $super(T_i)$ (i.e., no direct super processes are available), the super topic table contains information about processes interested in the first topic, according to the topic hierarchy level, that induces T_i. Once the process has joined a group, the underlying membership algorithm takes care of keeping the topic table up to date (see SectionV-A.1). As soon as one process detects that its super topic table is outdated (i.e. the information about the processes is not consistent anymore), it updates its super topic table and disseminates the modifications to the other processes (see SectionV-A.2).

C. Event dissemination

The dissemination of an event is performed as shown in Figure 2. Namely, a process p_1 sends its events to at least one process, p_2, from its super topic table and then p_1 gossips the event to the processes (p_3, p_4) in its group. When a process (p_2, p_3 or p_4) receives the event for the first time, it gossips the event within its group and, with a certain probability, disseminates the event to some process in its super topic table. As long as there is a supertopic with interested processes, the event shifts up to the next supertopic group. When the event reaches the root group, the processes receiving the event only gossip it in their group (as no super group exists).

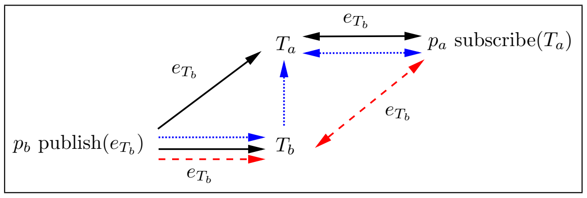

Fig. 1. Alternatives for publishing/receiving an event

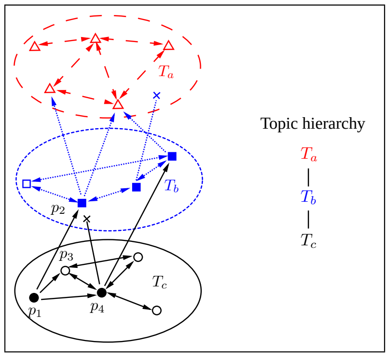

Fig. 2. The dissemination of an event in *daMulticast*

V. The algorithm

We present here the two principal parts of our *daMulticast* algorithm. These are (1) the management of membership information and, (2) the event dissemination scheme.

A. Membership

1) Membership tables: In *daMulticast*, the processes are all arranged into groups representing the topics they are interested in. Thus, every process interested in a topic T_i must maintain information about other processes interested in the topic T_i and the direct supertopic $super(T_i)$.

The identifiers (IDs) of processes interested in T_i are stored in a topic table (*Table*$^l_{T_i}$). As pointed out in Section I, we rely on an underlying gossip-based membership algorithm to populate and maintain the consistency of this table. This underlying algorithm is the “flat” membership algorithm presented in [10] which uses tables of size $(b_{T_i}+1)ln(S_{T_i})$ (b_{T_i} is a constant). The second table (supertopic table, *sTable*$^l_{T_i}$) contains IDs of several processes interested in the supertopic of the topic of interest.[4] This table has a constant size z_{T_i} (see Section V-A.2).

Super Topic Table	
$P_a \in \Pi_{super(T_i)}$	$P_d \in \Pi_{super(T_i)}$
$P_b \in \Pi_{super(T_i)}$	$P_e \in \Pi_{super(T_i)}$

Topic Table	
$P_a \in \Pi_{T_i}$	$P_d \in \Pi_{T_i}$
$P_b \in \Pi_{T_i}$	$P_e \in \Pi_{T_i}$
$P_c \in \Pi_{T_i}$	

Fig. 3. Supertopic and topic table for a process interested in topic T_i.

2) Linking topics and supertopics: If a process in the group T_i receives an event, it is responsible for disseminating this event to other processes of that group. The events are also disseminated to the processes interested in topic $super(T_i)$, because events of topic T_i are also of topic $super(T_i)$. The question here is how to make the link between the group T_i and the group $super(T_i)$. This problem can be separated into two sub-problems: (1) creating links between the different groups (Figure 4) and (2) keeping those links updated (Figure 6). Note that these links should not be the same throughout the entire lifetime of the group, for the sake of reliability and load-balancing.

a) Bootstrapping.: If a process that wants to join the system is provided with contacts belonging to the group $super(T_i)$, then the link is established (lines 5–8, Figure 4). This bootstrap mechanism is unfortunately not always feasible in dynamic systems. The second, and widespread (cf. [15], [10]) possibility is for the process to ask, via an initialization message specifying the topic of interest, other processes, about processes that are interested in $super(T_i)$, and so on recursively until a process interested in $super(T_i)$ is found. This is done via a task (FIND_SUPER_CONTACT, lines 14–28, Figure 4). As soon as a process is found, the supertopic table can be initialized and the FIND_SUPER_CONTACT task can be stopped (lines 31–32, Figure 4). Of course, it may happen that no such process exists. In this case, a new initialization message is sent through the FIND_SUPER_CONTACT task, with two topics of interests: $super(T_i)$ and $super(super(T_i))$. If no process has been found after some timeout period, the scope of the search is enlarged by adding, to the initialization message, the supertopic of the previous topic of interest, and so on until the root topic is contained in the initialization message (lines 19–27, Figure 4). As soon as a process interested in one of the topics specified in the initialization message is found, the supertopic table is initialized with this process. The FIND_SUPER_CONTACT task is stopped only if the process found is interested in $super(T_i)$ (lines 31–33, Figure 4), otherwise, the search continues, but is narrowed to the topic found and its subtopics (line 34, Figure 4). Figure 4 presents the pseudo-code for this part of the algorithm. Note that once a process has an initialized supertopic table, this information is disseminated, using the updates of the underlying membership algorithm, to the other processes of the group. The aim of disseminating the supertopic table, together with the membership algorithm, is to reduce the number of messages during the initialization. This optimization also reduces the number of messages disseminated to the supergroup. When a process receives a message containing a supertopic table, the process merges that information with its own supertopic table

[4]It may happen that the supertopic table does not contain IDs of processes interested in the direct supertopic of the topic of interest. See Section V-A.2 for a complete explanation.

(lines 6–9, Figure 6).[5] The dissemination of supertopic tables is not reported in the pseudo-code as it is part of the updates of the underlying membership algorithm. Note that this bootstrapping technique and algorithm relies here only on a weekly consistent global membership. A consistent overlay network ([15], [13]) would also make it easier to find processes interested in a specific topic.

b) Maintaining supertopic tables.: Each process, with a probability of $p^{sel}_{T_i}$ (see Section V-B for a precise definition of this probability)[6], tries to find out if the processes in its supertopic table are alive (lines 16–23, Figure 6)[7]. If the number of superprocesses that are alive is smaller than a certain threshold τ ($0 \leq \tau \leq z_{T_i}$), then the process asks all alive processes in its supertopic table to provide it with information (identifiers) about $z_{T_i} - \tau$ "new" processes belonging to the supergroup (lines 19–21, Figure 6). This information is then disseminated using the underlying gossip-based algorithm. When an event is published, only one process needs to disseminate this event to some other process of its supergroup; hence the constant size (z_{T_i}) of the supertopic table. It may happen that no direct superprocess is available. In this case, the task used during the initialization (FIND_SUPER_CONTACT) is restarted (lines 12–14, Figure 6).

B. Dissemination

Assuming that the membership has been successfully initialized, a process willing to disseminate an event of topic T_i proceeds as follows: the event is disseminated (1) to the processes of its supertopic table and (2) to the processes of its topic table. The superprocess dissemination (1) can be summarized as follows: with a probability $p^{sel}_{T_i} = \frac{g_{T_i}}{S_{T_i}}$ ($1 \leq g_{T_i} \leq S_{T_i}$, where g_{T_i} represents the number of processes that will try to contact processes that are in the supertopic table of the process, see Section VII), a process decides to take part in the dissemination of the event to the processes in its supertopic table (the process elects itself to do so, line 3 of Figure 7). If a process decides to act as link for a given event, the process sends the event to each of processes present in its supertopic table with probability $p^{a}_{T_i} = \frac{a_{T_i}}{z_{T_i}}$ ($1 \leq a_{T_i} \leq z_{T_i}$, where a_{T_i} determines the number of processes in the supertopic table that will receive the event, see Section VII, lines 4–6, Figure 7). The parameter a_{T_i} can be set according to the average probability of successful transmission. The three parameters g_{T_i}, a_{T_i} and z_{T_i} let the application choose between the overall reliability of the algorithm and the total number of events sent between the groups of processes. The dissemination of the events to the processes in the group of a process (2) can be summarized as follows: the process sends the event to $ln(S_{T_i}) + c_{T_i}$ processes, randomly selected in its topic table (lines 9–14, Figure 7). When receiving a new event for the first time, the processes (of either the supergroup or the group in which the event was initially published) forward the event using the dissemination algorithm (lines 5–10, Figure 5). Figure 2 illustrates the dissemination of an event in *daMulticast*.

VI. Analysis

We discuss the scalability of our algorithm with respect to message complexity and the amount of membership knowledge each process must store (i.e., memory complexity). We then analyze the overall reliability of our algorithm, and finally compare our algorithm with three alternative approaches. For the sake of readability, we do not give here the complete analysis development, the reader interested in details can refer to the Appendix Section.

[5]The MERGE function consists in keeping the "favorite" superprocesses in the table and replacing the failed ones with the fresh ones obtained with the membership messages.

[6]The *sel* in $p^{sel}_{T_i}$ stands for selected.

[7]The CHECK function consists in returning the total number of processes that are alive in the supertopic table. The detection of alive processes is done via timeouts.

SUPER TOPIC TABLE INITIALIZATION ALGORITHM executed by all $p_l \in \Pi_{T_i}$

```
1:  initMsg = $\emptyset$
2:  {Done only the first time the message is received}
3:  {$T_x$ includes $T_i$}
4:  upon RECEIVE(REQCONTACT,$p_l$,initMsg) by $p_m \in \Pi_{T_x}$ from $p_l$ do
5:    if $\Psi^m_{initMsg} \neq \emptyset$ then
6:      SEND(ANSCONTACT,$\Psi^m_{initMsg}$) to $p_l$
7:      RETURN
8:    end if
9:    {We try to find a contact up to a certain timeout}
10:   if initMsg has not expired then
11:     SEND(REQCONTACT,$p_l$,initMsg) to neighborhood($p_m$)
12:   end if
13: end upon

14: {executed each time a timeout occurs, if started}
15: task FIND_SUPER_CONTACT
16:   {A contact is known}
17:   if contact known then
18:     add the contact to sTable$^l_{T_i}$
19:   else
20:     {done at the first time}
21:     if initMsg = $\emptyset$ then
22:       initMsg = $T_i$
23:     else
24:       add $super(initMsg[initMsg.length])$ to initMsg
25:     end if
26:     SEND(REQCONTACT,$p_l$,initMsg) to neighborhood($p_l$)
27:   end if
28: end

29: {$T_x$ includes $T_i$}
30: upon RECEIVE(ANSCONTACT,$\Psi^m_{initMsg}$) by $p_l$ from $p_m \in \Pi_{T_x}$ do
31:   if $T_x$ == $T_i$ then
32:     stop FIND_SUPER_CONTACT
33:   else
34:     remove all $T_j$ in initMsg that include $T_x$
35:   end if
36:   sTable$^l_{T_i}$ = MERGE(sTable$^l_{T_i}$,$\Psi^m_{initMsg}$)
37: end upon
```

Fig. 4. Initialization algorithm used to find processes interested in topic T_x that includes T_i.

SUBSCRIPTION done by $p_l \in \Pi_{T_i}$

```
1: function SUBSCRIBE($T_i$) by $p_l$
2:   Start membership algorithm for $T_i$ if not already done
3:   Start maintain links algorithm for $super(T_i)$ if not already done
4: end
```

RECEPTION done by $p_m \in$ Table$^l_{T_i}$, $p_x \in$ sTable$^l_{T_i}$

```
5:  function RECEIVE($e_{T_i}$) by $p_m$, $p_x$
6:    if $e_{T_i}$ not received then
7:      DISSEMINATE($e_{T_i}$)
8:      deliver $e_{T_i}$ to the application
9:    end if
10: end
```

Fig. 5. Subscription/Reception algorithm.

MAINTAIN LINKS ALGORITHM executed by all $p_l \in \Pi_{T_i}$

```
1:  {$T_x$ induces $T_i$}
2:  upon RECEIVE(NEWPROCESS,$p_l$) by $p_m \in \Pi_{T_x}$ from $p_l$ do
3:    {The superprocess sends a set of available superprocesses to $p_l$}
4:    SEND(NEWPROCESS,$\Psi^m_{T_x}$) to $p_l$
5:  end upon

6:  upon RECEIVE(NEWPROCESS,$\Psi^m_{T_x}$) by $p_l$ from $p_m \in \Pi_{T_x}$ do
7:    {The supertopic table is updated}
8:    sTable$^l_{T_i}$ = MERGE(sTable$^l_{T_i}$,$\Psi^m_{T_x}$)
9:  end upon

10: {executed repeatedly}
11: task KEEP_TABLE_UPDATED
12:   if sTable$^l_{T_i}$ == $\emptyset$ then
13:     start FIND_SUPER_CONTACT
14:   else
15:     {Test for some processes if their supe processes are up}
16:     if RAND() $\geq p^{sel}_{T_i}$ then
17:       {If the total number of processes up is below a certain threshold, we send a message to the superprocesses that are
          up to receive new fresh membership information}
18:       if CHECK(sTable$^l_{T_i}$) $\leq$ $\tau$ then
19:         for all $p_y$ that are up $\in$ sTable$^l_{T_i}$ do
20:           SEND(NEWPROCESS,$p_l$) to $p_y$
21:         end for
22:       end if
23:     end if
24:   end if
25: end
```

Fig. 6. Algorithm used to maintain the link between a group T_i and a group T_x, T_x includes T_i.

DISSEMINATE done by $p_l \in \Pi_{T_i}$

```
1:  function DISSEMINATE($e_{T_i}$) by $p_l$
2:    SUBSCRIBE($T_i$)
3:    if RAND()$\geq p^{sel}_{T_i}$ then
4:      for all $p_y \in$ sTable$^l_{T_i}$ do
5:        with probability $p^a_{T_i}$ SEND($e_{T_i}$) to $p_y$
6:      end for
7:    end if
8:    $\Omega = \emptyset$
9:    for (j=0;j$\leq$log($S_{T_i}$)+$c_{T_i}$;j++) do
10:     {Send the message randomly to processes in our group}
11:     select randomly a process $p_y \in$ Table$^l_{T_i}$$-\Omega$
12:     SEND($e_{T_i}$) to $p_y$
13:     add $p_y$ to $\Omega$
14:   end for
15: end
```

Fig. 7. Dissemination algorithm.

A. Assumptions

We consider a topic T_i (here, $i \in \mathbb{N}^*$) that has a supertopic $super(T_i) = T_{i-1}$, which itself has also a supertopic $super(super(T_i)) = T_{i-2}$, and so on recursively until the root topic T_0. The maximal number of levels in the topic hierarchy is t, and the bottom-most topic is T_t. We assume in the analysis that each group representing a topic contains at least one process.[8]

B. Message complexity

We determine the total number of events sent in the system with our algorithm. First, in group T_i, all processes receive an event that is disseminated, in the ideal case (according to [3], cf. also [10]). Moreover, each process sends $ln(S_{T_i}) + c_{T_i}$ events (to every process in the view of topic T_i). The overall number of events sent in the group T_i is thus upper bounded by $S_{T_i} \cdot (ln(S_{T_i}) + c_{T_i})$. In T_i, several processes additionally disseminate the events to the processes of the supertopic. The number of events sent from one group T_i, to the next supergroup T_{i-1} is: *nbSuperMsg* $= S_{T_i} \cdot p_{T_i}^{sel} \cdot p_{T_i}^{a} \cdot z_{T_i} \cdot p_{T_i}^{succ}$.

This corresponds to the average sum of events sent by the processes of T_i (S_{T_i}), which have decided to act as links ($p_{T_i}^{sel}$), to the processes chosen ($p_{T_i}^{a}$) within those from the supergroup (z_{T_i}) and effectively received ($p_{T_i}^{succc}$)[9]. The total number of events sent from the group T_i all the way up to the group of processes interested in the root topic, is then: $\sum_{i=t}^{0}(S_{T_i} \cdot (ln(S_{T_i}) + c_{T_i})) + \sum_{i=t-1}^{0}(S_{T_i} \cdot p_{T_i}^{sel} \cdot p_{T_i}^{a} \cdot p_{T_i}^{succ} \cdot z_{T_i})$.

There are two sums because the processes interested in the root topic do not need to disseminate events to any higher level. In the worst case (in terms of message complexity), the values for $p_{T_i}^{sel}$, $p_{T_i}^{a}$ and $p_{T_i}^{succ}$ are equal to 1. We also upper bound the equation by z_{max} (where z_{max} represents the maximal value for all z_{T_i}), by $S_{T_{max}}$ (which denotes the number of processes in the biggest group T_{max} corresponding to the topic with the most subscribers) and by c_{max} (where c_{max} denotes the maximal value for all c_{T_i}). As $S_{T_{max}} > 1$, we can upper bound the equation again (by $ln(S_{T_{max}})$): *maxNbMsgSent* $\leq t \cdot S_{T_{max}} \cdot (ln(S_{T_{max}}) + c_{max}) + t \cdot S_{T_{max}} \cdot ln(S_{T_{max}}) \cdot z_{max} \leq t \cdot S_{T_{max}} \cdot ln(S_{T_{max}}) \cdot (1 + c_{max} + z_{max})$. As t can be upper bound by a constant, we have: *maxNbMsgSent* $\in O(S_{T_{max}} \cdot ln(S_{T_{max}}))$. Of course this message complexity holds iff t is constant (otherwise *maxNbMsgSent* $\in O(t \cdot S_{T_{max}} \cdot ln(S_{T_{max}})))$, which is not a limiting hypothesis.

C. Memory complexity

In the pattern we consider, i.e., where topics include one another, each process interested in a topic must maintain two tables. The only exception is for the processes interested in the root topic: these must take care of one table only. The size of the topic table depends logarithmically upon the number of processes interested in the topic. The supertopic table is of size z_{T_i}, which is constant. The number of membership tables depends neither on the number of supertopics of a topic of interest, nor on the number of its subtopics. The memory complexity of every process is: $ln(S_{T_i}) + c_{T_i} \leq$ *totalMbInfo* $\leq ln(S_{T_i}) + c_{T_i} + z_{T_i}$.

D. Reliability

By reliability we mean here the probability that *every* process interested in topic T_i receives a given event published for T_i. According to [3], if all the processes interested in the same topic T_i disseminate an event to $ln(S_{T_i}) + c_{T_i}$ processes (where c_{T_i} is a constant for the group T_i), then

[8]This is required for measuring message complexity and reliability.

[9]This probability depends on the availability of the processes together with the reliability of the links. For the sake of generality, we have decided to make this probability depend on the topic to be able to simulate weekly interconnected groups.

the probability that every process interested in T_i receives the event is $e^{-e^{-c_{T_i}}}$. The worst case is when the events must be disseminated at all levels of the topic hierarchy (i.e., in the t levels). This case occurs when an event is of the bottom-most topic and hence must be disseminated up to the group of processes interested in the root topic. This is the worst case because it sums the sensitive passing between topics and supertopics over the established links. This means that, to receive the event, *all* groups corresponding (recursively) to subtopics of the topic of that event must see their respective members receive the event, and must successfully relay the event to their respective supergroups. Before giving the overall reliability of *daMulticast*, we first introduce the number of processes *susceptible* to send an event from one group T_i to its supergroup: $\mathit{nbSuscProc}_{T_i} = S_{T_i} \cdot p^{sel}_{T_i} \cdot \pi_{T_i}$. We denote by π_{T_i} the proportion of processes that actually receive the event through the underlying gossip algorithm for a group T_i (cf. [4]) and hence are able to propagate the event to $super(T_i)$. The probability that no event is received by a member of $super(T_i)$ can now be calculated based on the number of susceptible processes ($\mathit{nbSuscProc}_{T_i}$): $\mathit{pbNoIntGrpMsg}_{T_i} = (1 - p^{succ}_{T_i})^{\mathit{nbSuscProc}_{T_i} \cdot p^{a}_{T_i} \cdot z_{T_i}}$. We recall here that $p^{succ}_{T_i}$ is the probability that an event sent from one group of processes is received in the supergroup and for the definition of the other values, we refer to Section V. The probability of the propagation of the message to a supergroup is: $\mathit{pit}_{T_i} = 1 - \mathit{pbNoIntGrpMsg}_{T_i}$. In this case, the probability that all processes belonging to a group T_j receive the event is:

$$\mathit{reliability} = \prod_{i=t}^{j} (e^{-e^{-c_{T_i}}} \cdot \mathit{pit}_{T_i}) \tag{1}$$

.

The first term of the reliability equation (i.e., $e^{-e^{-c_{T_i}}}$) comes from the underlying membership algorithm. It determines the reliability of the dissemination of an event of topic T_i in the group T_i and we can tune c_{T_i} to choose between the reliability of the dissemination in the group T_i and the message complexity of this dissemination. The second term of the reliability equation (i.e., pit_{T_i}) comes from the specificity of *daMulticast* (i.e., "data-awareness") as presented earlier. Interestingly, we can tune this parameter (via $p^{sel}_{T_i}$, $p^{a}_{T_i}$ and z_{T_i}) to trade between the reliability of the dissemination between a group T_i and its supergroup, and the number of messages sent between these groups.

E. Comparisons with other algorithms

In this section, we compare *daMulticast* with three a priori relevant alternative approaches we know of: (a) gossip-based broadcast, (b) gossip-based multicast and (c) hierarchical gossip-based broadcast. For fairness, all approaches use the same underlying membership algorithm (i.e., the one of [10]). According to approach (a), each time an event must be sent, it is broadcast in the entire system. This makes use of membership tables of size $ln(n) + c$, as explained in [3]. According to approach (b), the process has one membership table for every topic of interest (this is the approach where a group is created for the publishers of a topic, see Section IV-A). This approach is commonly used in several algorithms ([10], [1], [11]) and these do not take into account the topic inclusion relationships of the events. Approach (c) corresponds to the "hierarchical" technique presented in [10]. The basic idea is to create *small* subgroups (that do not depend on the interests of the processes in each group) and connect these groups to reduce the overall memory complexity. The system is split in two levels. The first level contains groups of processes that exchange events between them (intra group events). The second level is responsible for propagating the events between the groups. We refer the reader to [10] for the membership table size of this algorithm. Our comparisons focus on: (1) the message complexity of the algorithms, (2) the memory complexity of the algorithms and (3) the overall reliability of the algorithms.

1) Message complexity: The message complexity is $O(S_{T_{max}} \cdot ln(S_{T_{max}}))$ for all algorithms except for the gossip-based broadcast which has a message complexity of $O(n \cdot ln(n))$.[10] In other words, enhancing a gossip-based membership algorithm with *daMulticast* does not hamper its overall message complexity performance.

2) Memory complexity:

Gossip-based broadcast (a): An event is disseminated to all the processes in the system. Thus every process has one membership table only, but this table is of size $ln(n) + c$, where n represents the number of all the processes in the system (and $n \gg S_{T_{max}}$).

Gossip-based multicast (b): Every process must maintain a membership table for each topic it is interested in. With a maximum of t levels in a topic hierarchy, and assuming that each subtopic has exactly one supertopic (except the root), a process must deal with at most t tables. As each table is of size $ln(S_{T_i}) + c_{T_i}$, the total memory complexity of each process is: $\sum_{i=t}^{j}(ln(S_{T_i}) + c_{T_i})$.

Hierarchical gossip-based broadcast (c): Each process maintains two membership tables: one for disseminating the events to the processes that are randomly selected to "represent" their group, and a second membership table to disseminate events in the group itself. The first table has a size of $ln(N) + c_2$ and the second table has a size of $ln(m) + c_1$, where N represents the total number of groups (i.e., topics) and m is the number of processes inside a group. So each process has a memory complexity of: $ln(m) + c_1 + ln(N) + c_2$.

As shown in Section VI, the maximal number of membership tables in *daMulticast*, is 2 (and 1 if the process is interested in the root topic). This number does not depend upon the number of topics a process is interested in, when these include one another. If we try to compare our algorithm with the gossip-based broadcast one (i.e., (a)), the number of tables is just majored by one, which can be neglected given the huge gain obtained with *daMulticast* by avoiding any parasite messages. Finally, the memory complexity for a process in group T_i is $ln(S_{T_i})+c_{T_i}+z_{T_i}$ in *daMulticast*. This means that the memory complexity of a process is always smaller in our algorithm than in the other algorithms.

3) Reliability:

Gossip-based broadcast (a): With the memory complexity presented in Section VI-E.2, the reliability is: $e^{-e^{-c}}$.

Gossip-based multicast (b): With the memory complexity presented in Section VI-E.2, the reliability is: $\prod_{i=t}^{j} e^{-e^{-c_{T_i}}}$.

Hierarchical gossip-based broadcast (c): As explained in Section VI-E.2, the reliability is (see [10] for a complete analysis): $e^{-Ne^{-c_1}-e^{-c_2}}$.

As shown in Section VI-D, the reliability of our algorithm is $\prod_{i=t}^{j}(e^{-e^{-c_{T_i}}} \cdot pit_{T_i})$. In comparison with other algorithms, the probability that *all* processes receive an event is smaller with our algorithm, in the general case, especially for the processes interested in the root topic.[11] This comes from the fact that, in *daMulticast*, the reliability depends on the event propagation between groups. However, is it possible to tune this and achieve, in specific cases, the same reliability as other algorithms:[12]

Gossip-based broadcast (a): daMulticast achieves the same reliability as (a) when $0 \leq c \leq -ln(-t \cdot ln(pit))$. Here c denotes the constant used to determine the number of processes to disseminate events to in the gossip-based broadcast algorithm (e.g., $ln(n) + c$), see the Appendix Section. In this case, the memory complexity of our algorithm is smaller than the

[10]See the Appendix Section for a complete analysis.

[11]If we considered the *average* number of processes that receive an event, we would have a much better result (because we would make an average over the reliability of each group instead of a multiplication).

[12]For the sake of simplicity, we consider in the following of this analysis the average case, i.e., where, for every T_i, z_{T_i} is z, S_{T_i} is S_T and pit_{T_i} is pit.

memory complexity of the broadcast algorithm iff: $z \leq ln(n) + ln(1 + t \cdot e^c \cdot ln(pit)) - ln(S_T) - ln(t)$.

Gossip-based multicast (b): daMulticast achieves the same reliability as (b) when $0 \leq c \leq -ln(-ln(pit))$. Here, c denotes the constant used to determine the number of processes to disseminate events to in the gossip-based multicast algorithm (e.g., $ln(S_{T_i})+c_{T_i}$, where all c_{T_i} are the same and equal to c), see the Appendix Section. In this case, the memory complexity of our algorithm is smaller than the memory complexity of the gossip-based multicast algorithm iff: $z \leq (t-1) \cdot (ln(S_T) + c) + ln(1 + e^c \cdot ln(pit))$.

Hierarchical gossip-based broadcast (c): daMulticast achieves the same reliability as (c) when $-ln(\frac{t \cdot (1-ln(pit))}{N+1}) \leq c \leq -ln(\frac{-t \cdot ln(pit)}{N+1})$. Here c denotes the constant used to determine the number of processes to disseminate events to in the hierarchical algorithm, see the Appendix Section. In this case, the memory complexity of our algorithm is smaller than the memory complexity iff: $z \leq c + ln(N) + ln(N + 1 + t \cdot e^c \cdot ln(pit)) - ln(t)$.

VII. Simulation

We present in this section simulation results of *daMulticast*, conveying our claims of reliability and scalability, and confirming the previous analytical evaluation.

A. Setting

The number of levels t in the topic hierarchy is set to 3 (T_0, T_1, T_2 and $super(T_2) = T_1$, $super(T_1) = T_0$, T_0 being the root group). The number of subscribers, S_{T_i}, is 1000 for T_2, 100 for T_1 and 10 for T_0. b_{T_i} (determines the size of the topic table ($(b_{T_i}+1)ln(S_{T_i})$), is set to 3 for all groups. The number of processes any event is disseminated to, c_{T_i} (used in $ln(S_{T_i}) + c_{T_i}$) is equal to 5 for all groups. g_{T_i} (determines $p^{sel}_{T_i}$, the probability that a process elects itself to send events to the supergroup) is set to 5 for all groups. The number of processes, a_{T_i}, chosen in the supertopic table to disseminate the event in the supergroup, is equal to 1 for all groups. The size of the supertopic table, z_{T_i}, is equal to 3 for all groups. The probability for an event to be received is set to an arbitrary value of 0.85, to simulate unreliable, i.e. best effort, channels. The probability for a process to be failed varies. In the simulation, the membership tables (topic table and supertopic table) of a process are determined statically. These tables are initialized at the beginning of the simulation and do not change, during the entire simulation. Pessimistically, we assume that the membership algorithm does not "replace" a failed process, and that these fail at the very beginning (except for results in Figure 11, see below). Note that the events disseminated in the simulation belong to topic T_2. Our simulator written in *C#* simulates synchronous gossip rounds among processes in a Windows task. For doing the simulation we use a Pentium 4, 2.6GHz, 512MBytes of RAM on Windows 2000 SP3.

B. Results

Figure 8 depicts the maximal number of events sent within a group according to the number of processes having failed (here the state of a process (alive/failed) is set at the beginning of the simulation and does not change throughout the simulation; we depict the dynamic case later). The message complexity is of an order of $S_{T_i} \cdot ln(S_{T_i})$ as expected. We can also notice that a good reliability is achieved despite process failures.

Figure 9 depicts the number of events sent from group T_2 to T_1, and from T_1 to T_0 respectively. We can conclude that even if almost half of the processes fail, at least one event is sent to the group of processes interested in the supertopic. This is enough for disseminating the event to the upper groups.

Figure 10 depicts the probability for all processes to receive an event according to the percentage of processes having failed. Not surprisingly, the reception probability depends on the

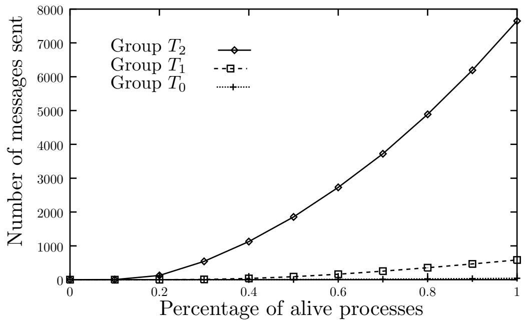

Fig. 8. Number of events sent in each group.

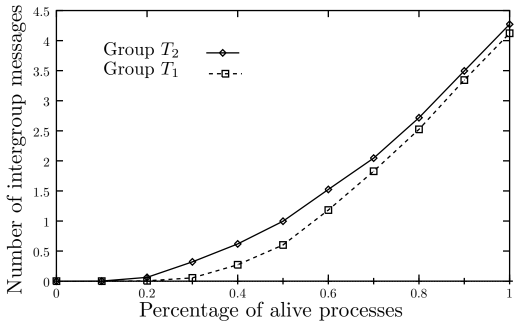

Fig. 9. Number of intergroup events.

overall probability of a process having failed. Of course, the reliability is smaller for processes interested in T_0 as the reception of an event of topic T_2, by the group T_0, depends on the success of the dissemination of this event in the group T_2 and T_1.

Figure 11 depicts the same results as Figure 10, except that now a process can appear to be failed for a process while appearing alive for another one (to simulate a weekly consistent membership algorithm). We achieve a much better reliability for a weakly connected system than in the preceding scenario (Figure 10). To achieve better reliability, we can easily adjust z_{T_i}, $p^a_{T_i}$ and g_{T_i}.

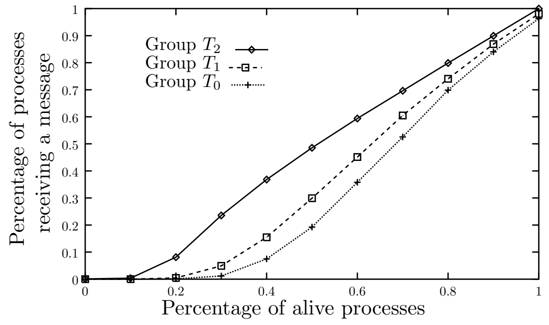

Fig. 10. Reliability (stillborn processes).

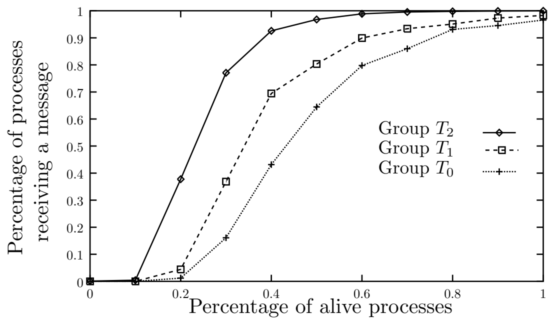

Fig. 11. Reliability (dynamically failed processes).

VIII. Concluding remarks

This paper presents *daMulticast*, an algorithm that adapts the traditional gossip-based membership approach to support hierarchical peer-to-peer topic-based publish/subscribe system. Our algorithm limits the membership information each process must maintain with regard to the topics it has subscribed to, does not introduce any single point of failure, and prevents processes from receiving events they have not subscribed to. In this paper we tackled the case where a topic has only one direct supertopic, mainly for presentation simplicity. Multiple supertopics (i.e., multiple inheritance) could be easily supported by either adapting the membership algorithm or by adding a supertopic table for each supertopic. Neither would hamper the overall performance of the algorithm.

References

[1] 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.

[2] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 219–227, July 2000.

[3] P. Erdös and A. Renyi. On the Evolution of Random Graphs. In *Mat Kutato Int. Közl*, volume 5, pages 17–60, 1960.
[4] P.T. Eugster, R. Guerraoui, A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh. From Epidemics to Distributed Computing. In *IEEE Computer, to appear*, 2004.
[5] P.Th. Eugster and R. Guerraoui. Probabilistic Multicast. In *Proceedings of the 2002 IEEE International Conference on Dependable Systems and Networks (DSN)*, pages 313–322, 2002.
[6] P.Th. Eugster, R. Guerraoui, S. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight Probabilistic Broadcast. In *Proceedings of the ACM Transactions on Computer Systems (TOCS)*, pages 341–374, november 2003.
[7] K. Jenkins, K. Hopkinson, and K. Birman. A Gossip Protocol for Subgroup Multicast. In *International Workshop on Applied Reliable Group Communication (WARGC)*, April 2001.
[8] B. Kantor and P. Lapsley. Network News Transfer Protocol, Request for Comments, 1986.
[9] R. M. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized Rumor Spreading. In *In Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 565–574, 2000.
[10] A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh. Probabilistic Reliable Dissemination in Large-Scale Systems. In *IEEE Transactions on Parallel and Distributed Systems*, volume 14, pages 248–258, March 2003.
[11] M.-J. Lin and K. Marzullo. Directional Gossip: Gossip in a Wide Area Network. In *Proceedings of the 3rd European Dependable Computing Conference (EDCC)*, pages 364–379, September 1999.
[12] 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)*, pages 185–207, April 2000.
[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures (SIGCOMM)*, 2001.
[14] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-Level Multicast Using Content-Addressable Networks. *Lecture Notes in Computer Science*, 2233:14–29, 2001.
[15] A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the 4th IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware)*, pages 329–350, November 2001.
[16] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The Design of a Large-Scale Event Notification Infrastructure. In *Proceedings of the 3rd International Workshop on Networked Group Communication (NGC)*, November 2001.
[17] D. Skeen. *Vitria's Publish-Subscribe Architecture: Publish-Subscribe Overview*. http://www.vitria.com, 1998.
[18] TIBCO. *TIB/Rendezvous White Paper*. http://www.rv.tibco.com/, 1999.

Appendix

We give here the full developments leading to the results presented in Section VI. The complete analysis is given for (1) the message complexity of each algorithm and (2), the size of the super topic table tuned in *daMulticast* to achieve the same reliability as the other algorithms.

1) Message complexity:

a) Gossip-based multicast: For a process interested in the top-most topic, the total membership information:

$$\textit{totalMbInfo} = \sum_{i=t}^{0}(ln(S_{T_i}) + c_{T_i}) \tag{2}$$

If a process wants to publish an event, the total number of disseminated events is:

$$\textit{nbMsgSent} = \sum_{i=t}^{0} S_{T_i}(ln(S_{T_i}) + c_{T_i}) \tag{3}$$

If we upper bound equation (3) by $S_{T_{max}}$ and c_{max} we have then:

$$(3) \leq t \cdot S_{T_{max}} \cdot (ln(S_{T_{max}}) + c_{max}) \tag{4}$$

As $S_{T_{max}} > 1$, c_{max} is a constant and t can be upper bounded by a constant, we have:

$$\textit{maxNbMsgSent} \in O(S_{T_{max}} ln(S_{T_{max}})) \tag{5}$$

b) Gossip-based broadcast: The total membership information each process has is given by:

$$\textit{totalMbInfo} = (ln(n) + c) \tag{6}$$

This means that the total number of events sent is:

$$\textit{nbMsgSent} = n \cdot (ln(n) + c) \tag{7}$$

As $n > 1$ and as c is a constant, we have:

$$\textit{maxNbMsgSent} \in O(nln(n)) \tag{8}$$

c) Hierarchical gossip-based broadcast: The total membership information each process has is given by (where N denotes the number of *small* groups and m the number of processes in each group):

$$\textit{totalMbInfo} = ln(N) + c_1 + ln(m) + c_2 \tag{9}$$

The total number of events sent in the system is:

$$\textit{nbMsgSent} = N \cdot m(ln(N) + ln(m) + c_1 + c_2) \tag{10}$$

If we upper bound equation (10) by $S_{T_{max}}$, we have:

$$\textit{nbMsgSent} \leq NS_{T_{max}}(ln(N) + ln(S_{T_{max}}) + c_1 + c_2) \tag{11}$$

And if $N < S_{T_{max}}$:

$$\textit{nbMsgSent} \leq NS_{T_{max}}(ln(S_{T_{max}}) + ln(S_{T_{max}}) + c_1 + c_2) \tag{12}$$

c_1, c_2 are constants and, we upper bound N by a constant (same assumptions as the one for t) we have:

$$\textit{maxNbMsgSent} \in O(S_{T_{max}} ln(S_{T_{max}})) \tag{13}$$

2) Trading membership with reliability:

a) Gossip-based multicast: For our algorithm to have the same reliability than in the gossip-based multicast algorithm, we must have:

$$\prod_{i=t}^{j} (e^{-e^{-c_{1_{T_i}}}} \cdot \textit{pit}_{T_i}) = \prod_{i=t}^{j} e^{-e^{-c_{2_{T_i}}}} \tag{14}$$

And in the worst case $j = 0$, because our algorithm has the worst performance when a process is interested in the top most topic. This means that (14) $\Leftrightarrow$

$$e^{\sum_{i=t}^{0} -e^{-c_{1_{T_i}}}} \prod_{i=t}^{0} \textit{pit}_{T_i} = e^{\sum_{i=t}^{0} -e^{-c_{2_{T_i}}}} \Leftrightarrow \sum_{i=t}^{0} e^{-c_{1_{T_i}}} - ln(\prod_{i=t}^{0} \textit{pit}_{T_i}) = \sum_{i=t}^{0} e^{-c_{2_{T_i}}} \tag{15}$$

If we assume (for simplification) that all the $c_{1_{T_i}}$ are equal to c_1, all $c_{2_{T_i}}$ are equal to c, and all $\textit{pit}_{T_i}$ are equal to pit, we have then (15) $\Leftrightarrow$

$$e^{-c_1} - ln(pit) = e^{-c} \overset{(①②)}{\Leftrightarrow} ln(e^{-c_1}) = ln(e^{-c} + ln(pit)) \Leftrightarrow c_1 = c - ln(1 + e^{c} ln(pit)) \tag{16}$$

With the following conditions:

- $\textcircled{1} \Leftrightarrow e^{-c} + ln(pit) > 0$, otherwise the equivalence in (16) does not hold. This means that $e^{-c} > -ln(pit) \Leftrightarrow c < -ln(-ln(pit))$. We can have the equivalence here between the two terms because $ln(pit)$ is always less then 0 as pit is always less or equal to 1 (remember that pit is a probability; in $\textcircled{3}$ we show what happens when pit is equal to 1).
- $\textcircled{2}$ c_1 must be greater or equal to 0, this means that $ln(e^{-c} + ln(pit)) \leq 0 \Leftrightarrow e^{-c} \leq 1 - ln(pit) \Leftrightarrow c \geq -ln(1 - ln(pit))$ (we can have the equivalence if $1 - ln(pit) > 0$ which is always the case, because $e > pit$). But $1 - ln(pit) \geq 1$ as $0 \leq pit \leq 1$, which means that $-ln(1 - ln(pit)) \leq 0$. In other terms, this means that the condition $c \geq 0$ encompasses the condition $c \geq -ln(1 - ln(pit))$.
- $\textcircled{3}$ In the case of $pit = 1$ this means that, according to (14) that $c_1 == c$.

From $\textcircled{1}$, $\textcircled{2}$ and $\textcircled{3}$, it is clear we can set c_1 with respect to c if and only if $0 \leq c \leq -ln(-ln(pit))$. If the previous property does not hold, we cannot set c_1 in *daMulticast* to have the same reliability as in the gossip-based multicast algorithm. This means that if c does not satisfy the previous property, no matter the size of the super topic table of our algorithm, it is not possible to achieve the same reliability. However, if c satisfies the property, we can replace c_1 with its value depending on c in the $ln(S_{T_i}) + c_1 + z_{T_i}$ equation (remember that all $c_{1_{T_i}}$ are equal to c_1) and this leads to (to simplify we assume that all $S_{T_i} = S_T$, that all $z_{T_i} = z$, which corresponds to the average case):

$$ln(S_T) + c - ln(1 + e^c ln(pit)) + z \overset{?}{\leq} \sum_{i=t}^{0} (ln(S_T) + c) \tag{17}$$

$\Leftrightarrow$

$$ln(S_T) + c - ln(1 + e^c ln(pit)) + z \overset{?}{\leq} t(ln(S_T) + c) \tag{18}$$

The total membership information of our algorithm is smaller than the total membership information of the gossip-based multicast algorithm iff:

$$z \leq (t-1)(ln(S_T) + c) + ln(1 + e^c ln(pit)) \tag{19}$$

Remember that the upper equation holds only for values of z that are greater than 0 (if z is less or equal to 0, no dissemination can be made to the upper levels and hence there is no point in trying to compare the different algorithms).

b) Gossip-based broadcast: For our algorithm to have the same reliability as in the gossip-based broadcast algorithm, we must have:

$$\prod_{i=t}^{j} \left(e^{-e^{-c_{1_{T_i}}}} \cdot pit_{T_i}\right) = e^{-e^{-c}} \tag{20}$$

In the worst case, $j = 0$, because our algorithm has the worst performance when a process is interested in the top most topic. From (20) it follows that:

$$e^{\sum_{i=t}^{0} -e^{-c_{1_{T_i}}}} \prod_{i=t}^{0} pit_{T_i} = e^{-e^{-c}} \Leftrightarrow \sum_{i=t}^{0} e^{-c_{1_{T_i}}} - ln\left(\prod_{i=t}^{0} pit_{T_i}\right) = e^{-c} \tag{21}$$

If we assume (for simplification purpose) that all $c_{1_{T_i}}$ are equal to c_1, and all pit_{T_i} are equals to pit, we have then (21) $\Leftrightarrow$

$$e^{-c_1}-ln(pit) = \frac{e^{-c}}{t} \overset{(\textcircled{1})}{\Leftrightarrow} ln(e^{-c_1}) = ln(\frac{e^{-c}+tln(pit)}{t}) \overset{(\textcircled{2})}{\Leftrightarrow} c_1 = ln(\frac{1}{e^{-c}+tln(pit)})+ln(t) \tag{22}$$

$\Leftrightarrow$

$$c_1 = c - ln(1+te^{c}ln(pit)) + ln(t) \tag{23}$$

With the following conditions:

- $\textcircled{1} \Leftrightarrow \frac{e^{-c}+tln(pit)}{t} > 0$ and as $t \geq 1$, this implies that $e^{-c}+tln(pit) > 0 \Leftrightarrow c < -ln(-tln(pit))$.
- $\textcircled{2}$ We want c_1 to be greater or equal to 0, this means that $ln(\frac{e^{-c}+tln(pit)}{t}) \leq 0 \Leftrightarrow e^{-c} + tln(pit) \leq t \Leftrightarrow^{\textcircled{3}} c \geq -ln(t(1-ln(pit)))$. As $1-ln(pit) > 0$ and as $t \geq 1$ (see $\textcircled{3}$), condition $\textcircled{2}$ implies $-ln(t(1-ln(pit))) \leq 0$. This means that $c \geq 0$ encompasses $c \geq -ln(t(1-ln(pit)))$.
- $\textcircled{3}$ This equation holds only if $t(1-ln(pit)) > 0$ which is always the case, as $t \geq 1$ and as $0 \leq pit \leq 1$.

From $\textcircled{1}$, $\textcircled{2}$ and $\textcircled{3}$ we can setup c_1 with respect to c if and only if $0 \leq c \leq -ln(-tln(pit))$. If the previous property does not hold, we cannot set c_1 in *daMulticast* to have the same reliability of the gossip-based broadcast algorithm. This means that if c does not satisfies the previous property, no matter what the size of the super topic table is in *daMulticast*, it is not possible to achieve the same reliability. However, if c satisfies the property, we can replace c_1 with its value depending in c in the $ln(S_{T_i}) + c_1 + s_i$ equation (remember that all $c_{1_{T_i}}$ are equal to c_1) and this leads to (to simplify we assume that all $S_{T_i} = S_T$, that all $z_{T_i} = z$, which corresponds to the average case):

$$ln(S_T) + c - ln(1+te^{c}ln(pit)) + ln(t) + z \overset{?}{\leq} ln(n) + c \tag{24}$$

The total membership information of our algorithm is smaller then the total membership information of the gossip-based broadcast algorithm iff:

$$z \leq ln(n) + ln(1+te^{c}ln(pit)) - ln(S_T) - ln(t) \tag{25}$$

So in order to have a gain, $ln(1+te^{c}ln(pit))$ must tend to its maximum (which is 0) and $ln(n) > ln(S_T) + ln(t)$.

c) Hierarchical gossip-based broadcast: For our algorithm to have the same reliability than in hierarchical gossip-based broadcast algorithm, we must have:

$$\prod_{i=t}^{j}(e^{-e^{-c_{1_{T_i}}}} \cdot pit_{T_i}) = e^{-Ne^{-c_1}-e^{-c_2}} \tag{26}$$

In the worst case, $j = 0$, because our algorithm has the worst performance when a process is interested in the top most topic. Moreover if we assume (again for simplicity purpose) that all $c_{1_{T_i}}$ are the same (equal to c_T), all pit_{T_i} are the same and equal to pit and that $c_1 = c_2 = c$ we have:

$$e^{\sum_{i=t}^{0} -e^{-c_T}} pit^t = e^{-(N+1)e^{-c}} \Leftrightarrow te^{-c_T} - tln(pit) = (N+1)e^{-c} \Leftrightarrow e^{-c_T} = \frac{(N+1)e^{-c}+tln(pit)}{t} \overset{(\textcircled{1})}{\Leftrightarrow} \tag{27}$$

$$c_T = -ln(\frac{(N+1)e^{-c} + tln(pit)}{t}) \overset{(\textcircled{2})}{\Leftrightarrow} c_T = ln(t) + c - ln(te^c ln(pit) + N + 1) \tag{28}$$

With the following conditions:

- $\textcircled{1} \Leftrightarrow \frac{(N+1)e^{-c}+tln(pit)}{t} > 0$ and as $t \geq 1$, this implies that $(N+1)e^{-c} + tln(pit) > 0 \Leftrightarrow c < -ln(\frac{-tln(pit)}{N+1})$.
- $\textcircled{2}$ We want c_T to be greater or equal to 0, this means that $ln(\frac{(N+1)e^{-c}+tln(pit)}{t}) \leq 0 \Leftrightarrow (N+1)e^{-c} + tln(pit) \leq t \Leftrightarrow e^{-c} \leq \frac{t(1-ln(pit))}{N+1} \Leftrightarrow c \geq -ln(\frac{t(1-ln(pit))}{N+1})$ (as t and N are greater then 0 and as $0 \leq pit \leq 1$).

From $\textcircled{1}$ and $\textcircled{2}$, we can setup c_T with respect to c if and only if $-ln(\frac{t(1-ln(pit))}{N+1}) \leq c \leq -ln(\frac{-tln(pit)}{N+1})$. If the previous property does not hold, we cannot set c_T in *daMulticast* to have the same reliability than in the hierarchical gossip-based broadcast algorithm. However, if c_T satisfies the property, we can replace c_T with its value depending in c in the $ln(S_{T_i}) + c_T + z_{T_i}$ equation (remember that all $c_{1_{T_i}}$ are equal to c_T) and this leads to (to simplify we assume that all $S_{T_i} = S_T$, that all $z_{T_i} = z$, which corresponds to the average case):

$$-ln(N + 1 + te^c ln(pit)) + ln(t) + z \overset{?}{\leq} ln(N) + c \tag{29}$$

In this case, the total membership information of our algorithm is smaller then the total membership information of the hierarchical gossip-based broadcast algorithm iff:

$$z \leq c + ln(N) + ln(N + 1 + te^c ln(pit)) - ln(t) \tag{30}$$