Data-Aware Multicast*

Sébastien Baehni Patrick Th. Eugster Rachid Guerraoui
Distributed Programming Laboratory, EPFL
{Sebastien.Baehni, Patrick.Eugster, Rachid.Guerraoui}@epfl.ch

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 and preserves processes from receiving messages related to topics they have not subscribed to. It also provides the application with 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.

1. Introduction

Many distributed applications are best supported by publish/subscribe infrastructures 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 [24] and Vitria [23], as well as more recent ones, e.g., TPS [7] and JORAM [16], organize these events in a hierarchical manner. Ideally, we expect from a topic-based publish/subscribe infrastructure 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., 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 ([13, 2]) 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 [8], while ensuring good overall reliability. Nevertheless, 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 topic, is an appealing alternative, but it increases memory complexity. Indeed, 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. As a consequence, this subscriber has also to be informed about the creation of any new subtopics of T_i. With (2), a publisher of a topic T_i has to publish its events not only within the group representing T_i, but within every group representing a supertopic of T_i. This 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 [6], might look like a viable alternative at first glance. However, this approach makes it impossible to reduce memory complexity without introducing parasite messages, i.e., without forcing processes to receive and participate in the dissemination of events beyond their own interests.

We present in this paper a decentralized multicast algo-

* 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 as well as by IST Pepito (OFES No 01.0227).

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.

rithm 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 membership technique, ensures the following properties. (1) The complexity memory of each process interested in a topic T_i is in the order of $ln(S_{T_i}) + c_{T_i} + z_{T_i}$ (here S_{T_i} denotes the number of processes which are interested in the topic T_i and c_{T_i}, z_{T_i} denote constant values for each T_i and are explained in more details later); (2) The application can 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 the order of $O(S_{T_{max}} \cdot ln(S_{T_{max}}))$, where T_{max} denotes the (super-)topic of T_i with most subscribers; (4) No parasite message is ever received; (5) No central server is relied upon. 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) falls back into a traditional membership algorithm (e.g., [2, 8, 10]) with no degradation (in terms of reliability, memory complexity, message complexity and latency complexity).

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 describes our model. Section 4 describes *daMulticast*. Section 5 analysis our algorithm by comparing its performance with alternative approaches. Section 6 gives some simulation results, and Section 7 some concluding remarks. For space limitations, the complete description of *daMulticast* and exhaustive evaluation is given in [1].

2. Related Work

2.1. The newsgroup propagation algorithm

NNTP (Network News Transfer Protocol, [12]) 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 comparison, *daMulticast* is completely decentralized.

2 Which is anyway available in most publish/subscribe systems we know of [7, 16, 24, 23].

2.2. Gossip-based algorithms

Various *gossip-based* algorithms, (e.g., [2, 15, 8, 10]) have been proposed in the literature. As pointed in the introduction, these offer good reliability while requiring a total number of messages that is only 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 [11] 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*.

2.3. P2P multicast algorithms

Publish/subscribe interaction can also be built on "traditional" P2P unicast algorithms, e.g., Scribe [22] (on Pastry [21]) and HiCAN [20] (on CAN [19]). These algorithms are all based on spanning trees and are sensitive to failures of processes located at the nodes of those trees. Even if fault-tolerance mechanisms can be used, these are resource-consuming and the node processes must have more bandwidth and processing power than regular processes. Moreover, just like the above-mentioned gossip-based approaches, none of these algorithms considers the hierarchical disposition of topics, leading to high memory complexity and/or parasite messages.

2.4. Content-based systems

SIENA [4] and Gryphon [17] are two examples of Internet-scale event notification services based on content-based routing mechanisms. None of them is comparable with our algorithm. In our case, we limit the content involved in filtering to a single topic. In addition, unlike in [4, 17], our algorithm does not rely on any network of dedicated application-level routers ("brokers") used to achieve efficient content-based filtering. In [17], process subscriptions are matched to IP multicast groups and therefore 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 [4], the published events are routed from the more

general filter to the most specific one: brokers responsible for a general filter are heavily loaded. In Hermes [18], the propagation of events is also done with the help of brokers and follows a top-down approach: a parent type must keep a reference to all its descendants. This can become cumbersome if the number of descendants changes continuously (issue not raised with *daMulticast*). In PMcast [6], the 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. Moreover, while PMcast addresses the problem of routing and filtering of messages based on their *individual* content, *daMulticast* deals with routing of messages associated explicitly with *global* topics.

3. System Model

3.1. Topics, processes and notations

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., "."). Overloading our symbols, 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*, *dsn04* 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).

A published event of a specific topic T_i is denoted by e_{T_i}. 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. The *supertopic table* for a specific topic T_i of a process p_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 includes T_i.

3.2. Topic/group pattern

Consider 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. 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 overloads the subscribers, whereas the second overloads the publishers (they must publish in several groups). *DaMulticast* provides an optimized variant of the second pattern to achieve a better load distribution, for both the publishers and the subscribers (dotted arrows of Figure 1).

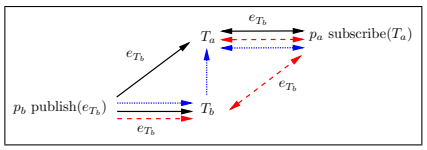

Figure 1. Publication/subscription alternatives.

3.3. Gossiping and membership protocols

DaMulticast relies on the gossiping technique of [14]. Basically, with this technique, each process gossips an event to $ln(S_{T_i}) + c_{T_i}$ target processes and the probability that a process receives the event goes to $e^{-e^{-c_{T_i}}}$ as S_{T_i} goes to infinity. Thus, any membership protocol that maintains, at each process, a membership table of minimal size $ln(S_{T_i}) + c_{T_i}$ (e.g., [14]), can be enhanced with *daMulticast*, to support topic hierarchies. Throughout the paper, we will assume such an underlying membership protocol.

4. The algorithm

In the following, we present our *daMulticast* algorithm. For space limitation we do not provide here the code of the algorithm. The interested reader can refer to the full paper [1].

4.1. Overview

4.1.1. Specification. In short, *daMulticast* is a probabilistic multicast algorithm. In this sense, it ensures the following properties: (1) Validity: if a correct process p_l ($\in \Pi_{T_i}$) gossips an event e_{T_i}, then some correct process p_k ($\in \Pi_{T_j}$, where T_j includes T_i, or $T_j == T_i$) eventually delivers e_{T_i}; (2) Probabilistic Integrity: for any event e_{T_i}, there is a high probability such that every correct process p_k ($\in \Pi_{T_j}$, where T_j includes T_i, or $T_j == T_i$) delivers e_{T_i}, at most once, and only if e_{T_i} was previously multicast by p_l ($\in \Pi_{T_i}$). (3) Probabilistic Agreement: If a correct process p_l ($\in \Pi_{T_i}$) delivers e_{T_i}, then there is a high probability such that every correct process p_k ($\in \Pi_{T_j}$, where T_j includes T_i, or $T_j == T_i$) eventually delivers e_{T_i}.

4.1.2. Process grouping and membership. We take into account the hierarchy of the topics to limit the membership information a process maintains. The processes are split 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 to initialize the topic and the supertopic tables (that compose the membership table) for that process (see Section 4.2.1). Once the process has joined a group, the underlying membership algorithm takes care of maintaining consistent topic tables and a pro-active algorithm is used to keep the supertopic table updated (see Section 4.2.2). Processes can dynamically join and leave the groups.

4.1.3. Event dissemination. The dissemination of an event is depicted in Figure 2. Namely, a process p_1 sends its events to at least one process, p_2, from its supertopic 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 supertopic 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. Note that it is not mandatory for a publisher (p_1) to ensure the propagation of the events into its supergroup. If it fails to do so, another process (here p_4) from the group does the job for it. This implies that, once a publisher (p_1) has transmitted its event to at least another process in its group, it can leave this group.

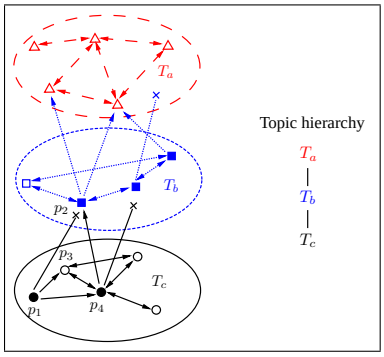

Figure 2. Dissemination in *daMulticast*.

4.2. Membership

4.2.1. Membership tables. In *daMulticast*, every process interested in a topic T_i maintains 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 ($Table^l_{T_i}$) and is maintained by the underlying membership algorithm. The second table (supertopic table, $sTable^l_{T_i}$) contains IDs of several processes interested in the supertopic of the topic of interest.[3] This table has a constant size z_{T_i}.

4.2.2. Linking topics and supertopics. If a process in 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 $super(T_i)$.[4] This problem can be separated into two sub-problems: (1) creating links between the different groups, in initializing the supertopic tables (Figure 3) and (2) maintaining the information of the supertopic tables consistent.

Taking care of only its direct supertopic is very appealing because in this case, new subtopics can be added dynamically into the system in a completely transparent manner for the superprocesses (the superprocesses do not have to maintain any membership information about subprocesses to receive events from them). Moreover, the processes have to

3 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 4.2.2 for a complete explanation.

4 Two groups (T_i and $super(T_i)$) are said to be linked if there exists at least one process in T_i that can send a message to a member in group $super(T_i)$.

take care of only two membership tables, irrespective of the total number of topics in the same topic hierarchy and allows dynamic changes of the topic hierarchy. If this hierarchy contains only one topic, *daMulticast* does not use the initialization (1) and maintenance (2) algorithms and hence simply falls back into the underlying membership algorithm with no degradation.

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 directly established. This bootstrap mechanism is unfortunately not always feasible in dynamic systems. The second 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 (the initialization message can contain a time to live indication (TTL) to not flood the network). As soon as a process is found, the supertopic table can be initialized.

Of course, it may happen that no such process exists. In this case, the fact that no process is interested in $super(T_i)$ does not imply that no process is interested in $super(super(T_i))$ or any of its supertopics.[5] A new initialization message is sent and it specifies two topics of interests: $super(T_i)$ and $super(super(T_i))$. Again, if no process interested in either $super(T_i)$ or $super(super(T_i))$ has been found, after the timeout of the message, 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.

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.[6] However, it may happen that this process is not interested in $super(T_i)$ but instead in a supertopic of $super(T_i)$. In this case the process interested in T_i keeps searching for processes interested in topic $super(T_i)$. Figure 3 presents the bootstrapping protocol.

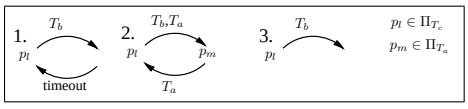

Figure 3. Bootstrapping.

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. When a process receives a message containing a supertopic table, it merges that information with its own supertopic table. This bootstrapping technique relies here only on a weakly consistent global membership. A consistent overlay network ([21, 19]) would also make it easier to find processes interested in a specific topic.

Maintaining supertopic tables. Once the supertopic table of a process p_l has been initialized, it has to be maintained. Indeed, it may happen that processes, whose IDs are stored in the supertopic table of p_l, crash or leave the group they have joined. In this case the supertopic table of process p_l is out-dated and it is not possible anymore for p_l to propagate events to its supergroup. For that purpose, each process, with a probability $p_{T_i}^{sel}$ (see Section 4.3 for a precise definition of this probability)[7], tries to find out if the processes in its supertopic table are alive[8]. 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. This information is then disseminated using the underlying membership algorithm. A pro-active protocol is used to avoid restarting the bootstrapping protocol if we detect that no superprocess is available when an event is published. The use of a reactive protocol would have implied a bigger dependency between the propagation of an event and the availability of the process responsible for that propagation (as the bootstrapping protocol can take some time). For the sake of reliability and load-balancing, it is also possible to replace superprocesses in the supertopic table even if those are available, to balance the propagation among all processes.

4.3. 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 $p_{T_i}^{sel} = \frac{g_{T_i}}{S_{T_i}}$ ($1 \leq g_{T_i} \leq S_{T_i}$, where g_{T_i} represents the number of processes that try to contact processes that are in the supertopic table of the process), a process decides to take part in the dissemination of the event to its supergroup (the process elects itself to do so). If a process decides to act as link for a given event, the process sends the event to each of the processes of its supertopic table with probability $p_{T_i}^{a} = \frac{a_{T_i}}{z_{T_i}}$ ($1 \leq a_{T_i} \leq z_{T_i}$, where a_{T_i} determines the number of pro-

5 In some sense, there is a lack of interested processes at a specific topic in the topic hierarchy.

6 In other words, the process keeps on sending initialization messages until it founds some other processes.

7 The *sel* in $p_{T_i}^{sel}$ stands for selected.

8 The detection of alive processes is done via timeouts.

cesses in the supertopic table that receive the event). The parameter a_{T_i} can be set according to the average probability of successful transmission. The dissemination of events within a group (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. When receiving a new event for the first time, every process (of either the supergroup or the group in which the event was initially published) forward once the event using the dissemination algorithm. This dissemination scheme (also called "infect and die") let the publisher crash or leave the group as soon as it has published its events and consequently does not impose any restriction on the availability of such processes.

5. Analysis

We discuss here the scalability of our algorithm with respect to message complexity, memory complexity, reliability and latency complexity. We compare our algorithm with three alternative approaches.

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-1}. We assume in the analysis that each group representing a topic contains at least one process.[9]

5.1. 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 [5], cf. also [14]). Moreover, each process sends $ln(S_{T_i}) + c_{T_i}$ events thus the overall number of events sent in the group T_i is 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 interested in the supertopic. The number of events sent from one group T_i to the next supergroup T_{i-1} is: $nbSuperMsg_{T_i} = S_{T_i} \cdot p_{T_i}^{sel} \cdot p_{T_i}^{a} \cdot z_{T_i} \cdot p_{T_i}^{succ}$.

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 ($p_{T_i}^{succ}$)[10]. 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: $\sum_{i=t-1}^{0}(S_{T_i} \cdot (ln(S_{T_i}) + c_{T_i})) + \sum_{i=t-2}^{0}(S_{T_i} \cdot p_{T_i}^{sel} \cdot p_{T_i}^{a} \cdot p_{T_i}^{succ} \cdot z_{T_i})$.[11] 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 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 (according to [25]). Note that we consider here the message complexity and not the actual value of the total number of messages sent: this value depends on t.

5.2. Memory complexity

In the pattern we consider, topics include one another and each process interested in a topic maintains two tables. The only exception is for the processes interested in the root topic: these care about 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: $ln(S_{T_i}) + c_{T_i} \leq totalMbInfo \leq ln(S_{T_i}) + c_{T_i} + z_{T_i}$.

5.3. 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 [5], if all the processes interested in the same topic T_i disseminate an event to $ln(S_{T_i}) + c_{T_i}$ processes, then 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 are disseminated at all levels of the topic hierarchy (i.e., in the t levels). This occurs when an event is of the bottom-most topic and has to be disseminated up to the group of processes interested in the root topic. This is the worst case because it sums the passing between topics and supertopics over the established links. Before measuring the reliability of *daMulticast*, we first compute the number of processes *susceptible* to send an event from one group T_i to its supergroup: $nbSuscProc_{T_i} = S_{T_i} \cdot p_{T_i}^{sel} \cdot \pi_{T_i}$. We denote by π_{T_i} the proportion of processes that actually receive

9 This is required for measuring message complexity, reliability and latency complexity.

10 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 simulate weakly interconnected groups.

11 There are two sums because the processes interested in the root topic do not need to disseminate events to any higher level.

the event through the underlying membership algorithm for a group T_i (cf. [9]) 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 ($nbSuscProc_{T_i}$): $pbNoIntGrpMsg_{T_i} = (1 - p_{T_i}^{succ})^{nbSuscProc_{T_i} \cdot p_{T_i}^{a} \cdot z_{T_i}}$. We recall here that $p_{T_i}^{succ}$ 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 4. The probability of the propagation of the message to a supergroup is: $pit_{T_i} = 1 - pbNoIntGrpMsg_{T_i}$. In this case, the probability that all processes belonging to a group T_j receive the event is: $reliability = \prod_{i=t-1}^{j}(e^{-e^{-c_{T_i}}} \cdot pit_{T_i})$. The first term of the reliability equation (i.e., $e^{-e^{-c_{T_i}}}$) comes from the gossiping technique we use (i.e., [14]). 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 trade the reliability of the dissemination in the group T_i and the total number of messages sent in the topic group 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"). We can also tune this parameter (via $p_{T_i}^{sel}$, $p_{T_i}^{a}$ and z_{T_i}) dynamically to trade the number of messages sent between a group T_i and its supergroup. This tunability might turn out to be important in dynamic systems where the number of processes are constantly changing. For example, if the number of processes is growing in a group, we can reduce pit_{T_i} to reduce the total number of intergroup messages sent but without hampering the reliability (as there are a lot of processes). If the number of processes in a group becomes very small, we enforce all the processes to propagate the events to their supergroup.

5.4. Latency

By latency we mean here the number of rounds needed by our algorithm to infect the entire system. To measure the latency of a specific topic, we assume that the event is propagated from one subtopic to its supertopic with probability 1 (pit_{T_i}).[12] According to [3], the number of rounds needed to infect a group of S_{T_i} processes is: $R_{S_{T_i}} = \frac{ln(S_{T_i})}{ln(ln(S_{T_i}))} + O(1)$. Applying this equation to our algorithm, we compute three cases: (1) the best case, (2) the average case and (3) the worst case.

In (1), the event is directly propagated from one group to a supergroup. In this case, to propagate an event from group T_j to group T_i ($i \leq j$): $latency_{min} = (j - i) + R_{S_{T_i}} \leq t - 1 + R_{S_{T_{max}}}$ and hence, $latency_{min} \in O(t + R_{S_{T_{max}}})$ and if t is a constant, $latency_{min} \in O(R_{S_{T_{max}}})$.

12 It does not make sense to calculate a latency of an event if this event is never received by a process.

In (3), the event is propagated entirely to one group before being sent to its supergroup. This means that: $latency_{max} = \sum_{k=j}^{i} R_{S_{T_k}} + (j - i) \leq t \cdot R_{S_{T_{max}}} + (t - 1) \leq t \cdot (R_{S_{T_{max}}} + 1)$ and hence, $latency_{max} \in O(t \cdot (R_{S_{T_{max}}} + 1))$. If t is a constant: $latency_{max} \in O(R_{S_{T_{max}}})$.

Finally, we compute the average latency (2) in which we assume that, after $\frac{R_{S_{T_i}}}{2}$ rounds, the event is propagated to the supergroup. In this case: $latency_{avg} = \sum_{k=j}^{i} \frac{R_{S_{T_k}}}{2} + (j - i) \leq t \cdot (\frac{R_{S_{T_{max}}}}{2} + 1)$ and hence, $latency_{avg} \in O(t \cdot (\frac{R_{S_{T_{max}}}}{2} + 1))$. If t is constant: $latency_{avg} \in O(R_{S_{T_{max}}})$.

These results do not depend on t if t can be upper bounded by a constant. However, it is clear that the value of the latency depends on t.

5.5. Comparisons with other algorithms

We compare *daMulticast* with three alternative approaches: (a) gossip-based broadcast, (b) gossip-based multicast and (c) hierarchical gossip-based broadcast. For the sake of fairness, all approaches use the same gossiping technique (i.e., that of [14]). According to (a), each time an event is sent, it is broadcast in the entire system. This uses membership tables of size $ln(n) + c$, as explained in [5]. 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 3.2). This approach is commonly used in several algorithms ([10, 2, 15]), which do not take into account the topic inclusion relationships of the events. Approach (c) corresponds to the "hierarchical" technique presented in [14]. 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 into 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. Our comparison focus on: (1) message complexity, (2) memory complexity, (3) reliability and (4) latency complexity.

5.5.1. 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 $O(n \cdot ln(n))$. In other words, enhancing a membership algorithm with *daMulticast* does not hamper its overall message complexity performance.

5.5.2. 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 processes in the system (and $n \gg S_{T_{max}}$).

Gossip-based multicast (b): Every process maintains 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 deals 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: $\sum_{i=t-1}^{j}(ln(S_{T_i}) + c_{T_i})$.

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 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 5, the maximal number of membership tables in *daMulticast* is 2 (1 if the process is interested in the root topic). This number does not depend on 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 (see Section 6). 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.

5.5.3. Reliability.

Gossip-based broadcast (a): With the memory complexity presented in Section 5.5.2, the reliability is: $e^{-e^{-c}}$.

Gossip-based multicast (b): With the memory complexity presented in Section 5.5.2, the reliability is: $\prod_{i=t-1}^{j} e^{-e^{-c_{T_i}}}$.

Hierarchical gossip-based broadcast (c): As explained in Section 5.5.2, the reliability is (see [14] for a complete analysis): $e^{-Ne^{-c_1}-e^{-c_2}}$.

As shown in Section 5.3, the reliability of *daMulticast* is $\prod_{i=t-1}^{j}(e^{-e^{-c_{T_i}}} \cdot pit_{T_i})$. In comparison with other algorithms, the probability that *all* processes receive an event is smaller with *daMulticast*, in the general case, especially for the processes interested in the root topic.[13] This comes from the fact that, in *daMulticast*, the reliability depends on the event propagation between groups. However, it is possible to tune this and achieve, in specific cases, the same reliability as other algorithms:[14]

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 [1]. In this case, the memory complexity of *daMulticast* is smaller 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 [1]. In this case, the memory complexity of *daMulticast* is smaller 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 number of processes that disseminate events in the hierarchical algorithm, see [14, 1]. In this case, the memory complexity of *daMulticast* is smaller iff: $z \leq c + ln(N) + ln(N + 1 + t \cdot e^c \cdot ln(pit)) - ln(t)$.

Achieving the same reliability than the other algorithms is only possible if we tune the parameters of pit_{T_i}, namely $p_{T_i}^{sel}$, $p_{T_i}^{a}$ and z_{T_i}. Of course, increasing those values impact the total number of intergroup messages sent as well as the total memory complexity.

5.5.4. Latency complexity.

The latency complexity is $O(R_{S_{T_{max}}})$ for all algorithms except for the gossip-based broadcast which has a latency complexity of $O(R_n)$. This means that *daMulticast* is equivalent, in terms of latency complexity, to all the other algorithms.

6. Simulation

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

6.1. 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

13 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).

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

for T_2, 100 for T_1 and 10 for T_0. The number of processes any event is disseminated to, c_{T_i}, is equal to 5 for all groups and g_{T_i} (which determines $p_{T_i}^{sel}$) is set to 5 for all groups. The number a_{T_i} (which determines $p_{T_i}^{a}$) is 1 for all groups. The size of the supertopic table, z_{T_i}, is 3 for all groups. The probability for an event to be received is set to an arbitrary value of 0.85, to simulate unreliable, e.g. best effort, channels. The probability for a process to crash 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 crashed process, and that these crash at the very beginning (except for results in Figure 5, see below).[15] 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. We use a Pentium 4, 2.6GHz, 512MBytes of RAM on Windows 2000 SP3.

6.2. Results

Figure 4 depicts the probability for all processes to receive an event according to the percentage of processes having crashed.[16] Not surprisingly, the reception probability depends on the overall probability of a process having crashed. 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.

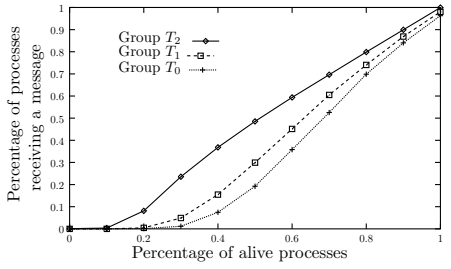

Figure 4. Reliability (stillborn processes).

Figure 5 depicts the same results as Figure 4, except that now a process can appear to be crashed for a process while appearing alive for another one (to simulate a weakly consistent membership algorithm). We achieve a much better reliability for a weakly connected system than in the preceding scenario (Figure 4). To achieve better reliability, we can easily adjust z_{T_i}, $p_{T_i}^{a}$ and g_{T_i}.

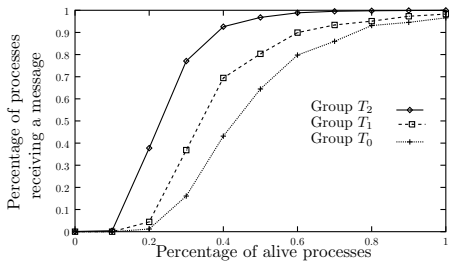

Figure 5. Reliability (dynamically crashed processes).

We consider in Table 1 the average number of rounds needed to disseminate an event from topic T_j to topic T_0. Three different topologies are considered: (a) $T_2 = 1000$, $T_1 = 100$ and $T_0 = 10$, (b) $T_j..T_0 = 100, j = 1..2$ and (c) $T_j..T_0 = 100, j = 1..4$. We compare (1) our approach with (2) a general gossip-based protocol [5] and with (3) PMcast [6] (for (2) and (3), the values have been calculated using the analytical equations and are not taken from simulations). The results confirm the analytical results given in Section 5.4 and convey the impact of the number of hierarchies on the latency.

	T_j,T_{j-1},...,T_0		
	1000, 100, 10	100 ($j = 2$)	100 ($j = 4$)
(1)	8.91	8.83	13.08
(2)	4.63	4.39	4.31
(3)	6.61	5.44	5.89

Table 1. Average number of rounds.

In Table 2, we consider the average total number of parasite messages sent in the system. To calculate the maximal number of parasite messages, we assume that a publisher publishes an event of the root topic T_0. For PMcast, the results obtained come from analytical results. Table 2 depicts the gain of *daMulticast* with respect to alternative approaches.

	T_j,T_{j-1},...,T_0		
	1000, 100, 10	100 ($j = 2$)	100 ($j = 4$)
(1)	0	0	0
(2)	11216	1699	3739
(3)	453	369	615

Table 2. Total number of parasite messages.

15 We know that, according to Section 5, the weakest performance of *daMulticast* are obtained when the supertopic tables are not updated. Thus not replacing crashed processes will give the worst performance of *daMulticast* and this is exactly what we want to measure in this section.

16 We do not give the performance simulation for the maximal number of events sent in a group as well as the maximal number of events sent between group (in [1]).

7. Concluding remarks

This paper presents *daMulticast*, an algorithm to disseminate events in a 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. This last solution would not hamper the overall performance of the algorithm, as it is typical in object-oriented programming languages that the maximal number of multiple supertopics is usually bound, on average, to 2 ([25]).

References

[1] S. Baehni, P.Th. Eugster, and R. Guerraoui. Data-Aware Multicast. Technical Report IC/2003/73, EPFL, http://lpdwww.epfl.ch/publications/, november 2003.

[2] 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.

[3] B. Bollobás. *Random Graphs*. Cambridge, 2001.

[4] 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.

[5] P. Erdös and A. Renyi. On the Evolution of Random Graphs. In *Mat Kutato Int. Közl*, volume 5, pages 17–60, 1960.

[6] 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.

[7] P.Th. Eugster, R. Guerraoui, and C.H. Damm. On Objects and Events. In *Proceedings of the 16th ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 2001)*, pages 131–146, October 2001.

[8] 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.

[9] P.Th. Eugster, R. Guerraoui, A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh. From Epidemics to Distributed Computing. In *IEEE Computer, to appear*, 2004.

[10] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. Scamp: Peer-to-peer lighweight membership service for large-scale group communication. In *Proceedings of the 3rd International Workshop on Networked Group Communication (NGC)*, 2001.

[11] K. Jenkins, K. Hopkinson, and K. Birman. A Gossip Protocol for Subgroup Multicast. In *International Workshop on Applied Reliable Group Communication (WARGC)*, April 2001.

[12] B. Kantor and P. Lapsley. Network News Transfer Protocol, Request for Comments, 1986.

[13] 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.

[14] 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.

[15] 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.

[16] Frédéric Maistre. Joram 3.6.3 administration guide. http://joram.objectweb.org, oct 2003.

[17] 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.

[18] P. R. Pietzuch and J. M. Bacon. Hermes: A distributed event-based middleware architecture. In *Proceedings of the International Workshop on Distributed Event-Based Systems (DEBS'02)*, July 2002.

[19] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *Proceedings of the ACM Conference on Special Interest Group on Data Communications (SIGCOMM)*, 2001.

[20] 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.

[21] 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.

[22] 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.

[23] D. Skeen. *Vitria's Publish-Subscribe Architecture: Publish-Subscribe Overview*. http://www.vitria.com, 1998.

[24] TIBCO. *TIB/Rendezvous White Paper*. http://www.rv.tibco.com/, 1999.

[25] Y. Zibin and J. Gil. Efficient Subtyping Tests with PQ-Encoding. In *Proceedings of the 16th ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 2001)*, October 2001.