HAL
open science

A Semantic Overlay for Self- Peer-to-Peer Publish/Subscribe

Emmanuelle Anceaume, Ajoy K.Datta, Maria Gradinariu, Gwendal Simon, Antonino Virgillito

To cite this version:

Emmanuelle Anceaume, Ajoy K.Datta, Maria Gradinariu, Gwendal Simon, Antonino Virgillito. A Semantic Overlay for Self- Peer-to-Peer Publish/Subscribe. Proceedings of the 26th International Conference on Distributed Computing Systems (ICDCS), Jul 2006, Lisboa, Portugal. pp.22 - 22, 10.1109/ICDCS.2006.12 . hal-01099200

HAL Id: hal-01099200

https://hal.science/hal-01099200v1

Submitted on 6 Jan 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Semantic Overlay for Self-∗ Peer-to-Peer Publish/Subscribe

E. Anceaume[1], A. K. Datta[2], M. Gradinariu[1], G. Simon[3], and A. Virgillito[4*]
[1]IRISA, Rennes, France
[2] School of Computer Science, University of Nevada Las Vegas, USA
[3] France Telecom R&D, Issy les Moulineaux, France
[4] Università di Roma "La Sapienza", Italy

Abstract

*Publish/Subscribe systems provide a useful platform for delivering data (events) from publishers to subscribers in an anonymous fashion in distributed networks. In this paper, we promote a novel design principle for self-∗ dynamic and reliable content-based publish/subscribe systems and perform a comparative analysis of its probabilistic and deterministic implementations. More specifically, we present a generic content-based publish/subscribe system, called DPS (*Dynamic Publish/Subscribe*). DPS combines classical content-based filtering with self-∗ (self-organizing, self-configuring, and self-healing) subscription-driven clustering of subscribers. DPS gracefully adapts to failures and changes in the system while achieving scalable events delivery. DPS includes a variety of fault-tolerant deterministic and probabilistic content-based publication/subscription schemes. These schemes are targeted toward scalability, and aim at reducing and distributing the number of messages exchanged. Reliability and scalability of our system are shown through analytical and experimental evaluation.*

1 Introduction

The publish/subscribe paradigm has emerged in the recent years as an effective technique for building distributed applications in which information has to be disseminated from *publishers* (event producers) to *subscribers* (event consumers). Users express their interests in receiving certain types of events by submitting a filter on the event contents, called a *subscription*.

When a new event is generated and *published*, the publish/subscribe infrastructure is responsible for checking the event against all current subscriptions and delivering it to all users whose subscriptions match the event. Content-based publish/subscribe systems allow complex filters on the event content, enabling the use of constraints such as ranges, prefixes, and suffixes. Combining expressiveness of subscription language and scalability of the infrastructure poses an interesting challenge that has inspired many researchers to explore this topic further. However, actual deployment of pub/sub architectures in real, large-scale systems is currently limited by their lack of self-∗ capabilities. In this work, self-∗ capabilities [2, 3] include 1) *self-organization* — the ability of the system to reduce the entropy of the system, for example, by making nodes form groups to improve or at least to maintain some global properties; 2) *self-configuration* — the ability of the nodes to set up their structural relationships; and 3) *self-healing* — the ability of the nodes to preserve their structural relationships despite the dynamicity of the system (joins, departures, or failures). Enhancing a pub/sub system with self-∗ capabilities allows an easier deployment and a more flexible adaptation in a larger spectrum of applications. Typical implementations of publish/subscribe systems (such as Siena [9], Gryphon [6] and, recently, Kyra [8]) rely on a network of dedicated servers (usually called brokers) that are controlled by administrators in charge of repairing and maintenance interventions. However, in dynamic decentralized scenarios such as peer-to-peer networks this architecture is not feasible, because the high dynamicity of the context requires the topology of the network to be frequently rearranged to face changes due to node joining, leaving and failing.

Scribe [10] and Bayeux [18] are two topic-based systems having self-∗ capabilities. They both make use of a DHT (Distributed Hash Table). A single node is responsible for matching and delivering all notifications related to a specific topic, which is the root of a dynamically-built diffusion tree for events. Combining content-based expressiveness with self-organizing capabilities of DHT-based overlays is discussed in several papers[16, 13, 5]. Meghdoot [13] and [5] work on top of a CAN and a Chord overlay, respectively to store the system subscriptions. The main difficulties in designing content-based pub/sub on top of DHTs are: *(i)* mapping content-based subscriptions into a

*Antonino Virgillito is partially supported by project MAIS, funded by Italian MIUR. Work was done while author was at IRISA, Rennes.

single key space and *(ii)* ensuring the persistence of subscriptions despite the dynamicity of the underlying overlay. Methods to map content-based subscriptions to DHT addresses are described in [16, 5]. The mapping requires subscriptions to be moved from the issuing node to a set of selected "rendezvous" nodes. Mapping may impose some restriction on the constraints applicable in subscriptions with respect to the general language supported by broker-based systems. Typically, string constraints like prefixes and suffixes cannot be easily mapped to a set of keys. Moreover, subscriptions are usually replicated on several rendezvous nodes, and large subscriptions (i.e., subscriptions that possibly match a large number of events) may have many copies. Replication is also used in order to maintain the persistence of subscriptions [13]. This is required because a subscriber can lose its subscriptions if a rendezvous node fails.

Contributions of the Paper In this paper, we promote a novel design principle for reliable content-based publish/subscribe architectures with self-$*$ capabilities. Our system, namely *DPS*, is not based on a network of brokers. Subscribers coordinate among themselves on a peer-to-peer basis to construct optimized event diffusion paths without any human intervention. More precisely, we propose a subscription-driven semantic overlay in which subscribers self-organize according to similarity relationships among their subscriptions. In DPS, two subscribers are considered similar when they share a common attribute of a subscription. Similar subscribers are logically connected into the same group. Groups of subscribers self-configure to form tree structures such that only one tree is built per attribute. No mapping of DHT-based overlay is needed. Virtually all types of attributes and constraints can be directly supported. Moreover, subscriptions are not replicated: a subscription is maintained only at the corresponding subscriber. Differently from a previous solution for semantic-driven pub/sub overlay [12], DPS does not assume the complete knowledge of the network to compute the neighbors of a node. Thus, each subscriber has to keep track of a limited number of its neighbors regardless of the size of the system, and the effect of node failures is confined within a bounded number of neighboring groups.

The general design principles of the DPS overlay can be instantiated with different algorithms to i) traverse the tree for propagating a subscription or a publication across the groups and ii) realize the communication inside a group and between groups. Tree traversal and communication approaches can be combined to design DPS implementations that cater the needs of different deployment contexts. In particular, we propose two different techniques for tree traversals (namely, *root-based* and *generic*) and two different approaches for communication (*leader-based* and *epidemic*). The epidemic approach is based on gossiping of events that obtains high probabilistic guarantees of delivery even in presence of frequent failures. With respect to the gossip-based algorithms for publish/subscribe described in [4], our system supports expressive content-based addressing. Gossiping techniques for content-based publish/subscribe have been exploited in [11], but without a self-organizing overlay.

DPS is evaluated through an extensive simulation and analytical study in which it has been tested using different types of workload that model realistic application scenarios, and comparing the different implementation styles. Results show that the DPS overlay allows to massively reduce the number of visited nodes with respect to a broadcast (from 75% to 90% of the nodes less). Moreover, simulations show the self-healing capabilities of DPS, even when subject to severe failure conditions, and the overall scalability of the approach, that can provide high degrees of reliability without cluttering the network with control messages.

The rest of the paper is organized as follows. In Section 2, we present the system model and the problem statement. In Section 3, we define the logical backbone of DPS, introducing the concept of similarity on which the logical structure of the DPS overlay is based. Several algorithms for subscriptions and publications, and for communicating within the groups are presented in Section 4. Analytical and experimental evaluations of DPS are studied in Section 5. We make some closing statements on the proposed and future work in Section 6. Finally, due to lack of space, the pseudo-code of the algorithms, proof of correctness of DPS and proof of its self-$*$ properties are presented in a longer version of this paper [1].

2 Framework

We assume a finite, yet unbounded dynamic set of nodes. The set is dynamic in the sense that nodes can join or leave at an arbitrary time. Each node is associated with a unique identifier. In a publish/subscribe system, Nodes cooperate to send (publish), relay, and receive (notify) special messages, namely events (or publications). The interest of a node in a set of events is referred to as subscription, and is expressed as a filter defined on the content of the events. In order to inform the system about their subscriptions, nodes invoke a `Subscribe` primitive. The invoking node is called a subscriber. Additionally, nodes can publish events by invoking a `Publish` primitive. We say that a node is notified of an event when the system invokes a `Notify` primitive on it.

We consider a content-based publish/subscribe data model [13] where both subscriptions and events use as building blocks a finite, yet unbounded universe of typed attributes. A *content-based subscription* (filter) is a conjunction of predicates, i.e., $F = AF_1 \wedge \ldots \wedge AF_j$, where AF_i is

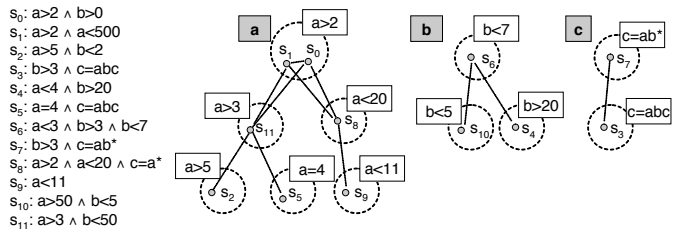


Figure 1. Logical Trees

defined as a tuple $AF_i = (name_i\ Op_i\ c_i)$ where $name_i$ is the name of the attribute, Op_i is an operator, and c_i is a constant value. The operator Op_i can be chosen from a set of basic operators that depends on the attribute type. For example, possible operators for numerical attributes are $\{=, <, >\}$, while string attributes can support prefix, suffix and substring wildcards. Complex filters can be expressed as the conjunction of two or more basic operators. For example, a range filter for an attribute a of the form $c_1 < a < c_2$ can be obtained as the conjunction of the two predicates $AF_1 = (a > c_1)$ and $AF_2 = (a < c_2)$. An *event* is a conjunction of equalities over the attributes' universe. More precisely, an event is denoted as $E = AV_1 \wedge \ldots \wedge AV_k$, where $AV_i = (name_i = v_i)$, where v_i is the value of the attribute. An event predicate AV matches a subscription predicate AF (denoted as $AV \in AF$) if the attribute names are the same in AV and AF and the attribute value in AV is in the range defined by AF. An event matches a subscription iff for all the predicates in the subscription, a corresponding matching value appears in the event.

Finally, it is important to note that the number of attributes for events and subscriptions is not fixed. That is, each single subscription or event can include an arbitrary number of predicates and no prior coordination among nodes is necessary to agree on the event space.

3 The DPS Overlay

In this section, we describe the construction and main features of the overlay scheme dedicated to the DPS pub/sub system. The DPS overlay is subscription driven: subscribers locally self-organize according to their interests and the resulting logical structure is a virtual forest of logical trees, where each tree is associated with an attribute and only one tree is maintained per attribute. Each vertex of a tree is labeled with a predicate (filter on the tree attribute). In the following, we present the relationships that enable subscribers to self-organize according to their subscription similarity. The two relationships lead to the construction of an overlay which is suitable for efficient event dissemination according to the following two principles: i) The more likely a subscriber is to receive a matching event, the higher it is placed in the tree; ii) All subscribers that are likely to receive a common subscription are made neighbors of each other. Efficient paths for events are thus created maintaining only a node's overlay neighbors: if a subscriber does not match an event, all its successors in the tree will neither and the event can be safely filtered out.

Two nodes are *similar* when they share at least one common predicate in at least one of their subscriptions.

Definition 1 (Similarity Relation $\bowtie$) *Let p and s be two nodes representing two subscription filters F_p and F_s, respectively. Assume that $F_p = \wedge_{i \in I} AF_i^{F_p}$ and $F_s = \wedge_{j \in J} AF_j^{F_s}$, where I and J are the sets of indices for predicates in F_p and F_s, respectively, and $AF_m^{F_q}$ is the predicate AF_m in the subscription filter F_q. p and s are* similar *with respect to a predicate AF, denoted as $p \bowtie_{AF} s$* iff $\exists k \in I, k' \in J :: AF_k^{F_1} = AF_{k'}^{F_2} (= AF)$.

A *semantic group* (or simply group) is identified through a *group predicate* which is the common predicate on which the members of the groups are similar. For example, the group labeled $A > 3$ refers to the group of all the subscribers that include $A > 3$ in their subscription predicate:

Definition 2 (Semantic Group) *Let G_{AF} be a set of nodes and AF a predicate. G_{AF} is an AF group* iff $\forall p, s \in G_{AF} :: p \bowtie_{AF} s$.

The *group predecessor* relation imposes a hierarchical ordering among the groups that is based on the *predicate inclusion* relation. A predicate AF_2 is included in a predicate AF_1 if all the events matching AF_2 also match AF_1.

Definition 3 (Predicate Inclusion) *Let AF_1 and AF_2 be two predicates and AV an event. $AF_2 \subset AF_1$*iff $\forall AV \in AF_2$, $AV \in AF_1$.

Two groups are related through the *group predecessor* relation when their respective group predicates are related by the above defined predicate inclusion relation.

Definition 4 (Group Predecessor Relation $\stackrel{pred}{\rightarrow}$) *Let G_1 and G_2 be two labeled groups with respect to the predicates (or labels) AF_1 and AF_2, respectively. Then $G_1 \stackrel{pred}{\rightarrow} G_2$* iff*$AF_2 \subset AF_1$ and $/\exists G_3$ such that $G_1 \stackrel{pred}{\rightarrow} G_3 \stackrel{pred}{\rightarrow} G_2$.*

Each attribute is "owned" by a unique subscriber. For instance, in Figure 1.b, the owners of the trees labeled "a", "b", and "c" are subscribers s_0, s_6, and s_7, respectively. Trees are connected among each other, for example by letting all owners know each other or by keeping at each node a cache of nodes belonging to other trees[1].

[1]Connections between trees are not shown in figures for clarity

A subscriber joins the tree corresponding to only one of the attributes of its subscription. This attribute can be arbitrarily chosen without affecting the correctness of the solution since each event is published in each logical tree that matches every attribute of the event. We decided to decompose a subscription into its attributes, rather than maintaining a single tree (as done in [12]), because of the generality of the content-based language that does not always allow to determine a predecessor between two subscriptions (for example, consider subscriptions $S_1 : (0 < a < 10) \wedge (b > 0)$ and $S_2 : (1 < a < 12) \wedge (2 < b < 4)$). The drawback of this choice is that subscribers also receive events that match only a part of their subscription (*false positives*). Moreover, the number of false positives is likely to grow when more attributes are present. Thus we deem our approach more effective especially when few attributes are considered.

Finally, following only the above definitions, some types of predicates like equality or substrings may be placed at different places in the tree. For example, the group for predicate $a = 4$ in the tree for attribute "a" may be placed below the group for predicates $a > 2$, $a > 3$, $a < 11$, or $a < 20$. These ambiguities may create problems while trying to locate a group upon subscription. To remove them, we impose two additional constraints on the tree construction as follows:

Constraint C1: All groups related to an ambiguous predicate must be placed in the tree following a unique consistent convention. For example, for numerical attributes, equality predicates are placed as successors of either the greater-than or the less-than groups.

Constraint C2: A group G related to an ambiguous predicate is placed as successor of its immediate predecessor G_m, s.t. there exist no groups that are predecessors of both G_m and G.

As previously said, our logical overlay is constructed by the self-organization of subscribers according to their subscription similarity. Other approaches [15, 8] group similar subscribers by applying a partitioning criteria over the event space. All nodes having subscriptions that fall into a common partition are grouped. We chose the similarity over the partitioning method because it does not require prior agreement among the nodes, does not depend on the number of nodes, and reduces the number of non-matching messages received by nodes in a group.

4 DPS Algorithms

In this Section, we present algorithms for the construction of the DPS overlay and the publication and subscription diffusion. These algorithms are organized into two classes: *tree traversal* algorithms, that concern how to locate the position of a group when creating, joining or publishing into a group, and *communication algorithms*, that regard how to exchange publications and subscriptions between groups and among nodes of a same group. The following data structures are used in the DPS algorithms:

- $groupview$: list of pointers to nodes inside the group.
- $predview$, $succview$: two ordered lists of K pointers to nodes in successor/predecessor groups. That is, they point not only to nodes in the direct successor group but also to successors/predecessors at upper/lower levels, in order to handle multiple concurrent failures involving a whole group at once. The lists are ordered according to the nodes predicates. If there are F nodes in the list and a new node is inserted, a node is removed from the bottom of the list. In groups with multiple branches, a node must have one $succview$ list for each of its successor groups.

4.1 Tree Traversal

Prior to subscribing or publishing an event, a node should traverse the tree in order to locate the position of the group it has to belong to, in case of subscription, and all the groups hosting subscribers for the event, in case of publication.

Traversal starts by a *contact point* in the trees related to the subscription or the publication. While subscribers only need to contact one tree, a publisher has to traverse all the trees corresponding to each attribute within the event. Being the trees connected among each other, it is always possible to locate a contact point in any of the trees, for example by propagating a request message with random walks. If there is no tree for an attribute (i.e., no contact point is found at the end of the random walks), a new tree is created and the first subscriber becomes its owner. It is possible that multiple trees for a same attribute are created, when two nodes subscribe concurrently. In order to further reduce the probability of this already rare event, the node that creates a tree starts periodically a new traversal, in order to detect duplicate trees and merge them into one.

In the *root-based* approach, visit of the tree starts from the root and proceeds only downwards, while in the *generic* approach any node can be chosen as a starting point and the visit goes in both directions. Root-based allows to obtain lower latency but imposes high stress on the root node and requires this node to always be known. In the generic approach a visit requires in general more messages but the load is more evenly balanced and the contact point can be any node in a tree.

Subscription Scheme. The subscription process exploits three primitives, namely FIND_GROUP, SUBSCRIBE_TO and CREATE_GROUP, respectively used for locating the group of similar subscribers, joining to it or, if it does not exist, creating a new one in the proper position in the tree. A subscriber s issuing a new subscription has to traverse a tree in order to find its position. Each step in the traversal consists in a call to FIND_GROUP on the next node.

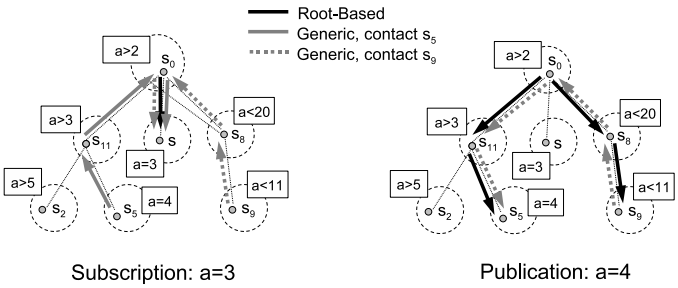


Figure 2. Tree Traversal Example

Figure 2 shows an example of a new group creation. The black line shows the path followed by the subscription $a = 3$ issued by subscriber s using the root-based approach. The subscription is received by group $a > 2$. Since this group is the smallest possible predecessor of group $a = 3$, it is considered the designated predecessor for the subscription (Constraint C2). As this group does not exist, it is created below $a > 2$ and s is added to it. The paths followed by the generic approach are represented by gray lines — solid and broken lines for the contact points s_9 and s_5, respectively.

When an appropriate group for s is located, a SUBSCRIBE_TO message is sent back to s to join the group. If no group matches s's predicate, then a CREATE_GROUP message is sent back s. Each time a new group is created, event propagation is blocked in the predecessor, by setting a proper variable. The variable is reset when data structures related to the new successor are updated. Allowing publications and subscriptions during group construction may result in events not delivered to the new successor group, or more seriously, creating incorrect groups in the trees in case of concurrent group creations.

Publication Scheme. The visit of the trees in a publication is invoked through the PUBLISH primitive from the contact points in all the trees corresponding to the attributes in the published event. An event received by a group is matched against the group predicate. If the event matches the group predicate, it is propagated inside the group (through the PUBLISH_GROUP primitive).

In the root-based approach downstream propagation along the tree continues as long as the event matches the group predicate; it stops otherwise. The successor relation between groups ensures that no matching subscribers are present in any successor groups. So, the entire branch of the tree can be safely excluded from the event propagation. Consider the right side of Figure 2. In the root-based approach (the black line), the publication $a = 4$ is forwarded downstream from the contact point to all the groups with predicates matching the publication. Differently, in the generic approach, if the event *does not* match the group predicate, it still has to be forwarded upstream to the predecessor. Otherwise, if the event has been received by j from its predecessor, it is forwarded to j's successor only if it *does* match j group predicate. In Figure 2 when the publication starts from the group $a < 4$, it is propagated up through both branches (dashed gray lines) through all the matching groups. Group $a > 2$ also needs to forward the publication to its successor in order to reach group $a = 4$.

4.2 Communication within the DPS System

4.2.1 Leader-Based Communication

Each group in each logical tree contains a special node which behaves as the leader of the group. Communication between different groups is realized via their respective leaders. K_c additional leaders are maintained to deal with the leader failure. A node becomes the leader of a group as soon as it creates its own group or remains the only member of a group. Co-leaders are selected as the first K_c nodes that joined the group directly after the leader. Only leaders and co-leaders maintain the $predview$ and $succview$ lists. They also maintain the whole group in their $groupview$. A regular member (i.e., neither a leader nor a co-leader) only has leaders and co-leaders in their $groupview$.

Leader-based Subscription. The subscription process is realized by implementing the CREATE_GROUP and SUBSCRIBE_TO primitives. Both are invoked on the new subscriber by the leader of the predecessor group. Upon receipt of such primitives, the new subscriber updates its variable $group$. If it becomes the leader or co-leader of a new group, it updates $succview$.

Leader-based Publishing. The publication process is realized by implementing the PUBLISH_GROUP primitive. An event received by a group through this primitive is always redirected to the group leader. The leader propagates all the events it receives to all the group subscribers. Each subscriber upon receipt of an event notifies its application only if the event matches one of its subscriptions.

4.2.2 Epidemic Communication

In epidemic communication [7, 14, 4], each member of a group communicates with a subset of members of other groups. In particular, each nodes stores its $predview$, $succview$ and $groupview$, that contain only a subset of the group's nodes. In contrast with the leader-based approach, several copies of a message may traverse the group. This guarantees a higher fault-tolerance at the price of message duplication. Data structures are updated for every change in the membership and maintained by periodic gossiping.

Epidemic Subscriptions. Similar to the leader-based approach, epidemic propagation of subscriptions is realized through the CREATE_GROUP and SUBSCRIBE_TO primitives. An additional primitive, GOSSIP_SUB, is required

to update the views and propagate the update within the group. Upon receipt of such primitives, the new subscriber updates its variables $groupview$ and $succview$. View update messages are gossiped by each node to F_s other nodes in the group. F_s is called the subscription fanout. When a gossip message is received by a node, it is forwarded with probability p, a parameter of the algorithm. To stop the propagation, probability p is reduced proportionally to the number of times the message is forwarded. Note that a node issuing a new subscription can receive more than one CREATE_GROUP or SUBSCRIBE_TO messages if the diffusion started from more than one contact points. This does not require any specific check in the algorithms.

Epidemic approach is prone to undesired behavior when two similar subscriptions are issued concurrently. In particular, if two different nodes in a particular group receive the subscription requests concurrently, two groups corresponding to the same predicate are created. According to our simulation study, this behavior is very infrequent and does not harm the correctness of the system. The system continues behaving according to its specification, only suffering from a non-optimal use of resources.

In order to limit these situations, a merge process is considered: nodes periodically send a $view_update$ message to their successors in the $succview$, containing the whole $succview$. Node receiving the update have the opportunity of adding to their $groupview$ some nodes in the group that they do not know, leading to a merge of disjoint groups.

Epidemic Publishing. Publications are diffused within a group with a simple gossiping. That is, each node forwards the message to k of its neighbors. As for subscriptions, the probability of forwarding an event in the group decreases proportionally to the number of times the event is forwarded.

4.3 Self-healing of DPS Overlay

The DPS overlay is able to self-heal when nodes in the overlay leave by voluntary departures (unsubscriptions) or failures (crashes), that can provoke partitioning between two groups in the tree or inside a same group. Nodes in the $predview$ and $succview$ structure are periodically monitored for failures. If one node has failed, it is immediately replaced by pulling a view update from the other alive nodes.

Self-healing in Leader-based approach When a group leader abruptly crashes, one co-leader (for example, the one with the lowest identifier) becomes the new leader. Its first task consists in promoting a regular member as co-leader. Then, it transmits to the whole group the new leader identity and the new co-leader. The new leader will be contacted by leaders in the adjacent groups that also detect the leader failures, that will be made aware of the new leader identity.

Self-healing in Epidemic Approach In epidemic-approach it is not easy to determine when a group has completely failed as nodes have in general divergent views about the group, predecessors and successors. We tolerate temporary situations in which the overlay is not consistent: for example two distinct groups for the same predicate exists in the tree and one of them does not point to any successor. However, this does not harm the connectivity of the tree, that is preserved at any time by the self-healing process, as shown by the simulation results in Section 5.2. The merge process described above eventually restores the overlay consistency.

5 Evaluation of DPS

5.1 Complexity Analysis

We discuss the scalability of DPS with respect to the message complexity. We also analyze the DPS reliability focusing on the probability that a subscriber interested in a particular filter receives events that match the filter. Moreover, we compare our different approaches (root-leader, root-epidemic, generic-leader, and generic-epidemic) with respect to these two complexity measures.

Reliability. We determine the probability that a new subscriber interested in a filter receives a given concurrently published event. Let us consider a publication e and a concurrent subscription s such that e matches s filter. Let T_s be the number of steps needed by s to find its similarity group, called the subscription turnaround time. Let T_e be the number of steps e needs to reach the s group, called the publication turnaround time. Without compromising the generality, we focus on a single attribute (one tree in the DPS logical structure). Note that subscription s may not "see" event e if the time needed for subscriber s to find its group is greater than that by the publication s, i.e., $T_s > T_e$.

In *root-based* DPS, T_s and T_e are very close since both s and e start at the root of the tree and subscriptions have a higher priority over publications for being processed. Thus subscriptions issued concurrently to events are aware of these events if these events match the subscription filter.

In *generic* DPS, both T_s and T_e depend on the chosen contact point. Hence, it may happen that concurrent publications/subscriptions have different turnaround times. Let p_i be the probability to choose a contact point on the i level of the tree and let s_k be the probability that the similarity group of subscription s is on the level k. T_s is greater than T_e if the number of steps between s contact point and s similarity group is greater than the number of steps between e contact point and s similarity group. More precisely, the probability p that s does not see e is the probability that s contact point is at level i, e contact point is at level j, and s similarity group is at level k, with $i < j < k$. Formally, $p = \sum_{i<j<k} p_i p_k s_k$. Among f events published concurrently with a new subscription, such that all the f

events match the subscription filter, only a fraction $f(1-p)$ are received by the subscriber. Clearly, the root-based DPS causes fewer lost events than the generic DPS scheme, and thus, is more reliable.

Message complexity We study the number of messages sent by the proposed algorithms. We focus only on one tree in the logical structure. Let h be the depth of the tree, S_i the maximal size of a group at level i of the tree, k the number of infected neighbors at each round of the epidemic algorithm, and k' the number of nodes contacted on the next level during the epidemic propagation along the tree.

Let us first consider the *leader-based* communication. In *root-based scheme*, the maximal number of messages corresponds to the traversal of a branch in the tree. Formally, the number is equal to $\sum_{i=0,(h-1)} S_i + (h-2)$. If S is the maximal size of a group, then the maximal number of messages is $h(S+1)-2$. In *generic-based scheme* as the contact point may be any node in the tree, we need to consider that an event may traverse, in the worst case, the current branch up to the root and the other subtree from the root down to the bottom. The maximal number of messages is then $2h(S+1)-4$.

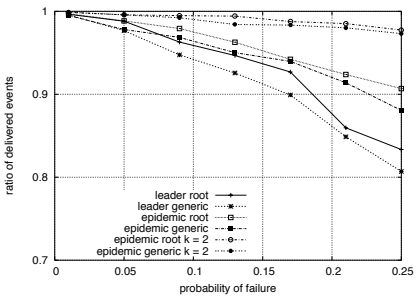
Let us now consider the *epidemic-based* communication. The *root-based scheme* produces in the worst case $kS_0 + kk' \sum_{i=1,(h-1)} S_i + k'(h-2)$ messages. If S is the maximal size of a group, the maximal number of messages is then $kS(1+k'(h-1)) + k'(h-2)$. Similarly, the *Generic based scheme* produces $2(kS(1+k'(h-1)) + k'(h-2))$.

5.2 Simulation of DPS

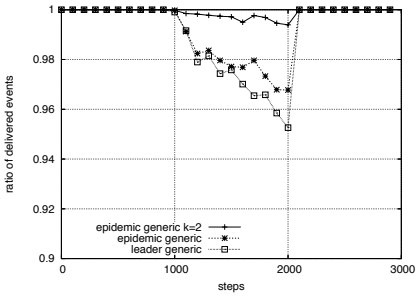
In this section, we present the results of an experimental evaluation of our system performed using an event-based simulator we developed. The aim of the simulation is threefold: i) supporting the basic motivation behind the DPS overlay, i.e., efficient content-based filtering; ii) showing the practical feasibility of our approach and iii) showing that self-* properties can be achieved in a scalable manner without introducing serious (e.g., exponentially growing) overheads on message delivery. Following this direction, we compare all the combinations of the approaches presented in the paper.

Simulation Context. The simulation is cycle based. The workload is characterized by the number and arrival/departure pattern of publishers and subscribers and by the distribution of publications and subscription they issue. Each experiment uses a different workload. Heartbeat-based failure detection between neighbors and recovery mechanism are implemented, with failure detection interval varying randomly from 10 to 25 steps.

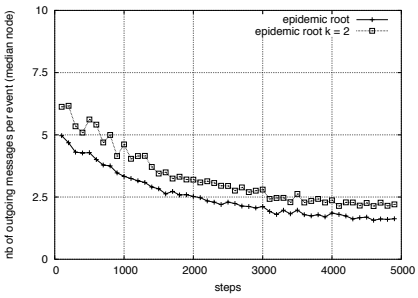
False Positives. First we concentrate on supporting our claim about the beneficial effect of the DPS organization on event dissemination. For each simulation run, we first issued 10,000 subscriptions (one per node) to build the over-



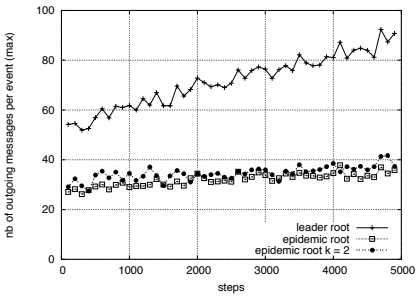
(a) Dependability



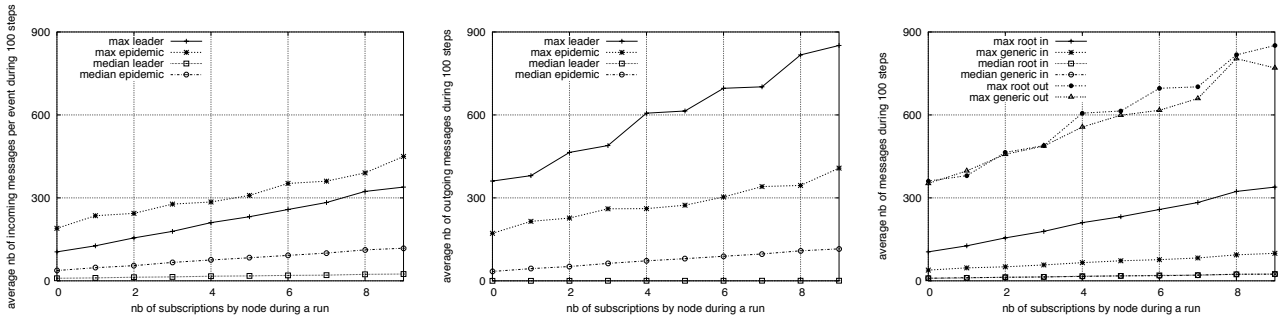
(b) Recovering from failures (generic)



(c) Scalability: outgoing messages per event (median)



(d) Scalability: outgoing messages per event (max)



(e) Leader vs. Epidemic Approaches:Received Messages

(f) Leader vs. Epidemic Approaches:Sent Messages

(g) Root-based vs. Generic Approach

Figure 3. Experimental Results

lay and then we issued 10,000 events[2]. The approach is generic, leader-based (not influencing results). We compute the number of visited nodes per event diffusion, evaluating the number of false positives. The overall number of messages is not considered in this experiment, nor failures and message losses. As the number of false-positives strongly varies according to the workload used, we considered a variety of synthetic workloads in order to cover a representative spectrum of different realistic situations. Synthetic workloads are mostly used in pub/sub simulation studies [9, 13], while a trace of real-world data is used at the best of our knowledge only in [17]. Values for each attribute in subscriptions and events are generated by varying the following parameters: type (integer or string), distribution of values (uniform or zipf), average range size (for numerical subscriptions), percentage of equality predicates. Values for string attributes are chosen in a dictionary of 500 values. Details of the workloads are depicted in Table 1. Workload 1 uses distributions that have been discovered in [17] to model real-world pub/sub data of a stock exchange application. Workload 2 models a multi-player game where players subscribe to events occurring in zones of a bidimensional game plane, whose size can be also very large thus generating a large number of matches. Finally, Workload 3 models an alert monitoring application, where subscriptions are concentrated on a restricted set of critical values and the overall number of matches is very low. Table 1 shows for each workload the percentage of contacted nodes and of matching nodes with respect to the total number of nodes, on average over the number of events, when using all the three workload types. The number of false positives is shown as well. In overall, we observed that DPS allows to cut the number of the visited nodes with respect to a broadcast by at least of the 45%, by a 70% on average, up to the 87% in more realistic situations. The number of false positives falls below an acceptable value of 30%, almost reaching 10% for the realistic workload. In the following sets of simulations, only workload 2 is used, in order to test only conditions which are less favorable for our system (i.e., more false positives). Again, we point out that with workloads based on a larger number of attributes, false positives are more likely to occur.

Dependability. In these experiments, we test the ability of the system in delivering messages despite node failures. We built two different scenarios, where the system initially contains 1,000 nodes and the execution is 3,000 steps long. All nodes subscribe to three distinct subscriptions (different for each node)[3]. In the first scenario, node failures are uniformly distributed in time, with a frequency of $1/p$, with p varying between 0.01 and 0.25 resulting, at the end of the simulation, in a number of nodes in the system which is, on an average, 97% to 25% of the initial nodes. This scenario tests a realistic situation where nodes disappear independently and in an unpredictable manner. A new event is published every 10 steps. In the second scenario, execution is divided into three phases. Nodes do not fail until step 1,000. Then one node fails every two steps between steps 1,000 and 2,000. From that point onwards, the system executes normally without failures all the way to the end of the execution. This scenario tests the recovery capabilities of the system after a large number of concurrent node failures. In both cases, we measure the ratio of correctly delivered events, i.e., the percentage of published events that reaches a node with a matching subscription.

Figure 3(a) shows the results of the first scenario. In all the approaches the system can reach a number of delivered events which is at least 80%, also when most of the nodes have failed. This is obtained with the leader-based

[2]The number of events and subscription does not influence the results and is chosen as a sufficiently large sample

[3]Increasing the number of subscriptions per node does not change the nature of the results in this experiment

	Attr.	Ev. Distr.	Sub. Distr.	Range Size	Eq. Perc.	Matching	Contacted	False Positives
Workload 1	num string	unif unif	zipf zipf	10%	50% 50%	2.37%	13.56%	11.19%
Workload 2	2 × num	unif	unif	50%	0%	25.13%	54.74%	29.61%
Workload 3	3 × num	zipf	zipf	20%	20%	0.42%	17.15%	16.73%

Table 1. False Positives Experiment: Workload details and Results

approach, which is as expected the least 'robust'. Increasing the number of co-leaders may offer a way to further increase this figure. The epidemic scheme confirms the expected higher number of delivered events than the leader-based approach. In particular, with epidemic, k=2, the ratio is greater than 0.97 even with a significant probability of failures. Results of the second scenario are exhibited in Figures 3(b), where the ratio of delivered messages is still high as events are delivered in more than 95% of cases. This curve also show that the system is able to self-recover as the ratio quickly grows back to 1 after step 2,000 in all cases.

Scalability. We tested the ability of the system to scale by measuring the load managed by nodes when propagating events and subscriptions as the size of the system grows. In this scenario, the system initially contains 1,000 nodes. A new node enters the system every two steps and immediately emits a new subscription. Publications are produced at a regular rate along system execution (10 new events every 100 steps). Figures 3(c) and 3(d) provide the results for this scenario in leader-based and epidemic configurations using root-based traversal[4]. The two plots report the time vs. the number of messages sent by a node per each event.

Figures 3(c) refers to the *median* node, defined as the node that sends less messages than half of the nodes and more messages than the other half. Figures 3(d) refers to the most overloaded node. The two plots show that in general the number of messages per event does not increase with the number of nodes, confirming the overall scalability of our approach. The only exception is the most overloaded node in the leader-based approach which has to handle more messages as system grows, because the size of the groups increases accordingly.

5.2.1 Leader vs. Epidemic

In the following experiments, we compare the behavior of the different approaches for implementing DPS while increasing the load at each node. Experiments are conducted in the following scenario: the number of nodes is 1,000, and each node emits regularly a new subscription and a new publication. Publications are produced at a rate of 10 new events every 100 steps, while a new subscription is produced regularly every 300 steps. Hence, the number of subscriptions per node ranges from 0 to 10 (maximum 10,000 in total). We measure the number of incoming and outcoming messages on the most loaded and median nodes, respectively, sampled during a period of 100 steps. Messages include the ones due to publication (10 events), subscription, and management of the overlay. Results were produced with a root-based approach for tree traversal and are presented in Figures 3(e) and 3(f).

Incoming Traffic. As expected, in epidemic communication the overall number of processed messages is in general higher than in leader-based because while in leader-based approach, all events are received by the corresponding recipients only once, in the epidemic approach, some redundant messages are received. The difference remains constant when the number of subscriptions increases as it is only due to event delivery. The overall increase is due to the fact that groups become more populated. The median node in the leader-based scheme is only a receiver of the events. So, the received messages practically remain constant, growing only slightly because each node receives more matching events. On the contrary, in the epidemic approach, each node in the group is involved in overlay management. Besides processing the events, it also receives requests from the predecessor groups that grow as subscriptions grow. However, the increase in the number messages is much slower wrt the increase in the subscriptions. Again, the message redundancy motivates the higher number of received messages in the epidemic-based approach.

Outgoing Traffic. The drawback of the leader-based approach is evident when considering the outcoming traffic per node. The number of messages sent by the leader highly increases with subscriptions, following the increasing size of the groups and the higher number of recipients per event. Moreover, the load is highly unbalanced, with the median node showing no sending activity. As expected, the load is more balanced in the epidemic approach, because neighbors are distributed among the nodes in a group. However, we still experience a difference between the most overloaded node and the median, because the first nodes to join a group are more likely to be the contact nodes of numerous predecessors and successors. However, the most overloaded node in the epidemic approach handles less than half of the messages in the leader-based, although the overall number of processed messages is higher in the epidemic, confirming better load-balancing.

[4]Experiments performed using generic approach returned almost overlapping curves, that are not reported for the sake of readability

5.2.2 Root vs. Generic

The results of the comparison between root-based and generic approach, presented in Figure 3(g), were obtained in the same scenario as the preceding experiment[5].

Incoming Traffic. The most overloaded node in the root-based approach is obviously the owner of the attribute. As the number of subscriptions increases, the number of messages it should deal with also increases. The generic approach effectively manages to distribute the incoming load among nodes, maintaining an almost constant number of messages at higher load. Note that the subscriptions influence the incoming traffic more than the outcoming traffic. This indicates the positive effect of the generic approach on the subscription process.

Outgoing Traffic. These experiments reveal few differences between the two approaches. This can be explained by considering that the outgoing messages in the leader-based approach are mainly due to the events. The most overloaded node in both cases corresponds to the leader of the bigger group. As the number of subscriptions increases, the size of the group and, subsequently, the leader outgoing traffic increase as well. Using the leader-based approach, in both cases, the median node does not send any messages.

6 Conclusion and Discussion

We have presented DPS, a distributed reliable and scalable content-based publish/subscribe system that exhibits self-$*$ characteristics. We have proposed different methods of diffusion of subscriptions and publications that can be combined to obtain four different implementations of the system. Based on the simulation results, we can conclude that the leader-based approach is more suitable for a relatively small set of nodes that are less prone to failures. On the other hand, the epidemic-approach provides higher dependability, better scalability, and load balancing at the cost of higher message complexity. As for the tree traversal strategies, the generic approach is more suitable for the subscription process as it better distributes the load. On the contrary, the publication process benefits from the root-based approach that obviously provides lower latency. The possibility of choosing different implementations makes the proposed system very versatile, so it can be deployed in many applications (e.g., virtual worlds, virtual games, e-market, etc.). As a future research direction, we intend to explore the evaluation of DPS in other specific contexts, such as sensor networks.

References

[1] E. Anceaume, A. K. Datta, M. Gradinariu, G. Simon, and A. Virgillito. DPS: Self* dynamic reliable content-based publish/subscribe system. Technical Report 1665, IRISA, 2004.

[2] E. Anceaume, X. Defago, M. Gradinariu, and M. Roy. Towards a theory of self-organization. In *Proc. of OPODIS 2005*.

[3] O. Babaoglu, H. Meling, and M. A. Anthill: A framework for the developments of agent-based peer-to-peer systems. *Proc. of ICDCS 2002*, 2002.

[4] S. Baehni, P. Eugster, and R. Guerraoui. Data-Aware Multicast. In *Proc. of DSN '04*, 2004.

[5] R. Baldoni, C. Marchetti, A. Virgillito, and R. Vitenberg. Content-based publish-subscribe over structured overlay networks. In *Proc. of ICDCS 2005*, 2005.

[6] S. Bhola, R. Strom, S. Bagchi, Y. Zhao, and J. Auerbach. Exactly-once Delivery in a Content-based Publish-Subscribe System. In *Proc. of DSN 2002*, 2002.

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

[8] F. Cao and J. P. Singh. Efficient Event Routing in Content-based Publish-Subscribe Service Networks. In *Proceedings of the 23rd Conference on Computer Communications (IEEE INFOCOM 2004)*, 2004.

[9] A. Carzaniga, D. Rosenblum, and A. Wolf. Design and Evaluation of a Wide-Area Notification Service. *ACM Transactions on Computer Systems*, 3(19):332–383, Aug 2001.

[10] M. Castro, P. Druschel, A. M. Kermarrec, and A. Rowston. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8), October 2002.

[11] P. Costa, M. Migliavacca, G. Picco, and G. Cugola. Epidemic algorithms for reliable content-based publish/subscribe: An evaluation. In *Proc. of ICDCS 2004*, 2004.

[12] P. Felber and R. Chand. Semantic peer-to-peer overlays for publish/subscribe networks. In *Proc. of EUROPAR 2005*.

[13] A. Gupta, O. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: Content-based publish:subscribe over p2p networks. In *Proceedings of Middleware '04*, 2004.

[14] A.-M. Kermarrec, L. Massouli, and A. Ganesh. Probabilistic Reliable Dissemination in Large-Scale Systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3), 2003.

[15] A. Riabov, Z. Liu, J. Wolf, P. Yu, and L. Zhang. New algorithms for content-based publication-subscription systems. In *Proc. of ICDCS'03*, pages 678–686, 2003.

[16] P. Triantafillou and I. Aekaterinidis. Content-based Publish/Subscribe over Structured P2P Networks. In *Proceedings of DEBS 2004*, 2004.

[17] Y. Wang, L. Qiu, D. Achlioptas, G. Das, P. Larson, and H. J. Wang. Subscription Partitioning and Routing in Content-based Publish/Subscribe Networks. In *Proc. of DISC '02*, 2002.

[18] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proc. of the Int. Workshop on Network and OS Support for Digital Audio and Video*, 2001.

[5] Due to space limitations, only results for leader-based approach were presented, as epidemic follows the same trend