Content-Based Publish-Subscribe over Structured Overlay Networks

Roberto Baldoni, Carlo Marchetti, Antonino Virgillito
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
email: {baldoni,marchet,virgi}@dis.uniroma1.it

Roman Vitenberg
IBM Research, Haifa
email: romanv@il.ibm.com

Abstract

This paper introduces a novel architecture for implementing content-based pub/sub communications on top of structured overlay networks. This architecture overcomes some well-known limitations of existing infrastructures, i.e. lack of self-configuration and of adaptiveness to dynamic changes. This is achieved by devising a mediator stratum between the rich event subscription semantics of content-based pub/sub systems and the standard logical addressing scheme of overlays. The paper describes details of the design and provides considerations in selecting the subscription-to-node and event-to-node mappings suitable for the solution. We identify the lack of native support for one-to-many communication by overlay networks as the main impediment for efficient system operation. The paper introduces a novel primitive for one-to-many message delivery, showing through simulation how this can improve performance of the architecture. The simulation study also shows performance comparison between the different mappings proposed as well as evaluation of other optimizations discussed in the paper.

1 Introduction

It is well known that a large class of application domains ranging from enterprise application integration to sensor networks can benefit from the presence of publish&subscribe (pub/sub) systems [1]. The most general form of pub/subs supports *content-based* subscriptions, using which subscribers are able to express interest in events by specifying a set of constraints over event attributes. This subscription semantics is significantly more expressive and flexible than topic-based subscriptions, wherein a single *topic* attribute determines the relationship between information sources and sinks. However, content-based subscription is harder to implement because the source-sink correlation cannot be determined *a priori*: it has to be computed on a per-event basis. As a consequence, the significant body of research focuses on efficient matching and event routing algorithms, while the problem of devising a scalable and at the same time highly adaptive content-based implementation remains largely unresolved.

In this work, we propose an architecture for implementing content-based pub/sub that harnesses and leverages the scalable message routing and adaptive self-configuration of overlay networks, i.e. the properties so much sought by content-based pub/sub schemes. To the best of our knowledge, the proposed architecture is the first content-based pub/sub implementation not requiring any manual configuration and management apart from the setup of an overlay network itself.

Designing a content-based pub/sub infrastructure on top of the standard communication and programming model provided by common structured overlay networks (e.g. CAN, Chord, Pastry, and Tapestry) requires to address two main issues: (i) the gap between the rich language that describes an event or a subscription and the single key identifier by which a message is typically routed in overlay networks, and (ii) the inefficiences due to implementing one-to-many communications on top of the unicast primitive commonly provided by overlays.

In order to address the former issue, we introduce a new class of "subscription-static" mappings whose nature facilitates system adaptiveness to dynamic changes. We consider three specific mappings in this class and provide their analysis in terms of memory usage and message complexity.

In order to address the latter issue, we propose several complementary optimizations. At the overlay level, we propose to introduce a multicast primitive in order to eliminate the inefficiencies that arise when using the standard overlay unicast primitive for implementing content-based pub/sub semantics (namely, non-optimal routing paths and redundant deliveries of a same message). At the pub/sub level, we propose two optimizations, namely *mapping discretization* and *event collecting and buffering*, that can increase system scalability and performance.

This study is supported by comprehensive performance analysis, performed through simulation that shows the over-

Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05)
1063-6927/05 $20.00 © 2005 **IEEE**

all scalability of our approach as well as the effectiveness of the proposed optimizations. The paper is structured as follows: Section 2 presents related work, Section 3 provides background on overlay networks and pub/sub systems, Section 4 describes the general architecture and optimizations, Section 5 presents the experimental results, Section 6 concludes the paper.

2 Related work

Several pub/sub systems (e.g., Gryphon [8], Hermes [10], JEDI [4], LeSubscribe [6], SIENA [2]) have been proposed in the literature, featuring a content-based addressing scheme for subscriptions. All these systems rely on an application-level network of servers that share the load of determining the recipients for an event and routing events toward them. As a consequence, the research on content-based pub/sub focuses on scalable and efficient algorithms for carrying out these functions. None of current content-based pub/sub systems features self-organization capabilities, thereby requiring human intervention for set-up and management of the application-level network. This strongly limits the actual deployment of such systems in practical large-scale settings.

An alternative approach is the *event space partitioning*, presented in [16]. Here the event space is divided into a set of partitions, and each partition is assigned to a node. This approach minimizes event traffic by forwarding each event to just a single node. This simplifies the initial system setup and eliminates the need of propagating and keeping the knowledge about the system state (i.e., stored subscriptions), thereby rendering the architecture less stateful and vulnerable to failures. Yet, even in these favorable settings, providing system self-configuration as well as scalable and balanced routing of events and subscriptions in a dynamic environment is still non-trivial. This issue of self-configuration is the main focus of our work.

Self-organization and fault-tolerance capabilities characterize structured peer-to-peer overlay network infrastructures, such as CAN [11], Chord [13], Pastry [12] and Tapestry [17]. Such systems provide an addressing scheme (independent of the actual network addresses) that is used to implement scalable and efficient application-level routing mechanisms, which are adaptive to node joins and departures. Scribe [3] and Bayeux [18] are two pub/sub systems built on top of Pastry and Tapestry respectively, which leverage their scalability, efficiency and self-organization capabilities. However, both systems provide only a topic-based addressing, thus offering limited expressiveness to users.

A few previous works (e.g., [9], [10], [14], and [15]) have already recognized the potential of combining the self-organization capabilities of overlay networks with the expressive addressing schemes of content-based pub/sub. In particular, [15] and [9] describe methods for mapping general content-based subscriptions, including range constraints, to overlay addresses, respectively using Chord and CAN as a reference overlay. However, each of these works develops a single individual mapping that is dynamically adjusted in order to cope with routing inefficiencies and improve load balancing. In contrast, our paper considers a general architecture for implementing a pub/sub system that uses the standard interface and functionality of structured overlay networks. We introduce an abstract stateless mapping, which is instantiated through three different specific mapping methods. One of those methods (Attribute-split) resembles the solution in [15], but it is implemented with a different communication protocol (see Section 4.2).

3 An overview of overlay networks and content-based pub/sub systems

3.1 Overlay networks

The common idea behind self-organization and routing in most overlay networks is that instead of being routed directly using physical nodes' addresses ranging over a space N, messages are routed by logical *key* identifiers, defined over a space K. The overlay network manages the mapping $\mathcal{KN} : K \rightarrow N$ of keys to actual nodes (further denoted as the *KN-mapping*); in other words, each key is *covered* by some node (e.g., the one which maps to the closest key value.) The system automatically routes the message to the node which covers the key in the message.

While the keys are exposed to the application, the KN-mapping is the sole responsibility of the overlay network and is typically hidden from the user. Such key-based routing provides a convenient higher-level abstraction for the application. In addition, it allows the system to quickly adapt to dynamic changes, such as a failure or addition of individual data centers to the system.

Virtually all overlay schemes provide a similar interface for the applications, which consists of the following basic primitives: a) `send(m,k)` operation to send a message m to a destination determined by the key k, b) `join()` and `leave()` operations for a node to join or leave the system, and c) `deliver(m)` operation that invokes an application upcall upon message m delivery.

For the purposes of this paper, we now present the key features of a specific overlay routing protocol, namely the Chord protocol [13][1].

[1] Let us note that the publish-subscribe infrastructure presented in this paper is portable in the sense that it can use any overlay routing scheme mentioned above.

Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05)
1063-6927/05 $20.00 © 2005 **IEEE**

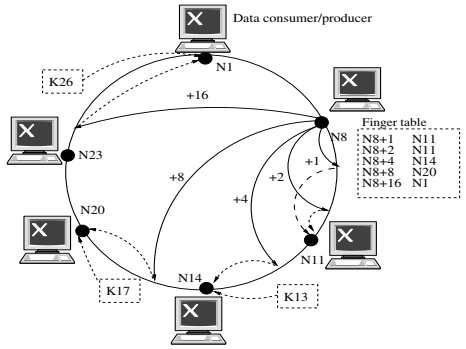


Figure 1. Chord: a content-based routing protocol for P2P networks

3.1.1 The Chord protocol

The Chord protocol is based on the fast distributed computation of a hash function that maps keys to the nodes covering them. A Chord node only maintains information about $O(\log n)$ other nodes in an n-node network.

The assignment of keys to nodes is done with *consistent hashing*. The consistent hash function assigns each node and key an m-bit identifier using SHA-1. These identifiers are ordered on an identifier circle modulo 2^m. The identifier circle is called the *Chord ring*. We will use the term "key" to refer to both the original key and its image under the hash function, as the meaning will be clear from context. Key k is assigned to the first node called the *successor node* of key k, whose identifier is equal to or follows k in the ring. Figure 1 shows a Chord ring with $m = 5$. Keys with identifiers $13, 17$ and 26 are assigned to nodes with identifiers $14, 20$ and 1.

For efficient lookup, Chord maintains a table called *finger table*. The i^{th} entry in the table at node n is the successor node s of the identifier $(n + 2^{i-1})$ modulo 2^m. The node s is called the i^{th} *finger* of node n. Each entry also contains the IP address and the port number of the relevant node. Figure 1 shows an example finger table for node 8. For example, the 4^{th} entry is $N20$ which is the successor node of identifier $(8 + 2^{4-1})$ mod $2^5 = 16$.

3.2 Content-based publish/subscribe

A distributed content-based publish/subscribe system comprises a set of nodes, each of which can act both as a producer and a consumer of information, playing the role of *publisher* and *subscriber*, respectively. Publishers and subscribers exchange information in form of *events* and *subscriptions*.

Events are defined according to a data model in which an event is defined as a set of attribute-value pairs. Each attribute $e.a_i$ has a *name*, a simple character string, and a *type*. The type is generally one of the common primitive data types defined in programming and query languages (e.g. integer, float, string, etc.). Events are thus defined over a d-dimensional *event space*, denoted as Ω.

On the subscribers' side, interest in specific events is expressed through *subscriptions*. A subscription σ is a query composed by a conjunction of constraints (disjunctive constraints can be treated as separate subscriptions). A single constraint is indicated with $\sigma.c_i$. Actual constraints depend on the specific data model and subscription language. Without loss of generality we consider a content-based language allowing range constraints over numerical attributes.[2] Queries defined according to this assumption are elements of the space Σ of all possible subscriptions. Therefore a query $\sigma \in \Sigma$ captures a subspace of the overall event space, i.e. $\sigma \subseteq \Omega$. We say that an event $e \in \Omega$ $matches$ a subscription $\sigma \in \Sigma$ iff it satisfies all the constraints in σ, i.e. iff $e \in \sigma$. When an event matches a subscription, the corresponding subscriber has to be delivered a *notification* for e.

In order to implement the matching, available content-based systems distribute across the nodes in the system the tasks of storing subscriptions, matching events against subscriptions, and delivering notifications to subscribers. Subscriptions in Σ and events in Ω are assigned to nodes through two mapping functions, namely $SN : \Sigma \rightarrow 2^N$ and $EN : \Omega \rightarrow 2^N$. In particular, given a subscription σ, $SN(\sigma)$ returns a set of nodes, named *rendezvous nodes of* σ, which are responsible for storing σ and forwarding events matching σ to all the subscribers of σ. $EN(e)$ complements SN by returning the *rendezvous nodes of* e, which are the nodes responsible for matching e against subscriptions registered in the system. These functions are used by nodes as follows: upon issuing a subscription σ, a consumer node sends σ to the nodes in $SN(\sigma)$, which store σ and the consumer identifier. Producer nodes send their events to the nodes in $EN(e)$, which match e against the subscriptions they host. For each subscription matched by e, e is forwarded to the corresponding subscriber. In order for the matching scheme to work and forward e to the consumers, it is necessary that the rendezvous nodes of e collectively store all the subscriptions matched by e, i.e., if $e \in \sigma$ for any subscription σ, then $EN(e) \cap SN(\sigma) \neq \emptyset$. We refer to this property as the "mapping intersection rule" in the rest of the paper.

4 Content-based Pub/Sub using Overlay Networks

We propose an architecture that is based on two major principles: 1) to utilize and leverage the self-organization

[2]For example, string values can be reduced to numbers by applying a hashing.

Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05)
1063-6927/05 $20.00 © 2005 **IEEE**

and scalable routing capabilities provided by overlay networks in order to design scalable and dynamically adaptable content-based pub/sub systems and 2) to employ a general form mapping that does not depend on the stored subscriptions (we call such mapping stateless). In summary, both principles contribute to the system adaptivity and self-configuration: the first principle makes it adaptive to node failures and joins whereas the second principle eliminates the need to propagate the knowledge about currently stored subscriptions.

4.1 Basic system architecture

The basic architecture we propose is depicted in Figure 2. According to it, the generic application using the content-based pub/sub infrastructure can perform subscriptions (sub()) and publications (pub()) as well as to be notified of incoming events matching some of its subscriptions (notify()). The lower layer in the architecture is formed by the overlay network that implements the behavior and provides the primitives we described in Section 3.1.

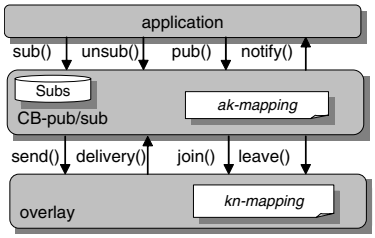


Figure 2. The proposed basic architecture

As a consequence, the *CB-pub/sub* layer has to map the event space into the universe of keys instead of nodes. In other words, the implementation need to provide $SK : \Sigma \rightarrow 2^K$ and $EK : \Omega \rightarrow 2^K$ mapping functions instead of SN and EN.

More precisely, the middle *CB-pub/sub* layer is responsible for implementing the functionality of a content-based pub/sub system, by exploiting the underlying overlay network infrastructure. To this end, this layer performs the following operations:

• implementing and computing the *SK* and *EK* mappings. In Figure 2, we abstract this functionality out into the *ak-mapping* module of the layer;

• forwarding subscriptions σ and events e to the keys in *SK(σ)* or *EK(e)*, respectively. This is done by invoking the unicast send() primitive provided by the overlay network. When sending a subscription, the key of the subscriber is also sent;

• receiving subscriptions and events through the delivery() upcall of the underlying overlay network. Subscriptions are stored along with the subscriber's keys whereas events are matched against the stored subscriptions;

• forwarding notifications when matches are found. If an event matches multiple subscriptions, the keys of the matching subscribers are determined and a notification is sent to all of the corresponding subscribers, again by utilizing the unicast send() primitive;

• managing node joins and departures.

While hiding the dynamic KN-mapping from the application greatly facilitates the design it creates a problem for stateful applications whose state distribution depends on the composition of nodes. For example, when a new node n joins the system, the subscriptions that map to its partition in the key space have to be moved to n from other nodes. Similarly, when a node leaves or crashes, the subscriptions that it stores should be relocated to its "neighbors" in the key space. Currently existing overlay networks neither manage the distribution of the application state nor expose information about node's neighbors.

Fortunately, each overlay network provides a proprietary way of sending messages to neighbors (e.g., *node successor* in Chord). This allows the joining node to pull the state from its neighbors. Failures can be handled by each node having its state replicated on a small number of neighbors.

4.2 Stateless mappings

We now consider three specific ak-mappings that satisfy the mapping intersection rule. The following notation is used to describe system parameters: d, the number of dimensions in the event space, Ω_i, projection of Ω on i's dimension, and K, the key space. For the Key Space-Split mapping, it is important that K be represented by a bit string, whose length we denote m. While this is consistent with most overlays, such as Chord and Pastry, a slightly more general representation is easy to obtain for other overlays, e.g., CAN, wherein a key is a discrete point in a multidimensional space.

Each our mapping is based on a collection of hash functions $h_i : \Omega_i \rightarrow [0,1]^l$; h_i maps attribute values in Ω_i to bit strings of length l. The hash functions as well as l are part of the mapping definition and may differ across the mappings. Given the set of h_i functions, we define a set of hash functions H_i for constraints $\sigma.c_i$ (both equality and inequality ones) to return sets of l-length bit strings as follows: $H_i(\sigma.c_i) = \{h_i(x) | x \in \Omega_i \wedge x \text{ satisfies } \sigma.c_i\}$.

The mapping definitions below specify l but use h_i as an external parameter. However, our implementation and performance analysis use a simple scaling function: $h_i(x) = x \cdot 2^l / |\Omega_i|$. Assuming that a constraint $\sigma.c_i$ spans the range of r_i values and that for every i, $2^l < |\Omega_i|$, this implies that $H_i(\sigma.c_i)$ returns $\lceil r_i \cdot 2^l / |\Omega_i| \rceil$ distinct values.

We use this fact below to compare the proposed mappings wrt. the number of keys to which subscriptions and publications map. While it is desirable that a subscription

Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05)
1063-6927/05 $20.00 © 2005 **IEEE**

	c_1	c_2
σ	$a_1 < 2$	$3 < a_2 < 7$
e	$a_1 = 1$	$a_2 = 6$

(a)

$SK(\sigma) = \{H(\sigma.c_1), H(\sigma.c_2)\}$
$H(\sigma.c_1) = \{h(0), h(1)\} = \{0000, 0001\}$
$H(\sigma.c_2) = \{h(4), h(5), h(6)\} = \{0100, 0101, 0110\}$

$EK(e) = \{h(e.a_1)\}$
$h(e.a_1) = h(1) = 0001$
$h(e.a_2) = h(6) = 0110$

(b) Mapping 1

$SK(\sigma) = \{H(\sigma.c_1) \times H(\sigma.c_2)\} = \{0010, 0011\}$
$H(\sigma.c_1) = \{h(0), h(1)\} = \{00, 00\}$
$H(\sigma.c_2) = \{h(4), h(5), h(6)\} = \{10, 10, 11\}$

$EK(e) = h(e.a_1) \circ h(e.a_2) = 0011$
$h(e.a_1) = h(1) = 00$
$h(e.a_2) = h(6) = 11$

(c) Mapping 2

Figure 3. Mapping Examples

map to multiple keys as it facilitates load-balancing and high-availability, mapping a subscription to a high number of keys is non-scalable in terms of both memory and bandwidth consumption.

In order to illustrate the mappings, we consider a simple example event space composed of 2 integer attributes taking values in the range of 0–7 each ($|\Omega_i| = 8$) so that $h_1 = h_2 = h$ and $H_1 = H_2 = H$. The key space in this example coincides with the attribute space so that $m = 4$. We take a subscription $\sigma = \{a_1 < 2, 3 < a_2 < 7\}$ and an event $e = \{a_1 = 1, a_2 = 6\}$ (Figure 3(a)) and show their processings for two of the mappings.

Mapping 1: Attribute-Split. The length of a bit sequence returned by h_i functions is equal to the number of bits in a key, i.e. $l = m$, so that h_i and H_i functions returns simply keys and sets of keys. The idea behind this mapping is to hash each constraint $\sigma.c_i$ within a subscription σ independently to a set of keys $H_i(\sigma.c_i)$, and then to send the subscription to the union of all these sets. In order to satisfy the mapping intersection rule, it suffices to choose a rendezvous key for an event by hashing just one of the event attributes. Formally, the SK function is defined as $SK(\sigma) = \bigcup_i H_i(\sigma.c_i)$, and the EK function is defined as $EK(e) = \{h_i(e.a_i)$ for some $i, 1 \leq i \leq d\}$. An example of this mapping is shown in Figure 3(b). The figure shows the 4-bit strings returned by SK and EK when invoked on σ and e, respectively. Since 2 values are spanned by c_1 and 3 by c_2, $SK(\sigma)$ returns a total of 5 keys.

Thus, the EK function returns just one key but the SK function returns a set of up to $O(\sum_{i=1}^{d} \lceil r_i \cdot 2^m / |\Omega_i| \rceil) = O(d + 2^m \cdot \sum_{i=1}^{d} (r_i / |\Omega_i|))$ distinct keys. This feature can be used to provide increased availability. However, if d is high, a subscription might be mapped to a large number of keys thereby impeding system scalability. The goal of reducing the number of keys to which a subscription is mapped motivates the following two mappings.

Mapping 2: Key Space-Split. This mapping is based on the idea of partitioning the m bits of the key space across the attributes so that $\lfloor m/d \rfloor$ bits are assigned to each attribute. Accordingly, $l = \lfloor m/d \rfloor$ in this mapping. The SK function returns all possible concatenations of bit strings: $SK(\sigma) = \{s_1 \circ \ldots \circ s_d | s_i \in [0,1]^l \wedge s_i \in H_i(\sigma.c_i)\}$. To satisfy the mapping intersection rule, the EK mapping is defined as $EK(e) = h_1(e.a_1) \circ \ldots \circ h_d(e.a_d)$, i.e. it returns a single rendezvous key. An example of this mapping is shown in Figure 3(c). The h function returns a 2-bit string for each value ($l = m/d = 2$).

The number of distinct concatenations is $\prod_{i=1}^{d} \lceil r_i \cdot 2^{\lfloor m/d \rfloor} / |\Omega_i| \rceil$. In particular, if $\forall i, r_i \cdot 2^{\lfloor m/d \rfloor} / |\Omega_i| > 1$, then this expression becomes $O(2^m \cdot \prod_{i=1}^{d} (r_i / |\Omega_i|))$.

Mapping 3: Selective-Attribute. This mapping is based on the observation that in many cases subscriptions may exhibit strong selectivity in one particular attribute, i.e., they filter out all but a small portion of all possible values for this attribute (i.e., the selective attribute). These selective constraints frequently occur in event spaces in practice [6], e.g., equality constraints on attributes such as 'type' or 'topic'. The idea behind this mapping is to map a subscription σ just by its most selective constraint $\sigma.c_s$, $r_s / |\Omega_s| = \min_{i=1}^{d} (r_i / |\Omega_i|)$. $l = m$, as in Attribute-Split. The SK function is simply defined as $SK(\sigma) = H_s(\sigma.c_s)$. However, each event has to be mapped by every attribute separately so that EK is defined as $EK(e) = \bigcup_{i=1}^{d} \{h_i(e.a_i)\}$.

Thus, a subscription is mapped to $\lceil 2^m \cdot \min_{i=1}^{d} (r_i / |\Omega_i|) \rceil$ keys by this mapping. This is at the very least d times better than the figure for Attribute-Split, even if no selective constraints are present. However, comparison with Key Space-Split is less straightforward. If all constraints are non-selective to the extent that $\forall i, r_i \cdot 2^{\lfloor m/d \rfloor} / |\Omega_i| > 1$, then Key Space-Split always outperforms this mapping. However, in the presence of at least one selective constraint, e.g., an equality constraint, Selective-Attribute maps the subscription to just a single key (or a few keys). At the same time, Key Space-Split may still return a huge number of all possible combinations if all other constraints are non-selective. In particular, Selective-Attribute is the least sensitive to partially defined subscriptions, i.e., subscriptions that specify constraints on only some of the attributes.

However, unlike the other two mappings, Selective-

Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05)
1063-6927/05 $20.00 © 2005 **IEEE**

Attribute maps an event to d keys in the worst case. This disadvantage might be significant if the workload is dominated by events.

Discussion. It is important to note that while devising good SK and EK functions is still a non-trivial task, it is simpler than providing SN and EN mappings. This is because the universe of keys is static and known by all nodes in advance. As a further advantage of using such static mappings, nodes do not need to coordinate their computation of the mapping, not in the beginning to bootstrap the system and not in presence of dynamic changes in the node composition. The underlying overlay network transparently handles and dynamically adjusts the KN-mapping and performs the routing accordingly. In addition to facilitating self-configuration, the proposed architecture also makes the system state less dependent on the node composition. Furthermore, most overlay networks are symmetric in the sense that they have no special purpose nodes (such as the root of a hierarchy). These two factors make the architecture highly resilient to failures because very little information is lost in the case of a node crash, and this information can be easily replicated on a small number of other nodes (see Section 4.1 for more details).

Unlike event space partitioning, we do not limit the EK-mapping in such a way that each event is mapped to just a single key (or only to keys that the underlying KN-mapping would map to a single node). This generalization alleviates an intrinsic drawback of event space partitioning. It should be also noted that static EK- and SK-mappings make handling dynamic hotspots, i.e., situations when all subscriptions and events fall into a small portion of the subscription/event space, more challenging. We suggest two complementary ways of fighting hotspots: a) by corresponding techniques at the level of KN-mapping; in particular, most overlay networks provide such mechanisms, and b) by providing nearly static EK- and SK-mappings in which infrequent changes may slightly alter the initially defined functions in order to accommodate hotspots. Since the knowledge about these changes would be disseminated very infrequently, it would not have any strong impact on the performance.

4.3 Optimizations

This section elaborates on various optimizations that improve the performance of the proposed architecture. While the first optimization (implementing a multicast primitive) aims at extending the standard interface and functionality of structured overlay networks, the other optimizations are implemented entirely within the CB-pub/sub layer. Our performance results in Section 5 quantify the improvement due to each optimization.

4.3.1 Extending the architecture with multicast

Upon the invocation of a sub(σ) or a pub(e), σ or e have to be propagated to all the rendezvous nodes. Being restricted to use only the unicast send primitive, this is done by a sequence of calls to send(). However, having the CB-pub/sub layer implementing one-to-many send with unicast primitives provided by the overlays suffers from the following inefficencies:

Multiple delivery: a single node can be assigned more than one key by the KN-mapping. However, since the CB-pub/sub layer is unaware of the underlying KN-mapping, it cannot detect that different keys map to the same node without sending probe messages. Thus, simulating multicast by a sequence of unicasts may result in redundant sends and deliveries of the same message.

Non-optimal paths: multiple copies of the message can traverse the same path several times. For example, let us consider a message multicast to two keys k_1 and k_2: it could happen that KN(k_1) is on the routing path toward KN(k_2) so that the message separately sent to k_1 and k_2 will traverse the same path section twice. Since the actual routing path are handled entirely by the overlay network, the CB-pub/sub layer has no means to detect such situations.

It should be noticed that the CB-pub/sub layer may endeavor implementing a multicast routing scheme itself (e.g., a multicast tree of keys). While this could alleviate the above problems, no optimal scheme could be constructed without knowledge of the KN-mapping.

We claim that a generally efficient solution to these problems can only be obtained by building a one-to-many send primitive *within* the overlay network. Specifically, we propose to extend the overlay layer of the basic architecture by providing an additional primitive, namely m-cast().

m-cast() receives a set of keys and a message as parameters and implements a multicast protocol in which every node to which at least one of the specified keys maps will receive the message. Each of such nodes will receive the message at most once. Furthermore, the implementation of m-cast() should also deal with finding an efficient routing path and maintaining a bound on the number of concurrently open connections at a node.

Building the CB-pub/sub functions above the extended overlay becomes significantly less complicated: for each event, notification, or subscription change, the ak-mapping returns the set of keys involved in the operation, and the actual send is performed by a single call to m-cast().

Implementing the dynamic multicast primitive with Chord. [5] shows how a broadcast mechanism may be implemented over Chord. This work uses a single specific addressing (range of keys starting from the sending node) for the sake of illustration. We now show how we imple-

Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05)
1063-6927/05 $20.00 © 2005 **IEEE**

ment a generic multicast primitive (m-cast(M,K) where M is the message to send and K = $\{k_1, \ldots, k_x\}$ is a set of target keys) and analyze its performance.

As described in Section 3.1.1, each node in Chord maintains a set of fingers $\{f_1, \ldots, f_l\}$ with exponentially increasing keys. In addition, each node n knows its successor $n.succ$ and predecessor $n.pred$ in the ring. To simplify the notation, we assume that f_1 is the node's successor in the ring and f_l is its predecessor.

The m-cast primitive is called both by the application that needs to send a message and the Chord deliver primitive upon message reception. The algorithm in Figure 4 piggybacks a (sub)set of target keys on a target message, which we denote as $M.K$.

```
extract−targets(K, n1, n2)
begin
   return {k ∈ K | k ∈ (n1.key, n2.key] on the ring};
end

n.m−cast(M,K)
begin
   K := M.K;
   targets := extract−targets(K,n.pred,n);
   if targets ≠ ∅
      n.deliver(M);
   targets := extract−targets(K,n,n.succ);
   if targets ≠ ∅
   begin
      M.K := targets;
      n.send(M,n.succ.key);
   end
   for each i ∈ [1, l − 1] do
   begin
      targets := extract−targets(K, fi, fi−1);
      if targets ≠ ∅
      begin
         M.K := targets;
         n.send(M, fi.key);
      end
   end
end
```

Figure 4. m-cast **implementation over Chord**

Note that the algorithm sends messages only to node's fingers so that each unicast message is delivered within a single hop. Furthermore, the algorithm preserves the $\log n$ limit on the number of neighbors that each node has to maintain connections with. Also, it is optimal in the sense that no node receives the same message twice.

Since our architecture mostly uses the primitive for sending a subscription to ranges of keys, we consider the message complexity and delivery dilation for the case when K represents a key range. Denote the number of nodes in the $[k_1, k_x]$ interval by $N_{[k_1,k_x]}$. Note that every message in this protocol is sent either to a finger node outside of $[k_1, k_x]$ or to a node that covers $[k_1, k_x]$ and delivers the message to the application. It is easy to see that the number of such finger nodes outside of the range is limited by $\log n$ in the worst case. Thus, the algorithms sends a total of $O(\log n + N_{[k_1,k_x]})$ one-hop messages in the worst case, with the maximal message delivery dilation being $O(\log n)$.

To quantify the benefits of using a multicast primitive that is natively supported by the platform, let us consider the performance of a unicast-based propagation to a range of keys. A unicast-based protocol may send messages conservatively in the following fashion: first, M is sent to k_1. Then, the algorithm operates recursively: each node that receives M being destined for a key k_i forwards M to $k_i + 1 \bmod 2^m$. The recursion stops when $k_i + 1 \bmod 2^m$ goes beyond k_x. Like in the multicast-based protocol, it is guaranteed that no node receives M twice. Therefore, this protocol has the same worst-case message complexity of $O(\log n + N_{[k_1,k_x]})$. However, $O(\log n + N_{[k_1,k_x]})$ is also message dilation of this protocol, which will be intolerable in most practical settings.

On the other hand, if a unicast-based protocol sends M aggressively, i.e., in parallel to all the keys, it will have the same appealing dilation of $O(\log n)$ as the above multicast-based protocol. However, it may send as many as $\Omega(\log n \times x)$ messages, which is clearly unacceptable.[3]

4.3.2 Buffering and collecting notifications

In our basic architecture, when a rendezvous node receives an event, it attempts to match it against each of the stored subscriptions. For each match found, a notification is immediately sent to the subscription source. While this provides a highly responsive system, at the sime time it may result in several short notification messages being sent (the number of events times the number of active subscriptions, in the worst case).

Note that in many cases (e.g., stock tickers, temperature sensors, and in general events produced by a data stream), consecutive events exhibit temporal locality, i.e., have close attribute values. Consequently, they map to the same node or neighbor nodes in a range. To improve performance in these settings, we propose the use of a buffering mechanism: each node accumulates notifications for a given amount of time and sends notifications periodically in batches (all the matches for each subscription are sent in a single message) [7]. Additionally, we introduce a mechanism that gathers notifications in a coordinated way for the case when events are mapped to nodes that lie close on the ring. If a subscription maps to a range of nodes, the middle node of the range serves as agent for this subscription and periodically forwards all collected notifications to the subscription source. Every other node in the range a) periodically sends detected matches to its neighbor that lies closer to the middle of the range, and b) aggregates all the matches it receives from its neighbor that lies farther from the middle of the range. Note that the cost of exchanging notifications between neighbor nodes is amortized across all stored subscriptions so that fewer exchange messages are sent but those messages are longer, which is typically more desirable.

[3] We leave out the details of analysis due to the lack of space.

Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05)
1063-6927/05 $20.00 © 2005 **IEEE**

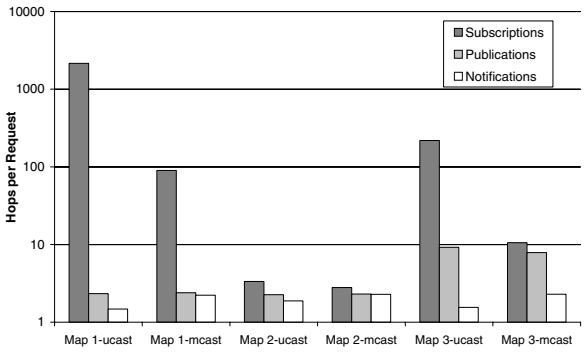


Figure 5. Total Number of Hops

4.3.3 Discretization of mappings

Associating a rendezvous node to each single value in a range may not be a feasible solution: for non-selective constraints, a subscription may be mapped to an excessively large number of nodes.

In order to cope with this problem, we propose a coarser subdivision of the event space by mapping intervals into keys rather than single values so that all the values within the same interval correspond to a single rendezvous. The coarser is discretization, the lower is the number of rendezvous nodes a large range is mapped to, subsequently requiring less hops for propagating a subscription.

The size of discretization intervals should be set considering that the total number of possible intervals (obtained by dividing the total size of event space for the size of the intervals) should be always higher than the number of the nodes in the system. If intervals are too large some nodes may be never considered as rendezvous, causing imbalance in the distribution of subscriptions.

5 Simulations

In this section we presents the results of a simulation study of our system, focusing on the performance comparison between the mappings and on the benefits obtained by using the multicast primitive and the other optimizations we introduced. The results are shown in terms of the following characteristics that are measured under varying conditions: a) the number of one-hop messages sent in the system and b) the number of subscriptions stored on the nodes.

We do not present any comparison with other existing pub/sub systems that are based on a network of brokers because the performance of those systems strongly depends on the broker network topology and distribution of subscriptions across the brokers, which are configured by the users. Hence, a meaningful comparison with our self-organizing system is deemed difficult to achieve.

5.1 Simulation Details

We implemented a simulator of our pub/sub architecture on top of the freely available Chord simulator. We exploited the Chord infrastructure for management of node joins, maintenance of finger tables, and point-to-point unicast primitive. Furthermore, we extended the Chord simulator by implementing the multicast primitive. We used a key space of size 2^{13}.

Experiments are conducted by generating and replaying subscriptions and publications defined over a 4 attribute event space. All attributes are integers, ranging from 0 to the maximum attribute value *ATTR_MAX* of 1,000,000. Each attribute is categorized as selective or non-selective for the purpose of workload generation on a per-experiment basis: each constraint in a subscription spans an independently chosen range that is generated as a random number between 1 and X, wherein X is 3% of *ATTR_MAX* for non-selective attributes and 0.1% for selective ones. For the sake of analyzing the performance of Selective-Attribute, it is important to note that the range of the most restrictive constraint out of the 4 constraints in a subscription is 0.6% of *ATTR_MAX* on average when all attributes are non-selective. Ranges are centered around a value that is chosen randomly following a uniform distribution for non-selective attributes and a Zipf distribution for selective ones. The characteristics of this workload are consistent with those of a real-world pub/sub application studied in [16].

When a subscription is stored at a rendezvous node, we set an expiration time after which the subscription is automatically removed. This simulates possible requests for unsubscriptions, which are common in real applications. Publications are generated with a given probability (that we call *matching probability*) to match at least one subscription. If not specified differently, the parameters used for simulations are as follows: the number of nodes n is 500; message delay is fixed to 50ms; subscriptions are injected at a regular rate of one each 5s, while publications follow a Poisson process with the average of 5s (subscriptions and publications are randomly interleaved); matching probability is 0.5.

Upon $n = 500$, the average number of hops it took the Chord simulator to deliver a single message between a pair of random nodes was about 2.5. This is better than $\log n$ due to the finger caching mechanism; this number showed little variation throughout the experiments.

5.2 Experimental Results

Network Performance. Figure 5 shows the number of hops per request (subscriptions, publications and notifications) obtained when using the three proposed mappings with unicast and multicast. Subscriptions never expire and all attributes are non-selective.

Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05)
1063-6927/05 $20.00 © 2005 **IEEE**

As expected, each publication was mapped to one key in mappings 1 and 2 and to four keys in mapping 3. For the above workload parameters, each subscription was mapped to slightly over one key in mapping 2. The number of mapped keys per subscription was about ten times higher for mapping 1 compared with mapping 3. These figures are directly reflected in the number of hops per subscription when using unicast. When the number of mapped keys is high (as in the case of subscriptions in mappings 1 and 3), using multicast significantly reduces the number of hops, by more than 90% in our experiments.

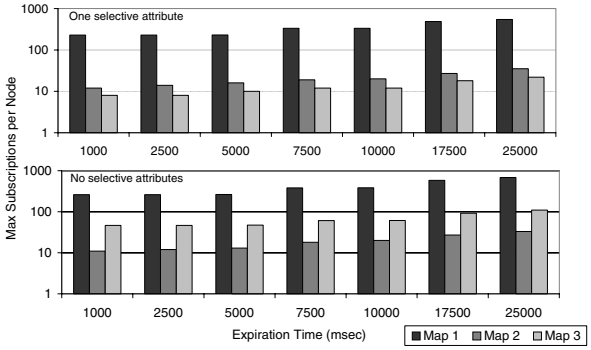


Figure 6. Memory consumption

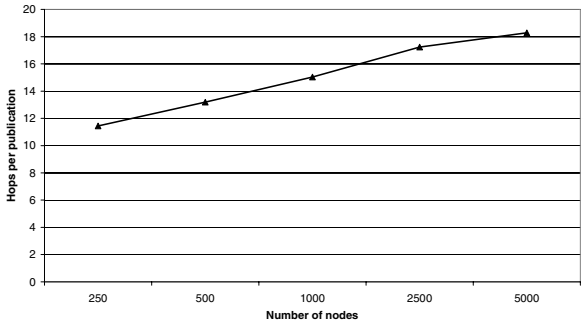


Figure 7. Scalability of bandwidth consumption

Memory consumption. We compared the memory occupation due to subscriptions for each of the three different mappings. In these experiments 25000 subscriptions and no publications were injected in the system. Figure 6 shows the maximum number of subscriptions per node in the system for different values of the expiration time of subscriptions, when zero (below) and one (above) selective attributes are present. The average number of subscriptions per node follows the same trend and it is not shown because of lack of space. Again, mapping 2 appears to have the best overall behavior when no selective attributes are present. However, we point out how mapping 3 can benefit from the presence of one selective attribute.

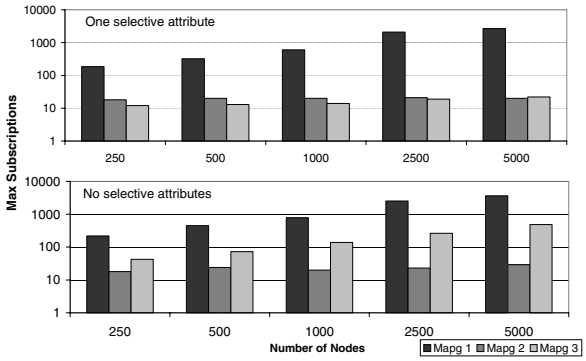


Figure 8. Scalability of memory consumption

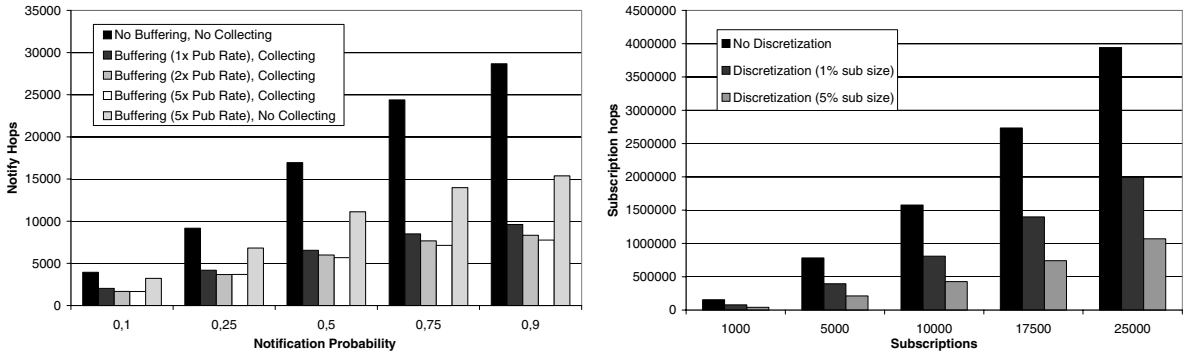
Scalability. Figure 7 shows how the number of hops per publication depends on the number of nodes n. The results are shown for mapping 3 when using unicast. In all cases, the number of hops grows logarithmically with n, which is determined by the basic scalability property of the underlying overlay network.

Figure 8 depicts the maximum number of subscriptions per node when 25000 subscriptions are injected in the system, with zero (below) and one (above) selective attributes. The overall number of subscriptions in the system increases with n because a same range is divided among a higher number of rendezvous, hence the corresponding subscriptions is copied several times. Mappings 1 and 3 are particularly sensitive to this phenomenon, exhibiting poor scalability when no selective attributes are present. On the contrary, when using mapping 2 the average number of subscriptions per node is almost constant. However, these experiments confirm the suitability of mapping 3 when selective subscriptions are present: in this setting, subscription duplication occurs rarely and the growth of the subscriptions per node is limited, allowing mapping 3 to perform better than mapping 2 when the number of nodes is less than 2500.

Optimizations. Figure 9(a) depicts the number of hops required for notifications with different matching probability. The different histograms correspond to different settings for notification buffering and collecting: no buffering and no collecting, buffering plus collecting (with a buffering period set to 1, 2 and 5 times the average publication period), and buffering and no collecting. Both buffering and collecting significantly reduce the number of hops due to notifications, introducing only a delay in the notification itself. However, the experiments show that most of the benefits of this optimization are achieved starting from small buffering periods.

Finally, Figure 9(b) shows the effect of discretization on the number of hops required for issuing subscriptions. The different histograms correspond to a discretization interval whose size is 1 (no discretization), 10% and 20% of the average range size. Both plots refer to mapping 3 with unicast,

Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05)
1063-6927/05 $20.00 © 2005 **IEEE**



(a) (b)

Figure 9. Effects of optimizations

but the same results apply to other mappings with multicast. Again, the positive effect of this optimization for a further reduction of subscription hops is evident.

6 Concluding remarks and future work

In this work we have described a P2P architecture for the content-based pub/sub paradigm, which relies on overlay network technologies. In particular, we described three different methods for mapping pub/sub subscriptions and events to overlay keys, analyzing their benefits and drawbacks. To the best of our knowledge, the proposed solution is unique in its self-configuration abilities and adaptiveness to dynamic changes.

In order to increase the efficiency of the proposed solution, we have proposed to enrich the existing overlay networks with one-to-many primitives, as well as to extend our infrastructure with notification buffering and range discretization capabilities. We have illustrated the feasibility and scalability of our approach with experimental results obtained from simulations.

References

[1] R. Baldoni, M. Contenti, and A. Virgillito. The Evolution of Publish/Subscribe Systems. In *Future Trends in Distributed Computing, Research and Position Papers*, volume 2584. Springer, 2003.

[2] 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.

[3] M. Castro, P. Druschel, A. 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.

[4] G. Cugola, E. D. Nitto, and A. Fuggetta. Exploiting an event-based infrastructure to develop complex distributed systems. In *Proceedings of ICSE '98*, April 1998.

[5] S. El-Ansary, L. Alima, P. Brand, and S. Haridi. Efficient Broadcast in Structured P2P Networks. In *Proc. IPTPS*, 2003.

[6] F. Fabret, A. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe. In *Proceedings of SIGMOD 2001*.

[7] R. Friedman and R. van Renesse. Packing Messages as a Tool for Boosting the Performance of Total Ordering Protocols. In *Proceedings of HPDC 1997*, 1997.

[8] Gryphon Web Site. http://www.research.ibm.com/gryphon/.

[9] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: Content-Based Publish/Subscribe over P2P Networks. In *Proceedings of Middleware 2004*, 2004.

[10] P. Pietzuch and J. Bacon. Peer-to-peer overlay broker networks in an event-based middleware. In *Proc. DEBS*, 2003.

[11] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. *LNCS*, 2233:14–34, 2001.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of Middleware '01*, 2001.

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of SIGCOMM*, 2001.

[14] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *Proceedings of DEBS'03*, 2003.

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

[16] 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 *DISC'02*, 2002.

[17] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 2003.

[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 *Int. Workshop on Network and OS Support for Digital Audio and Video*, 2001.

Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05)
1063-6927/05 $20.00 © 2005 **IEEE**