A Scalable Protocol for Content-Based Routing in Overlay Networks

R. Chand, P.A. Felber
Institut EURECOM
06904 Sophia Antipolis, France
{chand|felber}@eurecom.fr

Abstract

In content networks, messages are routed on the basis of their content and the interests (subscriptions) of the message consumers. This form of routing offers an interesting alternative to unicast or multicast communication in loosely-coupled distributed systems with large number of consumers, with diverse interests, wide geographical dispersion, and heterogeneous resources (e.g., CPU, bandwidth). In this paper, we propose a novel protocol for content-based routing in overlay networks. This protocol guarantees perfect *routing (i.e., a message is received by all, and only those, consumers that have registered a matching subscription) and optimizes the usage of the network bandwidth. Furthermore, our protocol takes advantage of* subscription aggregation *to dramatically reduce the size of the routing tables, and it fully supports dynamic subscription registrations and cancellations without impacting the routing accuracy. We have implemented this protocol in the application-level routers of an overlay network to build a scalable XML-based data dissemination system. Experimental evaluation shows that the size of the routing tables remains small, even with very large populations of consumers.*

1 Introduction

Content-based routing differs significantly from traditional unicast and multicast communication, in that messages are routed on the basis of their content rather than the IP address of their destination. This form of addressing is widely used in event notification or publish/subscribe systems [12] to deliver relevant data to the consumers, according to the interests they have expressed. By allowing consumers to define the type of messages they are interested in, data producers do not need to keep track of the consumer population and can simply inject messages in the network. In turn, consumers with scarce resources (e.g., mobile devices with limited bandwidth) can restrict the type and amount of data that they receive by registering highly-selective subscriptions, and hence limit their incoming network traffic. The complex task of filtering and routing messages is left to the network infrastructure, which consists typically of application-level routers organized in an overlay network.

In order to route messages to all, and only those, consumers that have registered a matching subscription, the distributed routers of a content-based network must keep track of the consumers' subscriptions in their routing table.With large numbers of consumers, the size of the routing tables can quickly become a bottleneck, as each router must match each incoming message against the subscriptions of its routing table at "wire speed" and the filtering speed is highly dependent of the number of subscriptions. It is thus of paramount importance for a scalable content-based network to incorporate a space- and bandwidth-efficient routing protocol, and highly-efficient filtering mechanisms.

In the paper, we present the XRoute content-based routing protocol that we have designed for our XNet XML-based data dissemination system [7]. Although XNet uses XML as data format and XPath as subscription language, our routing protocol can be readily applied to other subscription models, including simple IP prefixes. The protocol implements *perfect* routing, i.e., a message is routed only to the consumers that have registered a matching subscription, and to all of them. It takes advantage of subscription similarities to "aggregate" them in the routing tables, and hence minimize the space requirements and increase the filtering speed at the routers. Furthermore, the protocol allows consumers to register new subscriptions, and cancel them, at any time without impacting the routing accuracy. To the best of our knowledge, this is the first content-based routing protocol that takes advantage of subscription aggregation *and* fully supports subscription cancellations. Experimental evaluation demonstrates that subscription aggregation is effective and dramatically reduces the size of the routing tables.

2 Related Work

Selective event dissemination can be achieved by various means. The simplest approach, called *flooding*, consists in broadcasting events and filtering out unwanted data at the consumer (or at the consumer's local content router). This approach can quickly lead to network saturation. Alternatively, routers can be configured to match published events against all subscriptions and compute a destination list used to route events. This approach, called *match-first*, increases the space requirements and the filtering time at the routers, and does not scale well to large numbers of subscriptions. These two approaches are generally not classified as "content-based routing" because data is routed to all nodes in the first case, and according to a pre-computed list of addresses in the second case.

Several publish/subscribe systems implement some form of content-based routing (see [12] for a survey). Elvin [14] is architectured around a single server that filters and forwards producer messages directly to consumers, thus alleviating the need for a real content-based routing protocol. The authors mention a distributed extension of Elvin, but do not discuss how they plan to achieve distributed content routing.

IBM Gryphon [2] uses a set of networked brokers to distribute events from publishers to consumers. It uses a distributed filtering algorithm based on parallel search trees maintained on each of the brokers to efficiently determine where to route the messages. To construct or to

update the parallel search trees, each broker must have a copy of *all* the subscriptions in the system, which makes this approach unpractical with large number of subscriptions or when subscriptions are frequently registered and canceled.

Siena [4] also uses a network of event servers for content-based event distribution. The routing protocol of Siena [5] is most similar to ours. Each event server maintains a routing table that holds a subset of the subscriptions, and the associated subscribers and neighbor routers. Messages are matched against each subscription and forwarded along the paths corresponding to matching subscriptions. However, Siena's routing protocol does not support subscription cancellation (cancellations in Siena would degrade routing accuracy, and the system could eventually degenerate into a flooding approach). In addition, we could not determine the space- and time-efficiency of the protocol, and whether it can be extended to support more general subscription languages.

Jedi [10] proposes several variations for event routing among its networked event servers, including the *flooding* and *match-first* approaches. With the *hierarchical* approach, event servers are organized in an (arbitrary) tree; subscriptions are propagated upward the tree, and messages are propagated both upward and downward to the children that have matching subscriptions. This approach may lead to very large routing tables at the root of the tree, and unnecessary propagation of events upward the tree.

In [16], the authors propose an approach for content-based routing of XML data in mesh-based overlay networks. They introduce a routing protocol that reassembles data streams sent over multiple redundant paths to tolerate some node or link failures. The focus of this work is on reliable delivery of streaming data, and does not explicitly address subscription management.

In [15], the authors propose to add content-based routers at specific nodes of an IP multicast tree to reduce network bandwidth usage and delivery delays. They propose algorithms for determining the optimal placement of a given number of content routers. The routing protocol merely consists of propagating subscriptions upward the tree, until they reach the producer or are subsumed by other subscriptions. Subscription cancellation is not supported.

Note that, in this paper, we focus on the routing of messages in an overlay network, and we do not explicitly address the issue of efficiently matching the messages against subscriptions. This problem has been widely studied elsewhere (e.g., in [1, 13, 3, 7, 11]).

3 System Model and Definitions

Our protocol routes messages (or events) through the nodes of an overlay network, according to the messages' content and the subscriptions registered by the consumers. Each node of the overlay network acts as a content-based router. Each data consumer and producer is connected to some node in the network; we call such nodes *consumer* and *producer* nodes. To simplify the presentation, we assume that consumer and producer nodes are distinct, i.e., one cannot directly connect both a producer and a consumer to the same router node. Nodes that have no consumer or producer are *inner* nodes. A sample network topology is shown in Figure 4.

We assume that all routers know their neighbors, as well as the best paths that lead to each producer. We also assume that the number and location of the producer nodes is known. In contrast, the consumer population does not need to be known a priory.

Nodes communicate with their neighbors using reliable point-to-point transport such as TCP, and we assume that nodes and links do not fail. Each node has a set of *links*, or *interfaces*, that connects the node to its direct neighbors. We assume that there exists exactly one interface per neighbor (we ignore redundant links connecting two neighbors). For a given producer, we will generally denote by I_{up}, or *upstream interfaces*, the interfaces along the path up to the producer, and I_{down}, or *downstream interfaces*, the other interfaces (along the paths to the consumers). In general, we will discuss the properties and behavior of our protocol in the case of a single producer; it can be, however, trivially extended to the case of multiple producers.

The actual consumers are connected to consumer nodes via links that are not part of the overlay network, and therefore not associated with any of the node's interface. Furthermore, to simplify the presentation of the protocol, we assume that consumer nodes are edge routers with a single interface that connects them to the overlay network (this property can always be satisfied by introducing virtual consumer nodes at the edges of the overlay). Consumers register and cancel subscriptions via their consumer nodes. A consumer cannot cancel a subscription that it did not previously register (the consumer node will filter out such requests).

Consumer interests are expressed using a subscription language. Subscriptions allow to specify predicates on the set of valid events for a given consumer. Our XNet system was designed to use a significant subset of the XPath language [17] to specify complex, tree-structured subscriptions, and the XTrie filtering algorithm [7] for efficient matching of events against large number of subscriptions. However, our routing protocol can be used with any subscription language.

We say that a subscription S_1 *covers* another subscription S_2, denoted by $S_1 \supseteq S_2$, iff any event matching S_2 also matches S_1, i.e., $matches(S_2) \Rightarrow matches(S_1)$. The covering relationship defines a partial order on the set of all subscriptions. For XPath expressions, we have shown in [6] that covering relationships can be evaluated in $O(nm)$ time, where n and m are the number of nodes of the two expressions being compared.

4 Overview of the Protocol

Goals. Our routing protocol has been designed to achieve several goals. First, it should lead to *perfect* routing of data in the network, i.e., when an event is published, all the consumers that are interested in that event, and only those, must receive it. Second, routing should ideally be *optimal*, i.e., the link cost of routing an event should be no more than that of sending the event along a multicast tree spanning all the consumers interested in the event.

Third, the protocol should take advantage of subscription *aggregation* to minimize space and processing requirements at the nodes. Informally, subscription aggregation is a mechanism that enables us to reduce the size of the routing tables by detecting and eliminating subscription redundancies; it is a key technique to scale to very large populations of consumers in a publish/subscribe system.

Finally, the protocol should be efficient and allow consumers to register and cancel subscriptions at any time.

In particular, canceling a subscription should leave the system in the same state as if the subscription were not registered in the first place.

Routing. Routing works in a distributed manner. Each node N in the network contains in its routing table a set of entries that represent the subscriptions that its neighbor nodes are interested in. For each subscription S, node N maintains some information in its routing table in the form "if match S, send to $N_1, N_2, \ldots$". In other words, node N knows which neighbor nodes it must forward an event to, if that event matches S. When a node is a consumer node, it knows the consumers which are interested in receiving events matching S. The process starts when a publisher produces an event at its publisher node and ends when all consumer nodes that are interested in that event have received it.

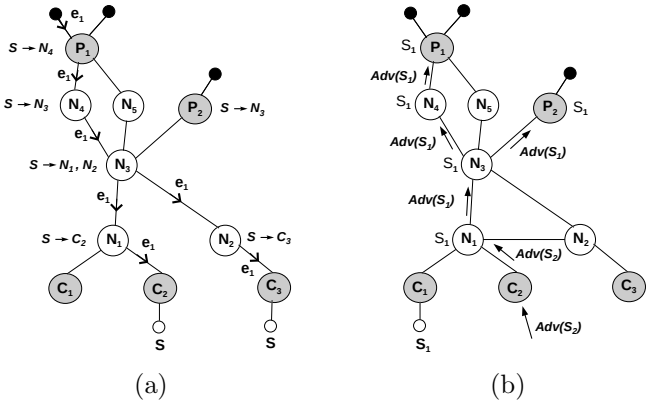

Figure 1: (a) A sample publish/subscribe network. Subscriptions are represented underneath the consumers that registered them, and routing table entries are listed next to the node they are associated with. (b) Subscription advertisements are propagated upward from the consumers to the publishers. They may be transformed along the propagation paths due to aggregation (here, we have $S_1 \supseteq S_2$).

Example 1 *Consider the network in Figure 1(a), with two publisher nodes P_1 and P_2, and three consumer nodes C_1, C_2, and C_3. The other nodes N_1, N_2, N_3, N_4, and N_5 are internal nodes. Nodes C_2 and C_3 have consumers interested in receiving events matching subscription S. Suppose that e_1, an event matching subscription S, is published at node P_1. Event e_1 will follow the path highlighted by the arrows.*

Principle of the Algorithm. When some consumer registers or cancels a subscription, the nodes of the overlay must update their routing tables accordingly; to do so, they exchange pieces of information that we call *subscription advertisements*, or simply *advertisements*. An advertisement carries a subscription, and corresponds either to a registration or a cancellation. From the point of view of node N, an advertisement for subscription S received from a neighbor node N' indicates that a consumer at N' or downstream from N' has registered or canceled subscription S. The subscription algorithm works by propagating advertisements recursively across the overlay, from the consumers toward the producers, following the best path (see Section 3). Note that subscriptions may be transformed along the propagation path due to aggregation, i.e., a subscription received as part of an incoming advertisement may be different from the subscription carried by the resulting outgoing advertisement.

The general principle of the algorithm is shown in Algorithm 1, and illustrated in Figure 1(b). The algorithm starts when a consumer registers or cancels a subscription S. It builds an advertisement corresponding to this subscription and sends it to its consumer node C. The algorithm ends when the publisher node has been reached. When a subscription should be registered by multiple producers, the advertisements are sent along the paths to each of the producers.

Algorithm 1 Sketch of the protocol at node N

```
1: when receive adv(S) from N' via interface I_down
2:    update routing table
3:    generate outgoing advertisement adv(S')
4:    send adv(S') via I_up upward to the producer
5: end when
```

Subscription Aggregation. Subscription aggregation is a key technique that allows us to minimize the size of the routing tables by eliminating redundancies between subscriptions, and consequently to improve the routing performance.

Consider the situation illustrated in Figure 1(b). At node N_1, two subscriptions S_1 and S_2 were advertised by consumer nodes C_1 and C_2, respectively. From the point of view of node N_3, this means that some consumers downstream N_1 are interested in receiving events matching S_1 or S_2. Now, assume that $S_1 \supseteq S_2$, that is, any event matching S_2 also matches S_1. The mechanism of subscription aggregation is based on the following observation: when an event e arrives at node N_3, it is only necessary to test e against S_1, because, by definition, any event matching S_2 also matches S_1, and any event that does not match S_1 does not match S_2 either.[1] Because of that property, S_2 becomes redundant and can be "aggregated" with S_1 (in particular, S_2 does not need to be propagated upstream from N_1 to N_3).

We distinguish between two forms of subscription aggregation. If S_1 and S_2 are registered through the same interface I^k (e.g., at Node N_3 in Figure 1(b)), we say that S_2 is *represented by* S_1 at interface I^k. If they are not registered through the same interface, we say that S_2 is *substituted by* S_1 (e.g., at Node N_1 in Figure 1(b)). In both situations, only S_1 is advertised upstream.

5 The Subscription Algorithm

In this section, we formally present our content-based routing protocol.

5.1 Data Formats

Routing Tables. Each node N maintains a routing table that consists of a set of entries. Each entry corresponds to one *distinct* subscription (two identical subscriptions share the same entry). We will write $entry(S)$ to refer to the entry corresponding to subscription S. It maintains information about all the registrations for subscription S that have been received by node N. More precisely, the information in $entry(S)$ represents N's view of its neighbor's interests in subscription S. Moreover, $entry(S)$ also contains the information required to implement the aggregation principle introduced in Section 4.

An entry $entry(S)$ in the routing table of node N has the following format:

$$S \; ; \; (T_S^1, \cdots, T_S^n) \; ; \; R_S \; ; \; Ptr_S$$

[1] An IP networking analogy would be that of network prefixes, where S_1 is a prefix of S_2.

where S is the subscription and n is the number of interfaces of node N. T_S^k represents the population of consumers downstream interface I^k that are interested in events matching S. Each T_S^k consists of a set of three integers that we will refer to as $T_S^k.x$, $T_S^k.y$, and $T_S^k.z$ (to be described shortly). $\overline{T_S^k}$ is defined by $T_S^k.x + T_S^k.y + T_S^k.z$ and is always greater than or equal to 0 (it is strictly greater than 0 iff there are consumers downstream interface I^k interested in receiving events matching S). Finally, R_S represents the total number of subscriptions that have been "aggregated" in S (either through representation or substitution), and Ptr_S, if non-null, points to another entry in the routing table that S is substituted by.

The sum of $T_S^k.x$ and $T_S^k.y$ represents the population of subscriptions S downstream interface I^k, i.e., the number of consumers interested in receiving events matching S (the distinction between $T_S^k.x$ and $T_S^k.y$ will be discussed later). $T_S^k.z$ corresponds to the number of subscriptions "aggregated" in S (either through representation or substitution) downstream interface I^k.

Advertisements. As mentioned previously, advertisement messages are exchanged between routers to register or cancel a particular subscription. From the point of view of node N, receiving an advertisement message $adv(S)$ from interface I^k means that a change about the population of subscriptions S has occurred downstream interface I^k. Node N must update its routing table to take this change into account; in particular, T_S^k needs to be updated. N also needs to generate and send an advertisement to the upstream neighbor node.

An advertisement message $adv(S)$ is a sequence of triples with the following format:

$$S\ ;\ n_S\ ;\ r_S$$

where S is the subscription advertised, and n_S is the number of times S should be registered ($n_S > 0$) or canceled ($n_S < 0$). r_S represents the number of subscriptions, distinct from S, that have been substituted by S downstream I^k, and that should be registered ($r_S > 0$) or canceled ($r_S < 0$) at node N. Finally, $adv(S)$ may contain additional triples, with the same format, indicating additional modifications to perform to the routing tables upstream.

Events. Events are messages whose content can be matched against consumer subscriptions. In our XNET system, events are formatted as XML documents. Once the routing table have been populated, routing an event is a trivial task. When node N receives event e sent by producer P from interface I_{up}, it matches e against the subscriptions in his routing table (in our system, efficient matching is implemented using the algorithms presented in [7]). For each interface I_{down}^k such that there is at least one subscription S with $\overline{T_S^k} > 0$, node N propagates e downstream that interface. Note that there cannot be cycles because each node always receives events through its I_{up} interface located on the best path (see Section 3) from the producer to the node, and never propagates them along that path.

5.2 Representation and Substitution

Before describing the subscription algorithm, we need to describe more formally the representation and substitution relations, and how they are implemented.

Definition 1 (Representation) *Consider entries for subscriptions S_1 and S_2 at non-consumer node N such that $S_1 \supset S_2$, $\overline{T_{S_1}^k} > 0$ and $\overline{T_{S_2}^k} > 0$, then S_2 must be* represented by S_1 *at interface I^k. This operation consists in modifying their entries as follows:*

1. $T_{S_1}^k.z \leftarrow T_{S_1}^k.z + \overline{T_{S_2}^k}$
2. $R_{S_1} \leftarrow R_{S_1} + \overline{T_{S_2}^k}$
3. $R_{S_2} \leftarrow R_{S_2} - T_{S_2}^k.z$
4. $\overline{T_{S_2}^k} \leftarrow 0$

Thereafter, we say that S_2 is represented by S_1 at interface I^k.

The representation operation implements the subscription aggregation mechanism introduced in Section 4. Indeed, having both $\overline{T_{S_1}^k}$ and $\overline{T_{S_2}^k}$ greater than zero is redundant, because it is not necessary to test an event against S_2 to know if it has to be forwarded down that interface. Therefore, when S_2 has been represented by S_1 at interface I^k, $\overline{T_{S_2}^k}$ becomes null, which is equivalent to say that no client is interested in receiving events matching S_2 downstream interface I^k.

Note that if some subscriptions were previously represented by S_2 at interface I^k, they now become represented by S_1 at I^k. Indeed, $\overline{T_{S_2}^k}$ represents the sum of the instances of S_2 registered at I^k *and* all the subscriptions that are represented by S_2 at I^k. At the time S_2 is represented by S_1 at I^k, S_1 takes control of all instances of S_2 and all the subscriptions that it represents (steps 1 and 2 in Definition 1), and S_2 loses control of the subscriptions it used to represent (steps 3 and 4).

Definition 2 (Substitution) *Consider entries for subscriptions S_1 and S_2 at node N such that: $S_1 \supset S_2$, $Ptr_{S_1} = null$, and $Ptr_{S_2} = null$. Then S_2 must be* substituted by S_1. *This operation consists in modifying their entries as follows:*

1. $Ptr_{S_2} \leftarrow S_1$
2. $R_{S_1} \leftarrow R_{S_1} + \sum_{k \le n} T_{S_2}^k.x + R_{S_2}$

Thereafter, we say that S_2 has been substituted by S_1, and S_2 must subsequently be advertised by S_1, i.e., any incoming advertisement $(S_2; n; r)$ yields an outgoing advertisement $(S_1; 0; n + r)$. Note that a subscription may be substituted by only one other subscription.

The signification of a substitution operation can be understood by observing the following scenario. Suppose that the conditions for substituting S_2 by S_1 are met, but we do not perform the substitution operation. If an incoming advertisement for S_2 (registering n_{S_2} subscriptions) arrives at node N, the outgoing advertisement sent to the upstream neighbor node N' at interface I^j will be $adv_{up}(S_2)$. Then, S_2 will be represented by S_1 at interface I^j of N'. Thus, by substituting S_2 by S_1 at node N, we anticipate this representation. The outgoing advertisement advertises S_1 and specifies that S_1 is to represent n_{S_2} additional subscriptions at interface I^j.

Although it adds some complexity to the protocol, the subscription substitution mechanism is *necessary* to guarantee perfect routing when canceling a subscription that acts as a substitute for some other subscriptions. In addition, it can help save bandwidth by propagating smaller advertisements.

Note that there may be multiple substitution relations between subscriptions. That is, subscription S can be

substituted by S', which is in turn substituted by S'', etc. We call such a sequence a *substitution chain*. For any subscription S_i, we denote by $h(S_i)$ the subscription at the top of the chain, i.e., the subscription S with $Ptr_S = null$. We denote by $tree(S)$ the set of all the subscriptions S_j that have been substituted, directly or indirectly, by S (including S). Figure 2 shows a subscription tree, where links represent substitutions (the child is substituted by its parent). For instance, $tree(S_1)$ contains all subscriptions, $tree(S_3)$ contains S_3, S_4, and S_5, and $tree(S_5)$ only contains S_5.

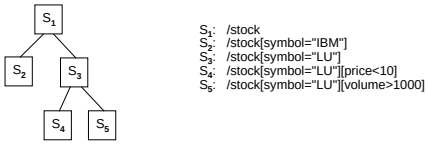

Figure 2: The substitution relations apply recursively. Subscriptions can be organized in a tree, where a link indicates that a child is substituted by its parent.

A substitution operation can only be performed between two subscriptions if none of them has already been substituted, in other words between two tops of chains. Let S_1 and S_2 be two such subscriptions. When S_2 is substituted by S_1, R_{S_1} is incremented by $\sum_{k \leq n} T^k_{S_2}.x$, which represents the number of subscriptions S_2 (step 2 in Definition 2). Indeed, as S_2 was not substituted before, $T^k_{S_2}.y = 0$ for all k. Besides, R_{S_1} is also incremented by R_{S_2}, which represents the number of subscriptions that are represented by S_2 at all interfaces, plus the ones that have been substituted by S_2, if any. Thus, recursively, R_{S_1} represents all the subscriptions in $tree(S_1)$, plus those that are represented by any of them. This is true for any subscription.

5.3 Protocol Description

Updating the routing table constitutes the main task of the subscription algorithm. The table must be updated at node N each time an advertisement for a subscription S arrives from an interface I^k, i.e., when a change has occurred in the population of the subscriptions S downstream interface I^k. The routing table at node N must be updated so that its entries are accurate enough to enable perfect routing. Moreover, the algorithm must make full use of subscription aggregation at all times. The details of the algorithm are given in Algorithms 2, 3, and 4, and described in the rest of this section.

Algorithm 2 — Routing Table Update

```
1:  if Ptr_S ≠ null then
2:    T^k_S.y ← T^k_S.y + n_S
3:    for all S' ancestor of S in tree(h(S)) do
4:      R_S' ← R_S' + n_S + r_S
5:    end for
6:    adv_out ← (h(S); 0; n_S + r_S)
7:  else
8:    T^k_S.x ← T^k_S.x + n_S
9:    adv_out ← (S; n_S; r_S)
10: end if
11: T^k_S.z ← T^k_S.z + r_S
12: R_S ← R_S + r_S
```

When an advertisement for a subscription S arrives at interface I^k of node N, we first update T^k_S. Then we try to establish some relations with the other subscriptions in the routing table, if possible. We now identify and discuss the various situations that may occur.

Establishing Subscription Relations. First we consider the following two properties (proofs in [8]):

Property 1 *When an advertisement for the registration of subscription S arrives from node N' at interface I^k of node N, S* cannot *be represented by any subscription at that interface.*

Property 2 *When an advertisement for the registration of subscription S_2 arrives at node N and S_2 can be substituted by another subscription S_1, then no subscription can be substituted by S_2.*

Now consider an advertisement $adv(S)$ for subscription S arriving at interface I^k of node N. If that advertisement corresponds to a subscription cancellation, it means that a registration advertisement for S has been received earlier at interface I^k (consumers cannot cancel subscriptions that they have not previously registered). Otherwise, if $entry(S)$ exists and is such that $\overline{T^k_S} > 0$, then some advertisement for the registration of S has been received earlier at interface I^k. In both situations, the possible aggregation (representation or substitution) relations between S and the other subscriptions have already been established.

Thus, we will only try to establish some relations when (i) $adv(S)$ corresponds to a registration and (ii) there is no entry for S or $entry(S)$ is such that $\overline{T^k_S} = 0$. Moreover, if S has an entry in the routing table, then some advertisement for the registration of S has been received earlier and substitution relations have already been established. We therefore try to build the following two relations when conditions (i) and (ii) above are met.

First, if there is no entry for S in the routing table, we try to substitute S by another subscription. If that is possible, then according to property 2, no other subscription can be substituted by S (lines $2-3$ in Algorithm 4). Otherwise, we try to substitute other subscriptions by S (lines $5-7$ in Algorithm 4).

Second, we try to represent other subscriptions by S at interface I^k (Algorithm 3). Recall that, according to property 1, S cannot be represented by another subscription.

Establishing the aggregation relations between S and the other subscriptions in the routing table may require modifying existing relations. We now identify these cases.

Modifying Subscriptions Relations. Consider an advertisement for the registration of subscription S arriving at interface I^k of node N, and suppose that we have $\overline{T^k_S} = 0$. A subscription can only have one substitution relation. Thus, establishing a substitution relation between S and some other subscriptions does not require extra modifications to be performed to the routing table.

On the other hand, a subscription can have multiple representation relations with other subscriptions. Consider the case where a subscription S_j is to be represented by S at interface I^k. There are $T_j = \overline{T^k_{S_j}}$ instances of subscription S_j. We have two cases:

First case: $S_j \in tree(S)$. The T_j instances of subscription S_j are now represented by S. For each subscription S_k ancestor of S_j in $tree(S)$, the T_j instances of subscription S_j are no longer substituted by S_k. Thus subscription S_k must have its R field decremented by T_j (lines $6-8$ in Algorithm 3). However, the subscriptions ancestor of S in $tree(h(S))$ (if any) are still a substitute

for the T_j instances of subscription S_j, and do not need to have their entry modified.

Second case: $S_j \notin tree(S)$. Then the T_j instances of subscription S_j (that are now represented by S at I^k) also have for substitutes every subscription ancestor of S in $tree(h(S))$ (if any). Thus those subscriptions must have their R field incremented by T_j (lines $23-25$ in Algorithm 3).

Algorithm 3 — Subscription Representation

```
1: declare A = 0
2: for all S_j subscriptions that can be represented by S at I^k do
3:    declare T_j = overline(T^k_{S_j})
4:    Represent S_j by S at I^k
5:    if S_j ∈ tree(S) then
6:       for all S_k ancestor of S_j in tree(S) do
7:          R_{S_k} ← R_{S_k} − T_j
8:       end for
9:    else
10:      if S_j ∈ tree(h(S)) then
11:         for all S_k ancestor of S_j in tree(h(S)) do
12:            R_{S_k} ← R_{S_k} − T_j
13:         end for
14:      else
15:         for all S_k ancestor of S_j in tree(h(S_j)) do
16:            R_{S_k} ← R_{S_k} − T_j
17:         end for
18:         if S_j ≠ h(S_j) then
19:            append (h(S_j); 0; −T_j) to adv_out
20:            A ← A + T_j
21:         end if
22:      end if
23:      for all S_k ancestor of S in tree(h(S)) do
24:         R_{S_k} ← R_{S_k} + T_j
25:      end for
26:   end if
27:   remove entry(S_j) if Σ_{p≤n} overline(T^p_{S_j}) = 0
28: end for
29: for all S_k ancestor of S in tree(h(S)) do
30:    R_{S_k} ← R_{S_k} + n_S + r_S
31: end for
32: R_S ← R_S + r_S
33: T^k_S.z ← T^k_S.z + r_S
34: if h(S) ≠ null then
35:    T^k_S.y ← T^k_S.y + n_S
36:    adv_out ← (h(S); 0; n_S + r_S + A) [+ appended triples]
37: else
38:    T^k_S.x ← T^k_S.x + n_S
39:    adv_out ← (S; n_S; r_S + A) [+ appended triples]
40: end if
```

Then, if S_j belongs to $tree(h(S))$, all subscriptions ancestor of S_j in $tree(h(S))$ (if any) must have their R field decremented by T_j (lines $11-13$ in Algorithm 3).

On the other hand, if S_j does not belong to $tree(h(S))$, then all the subscriptions ancestor of S_j in $tree(h(S_j))$ must have their R field decremented by T_j (lines $15-17$ in Algorithm 3). In addition, we necessarily have $h(S_j) \neq S_j$ (otherwise, S_j would have been substituted by S). Then, at the incoming interface of the upstream neighbor node, the T_j instances of subscription S_j are represented by subscription $h(S_j)$. This is incompatible with the fact that those T_j instances are now represented by S at node N. Thus, we must indicate that $h(S_j)$ should represent T_j fewer instances of subscription S_j at that node, whereas $h(S)$, should represent T_j additional instances of S_j. This information is appended to the outgoing advertisement in the form of two additional triples $(h(S); 0; T_j)$ and $(h(S_j); 0; -T_j)$ (lines $19-20, 36$ in Algorithm 3).

Dealing with Registrations. In this section, we detail the routing table updates performed by a node N when it receives from downstream interface I^k a *registration* advertisement for a subscription S: $(S; n_S, r_S)$. The process is different according to the value of $entry(S)$ in the routing table.

First case: $entry(S)$ exists and $\overline{T^k_S} > 0$ (Algorithm 2). As previously mentioned, no new relations can be established. All we have to do is to update T^k_S and R_S, as well as the entries of the subscriptions ancestor of S in $tree(h(S))$.

Algorithm 4 — Subscription Substitution

```
1: create a null entry(S)
2: if ∃S', S' ⊃ S, Ptr_{S'} = null then
3:    substitute S by S'
4: else
5:    for all S_k that can be substituted by S do
6:       substitute S_k by S
7:    end for
8: end if
9: call algorithm 3: "Subscription Representation".
```

Second case: $entry(S)$ exists and $\overline{T^k_S} = 0$ (Algorithm 3). We have to look for all the subscriptions that can be represented by S at interface I^k. We must also modify the existing relations and include those modifications in the outgoing advertisement, if necessary. When this is done, we update T^k_S and R_S, as well as the entries of the subscriptions ancestor of S in $tree(h(S))$ (lines $29-40$).

Third case: $entry(S)$ does not exist (Algorithm 4). We try to substitute S by another subscription that is not substituted (lines $2-3$). If that is possible, then we look for other subscriptions that can be substituted by S (lines $5-7$). When this is done, we are in the second case and we apply Algorithm 3.

Additional updates: The incoming advertisement may contain additional triples $(S'; 0; U)$. These triples are generated by Algorithm 3 (lines 19) at downstream neighbor node and are such that $U < 0$ and $Ptr_{S'} = null$. We are thus in the case where $entry(S')$ exists and $\overline{T^k_{S'}} > 0$, and we can apply algorithm 2 for each S'.

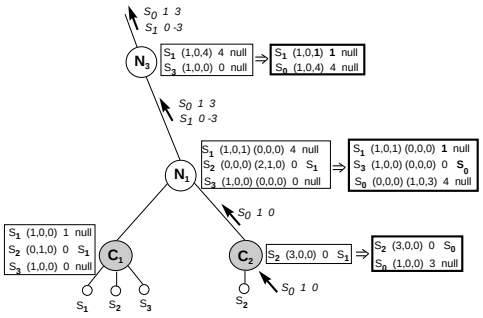

Figure 3: Example of the subscription algorithm. Registered subscriptions are represented below their corresponding client nodes. Routing tables (shown next to the nodes) are updated as a result of the registration of subscription S_0 (updated tables are shown with a thick frame). Here, we have $S_0 \supseteq S_2$, $S_1 \supseteq S_2$, and $S_1 \supseteq S_3$. There are no relationships between S_0 and S_1, and between S_2 and S_3.

Example 2 *Figure 3 illustrates the operation of the subscription algorithm on the publish/subscribe network of Figure 1(a). Four consumers have already registered some subscriptions. A consumer at client node C_2 registers subscription S_0, resulting in updates of the routing table at each node on the path from C_2 to each publisher. For the sake of clarity, we have only represented inner nodes N_1 and N_3.*

At nodes C_2, N_1, and N_3, entry(S_0) does not exist. Thus, algorithm 4 (which in turn calls algorithm 3) is called to update the routing table. The following relations are established: At node C_2, S_2 is substituted by S_0. At node N_1, S_3 is substituted by S_0, S_2 is represented by S_0 at the downstream interface to C_2, and entry(S_2) is removed. At node N_3, S_3 is represented by S_0 at the downstream interface to N_1 and its entry is removed.

Dealing with Cancellations. The cancellation algorithm is formally described in [8].

6 Protocol Evaluation

To test the effectiveness of our content-based routing protocol, we have conducted simulations using real-life document types and large numbers of subscriptions.

Simulation Setup. We have generated a network topology using the transit-stub model of the Georgia Tech Internetwork Topology Models package [18]. The resulting network topology, shown in Figure 4, contains 64 routers. We then added 24 consumers at the edges of the network and a single producer.

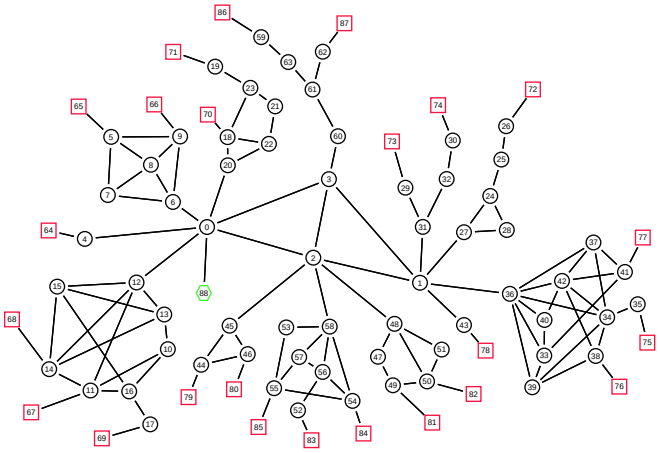

Figure 4: Simulated network topology with 64 routers (circles), 24 consumers (boxes), and 1 producer (hexagon).

We have simulated consumer load by registering subscriptions at the consumer nodes. The subscriptions were expressed using the XPath language [17]. To generate the set of XPath expressions, we have developed an XPath generator (described in [7]) that takes a Document Type Descriptor (DTD) as input and creates a set of valid XPath expressions based on a set of parameters that control: (1) the maximum height h of the tree patterns; (2) the probabilities p_* and $p_{//}$ of having a "$*$" or a "//" wildcard operator at a node of a tree pattern; (3) the probability p_λ of having more than one child at a given node; and (4) the skew θ of the Zipf distribution used for selecting element tag names. For our experiments, we have generated sets of tree patterns of various sizes, with $h = 10$, $p_* = p_{//} = 0.1$, $p_\lambda = 0.1$, and $\theta = 1$.

We have used the NITF (News Industry Text Format) DTD [9] to generate our sets of XPath expressions. The NITF DTD, which was developed as a joint standard by news organizations and vendors worldwide, is supported by most of the world's major news agencies and is used in several commercial applications. It contains 123 elements with 513 attributes (as of version 2.5). Note that the results of these experiment can easily be generalized to multiple DTDs. Indeed, as DTDs generally use distinct grammars, an XML document valid for a given DTD is unlikely to match a subscription for another DTD; thus, using multiple DTD essentially boils down to running separate experiments with each DTD and combining the results.

We have generated sets of subscriptions of various sizes (from 100 to 50,000 subscriptions). For each size, we have generated one set containing only distinct subscriptions, and a second set with possibly multiple occurrences of each subscription. We will refer to these as *unique* and *multiple* sets, respectively.

We have compared three routing protocols that implement perfect content-based routing. First, the *match-first* routing protocol that matches published events against all subscriptions and computes a destination list used to route events (see Section 2). As previously discussed, this protocol imposes a high storage and processing load on the publisher nodes and does not scale well. Second, we implemented a *simple* routing protocol that does not use subscription aggregation, except for suppressing multiple occurrences of a subscription. With that protocol, the size of the routing table at a node is equal to the number of distinct subscriptions that consumers registered downstream. Finally, our XRoute routing protocol that makes extensive use of subscription aggregation to minimize the size of the routing tables.

As all these protocols implement perfect routing, they will exhibit the same bandwidth usage. Therefore, we are interested in comparing their space requirements. Besides lowering the memory usage at the routers, keeping routing tables small is essential to implement efficient filtering: as the filtering speed typically decreases linearly with the number of subscriptions (whether matching subscriptions sequentially, or using sophisticated algorithms as in [7]), small routing tables can dramatically improve the overall performance of a content network.

We have specifically measured the *average* and the *maximum* sizes of the routing tables at the inner nodes with each protocol. The average sizes gives an indication of the overall efficiency of our aggregation techniques, and the maximum sizes can help dimensioning the resources allocated to routers in the network (in particular at the producer nodes, which typically have the largest routing tables). We study the variation of these sizes according to the number of subscriptions injected in the system.

Results and Interpretations. Figure 5 shows the average size of the routing tables of the XRoute and the *simple* routing protocols, with both unique and multiple sets. It appears clearly that, in both cases, XRoute reduces the average size of the routing tables dramatically (by more than a factor of 5).

Figure 6 shows the relative space gain of XRoute vs. *simple* routing. We can observe that the gap between both protocols widens significantly with large number of consumers, demonstrating that our content-based routing protocol is extremely scalable. Note that XRoute is even more efficient with multiple subscriptions instances because of the increased number of covering relations (even though the *simple* routing protocol also benefits from multiple sets).

Figure 7 shows the maximum size of the routing tables of the XRoute, *simple*, and *match-first* protocols, with multiple subscriptions instances. Here again, we observe that XRoute is very space-efficient: it outperforms the

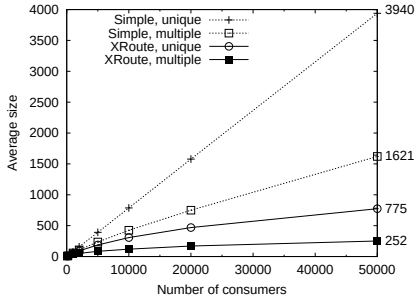

Figure 5: Average size of the routing tables with the XROUTE and *simple* routing protocols, with unique and multiple sets.

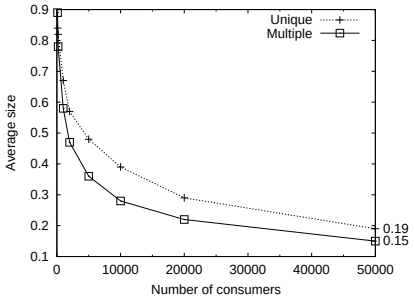

Figure 6: Average size ratio of XROUTE vs. *simple* routing.

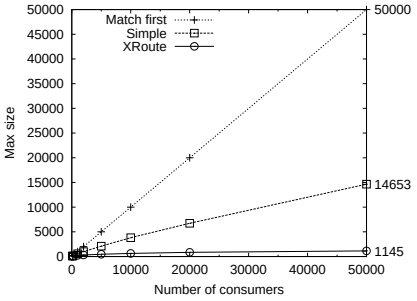

Figure 7: Maximum size of the routing tables with the XROUTE, *match-first*, and *simple* routing protocols.

other protocols, by factors of up to 14 (w.r.t. *simple*) and 43 (w.r.t. *match-first*). One can also notice that, with the *simple* protocol, the maximum size of the routing tables is approximately 10 times larger than its average size; in contrast, with XROUTE, the maximum size is less than 5 times bigger that the average size. Thus, our protocol seems to better balance the load on the routers.

7 Conclusion

We have developed a novel protocol for content-based routing in overlay networks. Our protocol, XROUTE, implements perfect routing, optimizes usage of network bandwidth, and minimizes the size of the routing tables in the system. To the best of our knowledge, our content-based routing protocol is the first to take full advantage of subscription aggregation *and* support registration cancellation, without impacting routing accuracy.

Although our protocol was designed for, and tested with, tree-structured XPath subscriptions, it can be readily applied to other subscription models. The experimental evaluation that we conducted shows that our protocol dramatically reduces the sizes of the routing tables and scales to very large consumer populations.

We are currently deploying our content-based routing protocol in the XNET XML content dissemination system, and integrating it with our highly-efficient XTRIE filtering algorithms [7] in application-level routers. We are also trying to extend the protocol to take advantage of lossy aggregation (as described in [6]) for further compression of the routing tables, but at the price of some deterioration in the routing accuracy and bandwidth usage.

References

[1] M. Altinel and M. Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB 2000)*, pages 53–64, Sept. 2000.

[2] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, and D. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS'99)*, 1999.

[3] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient filtering in publish-subscribe systems using binary decision. In *International Conference on Software Engineering*, pages 443–452, 2001.

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

[5] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Content-based addressing and routing: A general model and its application. Technical Report CU-CS-902-00, Department of Computer Science, University of Colorado, Jan. 2000.

[6] C.-Y. Chan, W. Fan, P. Felber, M. Garofalakis, and R. Rastogi. Tree Pattern Aggregation for Scalable XML Data Dissemination. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB 2002)*, Hong Kong, China, August 2002.

[7] C.-Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient Filtering of XML Documents with XPath Expressions. *VLDB Journal*, 11(4):354–379, 2002.

[8] R. Chand and P. Felber. A scalable protocol for content-based routing in overlay networks. Technical Report RR-03-074, Institut EURECOM, Feb. 2003.

[9] I. P. T. Council. News Industry Text Format.

[10] G. Cugola, E. D. Nitto, and A. Fugetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *IEEE Transactions on Software Engineering*, 27(9):827–850, Sept. 2001.

[11] Y. Diao, P. Fischer, M. Franklin, and R. To. YFilter: Efficient and Scalable Filtering of XML Documents. In *Proceedings of the 18th International Conference on Data Engineering (ICDE 2002)*, San Jose, CA, February 2002.

[12] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*. To appear.

[13] F. Fabret, H. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha. Filtering Algorithms and Implementations for Very Fast Publish/Subscribe Systems. In *Proc. of ACM SIGMOD*, pages 115–126, Santa Barbara, California, May 2001.

[14] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content Based Routing with Elvin4. In *AUUG2K*, Canberra, Australia, June 2000.

[15] R. Shah, R. Jain, and F. Anjum. Efficient Dissemination of Personalized Information Using Content-Based Multicast. In *Proceedings of INFOCOM 2002*, New-York, June 2002.

[16] A. Snoeren, K. Conley, and D. Gifford. Mesh Based Content Routing using XML. In *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP 2001)*, pages 160–173, Alberta, Canada, October 2001.

[17] W3C. XML Path Language (XPath) 1.0, Nov. 1999.

[18] E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proceedings of INFOCOM 1996*, San Francisco, March 1996.