Institut EURECOM
Research Report N^o 74 — RR-03-074

A Scalable Protocol for Content-Based Routing in Overlay Networks

R. Chand, P.A. Felber

February 26, 2003

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 [11] 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.[1] 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

[1] In contrast, in broadcast networks, filtering must be performed at the consumers. This approach has the drawback of misspending network bandwidth and consumer resources (consumers also receive, and must filter out, the messages they are not interested in).

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.

The rest of this paper is organized as follows: We first discuss related work in Section 2, and we introduce the system model and definitions in Section 3. In Section 4, we present the general principle of our content-based routing protocol, and we formally describe it in Section 5. Section 6 presents results from experimental evaluation. Finally, Section 7 concludes the paper.

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 [11] for a survey). Elvin [13] 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 [9] 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 [15], 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 [14], 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, 12, 3, 7, 10]).

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 5.

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 [16] 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.[2]

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.

[2]Most publish/subscribe systems use custom subscription languages with *name-value* pairs of properties, basic comparison operators ($=$, $<$, $\leq$, $>$, $\geq$), and logical operators (and, or, not).

4 Overview of the Protocol

4.1 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.

4.2 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.

Example 1. *Consider the network in Figure 1, 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.*

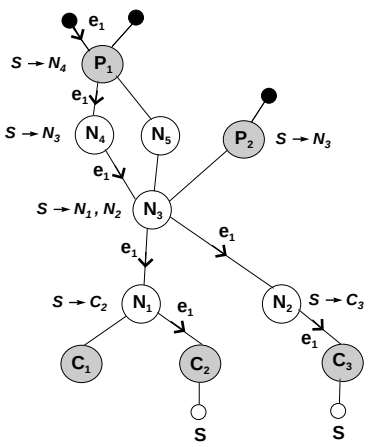

Figure 1: A sample publish/subscribe network. Subscriptions are represented underneath the consumers that registered them, and routing table entries are listed next to the node they are associated with.

4.3 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 2. The

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
```

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.

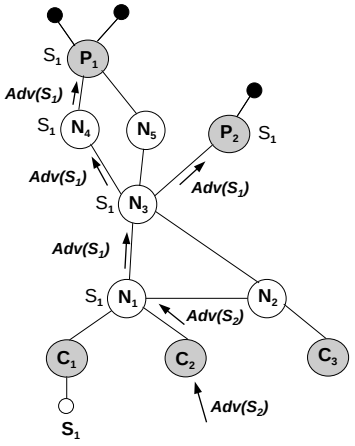

Figure 2: Subscription advertisements are propagated upward from the consumers to the publishers. They may be transformed along the propagation paths due to aggregation (here, we have $S_1 \supseteq S_2$).

4.4 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 2. 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.[3] 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 2), 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 2). 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:

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

[3] 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. $\overline{T_{S_2}^k} \leftarrow 0$

When the operation has been done, 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, $\overline{T^k_{S_2}}$ 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. $R_{S_1} \leftarrow R_{S_1} + \sum_{k \leq n} T^k_{S_2}.x + R_{S_2}$

When the operation has been done, 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.

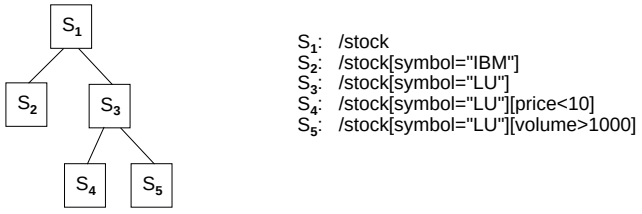

Figure 3: The substitution relations apply recursively. Subscriptions can be organized in a tree, where a link indicates that a child is substituted by its parent.

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 3 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.

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.

We can identify the following properties on the representation and substitution relations.

Consider node N. Let node N_{down} be the node downstream interface I^k and N_{up} be the upstream neighbor node.

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.*

Proof. Suppose that S can be represented by subscription S' at interface I^k. That means that we have $\overline{T^k_{S'}} > 0$, which means that S' has an entry at node N_{down}. But then at that node, subscription S would have been substituted by S' and no advertisement for S would have reached node N. Contradiction. □

It is very important to note that $T^k_{S_1}.z$ indicates how many subscriptions are represented by S_1 at interface I^k, but that we do not know which subscriptions are represented by S_1.

Corollary: It follows from property 1 that the only case when S_2 can be represented by S_1 is when $\overline{T^k_{S_1}} = 0$ and an advertisement for S_1 arrives at interface I^k. Then $\overline{T^k_{S_1}}$ becomes strictly positive and S_2 is represented by S_1 at interface I^k. In other words, an advertisement for S_2 has arrived before the advertisement for S_1.

Property 2. *Assume that $Ptr_{S_1} = null$. If S_2 is being represented by S_1 at any interface I^k of node N, then S_2 will also be represented by S_1 at the upstream neighbor node N_{up}, at incoming interface I^j (toward N).*

Proof. This property comes partly from the operation of the routing protocol. We have seen in the corollary of property 1 that if S_2 is being represented by S_1, then the advertisement for S_2 arrived before that for S_1. Suppose that S_2 was not substituted by any subscription at node N. Then S_2 has an entry at node N_{up}. At node N, S_1 is not substituted and thus the advertisement sent to N_{up} advertises S_1. Then at node N_{up}, the conditions for representing S_2 by S_1 are met. Now if S_2 is substituted by a subscription S' at node N, the operation of the routing protocol is such that S_2 will be represented by S at node N_{up}, at interface I^k. □

Property 3. *If S_2 is substituted by S_1 at node N, then S_2 is represented by S_1 at the upstream neighbor node, at incoming interface I^j (toward N).*

Proof. Suppose that S_2 arrived at node N before S_1. Then S_2 also has an entry at the upstream node N_{up}, because it has not bee substituted at node N. Because S_1 is not substituted, the outgoing advertisement advertises S_1. When it arrives at node N_{up} the conditions for representing S_2 by S_1 are met. Now suppose that S_1 arrived first at node N. Then when S_2 is substituted by S_1, according to the definition of the substitution relation, the incoming advertisement for S_2, $(S_2; n; r)$ yields an outgoing advertisement for S_1: $(S_1; 0; n + r)$. That means that S_1 is to represent $n + r$ instances of subscription S_2 at node N_{up}. □

Property 4. *If S_2 is represented by S_1 at interface I^k of node N, no registration advertisements for S_2 can be received at interface I^k.*

Proof. Suppose that an advertisement for S_2 arrives at interface I^k of node N. That means that at node N_{down}, S_2 is not substituted by any subscription. But if S_2 is represented by S_1 at interface I^k of node N, that means that we have $\overline{T^k_{S_1}} > 0$, and S_1 has an entry at node N_{down}. Thus S_2 would have been substituted by S_1 at node N_{down} and no advertisement for S_2 would reach node N. Contradiction. □

Property 5. *Consider subscription S. R_S represents the number of subscriptions that are represented by S at all interfaces, plus the number of all the subscriptions on $tree(S)$, plus the number of the subscriptions that any of them represent at any interface.*

Proof. According to the definition of the representation relation, R_S is incremented by $\overline{T^k_{S'}}$ when S' is being represented by S at interface I^k. Thus R_S represents at least the instances of all the subscriptions that are represented by S at all the interfaces.

Suppose that the height of $tree(S)$ is one. In other words, no subscriptions are substituted by S. Then the only way to increase R_S is by representation relations. Thus R_S represents exactly the instances of all the subscriptions that are represented by S at all the interfaces, and the property is true ($tree(S)$ comprises S only).

Now suppose that the height of $tree(S)$ is two. In other words, if $\{Substituted\}$ is the set of all the subscriptions that are substituted by S, no subscriptions are substituted by a subscription in $\{Substituted\}$. Consider $S_1 \in \{Substituted\}$. Then $tree(S_1)$ is of height one, and R_{S_1} is exactly

the number of subscriptions that are represented by S_1 at all the interfaces. When S_1 was substituted by S, R_S was increased by $\sum_{k \leq n} T_{S_1}^k.x + R_{S_1}$. Thus in R_S are included all the instances of subscription S_1 plus the instances of the subscriptions that are represented by S_1 at all the interfaces. This is true for all the subscriptions in $\{Substituted\}$, and the property is true.

Now suppose that the property is true for a height of h (recursive assumption). If the height of $tree(S)$ is $h+1$, then again let $\{Substituted\}$ be the set of all the subscriptions that are substituted by S. We have seen that R_S represents at least the instances of all the subscriptions that are represented by S at all the interfaces. Now for each $S_1 \in \{Substituted\}$, $tree(S_1)$ is of height at most h. When S_1 is substituted by S, R_S is increased by $\sum_{k \leq n} T_{S_1}^k.x + R_{S_1}$, that is by all the instances of subscription S_1 plus R_{S_1}. But because $tree(S_1)$ is of height at most h, according to the recursive assumption, R_{S_1} is exactly the number of subscriptions that are represented by S_1 at all the interfaces, plus the number of all the subscriptions on $tree(S_1)$, plus the number of the subscriptions that any subscription on $tree(S_1)$ represents at any interface. Thus the property is true for a height $h+1$, and according to the theorem of recursivity, it is always true. □

Property 6. *At node N, for any subscription S, $T_S^k.z$ is equal to R_S at node N_{down}.*

Proof. This property comes from the operation of the routing protocol. Indeed, the update of the routing table is such that $T_S^k.z$ corresponds to the number of subscriptions "aggregated" in S (either through representation or substitution) downstream interface I^k, that is at node N_{down} (as we have seen in the description of the format of a routing table). Then according to property 5, R_S represents exactly the number of those subscriptions. □

Corollary: From this property, it follows that part of the subscriptions that are represented by S at interface I^k of node N are subscriptions S_i at node N_{down} such that Ptr_{S_i} points to S. The other part consists of subscriptions that are represented by S at all the downstream interfaces of node N_{down}. Recursively we can completely identify them, because there are no representations at client nodes.

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.

When an advertisement for a subscription S arrives at interface I^k of node N, we first update T_S^k. 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 property:

Property 7. *When an advertisement for the registration of subscription S_2 arrives at node N, if S_2 can be substituted by another subscription S_1, then no subscription can be substituted by S_2.*

Proof. Suppose that a subscription S_3 can be substituted by S_2. This implies that S_3 is not substituted by a subscription yet. Also this implies that $S_3 \supset S_1$ (because of the transitivity property of the covering relation). Then S_3 would have been substituted by S_1. Contradiction. □

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 $\overline{T_S^k} > 0$, 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.

Algorithm 2 — Routing Table Update

1: **if** $Ptr_S \neq null$ **then**
2: $T_S^k.y \leftarrow T_S^k.y + n_S$
3: **for all** S' ancestor of S in $tree(h(S))$ **do**
4: $R_{S'} \leftarrow R_{S'} + n_S + r_S$
5: **end for**
6: $adv_{out} \leftarrow (h(S); 0; n_S + r_S)$
7: **else**
8: $T_S^k.x \leftarrow T_S^k.x + n_S$
9: $adv_{out} \leftarrow (S; n_S; r_S)$
10: **end if**
11: $T_S^k.z \leftarrow T_S^k.z + r_S$
12: $R_S \leftarrow R_S + r_S$

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 $\overline{T_S^k} = 0$. 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 relations when conditions (i) and (ii) above are met:

- 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 7, 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).
- 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 $\overline{T_S^k} = 0$. A subscription

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 $T_j = \overline{T_{S_j}^k}$
4: Represent S_j by S at I^k
5: **if** $S_j \in tree(S)$ **then**
6: **for all** S_k ancestor of S_j in $tree(S)$ **do**
7: $R_{S_k} \leftarrow R_{S_k} - T_j$
8: **end for**
9: **else**
10: **if** $S_j \in tree(h(S))$ **then**
11: **for all** S_k ancestor of S_j in $tree(h(S))$ **do**
12: $R_{S_k} \leftarrow R_{S_k} - T_j$
13: **end for**
14: **else**
15: **for all** S_k ancestor of S_j in $tree(h(S_j))$ **do**
16: $R_{S_k} \leftarrow R_{S_k} - T_j$
17: **end for**
18: **if** $S_j \neq h(S_j)$ **then**
19: append $(h(S_j); 0; -T_j)$ to adv_{out}
20: $A \leftarrow A + T_j$
21: **end if**
22: **end if**
23: **for all** S_k ancestor of S in $tree(h(S))$ **do**
24: $R_{S_k} \leftarrow R_{S_k} + T_j$
25: **end for**
26: **end if**
27: remove $entry(S_j)$ if $\sum_{p \leq n} \overline{T_{S_j}^p} = 0$
28: **end for**
29: **for all** S_k ancestor of S in $tree(h(S))$ **do**
30: $R_{S_k} \leftarrow R_{S_k} + n_S + r_S$
31: **end for**
32: $R_S \leftarrow R_S + r_S$
33: $T_S^k.z \leftarrow T_S^k.z + r_S$
34: **if** $h(S) \neq null$ **then**
35: $T_S^k.y \leftarrow T_S^k.y + n_S$
36: $adv_{out} \leftarrow (h(S); 0; n_S + r_S + A)$ [+ appended triples]
37: **else**
38: $T_S^k.x \leftarrow T_S^k.x + n_S$
39: $adv_{out} \leftarrow (S; n_S; r_S + A)$ [+ appended triples]
40: **end if**

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 $T_j = \overline{T_{S_j}^k}$ 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 ancestor of S_j in $tree(S)$, the T_j instances of subscription S_j are no longer substituted it it. 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.

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: $S_j \notin tree(S)$. 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).

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

Second case: $entry(S)$ exists and $\overline{T_S^k} = 0$ (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_S^k 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 $\overline{T_{S'}^k} > 0$. Therefore, we can apply algorithm2 for each S'.

Example 2. *Figure 4 illustrates the operation of the subscription algorithm on the publish/subscribe network of Figure 1. 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

In this section, we outline the principle of subscription cancellations.

Considerations. Consider node N and $entry(S)$ of its routing table. Let node N_{down} be the node downstream interface I^k and node N_{up} be the upstream neighbor node. Let $adv_{in}(S)$ be an advertisement for the cancellation of subscription S (thus $n_S < 0$), arriving at interface I^k from node N_{down}. In most situations, we can update the routing table using Algorithm 2. One scenario, however, must be dealt with differently.

Consider the case where, after applying Algorithm 2, we have $T_S^k.x + T_S^k.y = 0$ and $T_S^k.z > 0$. In other words, the incoming advertisement has canceled all instances of subscriptions S. Now consider an event e that arrives at node N and matches S. Because $\overline{T_S^k} > 0$, e will be forwarded downstream interface I^k. However, there are no more consumers interested in events matching S downstream I^k, and node N cannot determine whether e matches any of the subscriptions represented by S at I^k. Consequently, we may have imperfect routing, and in the worst case routing may degenerate into *flooding*. Therefore, we need to implement an additional operation to deal with this problem. This operation is implemented in algorithm 6.

This problem occurs when $T_S^k.x + T_S^k.y = |n_S|$ *and* $T_S^k.z > 0$. Let $\{S_i\}$ denote the set of subscriptions that S represents at interface I^k. To prevent imperfect routing, each subscription S_i must be such that $\overline{T_{S_i}^k} > 0$. In other words, the representations must be "undone". Moreover, all the possible relations between S_i and the other subscriptions in the routing table must be established, if not already. Thus we first have to identify these subscriptions $\{S_i\}$. Moreover, for each S_i, $T_{S_i}^k$ must be accurate, that is, it must account for the number of instances of S_i, and the number of subscriptions that are represented by S_i. Then all the possible relations must be established.

According to the corollary of property 6, part of the subscriptions that are represented by S at interface I^k of node N are those that are substituted by S at node N_{down} (i.e., their Ptr entry field points to S). The other part consists of the subscriptions that are represented by S at all of node N_{down}'s interfaces.

Operation at consumer node. Let CN be the consumer node where the consumer issues the cancellation of all instances of subscription S that it had registered before. Let N_{up} be the upstream neighbor node, and interface I^k such that CN is the node downstream that interface. We have seen that to prevent imperfect routing at node N_{up}, that node needs to know all the subscriptions that S represents at interface I^k. But because there are no representations at a consumer node, those subscriptions are the ones at node CN that are substituted by S. Thus at node CN, we first need to remove $entry(S)$. Then we have to reorganize the subscriptions that were substituted by S, in the sense that we have to establish the possible relations. Because CN is a consumer node, we will not try to establish representation relations. Let $\{L\}$ be the set of the subscriptions that were *directly* substituted by S, that is such that their Ptr field pointed to S. When $entry(S)$ has been deleted, they do not point to S anymore. First we try to establish substitution relations between the subscriptions in $\{L\}$. When this is done, we have a new set of subscriptions $\{L'\}$ (such that their Ptr field is null). Then we try to establish substitution relations between the subscriptions in $\{L'\}$ and the other subscriptions in the routing table. This leads to a new set of subscriptions: $\{L_{reinserted}\}$ such that their Ptr field is null. Let $reinsert(L)$ be the operation of reinserting the subscriptions in $\{L\}$ in such a way. Then we create an advertisement for the cancellation of subscription S, $adv_{in}(S)$, and we append to it the subscriptions in $\{L_{reinserted}\}$, in the form $(S_i; n_{S_i}; r_{S_i})$

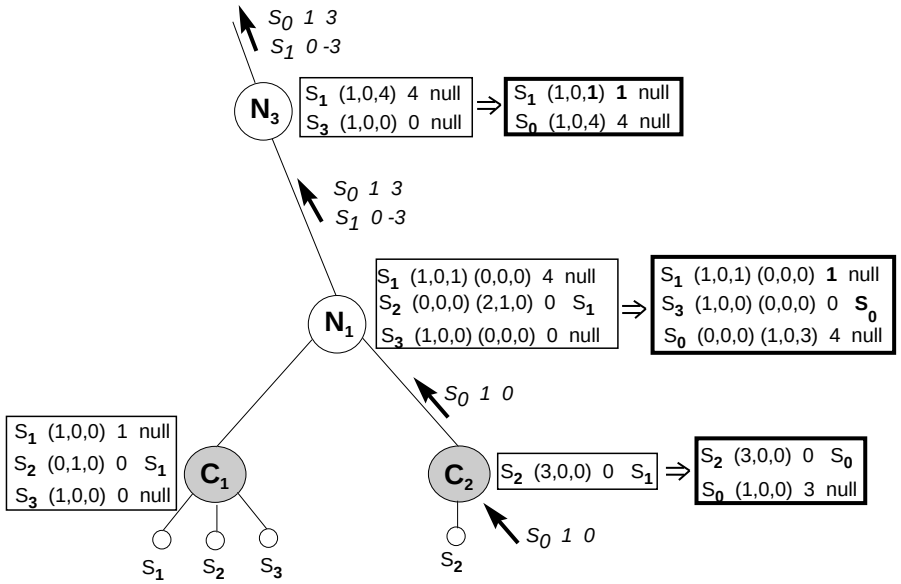

Figure 4: Example of the subscription algorithm. Registered subscriptions are represented below their corresponding client nodes. Routing tables (shown next to the nodes) are updated as a result of the registration of subscription S_0 (updated tables are shown with a thick frame). Here, we have $S_0 \supseteq S_2$, $S_1 \supseteq S_2$, and $S_1 \supseteq S_3$. There are no relationships between S_0 and S_1, and between S_2 and S_3.

(lines $86-91$). Those subscriptions are the result of the re-insertion in the routing table of node CN of the subscriptions that were substituted by S. In other words, we have: $\sum_{S_i \in \{L_{reinserted}\}} n_{S_i} + r_{S_i}$ is equal to R_S before $entry(S)$ was deleted. As we have seen in property 6, it is also equal to $T_S^k.z$ in the routing table of node N_{up}. To prevent imperfect routing at node N_{up}, those $T_S^k.z$ subscriptions in $\{L_{reinserted}\}$ must be inserted in its routing table after canceling subscription S. That is the object of the next paragraph.

Operation at inner nodes. Consider node N receiving the advertisement for the cancellation of subscription S, $adv_{in}(S)$, from the node downstream interface I^k, N_{down}. Let $\{L\}$ be the set of the subscriptions appended to $adv_{in}(S)$. We suppose (recursive assumption) that those subscriptions in $\{L\}$ represent the subscriptions that are represented by S at interface I^k of node N. Also, we suppose that those subscriptions are such that at the node downstream I^k, their Ptr field is null. When node N processes the advertisement, it sees that $T_S^k.x + T_S^k.y = |n_S|$ and $T_S^k.z > 0$. Thus it knows that is has to reinsert the subscriptions appended to $adv_{in}(S)$. A subscription S_i in $\{L\}$ may or may not have an entry in the routing table. If S_i does not have an entry, we create a null entry for it, and make the Ptr field point to S (lines $8-11$). Then for every subscription in $\{L\}$, we store those that belong to $tree(h(S))$ in a list, L_{keep}, and remove them from L (lines $12-15$). For every subscription S_j in L_{keep}, we update their $T_{S_i}^k$ field (note that S_i is necessarily substituted by another subscription) and their R field to take into account that n_{S_i} instances of S_i must be present at node N, and that S_i must represent r_{S_i} subscriptions. Also, $keep$ is a counter that represents the number of instances of the subscriptions in L_{keep} as well as the number of subscriptions that they represent (lines $18-23$). The subscriptions that remain in $\{L\}$ are the ones that already have an entry and do not belong to $tree(h(S))$. Because of the recursive assumption, we can call algorithm 2 to update the routing table for each of them, except that the

Algorithm 5 — Cancellation - Part 1

```
1: declare L_keep
2: declare L
3: declare L_reinsert
4: for all S_j ∈ adv_in(S) do
5:    push S_j in L
6: end for
7: for all S_j ∈ L do
8:    if S_j does not have an entry then
9:       create a null entry for S_j, entry(S_j)
10:      Ptr_{S_j} ← S
11:   end if
12:   if S_j ∈ tree(h(S)) then
13:      push S_j in L_keep
14:      remove S_j from L
15:   end if
16: end for
17: declare keep = 0
18: for all S_j ∈ L_keep do
19:   keep ← keep + n_{S_j} + r_{S_j}
20:   T^k_{S_j}.y ← T^k_{S_j}.y + n_{S_j}
21:   T^k_{S_j}.z ← T^k_{S_j}.z + r_{S_j}
22:   R_{S_j} ← R_{S_j} + r_{S_j}
23: end for
24: for all S_j ∈ L do
25:   call algorithm 2: "Routing Table Update"
26: end for
27: R_S ← R_S − T^k_S.z
28: T^k_S ← 0
29: if Ptr_S ≠ null then
30:   for all S_k ancestor of S in tree(h(S)) do
31:      R_{S_k} ← R_{S_k} + n_S − T^k_S.z
32:   end for
33:   for all S_j ∈ L_keep do
34:      for all S_k ancestor of S_j in tree(h(S)) do
35:         R_{S_k} ← R_{S_k} + n_{S_j} + r_{S_j}
36:      end for
37:   end for
38:   adv_out ← (h(S); 0; n_S − T^k_S.z + keep)
39:   if ∀p, T̄^p_S = 0 then
40:      for all S_k such that Ptr_{S_k} = S do
41:         Ptr_{S_k} = Ptr_S
42:      end for
43:      delete entry(S)
44:   end if
45: else
46:   see Part 2
47: end if
```

Algorithm 6 — Cancellation - Part 2

```
48: for all S_j ∈ L_keep do
49:   for all S_k ancestor of S_j in tree(h(S_j)) do
50:      R_{S_k} ← R_{S_k} + n_{S_j} + r_{S_j}
51:   end for
52: end for
53: if ∃p ≠ k, T̄^p_S ≠ 0 then
54:   adv_out ← (S; n_S; −T^k_S.z + keep)
55: else
56:   for all S_k such that Ptr_{S_k} = S do
57:      push S_k in L_reinsert
58:   end for
59:   delete entry(S)
60:   call reinsert(L_reinsert)
61:   adv_out ← (S; n_S; 0) [+ appended triples]
62: end if
```

result of the update is appended to the outgoing advertisement, as part of the additional triples optionally carried by an advertisement (lines $24-26$).

Now we consider different cases depending on the value of $entry(S)$.

Case 1: Ptr_S is not null

First we must update the entries of the ancestors of S, in a similar way to that in algorithm 3. $|n_S|$ instances of subscription S have been canceled. Thus those are no longer substituted by the ancestors of S. Also, $T^k_S.z$ subscriptions are no longer represented by S at interface I^k; they are no longer substituted by its ancestors neither. Thus we decrement the R field of every ancestor of S by $|n_S| + T^k_S.z$ (lines $30-32$) and decrement that of S by $T^k_S.z$. Also, we reset T^k_S (lines $27-28$).

Then, for every subscription S_i in L_{keep}, we have to update the entries of its ancestors in $tree(h(S_i))$ to take into account the n_{S_i} new instances of subscription S_i, and the r_{S_i} instances of the subscriptions that S_i is to represent at interface I_k. Those $n_{S_i} + r_{S_i}$ additional subscriptions are now substituted by every ancestor of S_i. Thus we increment their R field by $n_{S_j} + r_{S_j}$ (lines $33-37$).

Now if $\forall p, \overline{T^p_S} = 0$, then $entry(S)$ has to be deleted. But before doing this, we must take into account the possible subscriptions that are substituted by S. Because Ptr_S is not null, we can just make the Ptr of those subscriptions point to Ptr_S (up one level in the $tree(h(S))$). Then we delete $entry(S)$ (lines $39-44$).

Note that subscription $h(S)$ has seen its R field decremented by $|n_S| + T^k_S.z$ and incremented by

keep. At node N_{up}, subscription S is represented by $h(S)$ at the incoming interface. We are no longer in the situation where algorithm 6 must be applied. The only change to the routing table of node N that will have an impact at node N_{up} is the one made to the R field of $entry(h(S))$. At node N_{up}, subscription $h(S)$ is to represent $|n_S| + T_S^k.z - keep$ fewer subscriptions. Thus we just need to send the corresponding advertisement to node N_{up}, which will call algorithm 2 (line 38).

Case 2: Ptr_S is null

First we update the entries of the ancestors of the subscriptions in L_{keep}, as well as $entry(S)$, as in the previous case (lines $48-52$ and $27-28$). Then we have to consider the two same subcases.

If $\exists p \neq k$ such that $\overline{T_S^p} \neq 0$, then $entry(S)$ is not deleted. This implies that at node N_{up} we are no longer in the situation where algorithm 6 must be applied. As in the previous case, the only changes at N_{up} consists in updating $entry(S)$ (lines $53-55$).

Now if $\forall p, \overline{T_S^p} = 0$, $entry(S)$ is deleted. Then at node N_{up}, we will be in the same situation as node N, where algorithm 6 must be applied. This implies that node N_{up} must know the subscriptions that S represents at the incoming interface. But because $\forall p, \overline{T_S^p} = 0$, the only subscriptions that S represented at node N were the ones at interface I_k. Those were the subscriptions in $\{L\}$ and they have been reinserted in the routing table of node N either by algorithm 2 or as described in algorithm 6. Now the only subscriptions that S represents at the incoming interface of node N_{up} are the ones that are substituted by it at node N. Then we are in a situation similar to the one that we would have if N were a consumer node. Moreover, because of the recursive assumption and property 1, no representation relations can be established between the subscriptions in $\{L_{keep}\}$ and the other subscriptions. Then we call procedure *reinsert* to reinsert all the subscriptions that are directly substituted by S and append the result to the outgoing advertisement (lines $56-61$). Then those subscriptions are such that they represent the subscriptions that are represented by S at interface I_k of node N_{up}. Also, they are such that their Ptr field is null. Then at node N_{up}, the recursive assumption is true and algorithm 6 is called.

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.

6.1 Simulation Setup

We have generated a network topology using the transit-stub model of the Georgia Tech Internetwork Topology Models package [17]. The resulting network topology, shown in Figure 5, contains 64 routers. We then added 24 consumers at the edges of the network and a single producer.

We have simulated consumer load by registering subscriptions at the consumer nodes. The subscriptions were expressed using the XPath language [16]. 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 [8] 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

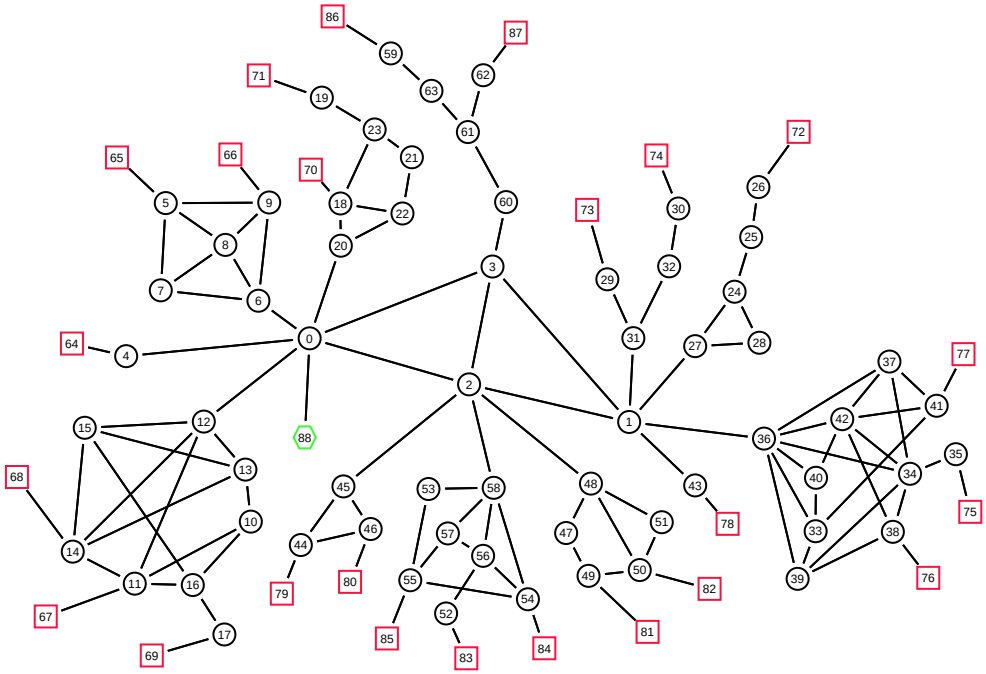

Figure 5: Simulated network topology with 64 routers (circles), 24 consumers (boxes), and 1 producer (hexagon).

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.

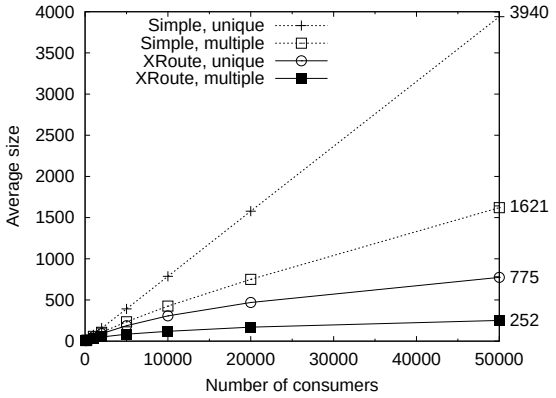

Figure 6: Average size of the routing tables with the XROUTE and *simple* routing protocols, with both unique and multiple sets.

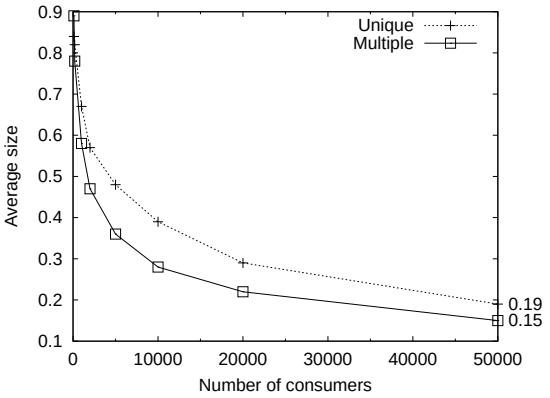

Figure 7: Average size ratio of XROUTE vs. *simple* routing.

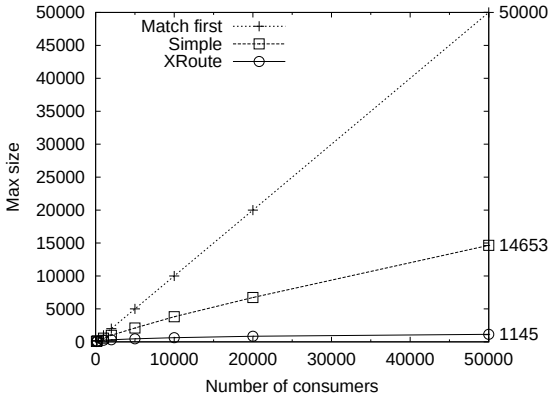

Figure 8: Maximum size of the routing tables with the XROUTE, *match-first*, and *simple* routing protocols.

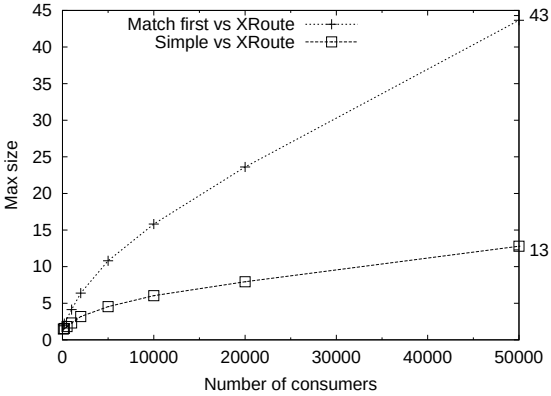

Figure 9: Maximum size ratios of *match-first* vs. XROUTE and *simple* vs. XROUTE.

6.2 Results and Interpretations

Figure 6 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 7 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. 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 8 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 much more space-efficient than the other protocols. 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.

Finally, Figure 9 directly compares the maximum

size of the routing tables of *simple* and *match-first* routing with that of XROUTE. Our protocol clearly outperforms the other protocols, by factors of up to 14 (w.r.t. *simple*) and 43 (w.r.t. *match-first*). Again, we can see that the difference between XROUTE and the other protocols increases with high numbers of consumers, demonstrating that our content-based routing protocol is extremely scalable.

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.J. 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, September 2000.

[2] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R.E. Strom, and D.C. 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.M. 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.S. Rosenblum, and A.L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Trans. on Computer Systems*, 19(3):332–383, August 2001.

[5] Antonio Carzaniga, David S. Rosenblum, and Alexander 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, January 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] International Press Telecommunications Council. News Industry Text Format.

[9] G. Cugola, E. Di 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, September 2001.

[10] 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.

[11] P.Th. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 2003. To appear.

[12] F. Fabret, H.A. Jacobsen, F. Llirbat, J. Pereira, K.A. 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.

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

[14] 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.

[15] A.C. Snoeren, K. Conley, and D.K. 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.

[16] W3C. XML Path Language (XPath) 1.0, November 1999.

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