A Routing Scheme for Content-Based Networking

Antonio Carzaniga   Matthew J. Rutherford   Alexander L. Wolf

Software Engineering Research Laboratory
Department of Computer Science
University of Colorado
Boulder, Colorado 80309-0430 USA
{carzanig,rutherfo,alw}@cs.colorado.edu

University of Colorado
Department of Computer Science
Technical Report CU-CS-953-03
June 2003

Abstract

This paper proposes a routing scheme for content-based networking. A content-based network is a communication network that features a new advanced communication model where messages are not given explicit destination addresses, and where the destinations of a message are determined by matching the content of the message against selection predicates declared by nodes. Routing in a content-based network amounts to propagating predicates and the necessary topological information in order to maintain loop-free and possibly minimal forwarding paths for messages. The routing scheme we propose uses a combination of a traditional broadcast protocol and a content-based routing protocol. We present the combined scheme and its requirements over the broadcast protocol. We then detail the content-based routing protocol, highlighting a set of optimization heuristics. We also present the results of our evaluation, showing that this routing scheme is effective and scalable.

1 Introduction

Content-based communication is a communication service whereby the flow of messages from senders to receivers is driven by the content of the messages, rather than by explicit addresses assigned by senders and attached to the messages [4]. Using a content-based communication service, receivers declare their interests by means of *selection predicates*, while senders simply publish messages. The service consists of delivering to any and all receivers each message that matches the selection predicates declared by those receivers.

In the content-based service model, message content is structured as a set of attribute/value pairs, and a selection predicate is a logical *disjunction* of *conjunctions* of elementary *constraints* over the values of individual attributes. For example, a message might have the following content

[class="alert", severity=6,
device-type="web-server",
alert-type="hardware failure"]

which would match a selection predicate such as this:

[alert-type="intrusion" $\wedge$ severity$>$2 $\vee$
class="alert" $\wedge$ device-type="web-server"]

Content-based communication is ideally suited for a variety of application domains, including news distribution, publish/subscribe event notification, system monitoring and management, network intrusion detection, service discovery, data sharing, distributed electronic auctions, and distributed games.

We believe that the best way to provide a content-based communication service is as a datagram, connectionless service, through a *content-based network*. We envision a content-based network as an overlay point-to-point network. Similarly to other traditional network services, routing in a content-based network amounts to synthesizing distribution paths throughout the network, while forwarding amounts to determining at each router the set of next-hop destinations of a message.

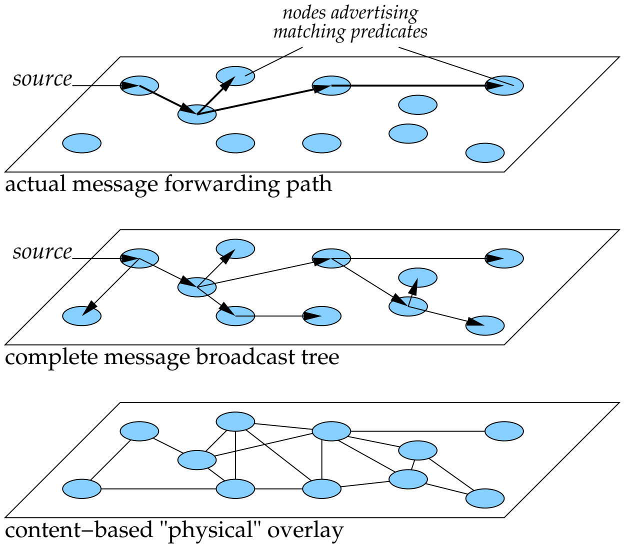

Figure 1: Network Overlay and High-Level Routing Scheme

In this paper we present a *combined broadcast and content-based (CBCB)* routing scheme for a content-based network. This scheme consists of a content-based layer superimposed over a traditional broadcast layer. The broadcast layer handles each message as a broadcast message, while the content-based layer prunes the broadcast distribution paths, limiting the propagation of each message to only those nodes that advertised predicates matching the message. This strategy is illustrated in Figure 1.

To implement this two-layer scheme, a router runs two distinct routing protocols: a *broadcast routing* protocol and a *content-based routing* protocol. The first protocol processes topological information and maintains the forwarding state necessary to send a message from each node to every other node. As it turns out, CBCB requires a broadcast layer that exhibits a specific topological property. Fortunately, this topological property can be satisfied by the most common broadcast schemes, with minimal modifications or with no modifications at all. In this paper we detail the requirements for the broadcast layer, and discuss how this layer can be implemented using protocols based on a global spanning tree, per-source minimal-paths spanning trees, or reverse-path broadcasting.

The second protocol processes predicates advertised by nodes, and maintains the forwarding state that is necessary to decide, for each router interface, whether a message matches the predicates advertised by any downstream node reachable through that interface. This second protocol, which is the main focus of this paper, is based on a dual "push–pull" mechanism that guarantees robust and timely propagation of content-based routing information.

These are the contributions of this paper:

- We present a routing protocol specifically designed for a content-based network. To the best of our knowledge, this is the first protocol that realizes a content-based communication service over a generic point-to-point network.

- We show that (1) the protocol is scalable to large networks, (2) the protocol realizes the content-based service with minimal missed deliveries and minimal unnecessary traffic, and (3) the protocol exhibits a stable behavior with respect to its control traffic.

In the next section we describe the basics of the content-based service model and the general architecture of a content-based network. We then detail the routing scheme, with particular attention to the mechanisms that realize the content-based layer. Following that, we present the main results of the experimental evaluation we conducted. We then discuss related work. We conclude indicating some future plans.

2 Content-Based Networking

A content-based network is a point-to-point, application-level overlay consisting of client nodes and router nodes, connected by communication links. By analogy with a physical network, we use the term interface to refer to the endpoint of a link. A content-based network accepts messages for delivery, and is connectionless and best-effort in nature. In a content-based network, nodes are not assigned unique network addresses, nor are messages addressed to any specific node. Instead, each node advertises a *predicate* that defines messages of interest for that node and, thus, the messages that the node intends to receive. The content-based service consists of delivering a message to all the client nodes that advertised predicates matching the message.

The abstract concept of a content-based network service is independent of the form of messages and predicates. To instantiate this concept, we define messages and predicates using the concrete syntax and semantics embodied in the Siena event notification service [3]. Note that in this regard, Siena is largely consistent with other publish/subscribe systems [2, 8, 14] and with existing standards for application-level publish/subscribe services [10, 12].

Thus, a *message* is a set of typed attributes. Each attribute is uniquely identified within the message by a *name*, and has a *type* and a *value*. For purposes of this paper, we consider the common types *string*, *integer*, *double*, and *boolean*. For example, [*string* carrier = UA; *string* dest = ORD; *int* price = 300; *bool* upgradeable = true;] would be a valid message. A *predicate* is a disjunction of conjunctions of constraints on individual attributes. Each constraint has a *name*, a *type*, an *operator*, and a *value*. A constraint defines an elementary condition over a message. A message matches a constraint if it contains an attribute with the same name and type, and if the value matches the condition defined by the operator and value of the constraint. For example, [*string* dest $=$ ORD $\wedge$ *int* price $<$ 400] is a valid predicate matching the message of the previous example.

In our model of content-based network, each node implements a service interface consisting of a *send_message(m)* function and a *set_predicate(p)* function. These functions are used by applications to send messages and to declare messages of interest respectively. The *set_predicate* function defines the content-based address of the node, overwriting any previous setting.

3 The CBCB Routing Scheme

A content-based network can be thought of as a dynamically-configurable broadcast network, where each message is treated as a broadcast message whose broadcast tree is dynamically pruned using content-based addresses. This observation forms the basis of the high-level design of the CBCB routing scheme.

CBCB consists of a content-based routing protocol implemented on top of a broadcast layer. The broadcast layer is necessary to make sure that a message flows from its source to all its destinations through loop-free and possibly minimal paths. The content-based layer is necessary to avoid sending a message towards nodes that are not interested in it. The broadcast layer is also used by the content-based protocol to propagate routing information.

The high-level routing strategy of CBCB is to establish content-based routes by advertising predicates from their issuer towards every other node, along the broadcast tree rooted at the issuer. This process produces forwarding state that "attracts" each message towards nodes that advertise predicates matching the message. In order to avoid loops, this forwarding state is used in combination with the broadcast tree rooted at the source of the message. Thus, the forwarding function proceeds by forwarding a message along the broadcast tree rooted at the sender, following only the branches that have matching predicates.

3.1 Broadcast System

CBCB uses a broadcast layer in combination with both its forwarding and routing functions. Without loss of generality, we will abstract this layer by assuming the availability of a *broadcast function* $B : N \times I \to I^*$ that, at each router, given a source node s and an input interface i, returns a set of output interfaces $B(s, i)$. The obvious requirement for the broadcast layer is to define a broadcast tree for each source node s through the recursive application of $B(s, \cdot)$.

The broadcast layer can be implemented using a global spanning tree (e.g., minimal spanning tree), per-source trees (e.g., shortest-paths trees), or other broadcast methods such as reverse-path forwarding [6]. For example, in the case of per-source shortest-paths trees, $B(s, i)$ would give the interfaces that are downstream from the current router, on the directed shortest-paths tree rooted in s, whereas, in the case of a reverse-path broadcast system, $B(s, i)$ would give either the complete set of interfaces, if i is on the unicast (reverse) path to s, or the empty set otherwise.

In addition to this basic requirement, the broadcast function B must satisfy a property that we call *all-pairs path symmetry*. This property is required by the high-level routing strategy of CBCB. In particular, the forwarding path of a message, from a sender to a receiver, is determined by the intersection of the broadcast tree rooted at the sender with the broadcast tree rooted at the receiver. Therefore, intuitively, the broadcast function must assure that such an intersection exists for each sender-receiver pair. Formally, a broadcast layer satisfies the all-pairs path symmetry property when, for each pair of nodes u and v, the broadcast function defines two broadcast trees T_u and T_v, rooted at nodes u and v respectively, such that the path $u \rightsquigarrow v$ in T_u is congruent to the reverse of the path $v \rightsquigarrow u$ in T_v.

Notice that this property is immediately satisfied by a broadcast layer based on a global spanning tree T, because for each u and v, $T_u = T_v = T$. A broadcast layer based on per-source shortest-paths trees can also immediately exhibit all-pairs path symmetry in all cases in which shortest paths are unique. This is because the shortest-paths trees T_u and T_v will contain the same (unique) shortest path between u and v. In the presence of multiple shortest paths between two nodes u and v, the broadcast function can be easily adapted to unambiguously select one of the paths (see Section 4.1 for some details). Similarly, a broadcast layer based on reverse-path forwarding will exhibit all-pairs symmetry as long as the underlying unicast routing protocol produces symmetric routes.

3.2 Preliminary Definitions

Before proceeding with the description of the content-based routing protocol, we briefly review the concept of *content-based address* [4] and that of *covering relation* between content-based addresses [3]. We define the content-based address of a node as a *predicate*—a total boolean function—that identifies the messages of interest for that node. In the following, we will use the terms content-based address and predicate interchangeably.

In the following, we will also need to refer to the messages and sets of messages that are selected by a content-based address. Thus, we write $p(m)$ to refer to the evaluation of a content-based address p over a message m, and we say that a content-based address p *selects* a message m when $p(m) = true$. Similarly, we refer to the set of all messages selected by a content-based address p as the *selection* of p (or *selection*(p)).

We also define a *covering* relation between content-based addresses: we say that content-based address p_1 *covers* content-based address p_2 iff $\forall m : p_2(m) \Rightarrow p_1(m)$, or, in other words, p_1 covers p_2 iff *selection*$(p_2) \subseteq$ *selection*(p_1). Notice that this covering relation defines a partial order between content-based addresses. For the sake of brevity, we also use the '$\prec$' symbol to indicate this relation. For example, $p_2 \prec p_1$ indicates that p_2 is covered by p_1.

3.3 Forwarding Scheme

The content-based part of CBCB maintains forwarding state in the form of a *content-based forwarding table*. This forwarding table associates a content-based address to each external interface and to the local application interface. From the router's perspective, it is convenient to represent the local application interface uniformly as an external interface. So, in the following, we will use interface I_0 as the link to local applications, and interfaces $I_1 \ldots I_k$ as the links to adjacent routers.

source
attribute$_1$ attribute$_2$ $\ldots$

Figure 2: High-Level Structure of a Message Packet

The content-based forwarding table is used by a content-based forwarding function F_C that, given a message m, selects the subset of interfaces associated with predicates matching m. The result of F_C is then combined with the broadcast function B, computed for the original source of m. (As shown in Figure 2, each message carries the identifier of the source node.) A message is therefore forwarded along the set of interfaces defined by the following formula:

$$(B(\text{source}(m), \text{incoming\_if}(m)) \cup \{I_0\}) \cap F_C(m)$$

We describe an efficient implementation of this formula elsewhere [5].

3.4 Routing State

The content-based routing module of a CBCB router maintains a routing table that associates a content-based address p_x to each interface I_x. Consistently with the forwarding table, we use interface I_0 to represent local applications (therefore, p_0 is the content-based address set by local applications through *set_predicate*), and $I_1 \ldots I_k$ to represent actual network links. The information stored in this routing table is conceptually identical to that stored in the content-based forwarding table. In fact, the forwarding table is constructed and updated by mirroring the routing table. We will however treat the two tables as separate objects because in reality they might be implemented by two independent, specialized data structures. It makes sense to separate these two tables also because the routing protocol might allow for them to be out of sync at times.

3.5 Content-Based Routing Protocol

The content-based routing protocol of CBCB consists of two mechanisms for the propagation of routing information. The first is a "push" mechanism based on *receiver advertisements*. The second one is a "pull" mechanism based on *sender requests* and *update replies*. In this section, we will initially present the main features of these two mechanisms, and then detail their behavior and discuss options and optimizations.

3.5.1 Receiver Advertisements

Receiver advertisements (RAs) are issued by nodes periodically and/or when the node changes its local content-based address p_0. RAs carry this content-based address as well as the identifier of their issuer. Their purpose is to push routing information from a receiver out to all potential senders, thereby setting up the forwarding state necessary to deliver messages of interest to the receiver.

issuer
predicate ...

Figure 3: A Receiver Advertisement (RA)

The structure of an RA packet is shown in Figure 3. RAs are propagated throughout the network using the following combined broadcast and content-based protocol:

- *Content-based RA ingress filtering*: a router receiving through interface i an RA issued by node r and carrying content-based address p_{RA} first verifies whether or not the content-based address p_i associated with interface i covers p_{RA}. If p_i covers p_{RA}, then the router simply drops the RA.
- *Broadcast RA propagation*: if p_i does not cover p_{RA}, then the router computes the set of next-hop links on the broadcast tree rooted in r (i.e., $B(r, i)$) and forwards the RA along those links.
- *Routing table update*: if p_i does not cover p_{RA}, then the router also updates its routing table, adding p_{RA} to p_i, computing $p_i \leftarrow p_i \vee p_{RA}$.

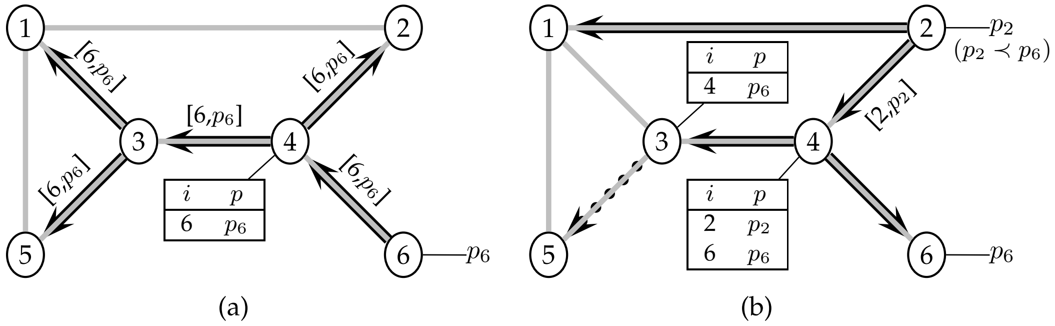

Figure 4: Receiver Advertisement Protocol (RA)

The example of Figure 4 illustrates the propagation of RAs. Intuitively, nodes in the graph represent routers, and light-colored edges represent direct (physical or overlay) links. Initially (Figure 4a) node 6, which is assigned content-based address p_6 by its local applications, issues the RA [6, p_6]. That RA propagates through the network following the broadcast tree rooted at node 6. The propagation path is represented in the figure by thick black arrows. The table attached to node 4 in Figure 4a represents the routing table of node 4 after node 4 has processed the RA. After this first RA gets distributed, node 2 issues another

RA carrying its content-based address p_2, which happens to be covered by p_6 (see Figure 4b). This second RA follows the broadcast tree rooted at node 2, however, using the ingress filtering rule, node 3 drops the RA. The result is that this second RA does not leave any trace at node 3, and is never forwarded along to node 5, as represented in Figure 4b by the thick dotted arrow. Figure 4b also shows the routing tables of node 4 and node 3 updated after the propagation of the second RA.

The content-based RA ingress filtering rule stops the propagation of redundant RAs. By applying content-based RA ingress filtering, routers avoid advertising content-based addresses along paths that are already set up with the necessary forwarding state.

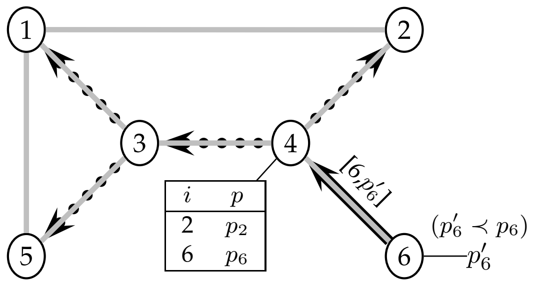

Figure 5: Address Inflation Caused by RA Ingress Filtering

Notice that, because of the ingress filtering rule, the RA protocol can only widen the selection of the content-based addresses stored in routing tables. In the long run, this may cause an "inflation" of those content-based addresses. For example, referring to the situation resulting from the propagation of the two RAs of Figure 4, if node 6 were to change its address to a more specific p'_6 (i.e., with $p'_6 \prec p_6$), then the resulting RA would be immediately discarded by node 4, according to the ingress filtering rule. The network would therefore maintain the forwarding state set by the first RA, which might cause the delivery of unwanted messages, that is, messages selected by p_6, but not by p'_6. (The forwarding function would never pass unwanted message up to applications though.) This situation is depicted in Figure 5.

3.5.2 Sender Requests

A router uses sender requests (SRs) to pull content-based addresses from all receivers in order to update its routing table. The results of an SR come back to the issuer of the SR through update replies (URs). The SR/UR protocol is designed to complement the RA protocol. Specifically, it is intended to balance the effect of the address inflation caused by RAs, and also to compensate for possible losses in the propagation of RAs.

issuer
request number
timeout

Figure 6: A Sender Request (SR)

An SR issued by s is broadcast to all routers, following the broadcast paths defined at each router by the broadcast function $B(s, \cdot)$. An SR carries the identifier of its issuer and a request number. The issuer identifier and the request number form a unique identifier of the SR, which is used to relate URs to SRs. An SR also carries a timeout that indicates the maximum amount of time that the sender is going to wait for a reply. The structure of an SR packet is shown in Figure 6.

A router processes an SR by forwarding it to downstream routers, and by generating a UR. Each UR carries a content-based address as well as the identifier of the SR that prompted it. URs are returned upstream

SR issuer
SR number
predicate
...

Figure 7: An Update Reply (UR)

to the issuer of the SR following the propagation path of the SR in reverse. The structure of a UR packet is shown in Figure 7.

Routers generate and process URs as follows:

- A leaf router in the broadcast tree immediately replies with a UR containing its content-based address p_0.
- A non-leaf router assembles its UR by combining its own content-based address p_0 with those of the URs received from downstream routers, and then sends its UR upstream.
- The issuer of the SR processes incoming URs by updating its routing table. In particular, an issuer receiving a UR carrying predicate p_{UR} from interface i updates its routing table entry for interface i with $p_i \leftarrow p_{UR}$.

In general, only the original issuer of the SR may use the URs generated by that SR to update its routing table. This is because the SR/UR protocol creates updates that are tailored to the issuer of the SR. We discuss a method to relax this restriction as well as other optimizations for the SR/UR protocol in Section 3.5.4.

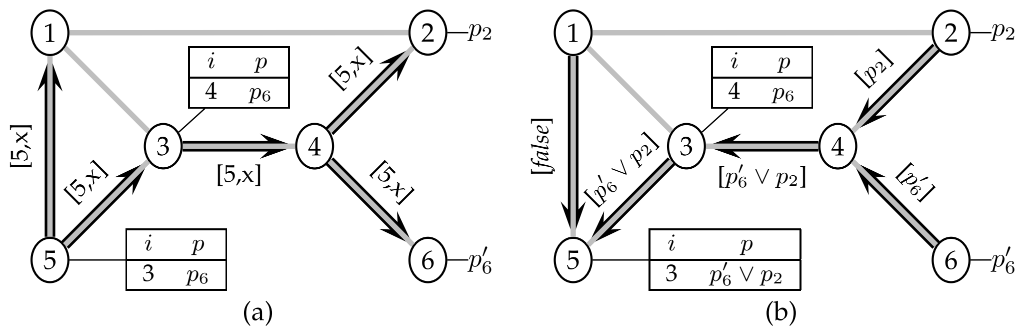

Figure 8: Sender Request Protocol (SR/UR)

The example of Figure 8 shows the propagation of SRs and of the corresponding URs. In the example, node 5 issues an SR, which is distributed to all other nodes following the broadcast tree rooted at node 5. Figure 8a shows this propagation as well as the routing tables of node 5 and node 3 at the time the SR is issued. Figure 8b shows the URs returned in response to the SR. For simplicity, we have omitted the SR issuer (5) and the SR identifier (x). Figure 8b also shows the routing tables of node 5 and node 4 after all the URs are processed.

3.5.3 Timeouts in the SR/UR Protocol

The propagation of SRs, from their issuer to every other node, is immediate. This means that a node receiving an SR must immediately forward the SR to all its neighbors that are downstream on the broadcast tree rooted at the issuer of the SR. The propagation of a UR, upstream towards the issuer of the SR, is instead triggered either by the receipt of all the downstream URs, or by a timeout.

Specifically, if the router is a leaf in the broadcast tree of the SR, then the router immediately sends its UR upstream. Otherwise, the router estimates a timeout t_d for the downstream routers, and forwards the SR downstream with timeout value t_d. The router computes the downstream timeout as $t_d = t - t_c - t_p$, where t is the (initial) timeout value reported by the SR, t_c is an estimate of the round-trip time from and to the router upstream (note that t_c includes the time already spent to get the SR from the upstream router, plus the time to ship the UR back to upstream router), and t_p is an estimate of the time necessary to compute the UR.

After forwarding the SR downstream, the router waits for the corresponding URs. The router computes and sends its own UR upstream as soon as it receives all the URs from downstream routers, or after a t_d timeout. If some of the expected URs are still missing after the t_d timeout, the router computes its UR using the downstream URs that were received, and proceeds by sending its UR upstream.

3.5.4 Optimizations for the SR/UR Protocol

The SR protocol may be quite expensive in terms of control traffic and computations within routers. In fact, an SR is essentially a broadcast packet that must reach, and be processed by, every router. In a network of N routers, each SR generates $2N$ packets (one SR packet and one UR packet per router). With every router periodically issuing SRs, the total amount of control traffic generated by SRs in a given time interval is proportional to N^2.

The amount of SR/UR traffic may therefore become a constraining factor for the scalability of CBCB to large networks. Fortunately, however, we can greatly reduce the amount of SR/UR traffic by reusing and caching URs, and by limiting the use of SRs to selected interfaces.

Caching and Reusing URs Two aspects of the SR/UR protocol contribute to the high cost of each SR. The first is the fact that an SR must reach every router in the network. The second is that only the issuer of the SR may use the resulting URs to update its tables. The optional optimization we discuss here aims at reducing the overall cost of SRs by limiting the propagation of each individual SR, and by allowing routers to share and reuse URs.

Recall from Section 3.5.2 that the SR/UR protocol mandates that only the issuer of an SR may use the corresponding URs to update its tables. This is the default behavior because, in general, each UR is specific to the broadcast tree of the issuer of the corresponding SR. As it turns out, this rule can be relaxed and, in some cases, a router may be able to use a UR requested by another router to update its own routing tables. The same router might also be able to cache that UR and use it later to immediately respond to another SR. Cached URs allow routers to block the propagation of SRs, greatly reducing SR/UR traffic.

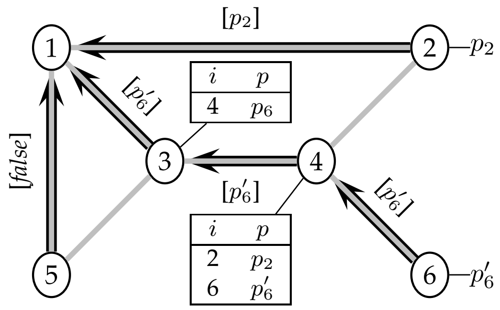

Figure 9: Opportunistic UR Processing

We use the example of Figure 9 to discuss the conditions under which routers can reuse and cache URs. Figure 9 shows the propagation of URs for an SR issued by node 1. For simplicity, the figure does not show the propagation of the SR. The configuration of routers corresponds to the scenario illustrated in Figure 8.

In this example, node 1 immediately updates its table upon receiving the UR from node 3, associating p'_6 to its interface to node 3. As specified in Section 3.5.2, in general, no other router may use the URs to update its table. The rationale for this restriction can be explained by considering node 3 in the example. The UR received by node 3 (from node 4) summarizes the addresses of the nodes that are downstream from node 3 on the broadcast tree rooted at node 1 (the issuer of the SR). This set of nodes includes nodes 4 and 6, and is different from the set that would be defined by an SR issued by node 3, which would include nodes 4, 2, and 6. It would therefore be a mistake for node 3 to use that UR to update its own table. In fact, as shown in Figure 9, node 3 maintains the address p_6 for its interface to node 4.

The UR mismatch of node 3 does not occur for every node. In fact, in the example of Figure 9, node 4 may safely use the UR received from node 6 to update its tables. Node 4 may also cache the predicate carried by that UR, and later respond to another SR, say from node 2, by immediately returning that predicate, without forwarding the SR to node 6. Note that no additional data structure is necessary to cache the UR, since that cached predicate is exactly what node 4 maintains in its routing table as the address of its interface to node 6.

The above examples can be generalized into a criterion that determines whether a UR can be reused and cached: A node u may reuse and cache a UR received from a downstream interface i, and sent in response to an SR issued by node s (where $s \neq u$), only if the set of nodes that are downstream from i on the broadcast tree T_s rooted at s is equal to the set of nodes, downstream from i, on the broadcast tree T_u rooted at u.

In practice, a router may be able to determine that it is safe to reuse and cache a UR by simply observing that one of its adjacent links is a *bridge* (i.e., a link that connects two otherwise disconnected parts of the network). This is the case for node 4 and node 6 in the example of Figure 9. In the CBCB scheme, the content-based layer does not have access to topological information, however the broadcast layer does, and may therefore be able to verify the caching criterion, or simply verify that a link is a bridge. Notice that in the particular case of a broadcast layer based on a single spanning tree, every link is a bridge, so every router may reuse and cache URs.

3.5.5 Controlled SRs

While caching and reusing URs allows routers to reduce the cost of individual SRs, a complementary optimization strategy is to limit the use of SRs. Instead of issuing SRs on a regular basis, and to all its interfaces (as specified in Section 3.5.2), a router might use SRs only when necessary, and/or only through a selected set of interfaces.

Recall that a router uses SRs mainly to trim its routing and forwarding tables, to compensate for the address inflation caused by the RA protocol, and ultimately to reduce the amount of unwanted messages. SRs are not useful per se, but rather become necessary in the presence of intense message traffic. This suggests that a router could control its use of SRs on the basis of the actual amount of messages output by the router. In particular, a router may use the following SR control policy: The router maintains a message counter C_i for each interface i. C_i is incremented every time a message is forwarded through interface i. The router issues an SR to an interface j whenever C_j exceeds a configurable threshold value $\overline{C}$. Every time the router issues an SR through interface j, the router also resets C_j to 0.

3.5.6 Address Simplification

In both the RA and SR/UR protocols, routers update their routing tables by combining content-based addresses. For example, in processing an RA carrying predicate p' from interface i, a router updates its routing table entry for i computing $p_i \leftarrow p_i \vee p'$. In this process, the router employs a simplifier to reduce the size and complexity of its tables. For example, assuming interface i is initially associated with predicate (price $>$ $50 \wedge$ price < 200), and assuming that an RA comes in from interface i carrying a predicate (price < 100), then the router would compute the new value for p_i as (price $> 50 \wedge$ price < 200) $\vee$ (price < 100) which can be simplified into (price < 200).

The simplification of the previous example is one that preserves the exact semantics of content-based addresses. It is also easy to imagine an optimization strategy whereby the router would use an *over-simplification* to trade some unwanted traffic for a reduction in the complexity of evaluating some predicates. For example, a predicate (price $> 50 \wedge$ price < 200) $\vee$ (price $= 250$) $\vee$ (price > 500) may be oversimplified into (price > 50). A precise description of the algorithms used for this type of over-simplification is outside the scope of this paper. Note, however, that this problem is conceptually very similar to the optimization and rewriting of queries in database systems.

3.5.7 Address Simplification

In both the RA and SR/UR protocols, routers update their routing tables by combining content-based addresses. For example, in processing an RA carrying predicate p' from interface i, a router updates its routing table entry for i computing $p_i \leftarrow p_i \vee p'$. In this process, the router employs a simplifier to reduce the size and complexity of its tables. For example, assuming interface i is initially associated with predicate (price $>$ $50 \wedge$ price < 200), and assuming that the an RA comes in from interface i carrying a predicate (price < 100), then the router would compute the new value for p_i as (price $> 50 \wedge$ price < 200) $\vee$ (price < 100) which can be simplified into (price < 200).

The simplification of the previous example is one that preserves the exact semantics of content-based addresses. It is also easy to imagine an optimization strategy whereby the router would use an *over-simplification* to trade some unwanted traffic for a reduction in the complexity of evaluating some predicates. For example, a predicate (price $> 50 \wedge$ price < 200) $\vee$ (price $= 250$) $\vee$ (price > 500) may be oversimplified into (price > 50). A precise description of the algorithms used for this type of over-simplification is outside the scope of this paper. Note however that this problem is conceptually very similar to the optimization and rewriting of queries in database systems.

4 Evaluation

Evaluation of the CBCB routing protocol was conducted by testing an implementation of the protocol within a simulated environment. The primary goals of our experiments were to understand the characteristics of the protocol with respect to three properties:

- *Main functionality:* does the protocol deliver messages to nodes that are interested in them?
- *Traffic filtering:* does the protocol prevent unnecessary message traffic?
- *Protocol scalability:* does the protocol produce a reasonable and stable amount of control traffic?

4.1 Experimental Setup

The topologies created for our experiments are flat, random, router-level topologies generated by BRITE [9] using the Waxman edge selection algorithm [13]. Bandwidth is assumed to be unlimited along all links.

In our experiments, we simulate the broadcast layer by implementing a global broadcast function, without simulating an actual broadcast protocol. In particular, we have tested a broadcast function based on a single spanning tree as well as one based on all-pairs shortest-paths trees. This latter function was implemented using Dijkstra's shortest-path algorithm, followed by a rewriting algorithm that, for each pair of nodes u and v, replaces the path $v \rightsquigarrow u$ with the reverse of the path $u \rightsquigarrow v$. This additional processing was put in place to satisfy the all-pairs path symmetry requirement.

In our simulations, every node in the network represents a CBCB router. Some nodes are also selected, at random with uniform distribution, to act as senders. Similarly, some nodes are selected to act as receivers. We fixed the density of senders at 20%, except in some experiments where we had no senders at all. The density of receivers was fixed for all the experiments at 75%. Both senders and receivers behave as Poisson processes. For senders, we experimented with rates of 2, 3, and 5 messages per minute. For receivers, we set the rate at which a receiver changes its predicate at 1 every 20 minutes, except in some experiments where we used static predicates. Notice that we (intentionally) used a relatively low output rate to single out the relative cost of control traffic.

Messages and predicates are composed randomly from distributions of names, values, types and operators. We do not discuss the numerous parameters that control the generation of messages and predicates

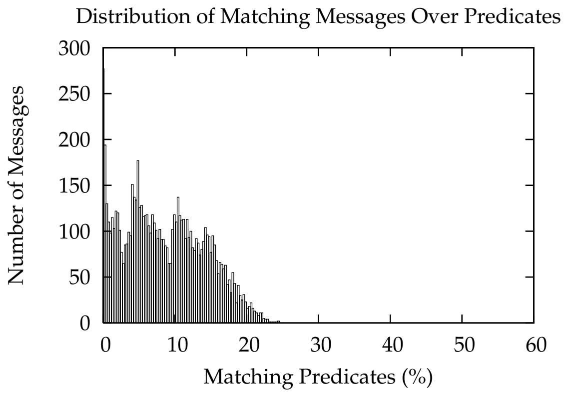

Figure 10: Matching Message Distribution

in this paper (we do that in our study of the performance of a forwarding algorithm [5]). Instead, we provide a high-level characterization of messages and predicates by showing, in Figure 10, the distribution of matching predicates for a typical workload. This figure shows that most messages match 5% to 15% of the predicates, that a significant number of messages do not match any predicate, and that no message matches more than 25% of the predicates.

4.2 Results

The main purpose of a content-based network is to get information to interested receivers. Our first set of experiments is therefore designed to verify that the network does indeed provide this basic functionality.

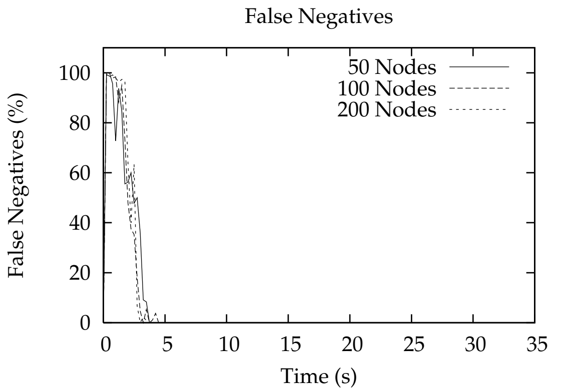

Figure 11: Percentage Of False Negatives Over Time

The results of this first set of experiments are plotted in Figure 11. The graphs show the percentage of *false negatives* over time. We compute this metric by computing the number of receivers that should receive each message. This value is computed by an "oracle" process that has perfect, instantaneous knowledge of all the predicates advertised throughout the network. For each message, we then compare the oracle count with the number of receivers to which the message is actually delivered by the CBCB network. The metric we show in Figure 11 is a moving average computed over a window of 0.25 seconds.

In these experiments, we use a static set of predicates. This means that each receiver advertises a predicate only once, at the beginning of the experiment. This setting is intended to highlight the time that it takes for CBCB (RAs) to establish the necessary routing information throughout the network. The experiments show that, in networks of up to 200 nodes, this initial routing latency is contained to under 5 seconds. We also performed experiments with periodically changing predicates. These experiments, however, produce essentially the same results as those reported in Figure 11, with a few, almost unnoticeable occurrence of false negatives after the initial setup period.

The data of Figure 11 show that, after a transient setup period, the CBCB protocol consistently delivers each message to all the interested receivers.

While the primary goal of a content-based network is to deliver messages to interested receivers, an implicit requirement is to limit the propagation of messages for which there are no interested receivers. In general, the effectiveness of CBCB is given by both the ability to reach all receivers, and the ability to do that without wasting too much bandwidth with unnecessary messages.

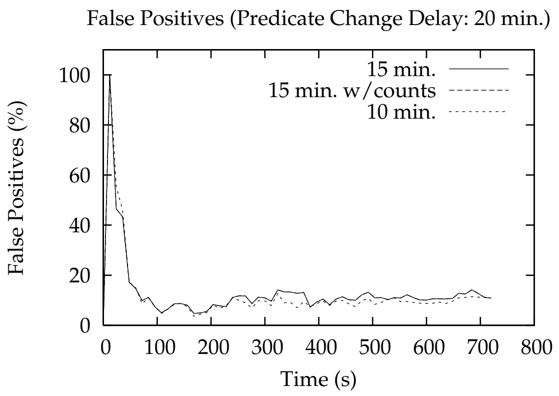

Figure 12: Percentage of False Positives Over Time

Figure 12 show the results of a series of experiments designed to highlight the amount of unnecessary messages transmitted by CBCB routers. The metric used in this figure is the percentage of *false positives* over the total number of messages processed by each router. A false positive is defined as a message received by a router that the router does not forward to any of its neighbors or to its local applications. Figure 12 shows a moving average computed over a window of 120 seconds.

The main conclusion we can draw from the graphs of Figure 12 is that, in steady state, the network incurs an acceptable, constant overhead of about 10% of the message traffic. The initial peak in false positives is due to the way we account for false positives in our experiments. In fact, at the beginning of the experiment, before the routing information is propagated throughout the network, routers have empty forwarding tables, and therefore count all the messages they receive as false positives. In these early stages of the experiment, this count includes all (and only) the messages that routers receive from their local applications, through the *send_message* function.

The three data sets of Figure 12 are obtained using different control parameters for the SR protocol. In particular, the first experiment uses periodic SRs issued every 15 minutes. The second experiment uses the score-based SR control policy described in Section 3.5.5, with a maximum frequency of one SR every 15 minutes. The third experiment uses periodic SRs every 10 minutes. Of these three variants, the one that yields the least amount of false positives is the one with the highest frequency of SRs (one every 10 minutes). This result is consistent with the main function of SRs within CBCB.

The accuracy afforded by the CBCB routing protocol is not without a cost in terms of control traffic. The amount of control traffic needed to keep predicates accurate across the network is driven both by the behavior of the receivers and by the parameters of the CBCB protocol. As receivers change their predicates more frequently, the routers must respond by sending more RA packets to alert senders of the new interested parties, and conversely the routers must send more SR packets to maintain predicate inflation and false positives under control.

Figure 13 shows the behavior of RA traffic over time. The metric used in Figure 13 is *percentage of RA*

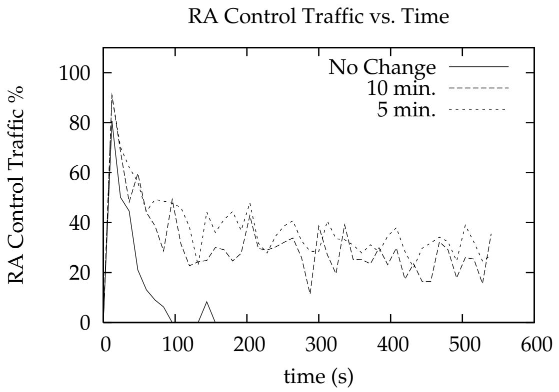

Figure 13: RA Control Traffic vs. Time

control-traffic which is defined as the number of RA packets over all network traffic. As in Figure 12, this is calculated as a moving average with a window of 120 seconds.

The three plots shown in this figure correspond to different receiver behaviors. All workloads are configured with a fixed output rate for senders. The line labeled "No Change" is the baseline where all receivers have a constant predicate. In this situation the RA traffic drops to zero after the initial startup period. The other experiments are configured with receivers that change their predicate on a regular basis. In the lines labeled "10 min." and "5 min." the receivers change their predicate an average of once every 10 and 5 minutes respectively. As expected, when receivers change their predicates more frequently, the RA traffic takes up a higher percentage of the overall traffic. However, as shown in Figure 13, the difference between these two situations is minimal. This is because routers limit the maximum frequency at which they release RAs. This policy is intended to prevent receivers that change their predicate very rapidly from overwhelming the network with RAs. Notice also that our particular choice of sender and receiver behaviors yields a high percentage of control traffic. This is because we experimented with only a few, slow senders. In a real application of content-based networking, we expect message rates of several orders of magnitude higher than the predicate-change rate. We also expect that message traffic would dominate the overall network usage.

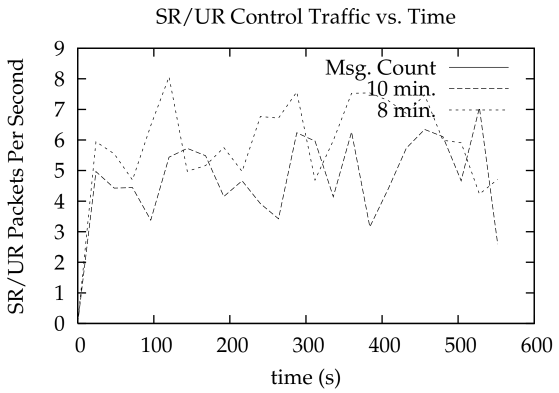

Figure 14: Average SR/UR Control Traffic vs. Time

The other facet of control traffic that must be considered are packets that are transmitted as part of the SR/UR protocol. In contrast to the RA control traffic, SR/UR traffic is largely driven by protocol parameters, and is therefore more directly controllable than the RA traffic. Figure 14 shows the average number of

SR/UR packets going through the entire network in one second at any given time. The experiments for Figure 14 were conducted without any message traffic.

The three data sets that are compared in Figure 14 correspond to the different SR control policies supported by our CBCB implementation. In the first data set labeled "Msg. Count", routers issue SRs only through their interfaces that exceed a given threshold of outgoing messages. Since these experiments were conducted in the absence of message traffic, this SR policy incurs no control traffic at all. The other two data sets correspond to routers that, irrespective of the message traffic they see, issue SRs at regular intervals of 8 and 10 minutes respectively. The experiments show that, even in these cases, the amount of SR/UR traffic remains stable at a very low rate of a few packets per second. (Notice that this rate is the aggregate traffic going through the entire network.) These results should be confronted with the measures of false positives reported in Figure 12. In particular, we note that a very small, constant flow of SRs is effective in keeping the volume of unnecessary message traffic under control.

5 Related Work

For its service model, content-based networking relates to a number of advanced network services and distributed-system technologies, including IP multicast [7], other rendezvous-based communication services such as the internet indirection infrastructure (*i3*) [11], intentional naming [1], and distributed publish/subscribe systems [3, 2]. A comparative analysis of the service model and general architecture of all these systems is available elsewhere [5]. However, as for the specific problem of routing in a content-based network, only a few of these systems offer solutions that are related to the work presented in this paper.

A large body of work has been devoted to advanced network services such as IP multicast. Indeed, IP multicast offers the most mature and scalable routing protocol among the systems we cited. Unfortunately, however, the service model of IP multicast is strictly less powerful than that of a content-based network, and there is no optimal way of using or adapting the multicast routing infrastructure to provide a content-based service. The same can be said of other extended multicast models such as *i3*.

The systems that are more closely related to this paper are the intentional naming system (INS) and distributed publish/subscribe systems such as Gryphon and Siena. In fact, all these systems offer a service model comparable to that of a content-based network, and they all use dynamic routing protocols.

INS, Gryphon, and Siena use a single data structure to serve as both a forwarding and routing table. We believe that this approach has serious problems. In fact, while conceptually simpler, it forces the use of data structures and algorithms that introduce unacceptable performance compromises for the forwarding function or the routing function, or both.

The general routing strategy is also a differentiator between our approach and both INS and Gryphon. Both INS and Gryphon propagate all routing information everywhere (intentional names in INS and subscriptions in Gryphon). Our primary routing strategy is instead limit the propagation of predicates using their semantic relations. Notice that this strategy is not applicable to INS because it conflicts with its name-resolution function. In fact, in order to be able to resolve names directly, every router in INS must maintain the name records for the entire network.

INS, Gryphon, and Siena, as well as all the distributed publish/subscribe systems that we know of, are restricted to an acyclic (overlay) topology. We believe that this is a major limitation, for obvious reliability problems, and for the administrative cost incurred in maintaining an acyclic network of servers. Also, the routing protocols of Gryphon, Siena, and INS are not designed to be robust with respect to failures in the communication of profiles. Two aspects of these routing protocols contribute to this lack of reliability: first, the protocols do not specify mechanisms to periodically refresh routing information. Second, the communication of routing information is stateful, meaning that servers exchange profiles that are incrementally merged into, or removed from, an existing set of profiles, and the protocol lacks a method to synchronize the routing tables as a whole.

6 Conclusion

In this paper we have presented the first routing scheme that realizes a content-based network service over a generic point-to-point network. This protocol consists of a traditional broadcast protocol combined with a specific content-based protocol. This latter protocol uses a "push–pull" mechanism for the propagation of routing information.

In order to evaluate the protocol, we implemented the protocol in a simulated environment and experimented with it, using various synthetic workloads. The results of these experiments confirm the validity of our design. In particular, we were able to show that the implements the content-based service with acceptable error rates, and with a stable amount of control traffic.

The routing protocol presented in this paper is part of, and builds upon our research work in the area of content-based networking, complementing our work on content-based forwarding [5]. As a natural progression of this work, we plan to study issues related policies and quality-of-service parameters in content-based networking, with the intent of incorporating these aspects into the design of improved routing and forwarding functions.

References

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *17th ACM Symposium on Operating Systems Principles (SOSP 99)*, volume 34 of *Operating Systems Review*, pages 186–201, Dec. 1999.

[2] G. Banavar, T. D. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *The 19th IEEE International Conference on Distributed Computing Systems (ICDCS '99)*, pages 262–272, Austin, Texas, May 1999.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, Aug. 2001.

[4] A. Carzaniga and A. L. Wolf. Content-based networking: A new communication infrastructure. In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, Scottsdale, Arizona, Oct. 2001.

[5] A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *SIGCOMM '03*, Karlsruhe, Germany, Aug. 2003. to appear.

[6] Y. K. Dalal and R. M. Metcalfe. Reverse path forwarding of broadcast packets. *Communications of the ACM*, 21(12):1040–1048, Dec. 1978.

[7] S. E. Deering and D. R. Cheriton. Multicast routing in datagram networks and extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–111, May 1990.

[8] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *ACM SIGMOD 2001*, pages 115–126, Santa Barbara, California, May 2001.

[9] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An approach to universal topology generation. In *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems MASCOTS '01*, Cincinnati, Ohio, Aug. 2001.

[10] Object Management Group. *Notification Service*, Aug. 1999.

[11] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 73–88, Pittsburgh, Pennsylvania, Aug. 2002.

[12] Sun Microsystems, Inc., Mountain View, California. *Java Message Service*, Nov. 1999.

[13] B. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9), Dec. 1988.

[14] T. W. Yan and H. Garcia-Molina. The SIFT information dissemination system. *ACM Transactions on Database Systems*, 24(4):529–565, Dec. 1999.