Fast Forwarding for Content-Based Networking

Antonio Carzaniga Jing Deng Alexander L. Wolf

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

University of Colorado
Department of Computer Science
Technical Report CU-CS-922-01 November 2001

© 2001 Antonio Carzaniga, Jing Deng, and Alexander L. Wolf

Abstract

This paper presents a new algorithm for content-based forwarding, an essential function in content-based networking. Unlike in traditional address-based unicast or multicast networks, where messages are given explicit destination addresses, the movement of messages through a content-based network is driven by predicates applied to the content of the messages. Forwarding in such a network amounts to evaluating the predicates stored in a router's forwarding table in order to decide to which neighbor router the message should be sent. We are interested in finding a forwarding algorithm that can make this decision as quickly as possible in situations where there are large numbers of predicates and high volumes of messages. We present such an algorithm and give the results of studies evaluating its performance.

1 Introduction

A content-based network is a novel communication infrastructure in which the flow of messages through the network is driven by the content of the messages, rather than by explicit addresses assigned by senders and attached to the messages [5]. In a content-based network, receivers declare their interests to the network by means of predicates called *filters*, while senders simply inject messages into the network at the periphery. The network is responsible for delivering to each receiver any and all messages matching the filter declared by that receiver. An ideal application for a content-based network is a publish/subscribe event notification service [1, 3, 8].

As in traditional address-based networks, the delivery function is performed incrementally by passing messages between intermediate nodes in the network. We say that messages flow from *upstream* nodes to *downstream* nodes until delivered. The delivery function consists of two interrelated subfunctions: *routing* and *forwarding*. Routing amounts to establishing flow paths through the network by compiling and positioning local forwarding tables at each node. A forwarding table contains the information necessary for a node to decide to which neighbor node or nodes a given message should be sent; the processing of a message at a node is the forwarding subfunction. Taken together, the forwarding performed at the nodes causes messages to be routed through the network. It is unfortunate that intermediate nodes are traditionally called "routers", when in fact they would be more appropriately called "forwarders".

Our concern in this paper is with the design of a fast forwarding algorithm for content-based networking. Clearly, the forwarding algorithm plays a critical role in the overall efficiency of the network. In the context of a content-based network, a forwarding table contains predicates representing the conditions under which a message should be forwarded to a particular neighbor. Forwarding amounts to evaluating the predicates against the content of each message arriving at the node. The predicates in a particular router's forwarding table are formed from the primitive filters presented to the network by receivers associated with nodes downstream in the message flow. We say that a message *matches* a filter if the predicate applied to it is true. The algorithm we seek must be well behaved in situations where there are large numbers of filters and high volumes of messages. We present such an algorithm and give the results of studies evaluating its performance. Our evaluation shows that our algorithm has good absolute performances in a variety of configurations of routers and application loads, including the extreme case of a single centralized router. We also show that, in the context of a network of routers, with a fixed number of neighbor nodes, our algorithm scales sublinearly in the number of filters, with almost no degradation of throughput over a certain number of filters.

There are two primary foundations for the work described here. The first is our development of a wide-area, content-based, publish/subscribe event notification service called Siena [4]. The overriding goal in the design of Siena is *scalability*, by which we mean support for large numbers of publishers, subscribers, and notifications spread across large networks. Our current implementation of Siena is structured as an overlay network of application-level routers leveraging any one of a number of transport-level networks, such as TCP or UDP.[1] Our focus has been on defining the basic routing principles of this overlay network, such as minimizing overall network traffic and calculating shortest paths, with little attention paid to forwarding.

The other primary foundation is the matching algorithm developed for the Le Subscribe publish/subscribe service [6]. Le Subscribe is a centralized system, in that there is a single server that is the target of all subscriptions and publications. This server is designed to efficiently match messages against subscriptions, essentially acting as a fast switch. The forwarding algorithm presented in this paper is an enhancement and extension of the Le Subscribe matching algorithm, tailored to the context of forwarding in a distributed network.

In the next section we provide some necessary details concerning content-based networking and give some examples of predicates and messages in terms of Siena subscription filters and notifications. Following that, we present our forwarding algorithm; the routing algorithm that generates, deploys, and determines when to update forwarding tables is beyond the scope of this paper. An experimental evaluation of the forwarding algorithm is then described. We conclude with a discussion of related work and future development plans.

[1] `http://www.cs.colorado.edu/serl/siena/`

2 Content-Based Networking

A content-based network is similar to a traditional address-based network in that it consists of *hosts* and *routers* connected by communication *links*. Hosts are nodes that have exactly one link, and act as senders or receivers of messages. Routers are nodes with more than one link, and act as dispatchers for messages that transit through them. Subsets of nodes may be directly connected to each other in *subnetworks*, thereby forming complete subgraphs, that are in turn connected to each other via routers. For simplicity, we ignore the internals of subnetworks and model them as single nodes.

As mentioned in the previous section, it is the mode of communication in content-based networking that differs significantly from traditional (unicast or multicast) address-based networking. A content-based network is a network in which 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 concept of content-based network is independent of the form of messages and predicates. Denoting the universe of messages as $\mathcal{M}$, and the universe of predicates over $\mathcal{M}$ as $\mathcal{P} : \mathcal{M} \to \{\mathit{true}, \mathit{false}\}$, we say that $\mathcal{P}$ and $\mathcal{M}$ define a *content-based addressing scheme*, which in turn defines the content-based network. Consistently we say that the predicate p_n advertised by n is the *content-based address* of the node n. We also say that a message m is implicitly addressed by its content to a node n with content-based address p_n if $p_n(m) = \mathit{true}$.

In the context of forwarding, we must refine these definitions somewhat. Notice that a router r may be connected to several neighbor nodes in the network. The role of r is to decide, for a given message received from an upstream neighbor, which downstream neighbors are to be forwarded a copy of that message. (The perspective of "upstream" and "downstream" for a particular router can either be fixed for all message traffic or it can differ for each individual message; this choice does not affect the discussion.) In essence, r is acting to its upstream neighbors as a proxy for the collective interests of its downstream neighbors. Following traditional networking terminology, we say that r presents an *interface* to its upstream neighbors. The content-based address of this interface is a predicate that is a disjunction of the predicates of its downstream neighbors. To distinguish the disjunction from its constituents, we refer to the constituent predicates as *filters*.

In our earlier work on the Siena event notification service we have defined what amounts to a content-based addressing scheme [5]. Because it has a convenient and concrete syntax and semantics, we use Siena *subscriptions* and *notifications*, respectively, to illustrate the more abstract concepts of filters and messages in this paper.

string	*carrier* = *UA*
string	*dest* = *MXP*
int	*price* = *600*
bool	*upgradeable* = *true*

Figure 1: Example of a Siena Notification Message.

A message is a set of typed attributes (Figure 1). Each attribute is uniquely identified within the message by a *name*, and has a *type* and *value*. For purposes of this paper, we consider the common types *string*, *integer*, and *boolean*. A filter is a conjunction of *constraints* on individual attributes (Figure 2). Each constraint

string	*dest* = *MXP*
int	*price* < *500*

Figure 2: Example of a Siena Subscription Filter.

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, the second constraint of the filter in Figure 2 matches those messages that contain an integer attribute named "price" with a value less than 500.

3 Forwarding Algorithm

The design of a forwarding algorithm involves the design of a forwarding table and of its processing functions. A schematic architecture of forwarding in a content-based network's router is depicted in Figure 3. A forwarding table is conceptually a map from predicates to interfaces of neighbor nodes

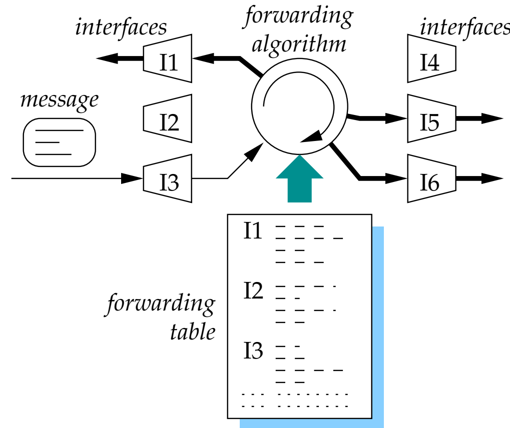

Figure 3: Forwarding in a Content-Based Network's Router

$$FwdTable : P \to I$$

where a predicate is a disjunction of filters, each one being a conjunctions of elementary conditions over the attributes of a message. An example of the logical contents of a forwarding table is shown in Figure 4, where the first column are the interfaces I_n of neighbors and the second column are the disjunction of filters $f_{n.m}$ mapping to interface I_n. Constraints on individual attributes within a filter are shown in the third column. This example is overly simple in that it is in general possible for the same filter to map to more than one interface. Forwarding an incoming message m amounts to computing the set of interfaces having

I_1	$f_{1.1}$	*string dest = Milano* *int price < 500*
	$f_{1.2}$	*string stock = DYS* *int quantity > 1000* *int price < 500*
I_2	$f_{2.1}$	*string airline = UA* *string orig = Denver* *string dest = Milano*
	$f_{2.2}$	*string dest = New York* *int price < 200*
	$f_{2.3}$	*string orig = Denver*
	$f_{2.4}$	*string airline = UA* *bool upgradeable = true*
I_3	$f_{3.1}$	*string stock = MSFT* *int price < 200*

Figure 4: Example Contents of a Forwarding Table

at least one filter matching m:

$$forward(m) = \{i \in I : m \quad matches \quad FwdTable(i)\}$$

The forwarding algorithm is based on a data structure representing the forwarding table. Figure 5 shows

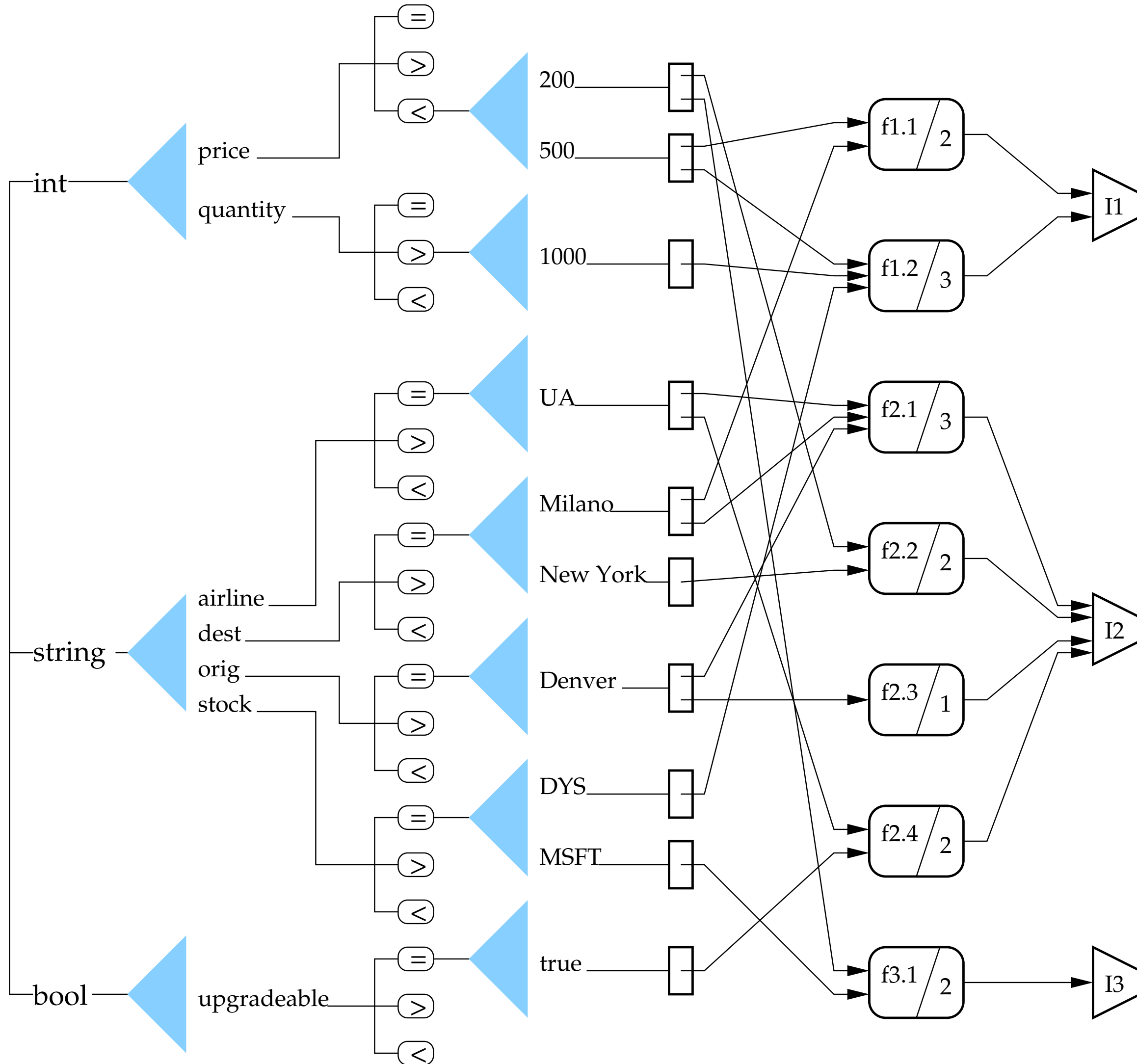

Figure 5: Representation of the Forwarding Table of Figure 4

a schematic view of that data structure for the example of Figure 4. Figure 6 shows a schematic view of a generic forwarding table.

With its four-level, left-to-right structure, the forwarding table reflects and supports the strategy adopted by the forwarding algorithm. At a high level, the algorithm works as follows: For a given message m, it considers each attribute $a_1, a_2, \ldots, a_k$ in m. For each attribute a_i, the algorithm starts from the left side of the structure and attempts to move forward through the table, first by satisfying single constraints, then by satisfying entire filters, which then lead to the choice of interfaces. The algorithm shortcuts the evaluation of filters whose interface has already been matched by other, previously evaluated attributes in the message.

The leftmost part of the forwarding table (see Figure 6) is a *constraint index*. The constraint index provides fast access to all the single constraints matching a given attribute a. The constraint index is organized

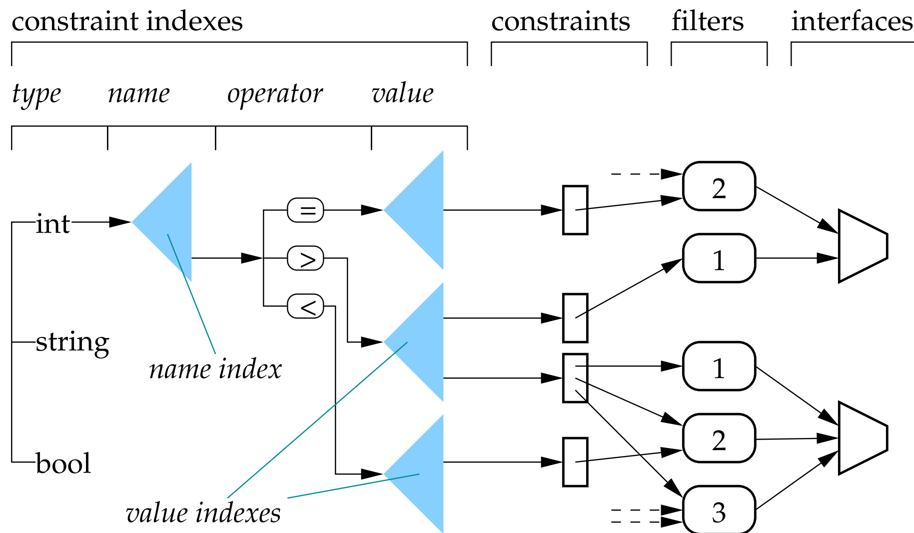

Figure 6: Generic Four-Level Representation of the Forwarding Table

as a chain of four subindexes based, respectively, on type, name, operator, and value. The *type index* is a simple switch that selects constraints based on their type. The *name index* uses attribute names as its keys, selecting from all the constraints having a's type, those that have a's name. The *operator index* splits the search into a number of branches that contain type-, and operator-specific *value indexes*. Every one of these operator-specific value indexes must be considered in the search. From each one of the value indexes, we can efficiently select those constraints that are matched by a's value.

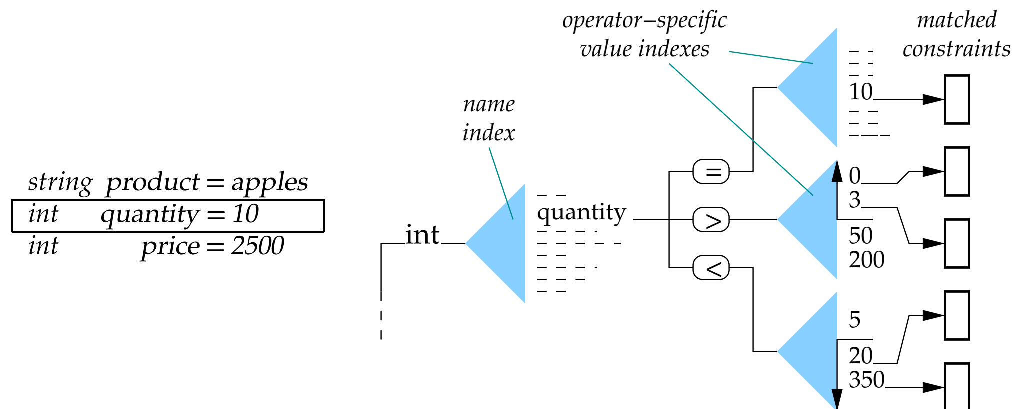

Figure 7: Example of Use of Constraint Index

The example of Figure 7 shows a selected fragment of a constraint index, processed while matching attribute [*int quantity* = *10*] of the given message. The search on type *int* and name *quantity* leads to the three operator-specific value indexes. Our current implementation supports constraint operators =, <, and >, and for each one of them uses sorted sets to index constraint values. The search process for the equality index is a simple binary search with exact match. The search process for less-than (greater-than) constraints performs a binary search to obtain the upper bound (lower bound) for the given value, and then returns all the values from there to the end (beginning) of the index. In the example of Figure 7, the upper bound for the attribute value 10 in the less-than constraint set is 20, and therefore the resulting matching constraints are [*int quantity* < *20*], and [*int quantity* < *350*], while the lower bound in the greater-than constraint set is 3, and the resulting matched constraints are [*int quantity* > *3*], and [*int quantity* > *0*].

Our current implementation supports a relatively limited set of operators, employing a simple indexing structure and search algorithm (i.e., binary search over sorted sets). Notice, however, that the constraint index is independent of the type- and operator-specific subindexes used to represent and search constraint values. In other words, specific indexing structures and search algorithms for other types and operators can be easily plugged into our constraint index. Examples of such structures are well-known indexes for prefix, suffix, and substring matching such as tries and PATRICIA trees, as well as indexes for ranges and multi-dimensional ranges for numeric values such as R-trees.

Processing an attribute a against the constraint index yields zero or more *constraint descriptors*. A constraint descriptor c holds a set of pointers to all the *filter descriptors* in which that constraint appears. Below, we refer to this set as *c.filters*. A filter descriptor f holds the total number of constraints contained in that filter ($f.size$), plus a pointer to the interface to which that filter is attached ($f.interface$).

In addition to the static data stored in the forwarding table, the forwarding algorithm uses two auxiliary dynamic data structures: a map $matched : I \rightarrow \{0,1\}$, implemented with a bit vector, that records which interfaces have already been matched, and a map $counters : filter \rightarrow N$ that counts the number of constraints matched for each processed filter.

```
proc forward(message m) {
    bitvector matched = ∅
    map<filter,int> counters = ∅
    set<interface> result = ∅
    foreach a in m {
        foreach c in matching_constraints(a) {
            foreach f in c.filters {
                if matched[f.interface] = 0 {
                    if f ∉ counters {
                        counters := counters ∪ <f,0>
                    }
                    counters[f] := counters[f] + 1
                    if counters[f] = f.size {
                        matched[f.interface] := 1
                        result := result ∪ {f.interface}
                        interface_count := interface_count + 1
                        if interface_count = total_interface_count {
                            return result
                        }
                    }
                }
            }
        }
    }
    return result
}
```

Figure 8: Pseudocode of the Forwarding Algorithm

The forwarding algorithm is shown in Figure 8. The outer loop iterates through the attributes of the input message. For each attribute a, the algorithm considers the filters partially matching a, that is those filters containing constraints matched by a. The algorithm shortcuts the evaluation of those filters whose interface has already been matched by a previous filter. The algorithm terminates as soon as the last interface is matched or when the last attribute is processed.

4 Evaluation

In order to evaluate our algorithm, we implemented it and studied its performance with a series of synthetic benchmarks, under varying combinations and types of filters, interfaces, and messages. In this section we present the results of our evaluation. We also made our implementation available on-line for further analysis and evaluation.[2]

4.1 Experiment Setup and Parameter Space

We implemented our algorithm in C++ and ran all the experiments on a 960Mhz computer with 512Mb of main memory. In addition to the main algorithm and data structures, we created some auxiliary programs to generate parameterized loads of filters and messages. In particular, we have identified and used the parameters listed in Table 1.

N	number of messages
I	total number of interfaces
F	total number of filters
AN	number of attributes per message
CN	number of constraints per filter
A	total number of distinct attribute names in messages
C	total number of distinct constraint names in filters
DT	distribution function for types in both filters and messages
DO	distribution function for operators in filters
SV	total number of distinct string values in messages
DSV	distribution of the given string values in messages
IV	total number of distinct string values in messages
DIV	distribution of the given string values in messages
DBV	distribution of boolean values in messages

Table 1: Scenario Definition Parameters

We performed all the experiments with 1000 messages ($N = 1000$). The total number of filters, F, is our primary independent variable, as well as the most important measure of scalability. Since our optimization strategy relies on the grouping of filters trough interfaces, another fundamental variable is I. Roughly speaking, I gives an indication of the characteristics of a router, its position, and it role in the larger content-based network. The six functions we selected for I are: $I = 2$, $I = 50$, $I = 100$, $I = 200$, $I = 500$, and $I = F$. $I = 2$ is an extreme case that represents a single sender or receiver endpoint. $I = 50, 100, 200, 500$ represent core routers, and in particular, lower values indicate a highly distributed network, or a router in the periphery of the network, while higher values are characteristic of more centralized networks or central routers. Finally the case of $I = F$ (one interface per filter) represents the case of a single centralized dispatcher.

For attribute names, we experimented with sets of 50, 100, and 1000 elements ($A = 50$, $A = 100$, and $A = 1000$). In order to use realistic names, we composed our sample sets by selecting random words out of a common dictionary. Then we used the same set of words for both attributes in messages and constraints in filters (therefore setting $C = A$). Notice that while this may be a simplification in defining the experiments, it in fact produces the most time-consuming scenarios for the forwarding algorithm. This is because having two completely overlapping sets of names maximizes the chances of having matching attributes and constraints. In the opposite extreme case of two completely disjoint name sets (one for attributes, and one for constraints) there would be no matches at all, and the time complexity for the forwarding algorithm would be $O(AN \log CN)$.

As for attribute values, we used a combination of dictionary values for strings, a range for integers, and a simple distribution for booleans. For strings, we compiled a list of words extracting SV words from the

[2] `http://www.cs.colorado.edu/serl/siena/forwarding/`

dictionary. For integer values we used a sequence of consecutive values from 0 to IV, and for booleans, we used a 50/50 distribution of values. For both integers and string values, we used a linear distribution to select values. As for attribute names, we used the same sets of values for both messages and filters. Notice once again that having a unified set of values and, moreover, using a nonuniform distribution for their random selection, increases the possibilities of having positive matches between constraints and attributes, thereby adding complexity to the matching process.

The distribution of types we used is 40% strings, 40% integers, and 20% booleans. The distribution of operators in filters is 60% equality, 20% less-than, and 20% greater-than. Other constant parameters or distributions are: uniform distribution in the range [1,5] for the number of constraints per filter (CN), and uniform distribution in [3,10] for the number of attributes per message (AN).

4.2 Results and Comments

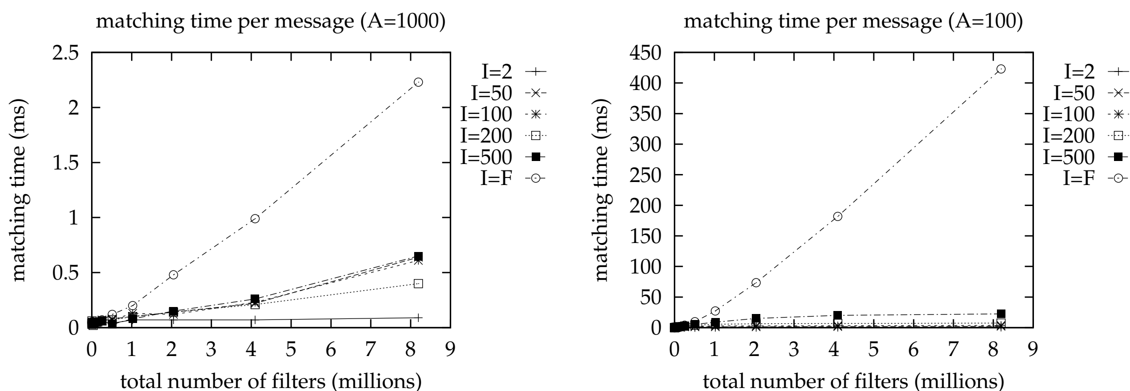

Figure 9: Overall Performance of the Forwarding Algorithm

Figure 9 shows a comprehensive view of the results of our experiments. The total number of filters ranges from one thousand to over eight million. First of all, notice that the algorithm shows reasonable performance even for the worst-case scenario, with matching time of about 400 milliseconds per message, in the presence of over 8 million filters and 8 million interfaces. As expected, the situation with $I = F$, which represents a centralized dispatcher, shows by far the worst behavior, whereas the scenarios with a fixed number of interfaces shows much better performance, with an almost flat cost curve. We will analyze those cost curves in detail below.

Figure 9 also shows two remarkably different behaviors corresponding to two different values of A ($A = 1000$ for the graph on the left, and $A = 100$ for the graph on the right), with over two orders of magnitude difference in matching time. As we can see from the distribution of matches shown in Figure 10, this difference is explained by the fact that the number of distinct attribute and constraint names A is a determinant factor in the overall distribution of matches.

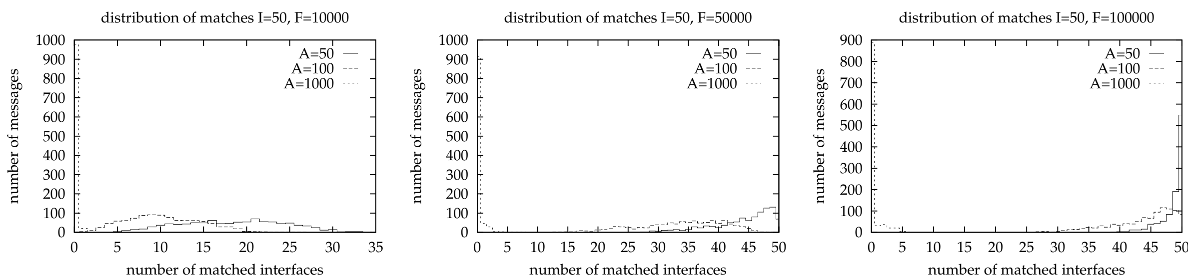

Figure 10: Distribution of Matches as a Function of the Total Number of Attribute Names

In particular, with large sets of attribute names (e.g., $A = 1000$), the probability of finding an attribute name in a constraint set is very low, which means that only a few messages will reach one or a few more interfaces. The forwarding algorithm is much faster in this case because most attributes from the input message will not find any corresponding constraints in the forwarding table, and therefore the forwarding algorithm will quickly skip through them.

The opposite situation occurs with smaller sets of names ($A = 100$ and $A = 50$): the probability of an attribute matching one or more constraints becomes much higher, thus keeping the forwarding algorithm busy, matching filters out of constraints. Notice that, in the presence of numerous filters (e.g., when $F = 100000$), messages tend to go to several interfaces, and a high percentage of them ends up going to all interfaces. As we show in detail below, this is also the situation in which our optimization becomes very effective.

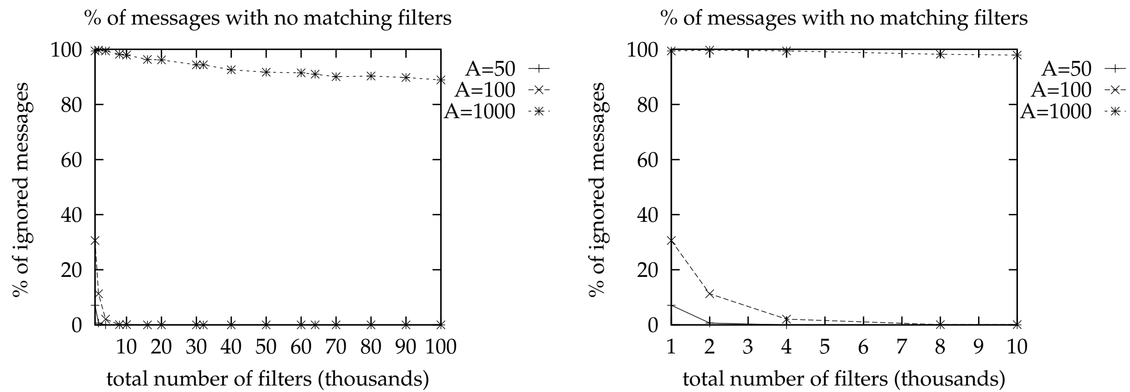

Figure 11: Percentage of Ignored Messages as a Function of the Total Number of Filters and Distinct Attribute Names

Figure 11 uses another reduction of the distribution data of Figure 10. This time we show the percentage of ignored messages—that is, messages for which there is no matching filter. As we said, the forwarding algorithm is very efficient in dropping those messages, so that percentage is a determining factor for the performance analysis. The two graphs show the same data set at two different scales over the number of filters. The graph on the left covers the range 1000–100000, and emphasizes the scenarios with $A = 1000$. As can be seen, in those cases most messages have no matching filter (90% of messages are ignored). The graph on the right focuses on a smaller scale of filters (1000–10000) to highlight the cases of $A = 100$ and $A = 50$. For those values of A, most messages are forwarded (30% down to 0% are ignored). In essence we can use A as a regulator for the percentage of matches.

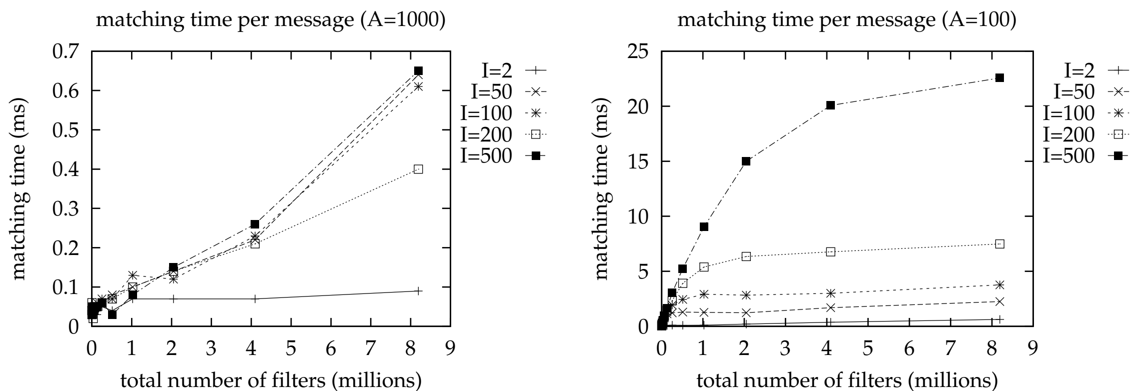

Figure 12: Performance of the Forwarding Algorithm with Fixed Sets of Interfaces

Figure 12 shows the same data sets of Figure 9, but without the extreme case of $I = F$. Again, notice how the two values of A produce very different results. Notice that the worst case of $A = 100$, in which all the messages are matched by one or more filters, shows a good matching time. In particular, even in the presence of as many as 8 million filters over 500 interfaces, the matching time remains within 25 milliseconds per message. More importantly, notice that all the curves become essentially flat for high numbers of filters, and that they level off at a value that is roughly proportional to the number of interfaces. This trend proves the effectiveness of our optimization based on a short-circuit evaluation of filters over interfaces.

4.3 Summary of Evaluation

Our experiments have shown that our forwarding algorithm has good absolute performance and good cost amortization over a variety of loads. In particular, we found that

- the *constraint index*, which provides fast lookup for attribute names over constraints, allows quick processing of attribute names that have no matching constraints, and
- the *short-circuit evaluation* of filters associated with the same interface greatly reduces processing time in the case where a single message may match a large number of filters.

The nice property of these two techniques is that they provide complementary optimizations in two application-dependent situations. The first one reduces processing time for messages matching a few constraints or no constraints at all, especially in the presence of a few filters, while the second one acts as a cutoff valve for the forwarding algorithm for messages that match several filters, and especially under heavy loads of filters.

5 Related Work

The idea of content-based networking is a natural evolution of our previous work on distributed event notification [4]. In particular, the problem of forwarding messages is similar to the problem of matching event notifications against subscriptions, which is also analogous to the problem of matching trigger rules in an active database. Other researchers have studied this problem, and proposed solutions based on various forms of decision trees and indexing structures for subscription predicates [1, 2, 6, 7].

The work presented in this paper is based on an algorithm and data structure proposed by Fabret et al. with their Le Subscribe system [6]. Fabret et al. focus on streamlining the matching algorithm by making it "cache conscious", and clustering filters for faster constraint checking. However, similar to all other optimized publish/subscribe matching techniques, they apply their algorithm in the context of a centralized server. By contrast, our general idea is to design a content-based communication *network*, in which several distributed routers cooperate to achieve the end-to-end effect of a single content-based notification dispatcher. In that context, we see and take advantage of an opportunity for further optimizations. In particular, we exploit the additional grouping of filters (by interface) with a shortcut evaluation procedure. Notice also that some of the clustering techniques proposed by Fabret et al. are applicable to this procedure in our forwarding algorithm.

Based on published experimental results [2, 6] and on the experiments we performed, we found that for the scenario with no grouping of filters (i.e., representing a centralized server) our forwarding algorithm is on a par with the state-of-the-art matching algorithms.[3] Our algorithm achieves additional cost (processing time) amortization when used in the proper context of a content-based router, as discussed in Section 4.2.

6 Conclusions

In this paper we have presented an algorithm and an associated data structure for fast content-based forwarding of messages. This algorithm is particularly suitable for the implementation of the forwarding

[3] A detailed and precise comparative analysis is beyond the scope of this paper.

algorithm of routers in a content-based network. Our algorithm, based on the general structure proposed for the Le Subscribe system, is specifically designed for content-based routers, and takes advantage of their fixed or limited number of output interfaces. In order to evaluate our algorithm, we have implemented it, and we have tested its performance under various configurations. From these experiments we found that the algorithm has good overall performance, even in the configuration corresponding to a single centralized server. The experiments also confirmed the validity of our optimization technique. We have also made our implementation available on-line for reference and further evaluation.

In the immediate future we plan to integrate our algorithm within our Siena distributed event notification service. As a natural progression of this work, we plan to attack the hard problem of routing in a content-based network. With Siena we have already defined the basic concepts of content-based subnetting and supernetting, and we have implemented what amounts to a routing table. Using that as a basis, we plan to study and develop optimized data structures for routing, as well as efficient and robust routing protocols for content-based networks.

Acknowledgments

The authors would like to thank David Rosenblum for his contributions to the design of Siena. The work of the authors was supported in part by the Defense Advanced Research Projects Agency, Air Force Research Laboratory, Space and Naval Warfare System Center, and Army Research Office under agreement numbers F30602-01-1-0503, F30602-00-2-0608, N66001-00-1-8945, and DAAD19-01-1-0484. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Air Force Research Laboratory, Space and Naval Warfare System Center, Army Research Office, or the U.S. Government.

References

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *Eighteenth ACM Symposium on Principles of Distributed Computing (PODC '99)*, pages 53–61, Atlanta, GA, May 4–6 1999.

[2] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient filtering in publish-subscribe systems using binary decision diagrams. In *Proceedings of the 23th International Conference on Software Engineering*, pages 443–452, Toronto, Canada, May 2001.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Proceedings of the Nineteenth ACM Symposium on Principles of Distributed Computing (PODC 2000)*, pages 219–227, Portland, OR, July 2000.

[4] 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.

[5] 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, AZ, Oct. 2001. In conjunction with the International Conference on Computer Communications and Networks ICCCN.

[6] 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, CA, May 2001.

[7] J. Gough and G. Smith. Efficient recognition of events in a distributed system. In *Proceedings of the 18th Australasian Computer Science Conference*, Adelaide, Australia, Feb. 1995.

[8] D. S. Rosenblum and A. L. Wolf. A design framework for Internet-scale event observation and notification. In *Proceedings of the Sixth European Software Engineering Conference*, number 1301 in Lecture Notes in Computer Science, pages 344–360. Springer–Verlag, 1997.