# CONTENT-BASED EVENT ROUTING WITH FILTER PROPAGATION

BY DAN DAVIS

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Prof. Manish Parashar

and approved by

New Brunswick, New Jersey

October, 2003

## ABSTRACT OF THE THESIS

# Content-Based Event Routing with Filter Propagation

**by DAN DAVIS**

**Thesis Director: Prof. Manish Parashar**

Publisher/subscriber is the emerging interaction paradigm in loosely coupled applications due to its inherent scalability. However, addressing the heterogeneity and dynamism in user interest and system state remains a key challenge in publisher/subscriber systems. Existing distributed event systems address these concerns at centralized hubs, at end points, or require flood broadcasted advertisements. Many systems have been implemented and tested on small overlay networks with the hope that the algorithms will scale to larger networks.

This thesis presents a distributed routing protocol for a publisher/subscriber event system that uses multicast routing trees to propagate content-based event filters toward the publishers in a widely distributed network. Filter propagation makes this system naturally distributed, efficient, and scalable. Network simulation results demonstrate that the protocol reduces unwanted event traffic and adjusts to changes in subscription and network topology. The compatibility of this protocol with wireless multicast routing protocols, and the minimization and approximation of content-based predicates are also discussed.

# Acknowledgements

I would like to thank my research advisor Dr. Manish Parashar for his invaluable guidance, support and encouragement during the course of this work and throughout my graduate studies at Rutgers. My friends have been patient with technical conversations and my need for distraction. My wife has proofread the document and supported and encouraged me patiently. My parents have somewhat refrained from asking about my progress.

## Dedication

To my loving wife, Sarah

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1 Introduction

Distributed publisher/subscriber systems route asynchronous events over a network from a publisher to multiple subscribers. These systems are typically implemented as middleware for use by applications. Publisher/subscriber systems promote loosely coupled application design, i.e. application processes on different computers communicate using the publisher/subscriber middleware. Each computer may publish events and/or subscribe to events.

Publisher/subscriber is one of the classic communication patterns. There are many applications that use the pattern and many middleware systems that provide publisher/subscriber semantics. Applications include system and network management, data collection in sensor networks, and document change distribution [1], [2], [3]. Publisher/subscriber systems can be as general purpose as IP multicast or as specific as the Salamander middleware for collaboratories [4]. The Open Grid Services Infrastructure (OGSI) specifies web-based event systems to improve the management of Grids [5]. Some even believe that solving the problems that prevent truly wide-scale event systems could enable a new wave of increased utility and autonomy of information technology [6].

To fulfill the needs of their target applications, distributed publisher/subscriber systems must meet different requirements. These requirements include subscription, filtering, aggregation, quality of service, scalability, and support for wireless and mobile environments. Scalability, subscription, and filtering on a wide area network are the key requirements that define the problem addressed in this thesis. Traditional addressing and routing as exemplified by IP multicast is based on publisher and subscriber addresses and not on message content. As a result these systems are not scalable. Publisher/subscriber systems that use content for both routing and filtering can provide distributed filtering and improve scalability. A scalable system performs better, and may more easily meet quality of service constraints. Moreover, distributed systems can be hard to manage and must meet a minimal level of self-configuration and tolerance of change to scale to large networks and support mobile clients [7].

In approaching this problem, we consider several research areas that involve publisher/subscriber routing: Peer-to-Peer research, Collaboratory and Grid research, Semantic routing research, and research on traditional internet routing protocols for IP multicast. In particular, peer-to-peer researchers often suggest that peer-to-peer designs are close to the original spirit of internet protocols and distributed systems [8]. In keeping with this observation, this thesis aims to use both IP multicast routing protocols and filtering event systems as design patterns to meet the needs of modern applications of distributed publisher/subscriber systems.

## 1.1 Requirements of Publisher/Subscriber Systems

### 1.1.1 Subscription, Filtering, and Aggregation

Publisher/subscriber systems differ in how they handle user subscriptions. For instance, IP multicast allows a subscriber to subscribe to a multicast address. The publisher sends events to the multicast address. Both the publisher and subscriber must be aware of the content of the data, but the multicast routers between them have no awareness of the content. CORBA's event service is somewhat similar: applications subscribe to channels and publish to channels. The channel, like the multicast address, defines the events the subscriber will receive.

However, a particular subscriber may only be interested in a subset of data sent to a multicast address or channel. If the publisher/subscriber system is aware of the content of the data, then the system may support a richer model of subscription called filtering. In such a model, each subscriber describes the data they want when they subscribe. Each publisher sends events conforming to a certain format so that the system is aware of message content. The system only delivers events that match the subscription. Both content-based routing as described by Siena and semantic-networking research here at Rutgers provide this kind of benefit.

A publisher/subscriber system can be easily layered on IP multicast to format data in a particular way when sent and then filtered when received. However, all the events, even unwanted events, will be sent to all computers subscribed for events. As illustrated in Figure 1.1, if most stock quotes are for other symbols, a large number of messages will be required unless content and filters are considered.

Rather than wanting a subset of data, a subscriber may be interested in a sequence of related events occuring close to the same time. If these events occur together, then a new event can be generated as the aggregate of the events in the sequence. This requires that a monitor in the network look for the sequence, receiving all events that might be in the sequence.

### 1.1.2 Quality of Service

Applications with quality of service (QoS) constraints, including real-time constraints, impose additional requirements on publisher/subscriber systems. For example, the

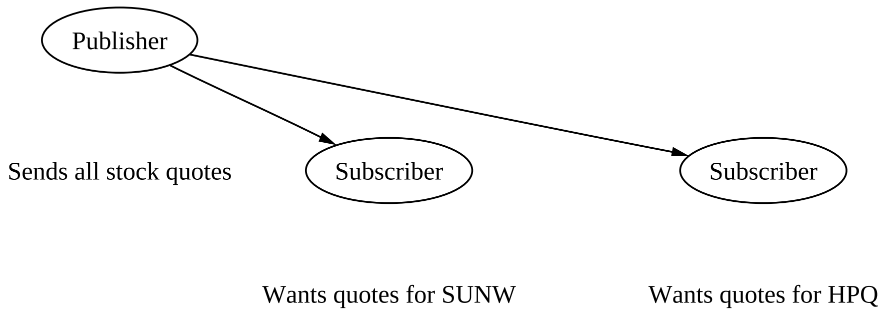

Figure 1.1: Why Routing Should Consider Filter Data

Salamander distributed system will store event data for some time so that all events can be delivered if the subscriber reattaches to the network. This provides a quality of service desirable to mobile and wireless subscribers. In contrast, both IP multicast and SIENA attempt a best-effort delivery, pushing QoS requirements to the applicaiton.

The architecture of a publisher/subscriber system may constrain the QoS qualities provided. Douglass Schmidt has enhanced CORBA's notication service to add support for real-time constraints [9]. Due to these changes, high priority events are delivered as quickly as possible regardless of the load on the real-time event service. However, even real-time behavior cannot make up for an architecture that is just too slow. A 250 millisecond deadline may be easily met with CORBA, but a 10 microsecond deadline will require a single process on a single computer solution.

Often, application specific quality of service requirements can be met by clever reuse of existing protocols. Van Jacobsen has shown how multiple IP multicast addresses can be used to represent the same video channel at different transmission rates to control network congestion [1]. Zhang's work on RSVP allows reliable media transmission by reserving buffer space for known multicast receivers [10].

The key differences between event systems and other publisher/subscriber systems lie in the qualities of service that are desirable. Events are likely to occur in exponential bursts whereas media-oriented publisher/subscriber data occur in uniform rate flows from one publisher to multiple subscribers. Qualities of Service like buffering for more reliable delivery also get at the difference between event and other publisher/subscriber systems. Finally, many event systems also have logging and accounting requirements that may be viewed as quality of service constraints. Van Jabobson's and Zhang's work primarily benefit media-oriented multicast. In contrast, the Salamander distributed system buffers events to avoid loss [4].

### 1.1.3 Scalability

Scalability is a difficult concept and is typically defined with respect to a number of independent and dependent variables. For example, as the number of publishers, subscribers, and routers increases, does the number of messages between participants and the latency for event delivery remain acceptable? How do increases to the size and rate of events affect the system?

The architecture and design of an event system govern its scalability. Each channel of CORBA's event service is hosted on a single computer, so CORBA's scalability is limited by the resources on that one computer. In contrast, IP multicast is widely scalable because each LAN subnet carries one copy of the message, and the routing information is distributed across multiple routers. Gnutella's success on college campuses can be viewed as a scalability experiment that underscores the need to consider communication patterns as well as distribution onto multiple servers.

A distributed system must also tolerate change and limit the amount of user configuration required at each computer involved. Most networks are not self organizing; when a computer is added to a network, some configuration is done to establish which computers are on the network, and which will act as a gateway to other networks. Most Gnutella clients require users to input a list of initial servers to contact. However, users are not required to input the topology and routes for the entire network. Peer to peer research shows that while some part of this configuration is essential, configuration can be avoided through further automation [7]. If well engineered, a distributed system can be more resilient than a centralized solution, but this is rarely achieved due to the modeling, testing, and forethought required.

### 1.1.4 Summary of Requirements

Table 1.1 lists desirable features of publisher/subscriber event systems. Support for mobile clients and security are included in addition to the features discussed above. Resiliency is included to clarify the interaction between reliable event delivery and other requirements, and to highlight the benefits of best-effort delivery.

| Requirement | Description |
|---|---|
| Subscription | tyically by channel, filter expression, or hybrid |
| Event Aggregation | supported in some event systems using filters |
| Reliable Delivery | may be optimized for bursty data flows |
| Resiliency | best-effort delivery reduces congestion |
| Real-Time | must be considered relative to typical system latency |
| Logging and Accounting | should be distributed in distributed systems |
| Scalability | with respect to what? |
| Mobile Client Support | mobile clients connect unreliably |

Table 1.1: Event System Requirements

## 1.2 Overview of the Thesis

This work studies algorithms for improved routing on a publisher/subscriber event system running over a connected overlay network. The algorithms improve scalability by providing increased selectivity through distributed filtering. The proposed algorithm is based on filtering using predicate expressions on typed tags attached to events. The filtering provided by the algorithm converges towards the optimal by propagating filters from the subscriber toward the publisher in a series of steps. The algorithm is studied in simulation on networks of a 100-nodes with from 10 to 80 publishers and subscribers. Experiments examine the steady-state efficiency of the distributed filtering, and the ability of the algorithm to adapt to changes in publishers and subscriber state and interests.

## 1.3 Contributions of the Thesis

The thesis makes the following contributions:

1. It introduces the concept of filter propagation as it applies to distributed publisher/subscriber systems.

2. It formulates filters as predicate expressions and clarifies concepts advanced by Siena and Semantic Networking by relating them to boolean algebra and probability.

3. It identifies new challenges for wireless ad-hoc networks by emphasizing the power savings available if more efficient filtering is used as compared to multicast routing.

## 1.4 Thesis Outline

The thesis consists of six chapters and is organized as follows. Chapter 1 serves as the introduction and presents the overview and contributions of this thesis. Chapter 2 outlines the related research and prior work in this field. Chapter 3 describes the routing algorithm based on typed tags and predicate expressions. Chapter 4 discusses the experimental study of this routing algorithm, the network topology, and implementation of the simulation framework. Chapter 5 details the experiments and analysis. Chapter 6 presents conclusions and directions for future work.

# Chapter 2
# Related Work

This chapter begins with an overview of several distributed event systems, focusing in particular on standards for computer system management. These are chosen from among the many middleware systems that provide event management because they are often used for managing computer resources, i.e. the consumer of events from these systems would care about events such as “filesystem full,” “disk media error,” or “computer shutting down.” As the sections below will demonstrate, all of these systems have problems with scalability. However, a massively powerful multicomputer, a Grid, and a modern enterprise may each include more than a thousand nodes, making scalability a critical issue.

Any event system that scales will have to be truly distributed. Consequently, I have studied the benefits and problems with IP multicast routing and peer to peer computing. IP multicast and its relationship to unicast routing are fundamental to a study of routing, and multicast is often used to improve scalability. Peer-to-peer systems focus on wide scale distribution of computation and data, and some general peer-to-peer frameworks explicitly address publisher/subscriber communication.

Some research has been done on publisher/subscriber event routing. This research treats publisher/subscriber as a part of the middleware. This chapter discusses Siena (Scalable Internet Event Notification Architectures) and Semantic Networking as examples of middleware targeted at publisher/subscriber semantics that are intended to scale to large networks.

## 2.1 Event Systems for System and Network Management

The cost of computer management grows as the number and capabilities of our computers grow. Businesses and institutions take various approaches to limiting the cost. For example, an organization may limit its purchases to a single vendor who will also provide support.

Event systems reduce the cost by reducing the chance of unanticipated failures and reducing the mean time to repair a system when a failure occurs. For example, if an administrator is aware of increasing hard disk seek errors ahead of time, he can install a replacement before a total failure. When an unexpected failure occurs, an administrator is aware of the failed system or module quickly. If a wide area event system is used, the vendor may be informed of the failure directly rather than by the administrator.

However, SNMP and CIM, industry standards for system and network management, are organized for traditional client/server computing and are not scalable. In contrast, Tru64 UNIX uses a publisher/subscriber event system whose events can have cluster scope. If computer systems are to become autonomic, in the sense proposed by IBM [6], then the control structures, the nervous system, must certainly be autonomic. One approach to the problem is an event system that includes all computers within an organization.

### 2.1.1 Simple Network Management Protocol Traps

The Simple Network Management Protocol (SNMP) envisions a client/server architecture with multiple clients contacting a larger number of servers. In this model, one client contacts multiple, widely deployed servers. The server is called an agent to distinguish from more traditional client/server designs [11], [12]. An SNMP trap is an asynchronous notification made by an SNMP agent to an SNMP network management client. However, one problem fundamentally limits the scalability of SNMP. There is no standardized way to subscribe to traps. American Power Conversion Corporation (APC) publishes an SNMP specification for their Uninterruptible Power Supply (UPS) devices [13]. This spefication allows an SNMP network manager to configure up to four addresses to which traps are sent. Tru64 UNIX's network management agent requires that trap receivers be manually entered in a file /etc/snmp.conf [14]. Even when an SNMP network manager may subscribe over the network, there is no standard for what SNMP object identifier should be used for such a subscription. As clients cannot subscribe to specific traps, many network management clients regularly poll a large set of agents. This can cause scalability problems on the network and within the computer on which the network management client resides.

One strength of SNMP is that a standard and registration process exists for trap descriptions. The process of registration allows multiple agents to broadcast SNMP traps to the same port unambiguously. This gets at the essential problem of namespace registration for wide-scale event systems.

Distributed event systems, such as that proposed in this thesis, aim to address the scalability issues of systems like SNMP without sacrificing the commitment to standards and data-description languages.

### 2.1.2 Common Information Model Indications

The Common Information Model (CIM), standardized by the DMTF, improves this model by allowing subscription via CIM to events called indications. In theory, indications can be received by anyone. In practice, CIM has problems with scalability despite providing subscriptions. Using Pegasus, the Open Group's CIM server, the receiver of a CIM indication must run a web server and parse the XML that encodes CIM data [15]. Since many network management clients will want the same indication, a CIM router for indications can save network bandwidth. However, each network management client is expected to register with the publishing server. The server sends a copy of the indication to each client.

This thesis offers a model for event routing that can be applied to CIM. The protocol proposed in this thesis could be implemented as a routing protocol for CIM indications.

### 2.1.3 Tru64 UNIX Event Manager (EVM)

HP's EVM system provides system wide event delivery with registration filters, a configurable persistent store for events, and a notification system that can include email, pagers, etc., at the system administrator's discretion. This system uses a shared root filesystem to extend the event delivery to a cluster. This limits EVM to the cluster boundary.

Working for HP and using SNMP and EVM, I have often wondered what might be possible in enterprise network management for a publisher/subscriber system that combined the benefits of SNMP, EVM, peer-to-peer self-organization, and traditional multicast routing. I believe that such a system would allow a more autonomic approach to system and network management [6].

## 2.2 Internet Routing Algorithms

A knowledge of internet routing, and multicast in particular, is critical to building distributed event systems for several reasons:

1. A study of internet unicast and multicast routing sheds light on any routing problem, in the sense that internet routing is fundamental to all distributed routing.

2. Multicast represents the best widescale effort to reduce unwanted message flow on the internet without building overlay topologies or restricting receivers.

3. Multicast is an implementation technology that can be mixed with overlay networks where necessary.

### 2.2.1 Routing Information Protocol

The Routing Information Protocol (RIP) was one of the earliest specified and standardized unicast dynamic routing protocols for the Internet. RIP is designed to dynamically determine the best route to a host through a network of routers that are each separately configured. RIP requires each router to periodically send its routing table to its neighbors [16]. A route consists of the network identifier, the cost, and the next hop router. Upon receiving a route message, a router finds its route to the network and compares it. If the new route will be better, the router updates its route to the network by setting the next hop router to the sender of the route and incrementing the cost. Networks to which the router is directly connected have a hop cost of 0. Routes with a lower aggregate cost are preferred to those with a higher cost. Routing algorithms that use an aggregate cost are called distance-vector routing protocols because the cost usually represents the distance in hops from the router to the target network.

RIP adjusts to changes in the network topology through a process called counting to infinity. If a network can no longer be reached due to a topology change, routers with a route to the network will exchange routes with increasing cost until the route is no longer considered reachable. This couting process is proven to converge due to the Bellman-Ford theorem from dynamic programming [17], [18]. In practice, infinity can never be reached. Therefore, some practical limit must be chosen so that the routing tables converge reasonably quickly. For this purpose, RIP designates 16 as infinity, limiting the width of the graph of networks. Other strategies such as split-horizon and poisoned entry also help distance-vector routing algorithms converge more rapidly [19].

### 2.2.2 Distance Vector Multicast Routing Protocol

The multicast capability of ethernet-based local-area networks has been exploited to provide standardized and supported multicast. By the time ethernet based local-area networks began to supplant token ring, IP multicast was standardized and protocols for multicast routing were in place. However, IP multicast requires hardware support in routers, and routers were not quick to support IP multicast.

To work around this, Stephen Deering [20] and other network scientists wrote software daemons to tunnel multicast traffic as normal point-to-point traffic. This created a network called the MBone. Networks with local multicast support could be connected through tunnels, using what in modern terms would be called an overlay network.

The canonical multicast algorithm is the Distance Vector Multicast Routing Protocol (DVMRP) [19], [20]. This algorithm leverages the routing information already exchanged for unicast routing to build multicast routing trees. When a router receives a multicast packet from a publisher that the router has never seen, the router uses the unicast routing information to determine what network interfaces point back towards the source. Other network interfaces are designated as child networks. The algorithm builds a distributed covering tree for each source host and multicast group combination as shown in Figure 2.1. This is accomplished via a process called Truncated Reverse Path Broadcast (TRPB) that treats the first packet received from a host/group combination as a reverse broadcast. Note that the tree followed by a multicast packet is not guaranteed to be a minimal spanning tree because the multicast packet is initially sent on all networks connected to the source, regardless of cost. Remarkably, the tree followed by a packet is not known to any node; instead, the routing state is distributed in the same manner as for unicast distance-vector routing. DVMRP uses the same route costs as RIP and therefore converges in the same manner.

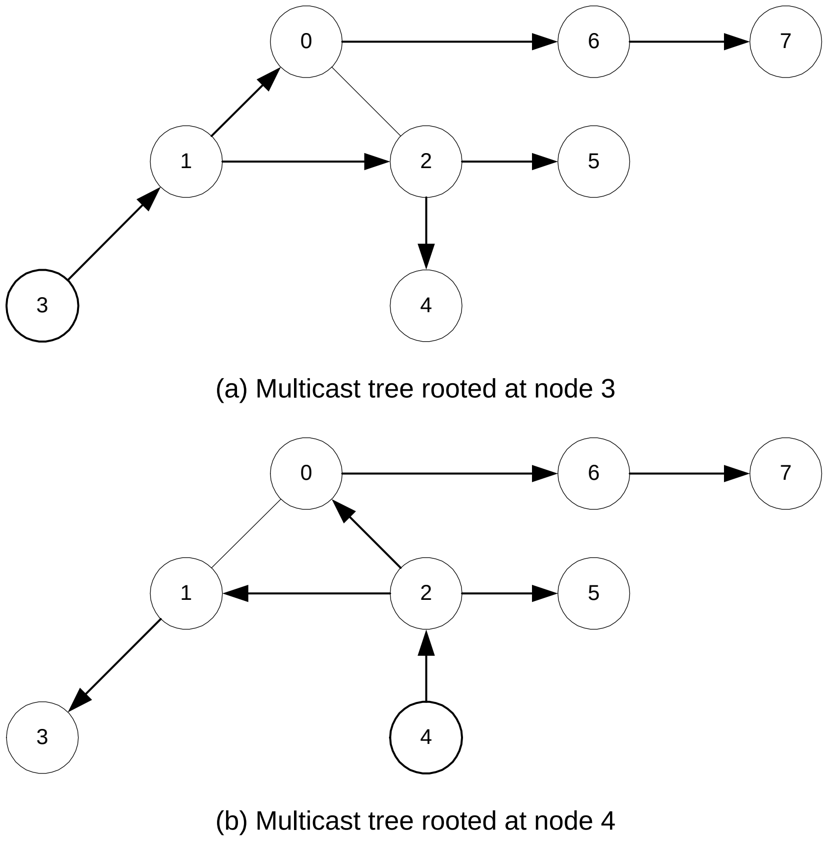
0
6
7
1
2
5
3
4
(a) Multicast tree rooted at node 3
0
6
7
1
2
5
3
4
(b) Multicast tree rooted at node 4

Figure 2.1: Multiple Multicast Trees on The Same Topology

In the standard for DVMRP, Deering suggest that periodic graft and prune messages allow a leaf router to tell routers closer to the source whether they have a subscriber. This suggestion is made not as part of the standard, but as a direction for further work. It works like this: upstream routers forward multicast packets until a prune message is received. The prune message indicates that the sending leaf router has no hosts subscribed to that multicast group. The prune message expires under the control of an associated timer. However, if a host served by the leaf router subscribes to the multicast group before the timer expires, then a graft message can cancel the previous prune. With respect to a particular multicast sender, a router has a list of child routes. If a prune is received from all child routes, the router can send a prune further upstream.

### 2.2.3 Link State Routing and Multicast

Other routing algorithms exploit plentiful memory, network bandwidth, and processing power to converge more quickly following changes in topology or state of networks. These algorithms broadcast topology changes throughout the network. Each router receiving a topology change updates its view of the network topology, calculates a minimum spanning tree or other tree, and uses this tree for routing. These algorithms are called link-state routing algorithms because they adjust more quickly to changes in topology, and can therefore adjust to changes in the state of a link, even if that change is temporary.

Open Shortest Path First (OSPF) is a large link-state routing standard designed for metropolitan and wide area networks [21]. OSPF specifies how link-state information is used for unicast, broadcast, and multicast on IP networks.

### 2.2.4 Issues with IP Multicast

The widespread availability and success of multicast has allowed researchers to identify issues with multicast. These include:

1. IP multicast expects a well-known list of multicast channels corresponding to television channels or radio stations. Today, meetings and presentations may require establishing a temporary channel such as is done by NetMeeting. Some RFCs have added ways to create a registry of temporary multicast addresses. However, there is no agreed upon standard for allocating temporary addresses.

   It would be desirable if allocating a temporary multicast address were as easy as allocating an ephemeral UDP/TCP port number. Allocating ephemeral UDP/TCP port numbers is an easier problem for two reasons. First, the port number does not affect network routing, but only delivery once a packet reaches its destination. Moreover, the operating system can be the sole authority on allocating addresses. In other words, there is no distributed protocol required before or after allocating an address. Because of this, it is only natural that there is a standard API for allocating temporary port numbers. However, it is possible for all operating systems to allocate temporary multicast addresses using a common API and protocol.

2. The standards for multicast include no provision for determining who may subscribe to a multicast channel. Ideally, the sender may want to restrict which hosts may listen to the channel directly or indirectly.

3. If a network's routers do not support multicast, then overlay daemons must be configured to tunnel multicast traffic to and from the network. `mrouted`, the multicast tunnel daemon, requires that the administrator know the address of another computer that runs `mrouted` and is connected to the MBone. More recent research on self-organizing overlays suggests that this may be needed only in the beginning, and that systems may optimize automatically once joining a multicast overlay.

4. Multicast does not inherently meet such demands as buffering, quality of service, and reliable delivery. I am aware of several papers that suggest that quality of service must be driven by the receiver. Van Jacobson has designed a protocol that controls and manages signal quality to match the bandwidth available for channel based applications such as television and radio [1]. Lixia Zhang designed the RSVP protocol to reserve buffer space within routers and thus assure packet delivery at a given packet rate [10]. Both of these address communication with a uniform packet rate, like a television station. However, a publisher/subscriber system oriented towards events should expect events to come in exponential bursts.

## 2.3 Peer-to-Peer Routing

Peer-to-peer systems such as Gnutella, Freenet, and JXTA are classified as data sharing peer-to-peer systems. However, there is a common aim between publisher/subscriber systems and the search operation on data-sharing peer-to-peer systems: both want to minimize the number of messages sent between nodes while meeting their other requirements. In event distribution, nodes receive messages if they are interested in acting on the event. In data-sharing peer-to-peer systems, nodes receive messages if they might contain the desired data. Changing the routing algorithm so that each node considers only the needed messages requires some understanding of a node's interest.

From a publisher/subsriber perspective, when a node searches Gnutella, the node publishes a search event to which other nodes subscribe. The search event is routed through the network. In Gnutella, the routing is a flooded broadcast with a hop limit. As the search request is forwarded, the route taken by each request is recorded. When a node has responses, these are source-routed to the originating node.

Gnutella's broadcast routing floods each query throughout the network. So, the broad scan done by Gnutella is likely to generate results. Some of the work on improving search overhead essentially involves improved routing based on response likelihood [22]. Other work improves routing by relating the topology of the network to data placement [23]. All of these seek to send fewer messages while tolerating the possibility of missing the desired result. Our work offers additional approaches to improving the efficiency of searching data-sharing peer-to-peer systems.

## 2.4 Multicast Routing on Ad-hoc Wireless Networks

An ad-hoc network is composed of mobile nodes without the presence of wired base stations to facilitate communication. Routing protocols for this environment are maturing quickly [24]. In this environment, mesh based protocols perform better than tree-based protocols in terms of reliable delivery under load and mobility [25]. Since many of the protocols studied refresh routing trees on demand, topology changing mobility can cause an increase in protocol overhead.

Due to the dynamic nature of ad-hoc wireless networks, on-demand algorithms rely on flooding for route discovery, and then store the collected routing information. Randomized ways to reduce the overhead of flooding without reducing its reliability offer a lot of benefits, especially when utilized for route discovery [26]. Some approaches taken for ad-hoc wireless sensor networks envision disposable units with high sensor density allowing more detailed study [27]. These devices use corner cube reflectors to receive messages. A response can be sent by modulating the reflection. However, the devices cannot rotate the reflectors. Inter-sensor protocols must rely on chance to find a neighbor with an appropriately aimed reflector. The constraints of this environment illustrate the challenges facing ad-hoc networks and the potential of randomized protocols.

The algorithm proposed in this thesis for filtering in a multicast tree may be adapted for mesh based multicast used for ad-hoc wireless multicast. The algorithm involves sending a subscription filter one hop up a multicast tree towards a publisher. This is only feasible in a tree, and so the wireless nodes routing information could be used to form an overlay tree embedded within the routing mesh. A more challenging approach to adapting the work of this thesis for a mesh-based routing algorithm would attempt to preserve the mesh, and treat the subscription filter as a request requiring confirmation by other nodes in the mesh.

## 2.5 Siena

Siena (Scalable Internet Event Notification Architecture)[28], [29], [30] and this thesis have many similarities:

1. Siena and this thesis do not make guarantees about event delivery. An effort is

made to deliver all events, but network congestion, topology and subscription changes can cause an event to be missed.

2. Siena and this thesis imagine an event as having a header consisting of name, type, and value tags that are used when routing the event. The event may contain additional information that is not available during routing.

3. Siena and this thesis view the interest of the subscriber as a boolean predicate that matches or does not match an event.

However, there are signifigant differences:

1. Siena advances the idea of event advertisements that form a routing tree based on the tags expected from different sources. This thesis forms a routing tree based on the sources that are producing events, and does not require broadcast flooded advertisements. The algorithm proposed in this thesis uses the publisher address to discover publishers and then builds a distributed routing tree per publisher. Siena will not scale as well in number of messages transmitted on the network, but should consume a lesser amount of memory as the number of publishers grows large.

2. This thesis does not automatically flood subscriptions up a routing tree, but instead propagates event filters towards an event source as unwanted events are received at leaf nodes.

3. Siena limits the subscription predicate to a conjunction of contraints on tags. When the same tag appears multiple times in a Siena subscription, the instances are always disjunctive [29]. Due to these constraints, Siena filters and subscriptions are easy to optimize when aggregating, but this also limits the expressiveness of the filtering language. For instance, the following expression cannot be exactly matched in a Siena subsrciption:

$$(symbol = HPQ) \wedge (price \geq 20.0) \wedge (price \leq 30.0)$$

Since the subscriber can only have one constraint on price, further filtering will be necessary after the event is received. If a subscriber used both tags by mistake, they would receive all events for symbol HPQ that have a price tag.

4. Siena supports subscriptions for patterns of mulitple events, which is a form of distributed event aggregation. Subscriptions for event aggregations can be propagated, but addressing this is beyond the scope of this thesis.

## 2.6 Semantic Routing

In his Masters thesis at Rutgers, Dhananjay Makwana pursued research on XML based semantic routing [31]. This work informs and inpsires mine in many ways. In particular, his thesis provides clear examples of the applications that benefit from publisher/subsrcriber patterns, and underlines that distributed routing is needed for scalable performance [2], [3].

The goal of semantic routing is to route data based on subscriber interest using XML to describe both data and interest. A subscriber's interest is called an interest profile. When a subscriber sends an interest profile, it causes a broadcast of interest profiles and a calculation of the subscription up a routing hierarchy. In this context, the decision where to route incoming data can be viewed as a transformation that outputs the downstream routes. The algorithm used is a link-state algorithm that works on an acyclic graph (hierarchy) of routers and subscribers.

Semantic routing relies on some of the same global information as DVMRP, showing that the problem of global information is essential to such distributed systems. For instance, even with multiple dimensions and containment hierarchies for data, problems can arise if two independent distributed agents decide to use a particular unique designator for communication with different meaning.

Makwana advances the idea of selectivity of interest profile in order to analyze the aggregate interest of the subscribers. I disagree on this point because selectivity is the inverse of probability. The selectivity of an interest profile is defined as the ratio of the number of input events to the number of output events matching the profile. That is, if a subscriber has selectivity 2, it will match 50% of the input events. Makwana then extends this to consider the selectivity of multiple subscribers. However, he assumes that the selection of one subscriber is independent of other subscribers. So, in his model, two nodes with selectivity 2 will together have selectivity 4 (will match 25% of the input events). However, if the two nodes both have the same interest profile, then each will match the same 50% of the input events. In general, the number of events that must be forwarded depends not only on the subscription, but also on the topology of the network and the distribution of input events.

## 2.7 Summary

| Feature | CORBA | SNMP | CIM | EVM | IP multicast | Siena | Semantic Networking |
|---|---|---|---|---|---|---|---|
| Subscription | channel | manual | filter | filter | channel | filter | XML profile |
| Event Aggregation | yes | no | no | no | no | yes | no |
| Reliable Delivery | yes | no | yes | no | no | no | no |
| Real-Time | yes | no | no | no | no | no | no |
| Logging and Accounting | no | no | no | yes | no | no | yes |
| Distributed | client/ server | client/ server | client/ server | cluster wide | WAN | WAN | designed networks |

Table 2.1: Comparison of Event System Features

Table 2.1 summarizes the features of the event systems discussed in this chapter. The features are taken from the requirements in Section 1.1. CORBA is included due to discussion of CORBA's features in that section. Most of the systems included are well established in industry. Siena and Semantic Networking represent evolving solutions to these problems. Any classification of this type is an oversimplication. CIM, EVM, and Siena support subscription by filter, but EVM's filter expression matches against a type hierachy with wildcards, and Siena and CIM use expression languages with different semantics. While EVM does not have real-time support, it supports a high event rate. To avoid oversimplification, the table tries to capture some of the details of system distribution. One row of the table lists not only whether each event systems is distributed, but how it may be distributed. While Semantic Networking supports large networks, these networks must be designed to be cycle free. IP multicast and Siena both support wide-area networks.

In addition to event systems, this chapter has also considered other topics. The discussion of multicast included history, current issues, and support on ad-hoc wireless networks. The relationship between peer-to-peer data sharing systems and event systems stimulates thought on how network server topology is related to publication and how predicate expressions might be applied to search routing.

# Chapter 3
# Filter Propagation

This chapter presents Filter Propagation, a publisher/subscriber event system closely based on DVMRP. The chapter begins with a conceptual overview of filter propagation. The chapter then discusses the data structures and messages corresponding to central concepts. These are events, subscription expressions, routing tables, and control messages. The chapter discusses the behavior of the system. As events and control messages are exchanged, nodes in the system react. Timeouts may be set and expire, leading to further actions. Finally, the chapter discusses improvements to filter propagation arising from optimization and approximation of subscription expressions.

## 3.1 Conceptual Overview

The underlying design principle of filter propagation is to forward an event too broadly rather than drop a desired event, and also to converge to perfect filtering in the network. The first event sent by a publisher is used as a discovery mechanism as it is forwarded throughout the network. As each node receives the event, the node builds its portion of a distributed routing tree specific to the publisher of the event. Then, the node forwards the event to additional nodes. This process continues until the event reaches all nodes in the network. A node forwards all events it receives until it knows enough to do otherwise. After that, the node sends a filter expression upstream towards the publisher of the event as soon as an unwanted message is received. Leaf nodes start the process off because a leaf node can immediately determine whether it is locally subscribed for the event. The following sections explain the details, but the concept is illustrated in Figure 3.1.

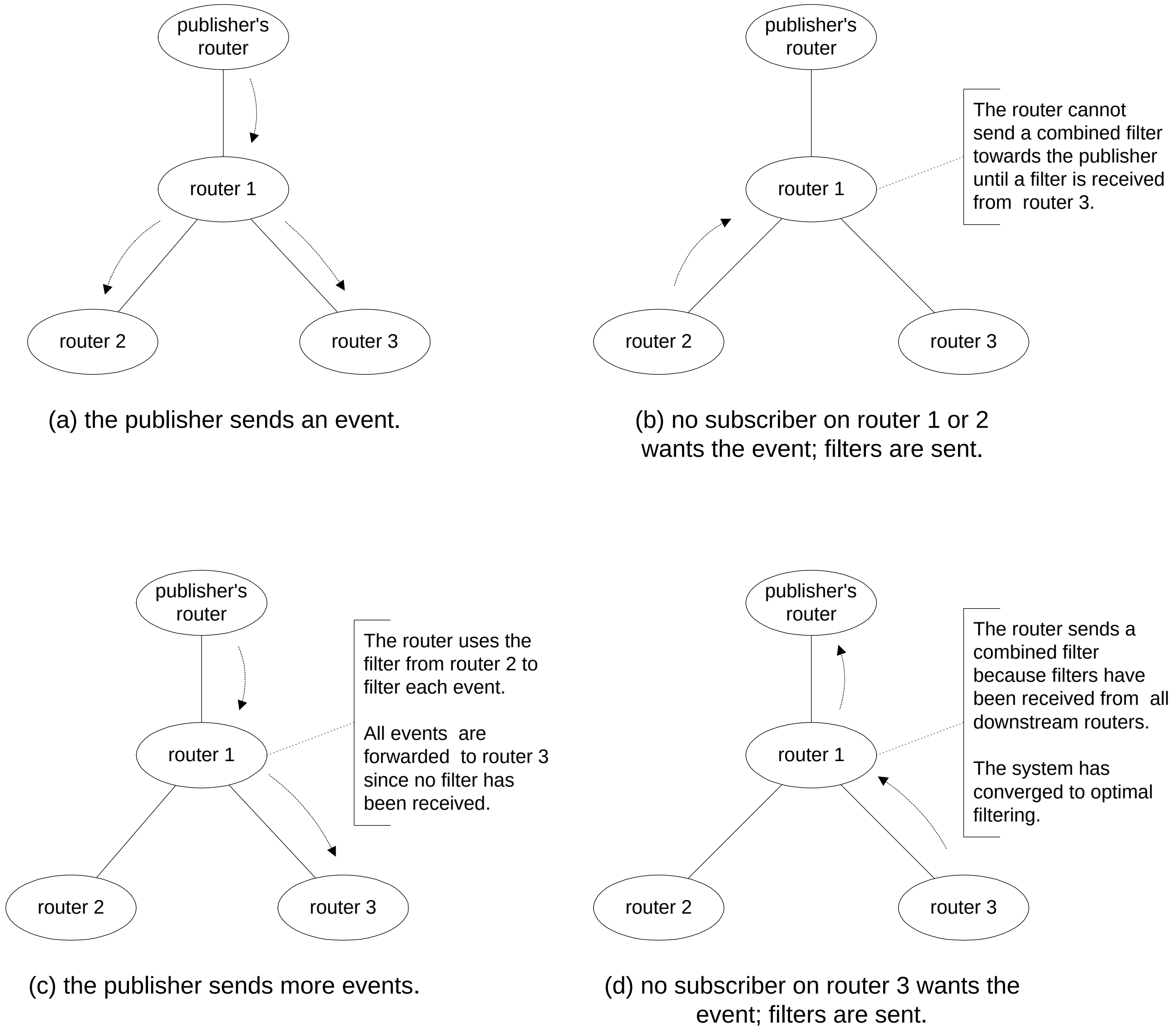
publisher's router
router 1
router 2
router 3
(a) the publisher sends an event.
publisher's router
router 1
router 2
router 3
The router cannot send a combined filter towards the publisher until a filter is received from router 3.
(b) no subscriber on router 1 or 2 wants the event; filters are sent.
publisher's router
router 1
router 2
router 3
The router uses the filter from router 2 to filter each event.
All events are forwarded to router 3 since no filter has been received.
(c) the publisher sends more events.
publisher's router
router 1
router 2
router 3
The router sends a combined filter because filters have been received from all downstream routers.
The system has converged to optimal filtering.
(d) no subscriber on router 3 wants the event; filters are sent.

Figure 3.1: Filter Propagation

## 3.2 Definitions and Data

### 3.2.1 Events

This thesis describes a distributed system for routing *events* from one or more publishers to zero or more subscribers. In 1.1.2, we discussed some key differences between event systems and other publisher/subscriber systems. This thesis focuses on scalability to the detriment of reliable delivery, logging and accounting.

In this system, an event consists of tags that describe the event. Each tag has a type, name, and value. The number and names of the tags are limited only by practical concerns. In addition, an event has a publisher address that is used to build routing trees. No timestamp is added to each event because there is no global notion of time in a widely distributed system. The event shown in Table 3.1 includes a time tag because stock publishers have a notion of when an event occurred. An event in a sensor network might instead include tags to identify the sequence and rate of samples.

| | | |
|---|---|---|
| publisher | address | 10.23.21.2 |
| string | symbol | “HPQ” |
| string | exchange | “NYSE” |
| float | price | 22.4 |
| int | change | 7 |
| time | when | 3/27/2003 8:47 PM |

Table 3.1: An Example Event

### 3.2.2 Subscription Expressions

A subscription defines the events that a client is interested in receiving. For example, in IP multicast, the multicast address is the subscription. The multicast address is like a channel in the CORBA event service in terms of how it affects received events.

In our system, a subscription is a predicate expression that can be evaluated using the tags in a specific event. The following are example expressions that are true when evaluated for the event shown in Table 3.1.

$$(when \geq 3/26/2003) \wedge (change \geq 5)$$

$$((symbol = \text{“}HPQ\text{”}) \vee (symbol = \text{“}MSFT\text{”})) \wedge (price \geq 20.0)$$

$$(symbol = \text{“}HPQ\text{”}) \wedge ((price \geq 30.0) \vee (change \geq 5))$$

Siena has a similar form of subscription expression, but restricts the form or class of the predicate expressions in an effort to simplify the problem of combining subscription expressions. This thesis does not restrict these predicate expressions; handling complicated expressions is essential to the problem of filtering.

For example, consider what happens in the network shown in Figure 3.2. Due to the subscriptions at router 2 and router 3, the upstream router must forward events matching the composite expression below. In this example, we imagine that router 2 and router 3 send positive subscriptions and the upstream router forwards events wanted by either router 2 or router 3.

$$\begin{aligned}
Subscription_2 =& (a \geq 5) \wedge (b \leq 5) \\
Subscription_3 =& (a \leq 2) \wedge (b \geq 10) \\
Subscription_{upstream} =& Subscription_2 \vee Subscription_3 \\
=& ((a \geq 5) \wedge (b \leq 5)) \vee ((a \leq 2) \wedge (b \geq 10))
\end{aligned}$$

The relationship between filtering and subscription is clarified by the use of boolean predicates. In our terminology, a filter is the negation of a subscription: a subscription describes what is wanted and a filter describes what is not wanted. However, it is important that this difference is not very important in designing a protocol. Due to the way the negation operation distributes over the conjunction and disjunction operation, a system operating on filters is functionally equivalent to one operating on subscriptions:

$$\begin{aligned}
Filter_2 &= \sim Subscription_2 \\
&= \sim ((a \geq 5) \wedge (b \leq 5)) \\
&= \sim (a \geq 5) \vee \sim (b \leq 5) \\
&= (a < 5) \vee (b > 5) \\
Filter_3 &= \sim Subscription_3 \\
&= (a > 2) \vee (b > 10) \\
Filter_{upstream} &= \sim Subscription_{upstream} \\
&= \sim (Subscription_2 \vee Subscription_3) \\
&= \sim Subscription_2 \wedge \sim Subscription_3 \\
&= Filter_2 \wedge Filter_3 \\
&= ((a < 5) \vee (b > 5)) \wedge ((a > 2) \vee (b > 10))
\end{aligned}$$

Therefore, subscription expressions can be sent to narrow the interest of an upstream router as if they were filtering expressions. They need not be converted, and are kept in the form most directly related to the interests expressed by end-users and applications.

### 3.2.3 Routing Tables

Every node that participates in the system can route events whether or not it is also a publisher or subscriber. However, it is most interesting to consider nodes with multiple interfaces. An event router's interfaces are its connections to other nodes in the system. These may be LANs, persistent connections in an overlay network, or even connections in a self-organizing overlay as described in other research [7].

A node uses the events it receives as a form of discovery to identify the publishers within the system. For each event publisher observed by a node, the node computes a set of child interfaces using distance vector information. This computation provides a distributed routing tree for each publisher that is rooted at the publisher, and this assures that event routing is loop free. For each combination of interface and publisher, the node keeps a subscription expression that is used to test whether to forward or deliver the event. The expression can be viewed as a list that is evaluated as a disjunction of sub-expressions. The expression for each interface and publisher combination is initially the literal “true.” Table 3.2 shows the routing table at router 1 in Figure 3.2 after the first event is received at router 1. Table 3.3 shows the routing table once the small network has converged. In an implementation where the child interfaces are LANs that may each contain more than one child node, an additional child node column would be needed in the routing table to distinguish the expressions belonging to each child node on an interface.

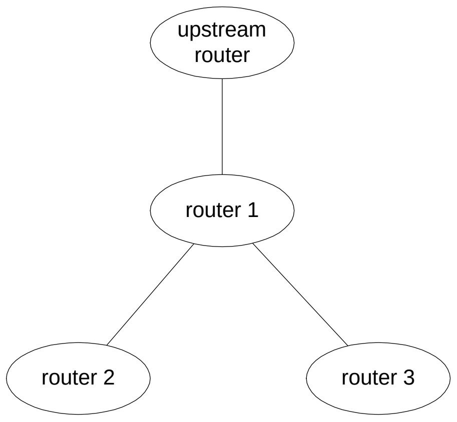

Figure 3.2: Composite Filtering at an Event Router

| publisher address | child interface | subscription expression |
|---|---|---|
| upstream router | router 2 | true |
| upstream router | router 3 | true |
| upstream router | self | false |

Table 3.2: Initial Routing Information

| publisher address | child interface | subscription expression |
|---|---|---|
| upstream router | router 2 | $(a \geq 5) \wedge (b \leq 5)$ |
| upstream router | router 3 | $(a \leq 2) \wedge (b \geq 10)$ |
| upstream router | self | false |

Table 3.3: Complete Routing Information

### 3.2.4 Control Messages

The control messages are called filter messages. Filter messages propagate subscription expressions towards a specific publisher. Each filter message is sent by one node to a neighbor node. The neighbor node is the parent in the distributed routing tree for a specific publisher. Table 3.4 shows a sample filter message. A filter message means that the child node wants some of the events, but not all, and the message therefore contains a composite subscription expression. The node wants any event that is true for the subscription expression. A filter message that contains only the literal "false" means that the node wants to recieve no events. A filter message that contains only the literal "true" means that a node wishes to cancel a previously sent filter and wishes to receive all events. The *sender address* is needed when multiple child nodes share the same interface. This is the case when the interface is a LAN segment rather than a point-to-point link to another node.

| | |
|---|---|
| publisher address | 10.23.21.2 |
| sender address | 10.24.1.2 |
| subscription expression | $(when \geq 3/26/2003) \wedge (change \geq 5)$ |

Table 3.4: A Sample Filter Message

## 3.3 Behavior

### 3.3.1 Receiving a New Event

When an event is received by a node, the node must first determine if the event is from a new publisher. If the publisher is new, then the list of child interfaces for that publisher is computed. The subscription expression for each child interface is intitialized to the literal "true." After initializing the list of child interfaces, processing continues as for any event.

The node looks up the child interfaces for this publisher. Each child interface has an associated subscription expression. For each interface, the node must decide whether or not to forward the event. If the subscription expression evaluates to "true," then the event is forwarded. However, if the expression evaluates to "false," then the node does not deliver the event.

Finally, if each interface has a subscription expression that is different from the literal "true," but the event was not forwarded to any interface, then the event was not wanted. If an event is not wanted, then a filter message is sent upstream towards the source of the event. The filter message contains a disjunctive composite of the subscription expressions for all of the child interfaces. To prevent storms of filter messages when a node is receiving frequent unwanted messages, a suppression timer is started as soon as a filter message is sent. The supression timer is associated with the publisher and the node to which the filter message was sent.

### 3.3.2 Receiving a Filter Message

When a node receives a filter message on an interface, it looks up the subscription expression for the publisher and interface, and replaces the subscription expression. After a filter message is processed, a timer is started to determine when the filter has expired.

In addition, the node may propagate the filter as soon as it receives it:

1. If the subscription expression is the literal “true,” then the receiving node must send a filter message with the “true” subscription expression if it has previously sent any filter message to the upstream router.

2. If the subscription expression is not the literal “true,” but all child interfaces associated with this publisher have subscription expressions different from the literal “true,” then the node must send a filter message with the composite subscription expression and set the suppression timer as described in Section 3.3.1. This allows the event system to converge rapidly when subscriptions change.

### 3.3.3 Changing Subscription

If a node changes its subscription, it must traverse the list of publishers and determine whether to send a filter message towards each publisher. If the node has previously sent a filter message towards a publisher, it must immediately send a filter message. The message is required because the node’s new subscription may match more events than the previous subscription. The simplest example of this is when a leaf node is subscribing for the first time. The node may already have sent a filter message indicating that it does not want to receive any events. The node must send a filter message towards each publisher to which it has previously sent a filter message. The logic is the same when an interior node changes its subscription. If any child interface’s subscription expression is the literal “true,” then the filter message will also contain the literal “true.” However, if this is the case, the interior node is unlikely to have sent a filter message for that publisher because the interior node already wanted all events.

### 3.3.4 Publishing

Because each node keeps a list of publishers, a new publisher may simply start sending events, and an existing publisher may stop sending events without any necessary upkeep. Of course, a publisher must communicate with one or more routers to join the system.

### 3.3.5 Timers

The origin and handling of three timers must be described to complete the design:

Without a timer on subscription expressions, a new node joining the system might never have an affect on the network filtering. So, each node sets a *filter timer* whenever a filter message is received. The filter timer is for the combination of publisher and child interface. When the filter timer expires, the node must discard the subscription expression and begin forwarding all events. No messages are required when the filter timer expires. The timeout for the filter timer should be long enough to allow network convergence. This depends on the diameter of the network, and is one of the areas of study.

Each subscriber keeps a *suppression timer* to prevent storms of filter messages. When a subscriber first receives an event that does not match its subscription expression, a filter message is sent and the suppression timer is initialized. The subscriber does not send another filter message until the suppression timer expires. The timeout for the suppression timer can be safely set to half the timeout for the filter timer. If the filter message is received successfully, no unwanted message will be received until the filter timer has expired. If not, the subscriber will send another filter message after the suppression timer expires.

Each node keeps a *publisher timer* so that the state associated with each publisher can be removed after the publisher stops sending events. The timer is reset whenever the node receives an event from that publisher. The timeout for this timer should be substantially longer than the other two timeouts. In this loosely coupled environment, a router cannot determine whethor a publisher has stopped sending events or has no events to send. The benefits of filter propagation depend on keeping track of the state associated with each publisher. Since memory is plentiful, we err on the side of remembering old publishers.

## 3.4 Predicate Optimization and Approximation

As subscriptions propagate towards publishers, the composite subscription becomes more complex. Exression optimization and approximation is desirable to bound the time necessary to make a forwarding decision for an incoming event.

An expression is optimized if it is put in a form that involves fewer comparisons. For example, the following expression may certainly be optimized:

$$(temperature < 0) \vee (temperature \geq 0)$$

Siena introduces a transitive, reflexive relation between subcriptions called covering, written using the square subset symbol, $\sqsubset$. From a logical perspective, this relation is the inverse of implication. However, it can be more intuitive to think of one expression covering another.

$$expr_A \sqsubset expr_B \Leftrightarrow expr_A \Leftarrow expr_B$$

An expression is approximated if another expression is found that is true more frequently than the original. It is desirable if the approximate expression also involves fewer comparisons. We can equally say:

$$approximation \sqsubset original$$

$$original \Rightarrow approximation$$

Given a predicate expression P, the following algorithm always finds an approximating expression with one or more fewer comparisons as long as an AND ($\wedge$) term appears in the expression. First, assume without loss of generality that no negation appears in P. We can make this assumption on the expression P because $\sim (a \leq 5) = (a > 5)$. P consists of terms p[0] through p[N], each with a depth. Take the AND term of maximum depth and replace it with either of the operand terms.

If no AND term remains, further approximation by this method of dropping terms may result in a subscription that is true for all events. In fact, this may occur if the terms are not chosen carefully, despite the limitation to AND terms. Terms should be eliminated if they involve tags that appear elsewhere in the expression P.

Consider the following example:

$$\begin{aligned}
&((a \geq 5) \wedge (b \leq 5)) \vee ((a \leq 2) \wedge (b \geq 10)) \\
\Rightarrow &(a \geq 5) \vee ((a \leq 2) \wedge (b \geq 10)) \\
\Rightarrow &(a \geq 5) \vee (b \geq 10)
\end{aligned}$$

At this point, the algorithm considered above would conclude that no further approximation beyond forwarding all events should be considered. This approximation algorithm may still conclude that all events should be forwarded, but that is not a poor conclusion for a heavily loaded, centrally located event router.

Algorithms similar to the above allow a router to self-optimize to reach desired performance. A router can be configured with a maximum number of comparisons per outgoing interface per event, and can optimize to that number of comparisons, conluding to always forward events on that interface if necessary. The maximum number of comparisons should be quite large because processor speed is plentiful relative to network bandwidth.

A good deal of research on minimization of boolean functions is available due to the importance of this minimization in automating digital circuit design [32]. In a publisher/subscriber system, our design may expect that subscriber interest will cluster. A composite subscription expression may be implemented as a list of sub-expressions. The list may be evaluated in any order due to the associativity of boolean disjunction. When any new subcription expression is added to an existing list, the list should be tested to make sure that the expression is not a duplicate of any other expression.

# Chapter 4
# Simulation Implementation

This chapter describes the simulation environment and the experimental framework so that the results can be more easily replicated or extended.

## 4.1 The Network Simulator

We used the Network Simulator 2, `ns`, for the experiments [33]. `ns` is a discrete object simulator that allows researchers to collaborate in the study of network protocols. `ns` is written in a combination of C++ and Object-Oriented Tool Control Language, `otcl`. To run an experiment using `ns`, a researcher writes an `otcl` program that uses `ns` specific classes to run an experiment. Then, the researcher executes the program using `ns`. `ns` interprets the program and exits.

To add a new protocol to `ns`, a researcher must implement one or more new classes. It is sometimes possible to do this entirely with `otcl` programming. Typically, the researcher adds one or more C++ classes to `ns`, and writes `otcl` experiments that access these new classes.

There are several advantages to using `ns` for the simulation:

1. Other researchers familiar with `ns` can easily replicate results since the results are produced using a standardized tool.
2. The separation between simulated experiment and simulated protocol allows researchers to easily vary experimental parameters that might otherwise be difficult to change.

3. Any new protocols simulated may become part of the standard repertoire of the research community that uses `ns`.

4. Our research benefits from others' contributions to `ns`.

One unforseen disadvantage is that `ns` does not support adding or removing network nodes or links once the simulation has started. To perform experiments that require nodes to "appear" and "disappear," `ns` supports changing the status of existing network links. While this meets all the essential needs for experiments in network dynamics, it also leaves in place some accidental complexity as all nodes and links must be created before the simulation begins. Also, since a node is never truly reset, simulated programs, called agents, running on the node, must each be individually reset by the researcher.

## 4.2 Topology Generation

The topologies used in the simulation were generated using the Georgia Tech Internet Topology Model (GT-ITM) [34], [35]. The GT-ITM can create random topologies by any of a number of standard random methods, but is distinguished by its ability to create random hierarchical topologies that simulate the actual topology of the Internet. When an organization joins the Internet, they pay one or more service providers for a connection to the wider world. Due to this, many organizations are connected to the Internet by one or two high bandwidth links to a more centrally placed service provider. The GT-ITM calls this the transit-stub model, where nodes called transit nodes form a central network of simulated service providers and nodes called stub nodes are organized into sub-networks at the periphery.

It is no accident that this type of hierarchical topology is similar to that generated by large self-organizing overlays [7]. Jain's research focuses on building a hierarchy where the cluster leaders form a central network.

Figure 4.1 shows the input file used with the GT-ITM. The input file instructs GT-ITM to create 10 random topologies starting with the seed 47. This creates 10 topologies that can be loaded into `ns`. Each topology is created with the same parameters. There should be 4 transit nodes and an average of 3 stub domains per transit domain, each having an average of 8 nodes. To make the network a little more random, and to simulate a more fault-tolerant topology, there is 1 extra network link from each stub domain to the transit domain and 1 extra network link within a stub domain.

GT-ITM topologies are converted for use with `ns` by invoking `sgb2ns`, a program that generates an `otcl` source file containing the procedure `create-topology`. To use the generated topology, an experiment must load the generated source file and then execute the procedure.

```
# <method keyword> <number of graphs> [<initial seed>]
# <# stubs/trans node> <#rand. t-s edges> <#rand. s-s edges>
# <n> <scale> <edgemethod> <alpha> [<beta>] [<gamma>]
# number of nodes = 1x4x(1+3x8) = 100
ts 10 47
3 1 1
1 20 3 0.5 1.0
4 20 3 0.6 1.0
8 10 3 0.42 1.0
```

Figure 4.1: GT-ITM Input File

## 4.3 Simulation Implementation

The experimental environment builds a new executable, `nsevt`, that includes additional classes in both C++ and `otcl`. Table 4.1 summarizes the added C++ and `otcl` classes and files.

To implement the filter propagation algorithm described in Chapter 3, I extended the classifier object used by `ns` to be a filtering replicator. The new class, `FilterReplicator`, is closely based on the `Replicator` class used to simulate multicast in `ns`. However, because there is no header file for the `Replicator` class, the `FilterReplicator` class inherits from the `Classifier` class directly. The class uses the `EvtFilter` class to evaluate the filter. A new packet header structure, `hdr_event`, transmits the tags and their values. For the simulation, only numeric tags are supported and no tag type is needed.

A new multicast protocol class called `CBM` completes the routing logic. In `ns`, each node has a multicast protocol object. The multicast protocol objects communicate with each other by sending prune and graft messages. These messages allow `ns` to simulate the full operation of DVMRP and later multicast algorithms as discussed in Section 2.2.2. The `CBM` class is closely based on the `DM` class that simulates DVMRP multicast. The `CBM` class operates in two modes: perfect and filter propagation. In the perfect mode, subscriptions are calculated from global knowledge whenever the node receives a graft message. The graft messages are passed up a routing tree towards the publisher, leading to perfect filtering. In the propagation mode, the protocol is as described in Chapter 3. The prune message is used for almost all filter messages, except that the graft message is used when the subscription expression is the literal "true."

Two more classes represent the publisher and the subscriber. The `EventSource` class inherits from the `UdpAgent` class and publishes events. It may be attached to the traffic generator classes, and adds tags randomly. The `EventSink` class inherits from `LossMonitor` and subscribes for events. It tracks the total number of received events and also how many match the subscription. It also performs a call from C++ to `otcl` to inform the `CBM` protocol object that the `EventSink` has changed subscription or received an unwanted event.

## 4.4 Experimental Framework

With this infrastructure, the filter efficiency of three forms of filtering can be studied:

1. An experiment that uses standard DVMRP simulation can simulate filtering at the client, the worst case for message efficiency. By clearing all counters after a warming period, the start-up overhead is discounted, and only the steady-state cost is included in message counts.

2. An experiment that uses the perfect mode of the `CBM` class can simulate perfect filtering at the publisher with no overhead. The publishers should start sometime before subscribers, and then message counters are cleared after a warming period.

3. An experiment that uses the filter propagation mode of the `CBM` class can simulate the Filter Propagation algorithm from Chapter 3. The `pruneTimeout` variable can be used to vary the filter timeout. The publisher timeout is not simulated, and the suppression timeout is set to half the filter timeout. Setting a filter timeout longer than the simulation disables the timeout.

Message counts are collected throughout the simulation. The `EventSource` class tracks the number of events sent by each publisher. The `FilterReplicator` class

| Classname | File(s) | Purpose |
|---|---|---|
| hdr_event | event.h and event.cc | an event header with tags |
| EventSource | event.h and event.cc | a publisher |
| EventSink | event.h and event.cc | a subscriber |
| EvtFilter | evtfilter.h and evtfilter.cc | a subscription expression |
| FilterReplicator | evtrouter.h and evtrouter.cc | a forwarder |
| CBM | CBM.tcl | Content-Based Multicast protocol |
| | setup.tcl and override.tcl | initialization logic |
| | exper.ns | perform an experiment |

Table 4.1: Source Files of Experimental Framework

counts the number of copies of events sent to child nodes. The `EventSink` class counts the number of events received at subscribers and how many of these matched the subsciption expression. The number of control messages are counted by the `CBM` class.

The `exper.ns` file extends tcl to seperate the experiment, the topology, the output and the random seed:

```
nsevt exper.ns exper/tiny-noto topology/tiny.tcl out 89
```

The above command loads the file in Figure 4.2 and applies it to a topology defined in `topology/tiny.tcl` that implements the `otcl` procedure `create-topology` generated by GT-ITM. Results are appended to `out` and the random number generator is initialized with seed 89. The experiment file in Figure 4.2 instructs `exper.ns` to run an experiment with 1 group and 0 transit nodes, setting a filter timeout of 32 seconds. The simulation is run for 8 seconds with the filter propagation algorithm. Out of the topology, 1 node is randomly chosen to publish to group 0, and 2 nodes are randomly chosen to subscribe to group 0. The additional parameters to the `publishers` and `subscribers` commands identify when the nodes should start and stop publishing and subscribing. A node may be both a publisher and a subscriber.

Any protocol or software must be debugged and verified. Figure 4.3 shows how diagnostics can be enabled on an experiment. The `dump-filters` function saves routing tables to a named file. The second argument controls whether the actual routing table or an ideal routing table is saved. The ideal routing table is computed by a depth-first search along each outgoing interface that finds subscribers. After the simulation, a script can compare the actual and ideal routing tables, or the routing tables of "perfect" filtering and filter propagation. This verifies that both algorithms converge to the same ideal routing table. The `trace-events` function enables a diagnostic log used to track down the inevitable bugs. Each node records diagnostic messages identified by timestamp, node, and logging object to the trace file. The diagnostic messages closely follow the prune and graft messages and their effects on the routing tables.

```
# experiment <groups> <transit> <filtertimeout> <time> <style> <seed>
experiment 1 0 32.0 8.0 propagate 47

# publishers <group> <nodes> <start> <stop> <size> <period> <color> {
#   <tags> }
publishers 0 1 0.5 7.0 1000 0.05 DarkGreen {
    temp random 0.0 20.0,
    current random 0.0 20.0,
    fish random 0.0 20.0,
}
# subscribers <group> <nnodes> <start> <stop> <percent> { <filters> }
subscribers 0 2 1.5 7.5  25% {
    (temp >= 10) && (current >= 10),
    (temp < 10) && (current >= 10),
    (current >= 10) && (fish >= 10),
    (fish >= 10) && (temp < 10),
    (temp < 2.5) || (temp >= 17.5),
}
warming-period 2.0
```

Figure 4.2: Specifying an Experiment

```
# dump routing tables to file fnotimeout 7.5 seconds into the simulation
# store both actual and ideal routing tables
set ns [Simulator instance]
$ns at 7.5 "dump-filters fnotimeout 1"
$ns at 7.5 "dump-filters fnotimeout 0"

# trace time/node based diagnostics on messages to file trace-notimeout
trace-events trace-notimeout
```

Figure 4.3: Diagnostic Functions

# Chapter 5
# Experiments and Results

## 5.1 Filter Efficiency

This section evaluates the scalability of filter propagation as the number of publishers and number of subscribers increase, using a synthetic simulation over networks of 100 nodes. To evaluate efficiency, we plot the average number of inter-node messages per event, including control messages, in addition to the copies of the event. Four protocols are compared using the experimental framework discussed in Chapter 4. For each experiment, 4 different topologies and seeds are used. On each topology, the four protocols are compared by using the same random seed to choose the publisher and subscriber nodes and the subscription expressions. The four protocols studied are:

1. Filtering at the subscriber, simulated by using normal DVMRP multicast. This is a control on the study of filter propagation showing the worst case with respect to eliminating inter-node messages. DVMRP is run with a timeout on prunes that is longer than the simulation.

2. Filtering at the publisher, simulated using the perfect mode of the `CBM` protocol class mentioned in Chapter 4. This filtering at the publisher is the best-case, but it is not realistically achievable because no control messages are counted. In a non-ideal world, each subscriber must transmit its subscription to each publisher, and the network must adapt to change.

3. Filter propagation without a timeout, simulated by setting the `CBM` class timeout

beyond the end of the simulation. The control overhead for filter propagation is included in the simulation.

4. Filter propagation with a timeout, simulated by setting the `CBM` class timeout to 8 seconds. Other experiments below show that the filtering converges in 5 seconds on a network of 100 nodes, and the simulation runs for 20 seconds, and so there are 2 timeouts following the initial convergence. These test cases represent steady-state filter propagation with a pessimistically short timeout.

Each experiment is run with 10, 20, 40, and 80 publishers and 10, 20, 40, and 80 subscribers. Each publisher generates events with three tags that vary uniformly. The subscribers randomly choose a filter that will match 25% of the events. The experiment file resembles that shown in Figure 4.2, but 39 subscription expressions are used and the number of publishers and subscribers are varied as mentioned above. The choice of 25% is neither conservative nor liberal. In a non-synthetic environment, some nodes would want many of the events for logging puposes, and other nodes would want very few of the events.

Figures 5.1 compares filter propagation to the best case protocol, publisher filtering, and the worst case protocol, subscriber filtering. Each protocol is run with 10 publishers as the number of subscribers increase. Figures 5.2, 5.3, and 5.4 are similar results for 20, 40, and 80 publishers respectively. We can see that as the number of subscribers increases, the efficiency of filter propagation approaches the best case. These results are an average of results for all 4 topologies.

The same results are shown differently to illustrate the scalability of filter propagation. Figure 5.5 plots 10, 20, 40, and 80 publisher experiments against each other to show how filter propagation scales with the number of subscribers. Figure 5.6 changes the x-axis to show scaling of 10, 20, 40, and 80 subscriber experiments as the number of publishers increases. As the number of publishers increases, the average messages per event decreases due to the density of publishers in the graph. The topology contains only 100 nodes, and so when there are 80 publishers, there is a higher probability that publishers are close to many of their subscribers. Each node may be both a publisher and a subscriber, which inreases the benefits of density.

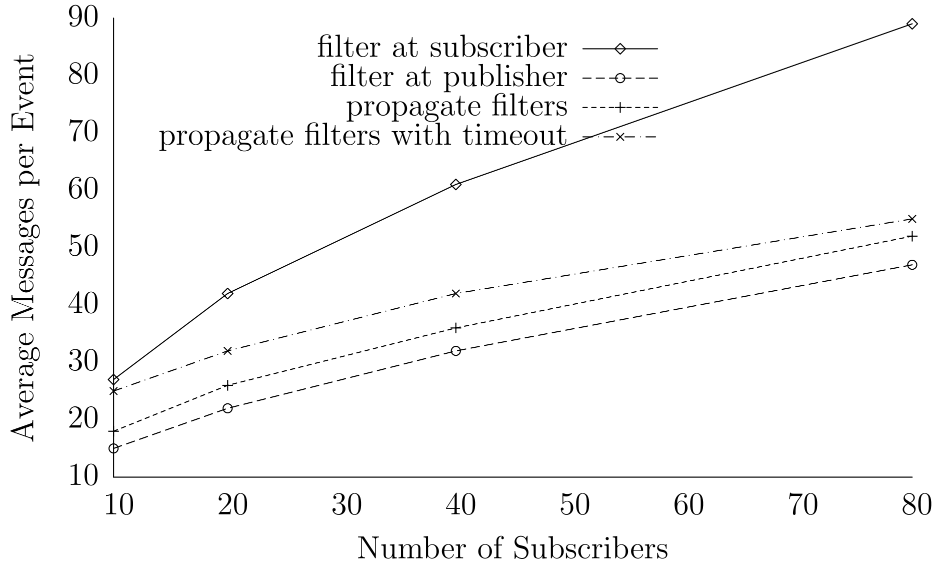

Figure 5.1: 10 Publishers on a 100 Node Network

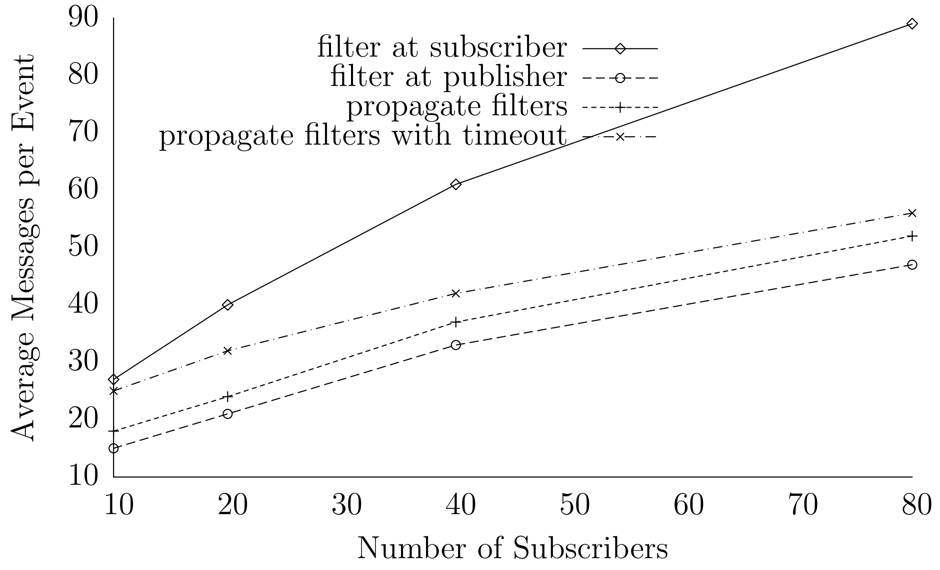

Figure 5.2: 20 Publishers on a 100 Node Network

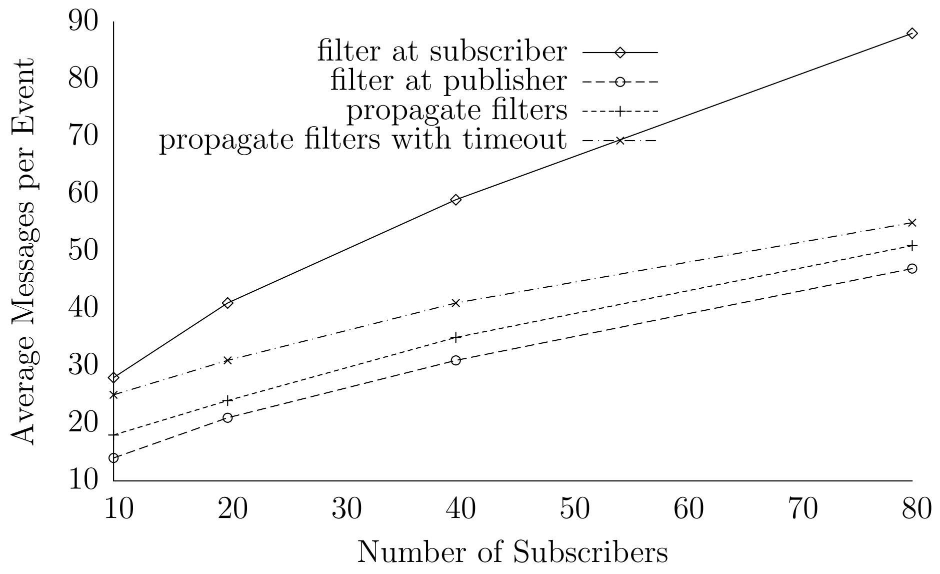

Figure 5.3: 40 Publishers on a 100 Node Network

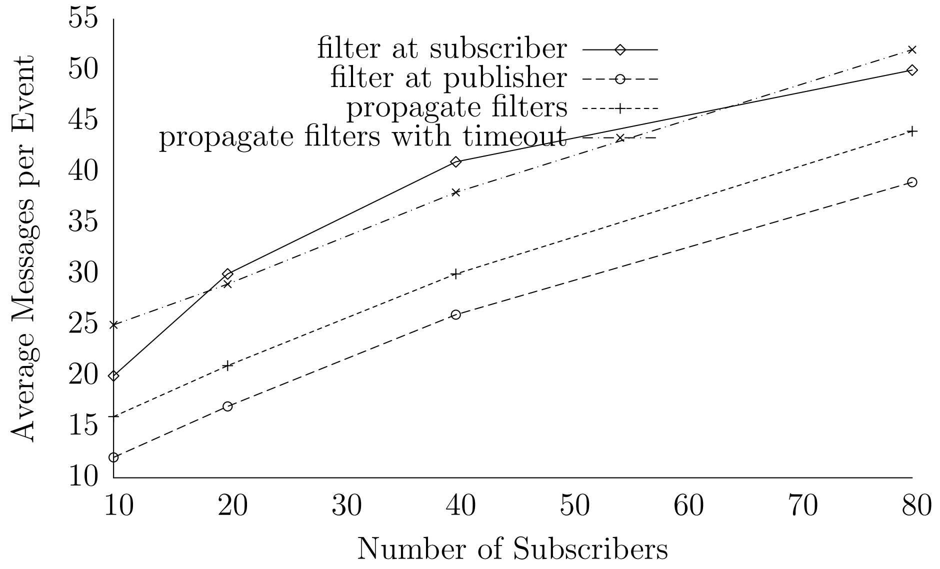

Figure 5.4: 80 Publishers on a 100 Node Network

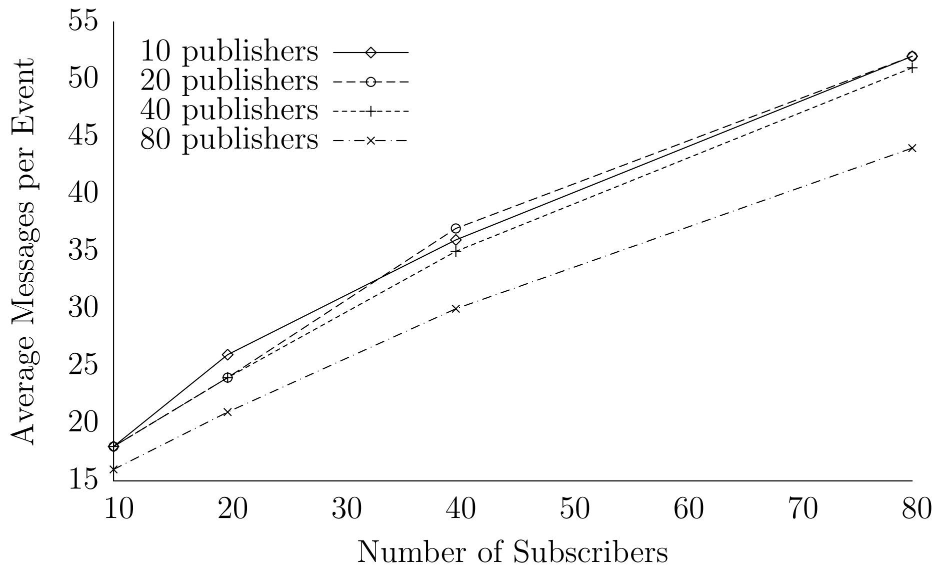

Figure 5.5: 10, 20, 40 and 80 Publishers as Subscribers Increase

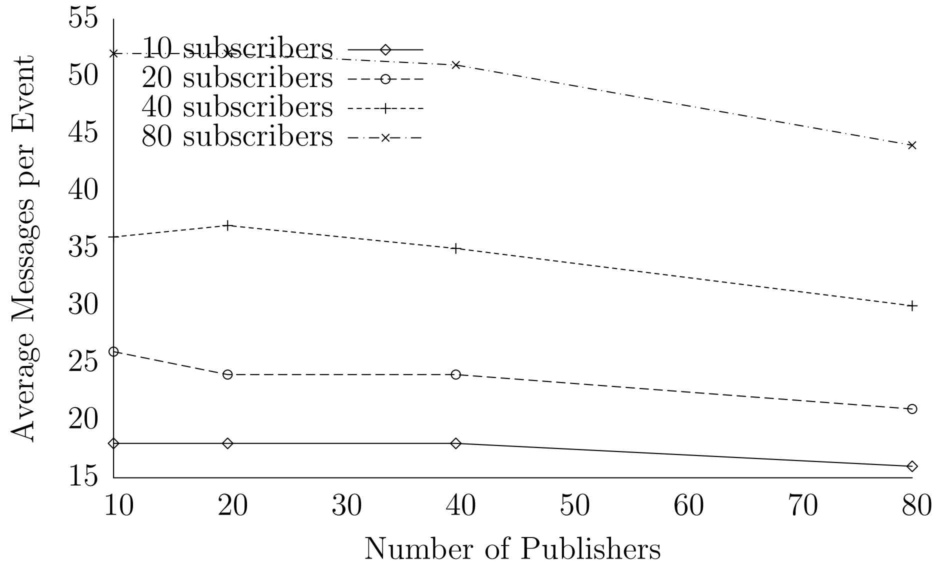

Figure 5.6: 10, 20, 40, and 80 Subscribers as Publishers Increase

### 5.1.1 Overhead of Filter Timeout

In the experiments described above, some of the messages needed to deliver each event are the protocol overhead, consisting of filter messages. These control messages can be plotted over time to show how quickly the protocol responds to change. Figure 5.7 shows that when an experiment is first started, filter messages flow through the network. Thereafter, if there is no timeout of the filter and no change of the filter, no additional control messages are required. For comparison, the control messages for DVMRP multicast are shown. These control messages are subtree prunes and grafts. DVMRP multicast requires more initial messages because filter propagation waits for a message that was not wanted; however, DVMRP has less total overhead.

As shown in Figure 5.8, there are additional control messages if there are filter timeouts, or prune timeouts for DVMRP multicast. The increase in control messages is both more gradual and less in magnitude than the initial convergence of the network due to the operation of the filter propagation protocol. During the first convergence (from 0 to 5 seconds), filter messages arrived at nodes at different times. Therefore, the filter timers on publisher nodes will expire later on average than the filter timers on subscriber nodes or router nodes. When the subscriber nodes' filter timers expire, the subscriber nodes will not immediately send filters, but will wait to receive an unwanted message. As publisher nodes' filter timers expire, subscribers will receive unwanted messages. So, the rate of filter messages will be more gradual than for initial convergence. Moreover, fewer messages will be needed because some state already exists in the network when the first filter messages are sent.

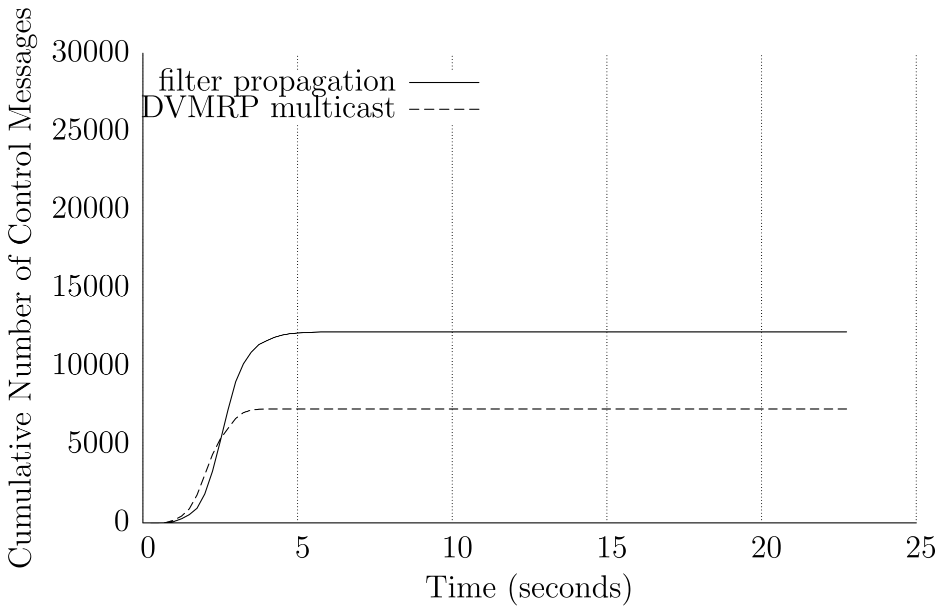

Figure 5.7: Cumulative Control Message Count with No Timeouts

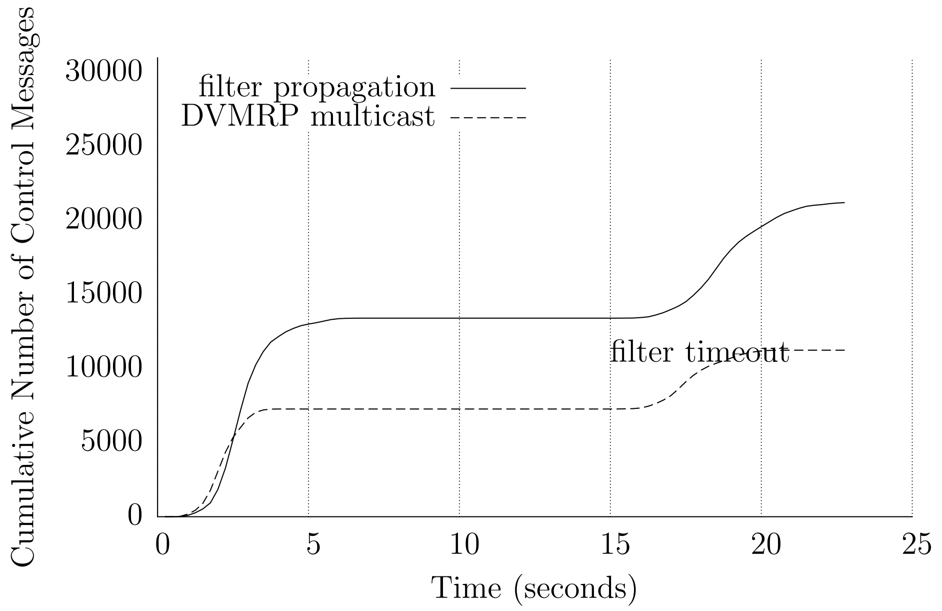

Figure 5.8: Cumulative Control Message Count with a Timeout

## 5.2 Adapting to Change

The protocol must adjust to many changes gracefully:

1. New publishers may begin to send data, or existing publishers may stop. Publishers stop for free because there are no control messages to send.
2. New subscribers may cancel pruning filters, or existing subscribers may stop wanting events.
3. Nodes may change their subscription without changing their subscriber status.
4. The network topology may change, causing the lowest latency routing tree to change, or even partitioning the network.

Figure 5.9 shows how quickly the filter propagation algorithm responds to new publishers, to new subscribers, and to subscription changes of existing subscribers. The initial convergence allows comparison with Figures 5.7 and 5.8.

When 10 more publishers are added, the overhead required is just as great because each publisher has its own routing tree, and the subscription expressions of the subscribers must be propagated again. This is a clear disadvantage of the filter propagation protocol over a protocol that uses a single routing tree. However, the addition of 20 subscribers requires few control messages. The protocol is slow to respond to the change in subscription because of the synthetic nature of the test. Each subscription filter matches 25% of the events. For the purposes of studying how rapidly the protocol adapts to change, it would be better to use a lower match percentage and to have more potential subscription filters.

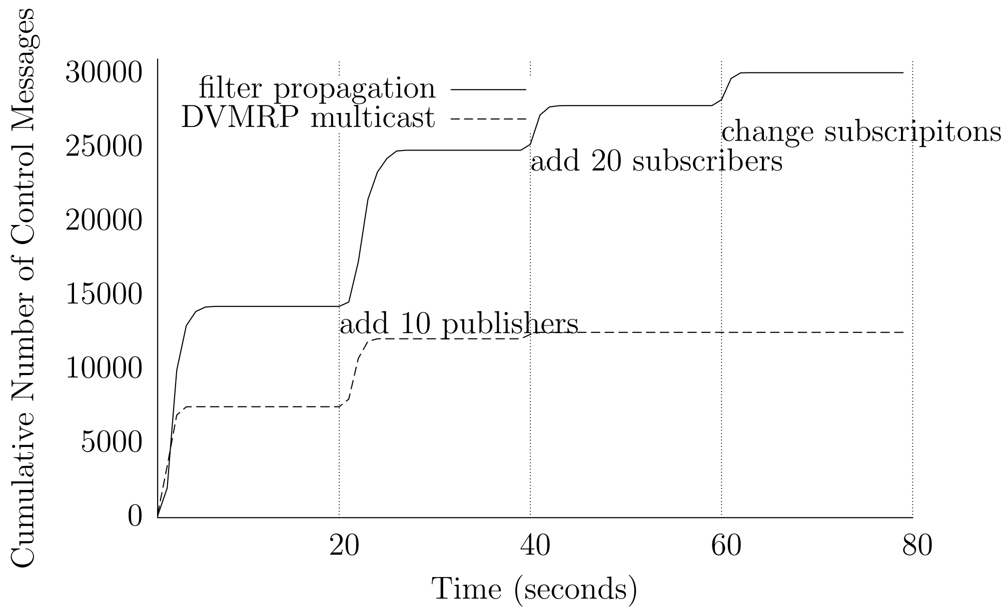

Figure 5.9: Cumulative Control Message Count with Injected Changes

# Chapter 6
# Conclusions and Future Work

## 6.1 Conclusions

This thesis explored large scale publisher/subscriber systems from the perspective of avoiding flooded publisher advertisements and subscriptions. After surveying current research on publisher/subscriber systems and widely deployed, standardized systems for system and network management, an algorithm was proposed that propagates subscription expressions towards publishers. The propagation allows filtering in the network. The algorithm eliminates the need to broadcast publisher interest at the cost that each subscriber must keep state specific to each publisher.

This algorithm was evaluated using simulation to demonstrate the algorithm's efficiency at reducing the transmission of messages, the algorithm's speed of convergence following a change, and general overhead. The network simulator, `ns`, was extended so that the multicast environment supports a notion of subscriber interest based on lists of expressions. The use of a research standard simulator makes it easy for others to reproduce and extend the results presented here, and therefore provides a valuable research tool. The results show that the system is scalable on a topology similar to the Internet, and that the system reacts well to change.

Predicate optimization and approximation allows routers to bound the amount of work needed in order to make a forwarding decision. Some simple heuristics were proposed that err on the side of forwarding events. These heuristics should be evaluated in the future.

## 6.2 Future Work

This thesis raised many questions that could not be investigated in the time available. The most interesting directions for future work are these:

1. Several additional experiments would help assess scalability. The relationship between the size of the network and both latency and efficiency should be studied. A more in depth investigation of the impact of topology changes to this and related algorithms would be desirable. `ns` made it difficult to evaluate the effects of topology changes on the algorithm.

2. The filter approximation heuristic should be compared with more aggressive techniques from the literature on boolean optimization.

3. A prototype implementation might bring insight into the use of multicast and the support of LAN environments in addition to overlay networks. In a prototype implementation, it would be interesting to investigate the accounting requirements of system and network management applications and to implement distributed event aggregation.

4. The choice made to avoid a publisher advertisement motivates a direct quantitative comparison between algorithms with advertisements and filter propagation.

5. The filter propagation algorithm rests upon the calculation of a multicast distribution tree. Because there is a separate tree for each publisher, each node must keep state associated with the publishers. One tree would address this shortcoming. However, many wireless routing algorithms use a routing mesh rather than a tree. These algorithms dynamically allow message duplication so as to be more resilient in highly dynamic environments. One possibility is to embed a tree within the mesh used for resilient routing. A more challenging approach would be to change the way that a filter propagates, so that it may propagate through the mesh towards the publisher.

# References

[1] S. McCanne, V. Jacobson, and M. Vetterli, “Receiver-driven layered multicast,” in *ACM SIGCOMM*, New York, NY, August 1996, vol. 26-4, pp. 117–130, ACM Press.

[2] Semandex Technical Staff, “Making Sense of Sensors,” White paper, Semandex, Princeton, NJ, 2002, [Online document], Available: http://www.semandex.com/library/.

[3] Semandex Technical Staff, “What is Content-Based Routing?,” White paper, Semandex, Princeton, NJ, 2002, [Online document], Available: http://www.semandex.com/library/.

[4] G. R. Malan, F. Jahanian, and S. Subramanian, “Salamander: A Push-based Distribution Substrate for Internet Applications,” in *Proc. of the USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, December 1997.

[5] S. Tuecke et al., “Open Grid Services Infrastructure,” Specification GWD-R, Global Grid Forum, June 2003, [Online document], Available: http://www-unix.globus.org/toolkit/.

[6] P. Horn, “Autonomic Computing: IBM’s Perspective on the State of Information Technology,” White paper, IBM, October 2001, [Online document], Available: http://www.research.ibm.com/autonomic/manifesto/.

[7] S. Jain, R. Mahajan, D. Wetherall, and G. Borriello, “Scalable Self-Organizing Overlays,” Tech. Rep. UW-CSE 02-06-04, Dept. of Computer Science, Washington University, May 2000.

[8] A. Oram, Ed., *Peer to Peer: Harnessing the Power of Disruptive Technologies*, O’Reilly and Associates, Sebastopol, CA, 2001.

[9] T. Harrison, D. Levine, and D. Schmidt, “The Design and Performance of a Real-time CORBA Event Service,” Tech. Rep. #WUCS-97-31, Dept. of Computer Science, Washington University, St. Louis, MO, 1997.

[10] L. Zhang, S. Deering, and D. Estrin, “RSVP: A new resource ReSerVation protocol,” *IEEE network*, vol. 7, no. 5, pp. 8–18, September 1993.

[11] M. Rose and K. McCloghrie, “Structure and Identification of Management Information for TCP/IP-based Internets,” RFC 1155, IETF, May 1990, [Online document], Available: http://www.ietf.org/rfc.html.

[12] J. Case, M. Fedor, M. Schnoffstall, and J. Davin, "A Simple Network Management Protocol (SNMP)," RFC 1157, IETF, May 1990, [Online document], Available: http://www.ietf.org/rfc.html.

[13] "PowerNet Manager," American Power Conversion Corporation (APC), [Online document], [cited April 11, 2003], Available: http://www.apcc.com/products/.

[14] HP Technical Staff, "snmpd(8)," Tru64 UNIX Version 5.1B Refererence Page, HP, 2003, [Online document], [cited April 11, 2003], Available: http://www.tru64unix.compaq.com/.

[15] DMTF Technical Staff, "CIM Operation over HTTP," Specification #DSP0200, Distributed Management Task Force (DMTF), August 1999, [Online document], Available: http://www.dmtf.org/standards/.

[16] C. Hedrick, "Routing Information Protocol," RFC 1058, IETF, June 1988, [Online document], Available: http://www.ietf.org/rfc.html.

[17] R. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

[18] L. R. Ford and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.

[19] D. Waltzman, C. Partridge, and S. Deering, "Distance Vector Multicast Routing Protocol," RFC 1075, IETF, November 1988, [Online document], Available: http://www.ietf.org/rfc.html.

[20] S. Deering, "Multicast Routing in Internetworks and Extended LANS," in *ACM SIGCOMM '88*, Stanford, CA, Aug. 1988, pp. 55–64.

[21] J. Moy, "OSPF Version 2," RFC 1583, IETF, March 1994, [Online document], Available: http://www.ietf.org/rfc.html.

[22] B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks," in *Proceedings of ICDCS '02*, Vienna, Austria, July 2002, pp. 5–14.

[23] C. Schmidt and M. Parashar, "Flexible Information Discovery in Decentralized Distributed Systems," in *Proceedings of the 12th International Symposium on High Performance Distributed Computing*, Seattle, WA, June 2003, pp. 226–235, IEEE Computer Society Press, [online], Available: http://www.caip.rutgers.edu/TASSL/.

[24] E. Celebi, "Performance Evaluation of Wireless Multi-hop Ad-hoc Network Routing Protocols," M.S. thesis, Dept. of System & Control Engineering, Bogazici University, Turkey, 2002, [online], Available: http://www.cmpe.boun.edu.tr/ emre/research/msthesis/.

[25] S. Lee, W. Su, J. Hsu, M. Gerla, and R. Bagrodia, "A Performance Comparison Study of Ad Hoc Wireless Multicast Protocols," in *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000, pp. 565–574.

[26] Z. J. Haas, J. Y. Halpern, and L. Li, "Gossip-based Ad Hoc Routing," in *IEEE INFOCOM 2002*, New York, NY, June 2002, [online], Available: http://wnl.ece.cornell.edu/wnlprojects.html.

[27] J. M. Kahn, R. H. Katz, and K. S. J. Pister, "Next century challenges: Mobile networking for "smart dust"," in *Proc. of MOBICOM '99*, Seattle, WA, July 1999, pp. 271–278.

[28] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Interfaces and algorithms for a wide-area event notification service," Tech. Rep. CU-CS-888-99, Department of Computer Science, University of Colorado, October 1999, revised May 2000, [online], Available: http://www.cs.colorado.edu/ carzanig/papers/index.html.

[29] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Content-based addressing and routing: A general model and its application," Tech. Rep. CU-CS-902-00, Department of Computer Science, University of Colorado, January 2000, [online], Available: http://www.cs.colorado.edu/ carzanig/papers/index.html.

[30] A. Carzaniga and A. L. Wolf, "A benchmark suite for distributed publish/subscribe systems," Tech. Rep. CU-CS-927-02, Department of Computer Science, University of Colorado, April 2002, [online], Available: http://www.cs.colorado.edu/ carzanig/papers/index.html.

[31] D. Makwana, "An Introduction to Semantic Networking and its Performance Evaluation," M.S. thesis, Dept. of Electrical & Computer Engineering, Graduate School, Rutgers University, 2002, [online], Available: http://www.caip.rutgers.edu/ makwana/Acads/Research/.

[32] E. J. McCluskey, "Minimization of Boolean Functions," *The Bell System Technical Journal*, vol. 35, no. 5, pp. 1417–1444, November 1956.

[33] ISI Technical Staff, "The Network Simulator - ns-2," Web page, Information Sciences Institute, University of Southern California, Marina del Rey, CA, 2002, [online], [cited September 1, 2002], Available: http://www.isi.edu/nsnam/ns/.

[34] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," in *IEEE INFOCOM 1996*, San Francisco, CA, March 1996, vol. 2, pp. 594–602, [online], Available: http://www.cc.gatech.edu/fac/Ellen.Zegura/papers/.

[35] K. L. Calvert, M. B. Doar, and E. W. Zegura, "Modeling internet topology," *IEEE Communications Magazine*, vol. 35, no. 6, pp. 160–163, June 1997.