

# Effects of Routing Computations in Content-Based Routing Networks with Mobile Data Sources

Vinod Muthusamy<sup>†</sup>, Milenko Petrovic<sup>†</sup>, Hans-Arno Jacobsen<sup>†‡</sup>

Middleware Systems Research Group

<sup>†</sup>Department of Electrical and Computer Engineering

<sup>‡</sup>Department of Computer Science

University of Toronto

{vinod,petrovi,jacobsen}@eecg.toronto.edu

## ABSTRACT

This paper presents the first quantitative evaluation of the role of routing computations on performance when mobility is introduced to a content-based routing network. Additionally, the paper identifies the factors that affect the performance of a distributed publish/subscribe architecture supporting mobile publishers, formalizes publisher mobility protocols for distributed publish/subscribe systems, and develops and evaluates protocols that reduce the costs associated with supporting mobile publishers in publish/subscribe systems. Our results show that ignoring route computation time paints a false picture of the scalability of content-based routing networks, but that with appropriate protocols the adverse effects can be mitigated.

## Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems—*Distributed applications*; C.4 [Computer Systems Organization]: Performance of Systems

## General Terms

Algorithms, Design, Experimentation, Performance

## Keywords

Publish/Subscribe, multicast, mobility, routing

## 1. INTRODUCTION

It is becoming simpler for non-computer oriented users to publish information on the Internet because of a myriad of user-friendly tools. For example, it is easy to keep an “online” diary (e.g., blogs). Collaboration tools, such as wikis, allow users to quickly publish information using a web browser, without requiring access to or knowledge of any additional applications. Also, applications for web page authoring are becoming easier to use. As a result of the

advances in web authoring tools, the number of information publishers has grown considerably. It is estimated that 32 million people in the US read blogs (which represents 27% of the estimated 120 million US Internet users) while 8 million have published blogs. It has also been reported that 6 million people use news aggregators to stay abreast of the latest changes on web sites (e.g., blogs, headlines, discussion forums)<sup>1</sup>. This trend clearly indicates that timely dissemination of information is becoming critical.

As the use of portable wireless devices with Internet access becomes more pervasive (as seen by the increasing number of wireless access points, the increasing use of 3G technologies and the availability of more capable PDAs and mobile phones), we foresee the publishing trend on the Internet to apply in the realm of portable computing.

Wireless monitors in the form of embedded wireless sensor networks further increase the number of mobile publishers. For example, automobiles could publish various safety-critical information about failures resulting from various conditions (traffic, weather, parts failures, etc.).

The quick notification of various interested parties is very important. For example, after a car accident, police would be interested in knowing about vehicles that were involved in, or were near, the accident. The severity of the accident would determine what kind of information would be published. Large or serious accidents would notify nearby fire departments and paramedics, while small ones could notify insurance companies and police collision offices only.

Mobile publishing systems allow information to be published from devices at various locations and they enable users interested in receiving information from the mobile publishers to receive it regardless of the location of the publisher. In other words, the system is responsible for routing published information to the interested users.

The publish/subscribe (pub/sub) paradigm has been used in the context of selective information dissemination on the Internet [11, 10, 1, 6, 8, 3, 14, 13, 24], where publishers act as information providers, subscribers as information consumers, and a broker mechanism routes relevant publications to interested subscribers.

Pub/Sub systems have a number of desirable characteristics for mobile information dissemination applications. They

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*MobiCom '05*, August 28–September 2, 2005, Cologne, Germany.  
Copyright 2005 ACM 1-59593-020-5/05/0008 ...\$5.00.

<sup>1</sup>Reported by Pew Internet & American Life Project ([www.pewinternet.org](http://www.pewinternet.org)), an organization that produces reports that explore the impact of the Internet on families, communities, the daily life. Also reported by “RSS at Harvard Law” ([blogs.law.harvard.edu/tech/](http://blogs.law.harvard.edu/tech/))

efficiently filter and disseminate large volumes of data to many users [11, 1, 7]. Also, they decouple communication, both in time and space, allowing publishers and subscribers to communicate without having to be connected simultaneously or having to know about each other. Therefore, the pub/sub paradigm naturally supports mobile publishers.

Many applications can benefit from selective and decoupled dissemination including stores sending coupons to shoppers at a mall, reporters at a news conference sending real-time photos and news to interested readers, teammates communicating with each other in a strategic game, and commuters propagating traffic jam information to drivers behind them. Other more critical applications include sending information about arriving patients in a hospital or disaster scenario to those doctors and staff who are able to treat the patients' symptoms, and disseminating traffic accident data to nearby police and ambulance.

Most existing research assumes that both publishers and subscribers are not mobile. Work such as Burcea *et al.* [5], and Cugola *et al.* [8] have looked into supporting mobile clients. However, both of these studies ignore the effects of routing computations and mobile data sources.

Content-based routing protocols are intrinsically data-centric. Data-centric networking protocols use content addressing instead of host (e.g., IP) addressing for participating nodes, thus decoupling application communication from the underlying network. Data-centric networking is particularly important for networks that need to support mobile publishers. In such applications, users want to receive data of interest to them, regardless of where that data is located in the network; and as publishers become more mobile, so will the need for efficient routing of information from mobile publisher to subscribers.

Data-centric routing is intrinsically different from host-based routing in that data is routed based on users' specified interests. How this interest is expressed becomes important, as more complex interest-specification (subscription) languages will require more time consuming routing computations. This is in stark contrast with host-centric routing, where data is routed based on host IDs. Route computations based on host IDs can be implemented very efficiently. This is not true for content-based routing protocols, and as we show in the paper, computation time must be taken into account when analyzing scalability of content-based routing protocols.

We are not aware of any studies that experimentally evaluate and quantify the effects of routing computations in networks that use content-based routing protocols and support mobile publishers. In fact, prior work evaluating content-based routing completely ignores this cost [6, 8, 3, 15, 5]. We will see that routing computations can be a limiting factor when it comes to scalability of pub/sub systems that support mobility. The most important contributions we make in this paper are as follows:

1. We study and quantify the effects of routing computations on the performance of content-based routing protocols in networks supporting mobile data sources, and identify scalability misconceptions that stem from ignoring these effects.
2. We demonstrate that counter to traditional assumptions, advertisements can have a significant impact on the pub/sub network when data sources are mobile,

and we develop protocols that reduce the costs associated with mobile publishing.

3. We thoroughly evaluate the routing protocols and quantify the performance degradation due to mobile publishers.

The rest of the paper is organized as follows. Section 2 provides a brief overview of the pub/sub paradigm, the operation of a pub/sub content-based router, and the organization of a distributed pub/sub network. Section 3 discusses the modeling of the matching algorithms for routing computations used in this paper. Section 4 describes the protocol to support mobile sources, and develops four extensions to this protocol. Section 5 presents an experimental evaluation of the mobility protocols. Section 6 discusses related work and puts our work in perspective. Finally, Section 7 concludes the paper and discusses directions for future work.

## 2. BACKGROUND

In order to keep the paper self-contained, this section provides a detailed description of the pub/sub model and content-based routing.

### 2.1 Publish/Subscribe Model

The pub/sub paradigm is effective at supporting data-centric, information dissemination applications [1, 2, 3, 8, 9, 11]. Clients in a pub/sub systems are autonomous components that exchange information by publishing content and by subscribing to content of interest. Clients that produce information are referred to as *publishers*, while clients that consume information are referred to as *subscribers*<sup>2</sup>. Publishers generate messages (publications) to inform the external world that a certain condition has occurred. Subscribers, which express their interest in publications by means of subscriptions, are then notified of the occurrence of these publications. The central component of a pub/sub system is the *broker*, which records (persists) all subscriptions in the system. When a certain publication is published, the broker matches it against all subscriptions. On a match, the broker notifies the corresponding subscriber. It is important to note that messages from publishers (publications) do not contain any address; instead, they are routed through the system based on their content (for a content-based system). A network of brokers can be formed to create a content-based routing system.

Pub/Sub systems can be classified, based on the expressiveness of their subscription language, such as those based on *topics* (or *subjects*), *types*, *content*, or *subject spaces* [12]. In topic-based pub/sub, clients can subscribe to several topics and receive notifications about all publications within these topics. These systems usually offer flat or hierarchical addressing. In flat addressing, all topics are disjoint, while in hierarchical addressing topics are organized in hierarchies; subscriptions can address any node in the hierarchy, implicitly addressing all subtopics of the node. *Type-based* publish/subscribe systems are similar to topic-based, but use publication types instead of topics for matching. *Content-based* pub/sub systems increase the expressiveness of subscriptions by allowing more complex queries on the publication content [11]. The subject space model further

<sup>2</sup>Note that a client can be both a publisher and a subscriber.

Subscription $s_1$	Subscription $s_2$	Covering Relation
(product = "computer", brand = "IBM", price $\leq$ 1600)	(product = "computer", brand = "IBM", price $\leq$ 1500)	$s_1$ covers $s_2$
(product = "computer", brand = "IBM", price $\leq$ 1600)	(product = "computer", price $\leq$ 1600)	$s_2$ covers $s_1$
(product = "computer", brand = "IBM", price $\leq$ 1600)	(product = "computer", brand = "Dell", price $\leq$ 1500)	$s_1$ does not cover $s_2$ $s_2$ does not cover $s_1$

Table 1: Examples of subscriptions and covering relations

increases the expressiveness of subscriptions by persisting both subscriptions and publications [12].

## 2.2 Publish/Subscribe Router

A pub/sub router performs the following three operations: (1) forwarding of advertisements, (2) forwarding of subscriptions and (3) forwarding of publications.

The details of the three operations are highly dependent on the expressiveness of the subscription and publication representation languages.

To better understand the operation of a pub/sub router, we give a more formal definition of a subscription and a publication representation language most commonly used in the research literature. The formal representation of a publication is given by the following expression:  $\{(a_1, val_1), (a_2, val_2), \dots, (a_n, val_n)\}$ . Subscriptions are expressed as conjunctions of Boolean predicates. In a formal description, a simple predicate is represented as (*attribute\_name relational\_operator value*). A predicate ( $a \text{ rel}_{op} val$ ) is matched by an attribute-value pair ( $a, val$ ) if and only if the attribute names are identical ( $a = a$ ) and the ( $a \text{ rel}_{op} val$ ) Boolean relation is true. A subscription  $s$  is matched by a publication  $p$  if and only if all its predicates are matched by some pair in  $p$ .

We are now ready to look at the details of the pub/sub router operations.

### 2.2.1 Forwarding of Advertisements

Advertisements are used by publishers to announce the set of publications they are going to publish. Consequently, advertisements create routing paths for subscriptions from subscribers to publishers, whereas subscriptions build routing paths for publications from publishers to subscribers. Usually, both subscriptions and advertisements have the same formal representation. However, there is an important distinction between the predicates in an advertisement and those in a subscription: the predicate in a subscription are considered to be in a conjunctive construction, while those in an advertisement behave as in a disjunctive one.

An advertisement  $a$  *matches* a publication  $e$  if and only if all attribute-value pairs match some predicates in the advertisement. Formally, an advertisement  $a = \{p_1, p_2, \dots, p_n\}$  determines a publication  $e$ , if and only if  $\forall(attr, val) \in e, \exists p_k \in a$  such that ( $attr, val$ ) matches  $p_k$ .

An advertisement  $a$  *intersects* a subscription/advertisement  $s$  if and only if the intersection of the set of the publications determined by the advertisement  $a$  and the set of the publications that match  $s$  is a non-empty set. Formally, at the predicate level, an advertisement  $a = \{a_1, a_2, \dots, a_n\}$  intersects a subscription/advertisement  $s = \{s_1, s_2, \dots, s_n\}$  if and only if  $\forall s_k \in s, \exists a_j \in a$  and there exists some attribute-value pair ( $attr, val$ )<sup>3</sup> such that ( $attr, val$ ) matches both  $s_k$  and  $a_j$ . Table 2 presents some examples of subscriptions

<sup>3</sup> $s_k$  and  $a_j$  refer to the same attribute  $attr$

and advertisements and the corresponding intersection relations.

The following are the steps performed by the pub/sub router upon receiving an advertisement:

1. For the advertisement received, check if there are *covering* advertisements in the advertisement table. If there are, then we do not need to forward the advertisement.
2. If there is no covering advertisement, *insert* incoming advertisement in the advertisement table, and forward the advertisement to all neighbors.
3. Check if there are *intersecting* subscriptions in the subscription table. If there are, forward the intersecting subscriptions to the neighbor from which the advertisement was received.

### 2.2.2 Forwarding Subscriptions

Subscription processing is similar to advertisement processing. Given two subscriptions  $s_1$  and  $s_2$ ,  $s_1$  *covers*  $s_2$  if and only if all the publications that match  $s_2$  also match  $s_1$ . In other words, if we denote with  $E_1$  and  $E_2$  the set of publications that match subscription  $s_1$  and  $s_2$ , respectively, then  $E_2 \subseteq E_1$ .

At the predicate level, the covering relation can be expressed as follows: Given two subscriptions  $s_1 = \{p_1^1, p_2^1, \dots, p_n^1\}$  and  $s_2 = \{p_1^2, p_2^2, \dots, p_m^2\}$ ,  $s_1$  covers  $s_2$  if and only if  $\forall p_k^1 \in s_1, \exists p_j^2 \in s_2$  such that  $p_k^1$  and  $p_j^2$  refer to the same attribute, and if  $p_j^2$  is matched by some attribute-value pair ( $a, val$ ), then  $p_k^1$  is also matched by the same ( $a, val$ ) attribute-value pair. In other words,  $s_2$  has potentially more predicates and they are more restrictive than those in  $s_1$ . Table 1 presents some examples of subscriptions and covering relations.

Informally, when a broker  $B$  receives a subscription  $s$ , it will send it to its neighbors if and only if it has not previously sent them another subscription  $s'$ , that covers  $s$ . Broker  $B$  will receive all publications that match  $s$ , since it receives all publications that match  $s'$  and the publications that match  $s$  are a subset of the publications that match  $s'$ .

Table 1 presents some examples of subscriptions and the corresponding covering relations. The goal of subscription covering is to quench subscription propagation, thereby reducing network traffic and trimming the size of subscription (i.e. routing) tables.

The processing of the subscriptions at the pub/sub router then proceeds as follows:

1. For each incoming subscription, check if there are *covering* subscriptions in the subscription table. If there are, then we do not need to forward the subscription.
2. If there is no matching subscription, *insert* incoming subscription in the subscription table.

Subscription $s$	Advertisement $a$	Intersection Relation
(product = "computer", brand = "IBM", price $\leq$ 1600)	(product = "computer", brand = "IBM", price $\leq$ 1500)	$a$ intersects $s$
(product = "computer", price $\leq$ 1600)	(product = "computer", brand = "IBM", price $\leq$ 1600)	$a$ intersects $s$
(product = "computer", brand = "IBM", price $\leq$ 1600)	(product = "computer", brand = "Dell", price $\leq$ 1500)	$a$ does not intersect $s$

**Table 2: Examples of subscriptions, advertisements and intersection relations**

3. Check if there are *intersecting* advertisements in the advertisement table. If there are, forward the subscription to those neighbors from which we received the matching advertisements.

### 2.2.3 Forwarding Publications

Finally, publications are processed as follows.

- For each incoming publication, check if there are *matching* subscriptions in the subscription table. If there are, forward the publication to all the neighbors from which each of the matching subscriptions was received.

## 2.3 Publish/Subscribe Broker Network

A number of pub/sub brokers can be organized into a content-based routing network. In such a network, one of the most important problems is the routing of a publication to interested subscribers based on the content of the publication. There are several routing protocols proposed in the literature [6, 8, 3], all of which follow the described pub/sub broker architecture to some extent. Some of them do not use advertisements and some of them do not perform subscription covering.

Generally, the proposed solutions always involve a network of brokers that collaborate in order to route the information in the network based on its content. Advertisements from publishers are flooded throughout the network, and stored at each broker’s routing table in order to build a distributed advertisement tree. When a subscriber receives an advertisement, it sends covering subscriptions along the reverse path of the advertisement tree. These subscriptions are stored in the routing table of each broker along the subscription path, and result in a distributed multicast tree. Finally, publications from a publisher follow the reverse path of all matching subscriptions (i.e. the multicast tree rooted at the publisher) and delivered to interested subscribers.

A fundamental assumption of pub/sub systems is that the number of advertisements is much less than the number of subscriptions which is much less than the number of publications. This justifies the flooding of advertisements. However, we will show that this assumption is not valid when publishers are mobile. Moreover, when routing computations are taken into account, the effects of mobility are even more pronounced.

## 3. MODELING ROUTING COMPUTATIONS

*Matching, insertion, intersection, and covering*, as defined in Section 2.2, are the key *routing computations* in a pub/sub router. A number of algorithms have been proposed to efficiently perform some of these operations [11, 20, 9, 13]. The performance of the algorithms is typically evaluated only on a single broker (router) and usually the performance results for matching, insertion and covering are reported in isolation. However, as Section 2.2 shows, a realistic pub/sub

router usually needs to perform a series of steps that frequently involve several routing computations (e.g., covering followed by insertion or matching). Quantitative evaluations of distributed pub/sub networks *ignore* routing computations and they use the good published performance results of algorithms on a single router to justify this [6, 5, 8]. But as our experimental results indicate, the performance results of routing computations on a single router *do not reflect their true effect on a network of routers*.

To truly evaluate the effects of routing computations, we develop two models based on published matching, insertion, intersection, and covering algorithms. We strived for simplicity that includes only those components in route computation contributing the most to the processing delays. The matching, insertion, intersection, and covering algorithms can be classified according to the expressiveness of the subscription and data representation language. In this paper, we define and study two representative algorithm classes: FAST and COMPLEX.

The first model represents a class of algorithms for subscription and publication representation languages with relatively efficient matching algorithms. We identify these with the FAST algorithm class. This models a group of algorithms such as [11] and [7]. Usually, FAST class algorithms represent publications as lists of attribute-value pairs and subscriptions as a conjunction of predicates, as described in the Section 2.2.

The second model represents a class of algorithms that provide more expressive subscription and publication representation languages such as [20] and [9]. We represent this class with the COMPLEX algorithm. Usually, COMPLEX class algorithms represent a publication as a directed labeled graph or tree path. A COMPLEX subscription is a directed graph pattern specifying the structure of the publication graph with optional constraints on some vertices. Some of the more expressive COMPLEX class algorithms have SQL-like subscription languages [9]. Expressiveness of a COMPLEX class subscription language allows efficient filtering of XML/RDF-expressed metadata. Common applications include filtering of RSS documents on the Internet and PDF documents in the enterprise. According to [pewinternet.org](http://pewinternet.org), currently there a 6 million people in the US who use RSS aggregators to receive published information. Consequently, there is a growing need for efficient content-based routing protocols that can support the kind of subscription and data language expressiveness provided by the COMPLEX algorithm class.

In developing both models, we use the simplified architecture of a pub/sub router as our guide (see Section 2.2). For FAST we take the *best* reported results (i.e., under the most favorable workload for FAST) for publication matching and conservatively assume that this time is independent of the number of subscriptions. For the COMPLEX algorithm we also take the *best* reported results (i.e., under the most favorable workload for COMPLEX) but assume the matching time

increases linearly with the number of subscriptions. The linear scalability in the number of subscriptions is based on actual published results [20, 9].

To the best of our knowledge, a quantitative evaluation of a FAST or COMPLEX class covering or intersection algorithm has not been published. Consequently, we model the time to perform subscription covering and intersection as being proportional to the publication matching time. For the FAST class this means that calculating the covering or intersection relation takes *constant* time regardless of the number of subscriptions and for the COMPLEX class, the time increases linearly with the number of subscriptions.

Finally, we assume that the insertion times are also proportional to publication matching time. Since publication matching time is what these algorithms are generally optimized for, they usually do it at the expense of insertion time. Therefore, we feel that assuming insertion and matching times are proportional is a very conservative assumption. Furthermore, we assume that the number of subscriptions in the network will dominate the number of advertisements, so we *ignore* the computations involving the advertisement table (also a very conservative assumption).

We believe that the models are based on very conservative assumptions about the performance of routing computation algorithms. Since the most favorable workload conditions for the FAST and COMPLEX classes are different, due to differences in the expressiveness of their respective subscription languages, FAST and COMPLEX classes cannot be directly compared. Our results merely indicate the performance of a content-based routing network for the two classes of routing computation algorithms under their respective most favorable workloads.

## 4. MOBILE PUBLISHERS

Client mobility is based on MOVEIN and MOVEOUT operations [8] that offer clients the ability to disconnect from and reconnect to the system. To support the disconnected operation of subscribers, brokers must store publications for the subscriber, and replay them to the subscriber when it reconnects to the network. However, there are no special protocols to handle publisher mobility. Unlike disconnected operation with subscribers, there is no information that the publisher has missed while disconnected. It is perhaps for this reason that no new protocols have been proposed for handling publisher mobility. We will show in Section 5 why this is a problem.

Below we describe the standard publisher mobility protocol as well as some optimizations that we propose.

### 4.1 Standard Protocol

Figure 1 illustrates a timeline of a publisher going into a period of disconnection and reconnecting to a different broker. During period  $t_1$ , the publisher is connected to Broker 1, the publisher rooted advertisement and multicast trees have been built, and publications are correctly multicast to all interested subscribers. At the end of period  $t_1$ , the publisher disconnects from Broker 1 and reconnects after period  $t_3$  to Broker 2. Period  $t_2$  is used by the PREFETCHING and PREFETCH-DELAYED optimizations below. Period  $t_4$  is required to complete the reconnection phase, which involves rebuilding advertisement and multicast trees. Finally, during period  $t_5$  publications from the publisher are again correctly multicast to all interested subscribers.

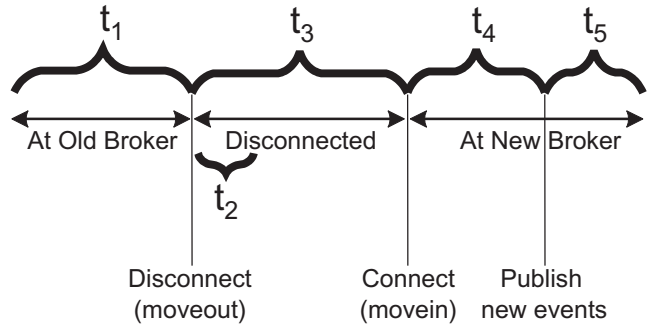


Figure 1: Publisher mobility timeline

The objective of the publisher mobility protocol is to reconfigure the advertisement and multicast trees to account for publisher mobility. We assume the publisher always reconnects to its physically closest broker. When a publisher disconnects from Broker 1, Broker 1 sends an unadvertisement message for any advertisements it has received from the publisher. Note that these unadvertisements may induce unsubscriptions to be propagated in the reverse direction of the unadvertisement propagation (to tear down the multicast tree) just as advertisements induce subscriptions to be sent. This tree teardown occurs during period  $t_2$  after which there is no state associated with the publisher in the system. At the end of period  $t_3$  the publisher connects to Broker 2, and upon reconnection sends its advertisements again. During period  $t_4$  the advertisement and multicast trees are rebuilt, and finally publications sent during period  $t_5$  are delivered to all interested subscribers. It is this mobility induced tree teardown and reconstruction that makes the assumption of few advertisements invalid in this context.

Publications sent during period  $t_4$  may not be delivered to interested subscribers since the multicast tree has not been rebuilt yet. However, in this protocol, there is no way to know when period  $t_4$  is complete; Broker 2 does not know for certain when the multicast tree has been rebuilt. This is a fundamental problem arising from the decoupling of publishers and subscribers in the pub/sub model; publishers do not know the set of subscribers that are interested in their content. In fact, no one node knows the set of participants in a given multicast tree. This problem is exacerbated by the fact that it is difficult to distinguish between new subscriptions that enter the system shortly after the publisher moves in, and old subscriptions that simply arrive slowly at the publisher. The pub/sub semantics require the former subscribers to receive publications from the publisher, while the former is allowed to miss some publications until they propagate through the system. Since the length of period  $t_4$  is unknown, we would like to minimize this period so as to minimize the probability that a publication sent soon after reconnection is not delivered to an interested subscriber.

Each broker has an `inAds` table to store  $(ad, nodeid)$  pairs, and an `outAds` table to store  $(ad, nodeid, sent)$  tuples. The contents of these tables, along with the corresponding `inSubs` and `outSubs` tables are summarized in Table 3. Figures 2 and 3 show the pseudo-code for handling advertisements, and subscriptions, respectively. Handling unadvertisements and unsubscriptions follow in a similar manner. Notice that (un)advertisement propagation can induce (un)subscription

**Algorithm receive(Advertisement  $a$ )**  
 (\* Store and forward an advertisement. \*)

1. (\* Store in table \*)
2.  $\text{inAds} \leftarrow \text{inAds} \cup (a, a.sender)$
- 3.
4. (\* Forward ad to neighbours \*)
5. **for** each neighbour  $n$  where  $n \neq a.sender$
6.     **do if**  $\exists(ad, n, true) \in \text{outAds}$  where  $ad.covers(a)$
7.         **then**  $\text{outAds} \leftarrow \text{outAds} \cup (a, n, false)$
8.         **else**  $\text{outAds} \leftarrow \text{outAds} \cup (a, n, true)$
9.              $\text{send}(a, n)$
- 10.
11. (\* Send matching subscriptions back towards ad \*)
12. **for** each  $(s, n) \in \text{inSubs}$  where  $s.intersects(a)$  and  $n \neq a.sender$
13.     **do if**  $\exists(sub, n, true) \in \text{outSubs}$  and  $sub.covers(s)$
14.         **then**  $\text{outSubs} \leftarrow$
15.              $\text{outSubs} \cup (s, a.sender, false)$
16.         **else**  $\text{outSubs} \leftarrow$
17.              $\text{outSubs} \cup (s, a.sender, true)$
18.              $\text{send}(s, a.sender)$

**Figure 2: Advertisement handler**

**Algorithm receive(Subscription  $s$ )**  
 (\* Store and forward a subscription. \*)

1. (\* Store in table \*)
2.  $\text{inSubs} \leftarrow \text{inSubs} \cup (s, s.sender)$
- 3.
4. (\* Forward toward reverse path of advertisements \*)
5. **for** each  $(a, n) \in \text{inAds}$  where  $a.intersects(s)$  and  $a.sender \neq s.sender$
6.     **do if**  $\exists(sub, n, true) \in \text{outSubs}$  and  $sub.covers(s)$
7.         **then**  $\text{outSubs} \leftarrow$
8.              $\text{outSubs} \cup (s, a.sender, false)$
9.         **else**  $\text{outSubs} \leftarrow$
10.              $\text{outSubs} \cup (s, a.sender, true)$
11.              $\text{send}(s, a.sender)$

**Figure 3: Subscription handler**

propagation. That is, the (de)construction of the advertisement tree induces the (de)construction of the multicast tree.

## 4.2 Prefetching Protocol

The PREFETCHING protocol exploits knowledge of future mobility patterns. It is similar to the STANDARD protocol except that the advertisement and multicast trees are rebuilt during period  $t_2$  instead of period  $t_4$ . Therefore, the length of period  $t_4$  is now zero, and any publications sent immediately after reconnection are delivered to interested subscribers. Note that  $t_2$  is the time it takes to rebuild trees, and is independent of the disconnection time.

This protocol has the advantage of hiding tree rebuilding time from the publisher since it occurs while the publisher is disconnected. Also, since the old tree (at Broker 1) is being torn down concurrently with the building of the new tree (at Broker 2), it may occur that the new tree grafts onto the old tree before it is torn down, obviating the need to tear down the old tree completely. The DELAYED protocol below tries to force this case, which only occurs by chance in the PREFETCHING protocol.

## 4.3 Proxy Protocol

The PROXY protocol is an extension of the PREFETCHING protocol. The assumption here is that publishers tend to move within a restricted area. For example, a taxi driver

inAds	Store advertisements received from neighbors
$t.ad$	Advertisement pattern
$t.nodeid$	Address of neighbor that sent $ad$
outAds	Store advertisements sent to neighbors
$t.ad$	Advertisement pattern
$t.nodeid$	Address of neighbor that $ad$ should be sent to
$t.sent$	True if $ad$ was actually sent to $nodeid$ (may be false if there is an $ad'$ that covers $ad$ )
inSubs	Store subscriptions received from neighbors
$t.sub$	Subscription pattern
$t.nodeid$	Address of neighbor that send $sub$
outSubs	Store subscriptions sent to neighbors
$t.sub$	Subscription pattern
$t.nodeid$	Address of neighbor that $sub$ should be sent to
$t.sent$	True if $sub$ was actually sent to $nodeid$ (may be false if there is a $sub'$ that covers $sub$ )

**Table 3: Broker tables**

may service only certain regions of a city. (The taxi may be publishing location updates to a dispatcher or potential customers). The PROXY protocol assigns a set of brokers to act as proxies for the publisher. These proxies always maintain a tree for the publisher. This way, there is no teardown or rebuilding of the tree when the publisher disconnects from or connects to one of its proxies. However, tree rebuilding does need to occur if the publisher connects to a non-proxy broker.

## 4.4 Delayed Protocol

The DELAYED protocol exploits the fact that the old tree (rooted at Broker 1) and the new tree (rooted at Broker 2) have significant overlap. This is especially true if Broker 1 and 2 are nearby, as in the case of an always connected mobile publisher. In the DELAYED protocol, the teardown of the old tree at Broker 1 is delayed for some time after MOVEOUT. This time is to allow the publisher to reconnect to another broker, and graft the new publication tree to the old one. After the delay, the old broker tears down only the extraneous portions of the combined tree.

## 4.5 Prefetch-Delayed Protocol

The PREFETCH-DELAYED protocol is a combination of the PREFETCHING and DELAYED protocols. As in the PREFETCHING protocol, PREFETCH-DELAYED initiates tree rebuilding at Broker 2 when the publisher disconnects from Broker 1, thereby hiding the tree reconstruction time from the publisher. However, since the tree rooted at Broker 1 is torn down concurrently with the construction of the tree rooted at Broker 2, it may occur that the old tree is completely torn down before the new tree is built. In the PREFETCH-DELAYED protocol, the teardown of the old tree is delayed to allow the new tree to graft onto the old one. In this way, PREFETCH-DELAYED offers the advantages of both the PREFETCHING and DELAYED protocols.

## 4.6 Discussion

These optimizations make minimal assumptions about the underlying system, and can be used with any type of distributed pub/sub system. Moreover, the optimizations can be combined. For example, PROXY can be combined with DELAYED to potentially achieve even better performance.

It is instructive to notice that the STANDARD protocol does

not distinguish between a moving publisher and a publisher that leaves and enters the system. Therefore it discards all state (advertisement and multicast trees) associated with a publisher on MOVEOUT, and must completely rebuild it on MOVEIN. Our optimizations address this issue.

Since publisher mobility causes expensive reconstruction of advertisement and multicast trees, it may be tempting to eliminate advertisement flooding and flood subscriptions instead. However, subscribers typically outnumber publishers, so the savings in publisher mobility induced tree rebuilding cost do not justify subscription flooding. Furthermore, subscriber mobility will now cause multicast tree reconstruction; this reconstruction is much more expensive than when advertisements are used, since the multicast trees now span the whole network rather than being a minimal tree from the subscriber to interesting publishers.

## 5. EVALUATION

In this section we first discuss the factors that affect the evaluation of a distributed pub/sub system that supports mobility. We then present an experimental evaluation of the protocols introduced in Section 4.

### 5.1 Parameters

Several factors must be considered when designing or evaluating a distributed pub/sub system supporting mobility. We classify these factors into three categories: network, mobility, and application. Note that several of these factors are also issues in pub/sub systems that do not support mobility. The discussion below is meant to give insights on how various factors may affect performance.

Network characteristics are those concerned with the network infrastructure:

- **Link bandwidth and latency.** This includes the links between brokers and those between brokers and clients. Faster links can perform tree reconstruction quicker.
- **Placement of brokers.** This is related to availability and load balancing issues: it is important to place more brokers where user demand is high. Moreover, strategic placement of brokers can help isolate traffic in the network when publishers and subscribers exhibit interest locality.
- **Broker topology.** Intuitively, the “deeper” the topology (more hops between any two nodes), the taller the advertisement and multicast trees tend to be, and hence the longer it takes for tree rebuilding. In contrast, a wider topology increases the load on each broker since it serves a greater number of neighbor (descendant) brokers in the tree.
- **Number of brokers.** With an increased broker density, physical mobility is more likely to lead to network mobility, that is, the need to connect to a different broker. The tree rebuilding costs resulting from this network mobility, will therefore increase.

Mobility characteristics are related to the movement and disconnection patterns of users:

- **Connection and disconnection times.** Long connection or disconnections periods (i.e., slow mobility)

result in fewer tree rebuilding. On the other hand, short disconnection time can be exploited by the DELAYED and PREFETCH-DELAYED protocols.

- **Mobility patterns.** It is possible to develop optimizations tuned for certain mobility patterns. For example, with a repetitive pattern we can predict user movement, and thus use the PREFETCHING or PROXY optimizations. Other optimizations might take advantage of group mobility patterns where a large number of users (perhaps with common subscription interest) move together.

Application specific characteristics are associated with the nature of publishers, subscribers, publications and subscriptions:

- **Routing computations.** The expressiveness of the subscription and publication representation language determines the processing delays. Simple subscription and publication languages are able to have low delays and are more resilient to fluctuation in message loads. However, limited expressiveness also limits their usefulness. Representation languages considered more useful, such as those dealing with XML/RDF, are also more expressive and hence more expensive in terms of routing computations. As a result, unless effective mobility protocols are used, the network reconfiguration traffic causes excessive processing delays.
- **Number of publishers and subscribers.** The implications of this factor are captured in several other factors below. For example, an increase in the number of publishers would increase the aggregate publication rate, as well as the aggregate mobility, and hence the total number of tree reconstructions. Increased subscribers also causes more expensive tree rebuilding. Increasing the number of publishers or subscribers can lead to an increase or decrease in subscriber and interest locality.
- **Publishing rate.** A higher publication rate increases the regular pub/sub multicast traffic, as well as the need to rebuild trees faster.
- **Specificity of subscriptions.** A very general subscription will find more interesting publishers and increase the cost of the second tree rebuilding phase (subscription propagation).
- **Subscriber locality.** This relates to whether subscribers tend to be localized to a set of brokers or spread throughout the network. It is also concerned with the (network) distance between subscribers and the publishers publishing events they are interested in. This can influence the balance of load across the brokers in the network.
- **Subscription (interest) locality.** This looks at the correlation of subscriptions in the network. Lots of similar subscriptions leads to more efficient multicast trees and makes the second phase of tree rebuilding (subscription propagation) less costly.
- **Publisher and publication locality.** Many publishers publishing similar content from the same portion of the network will result in highly overlapping

advertisement and multicast trees. This makes mobility cheaper. To some extent, the PROXY protocol tries to force this situation.

- **Event (publication) size.** The contents of a publication might consist of only the predicates describing that event, or might include other data such as a video. Larger publications lead to more traffic which may cause network congestion and slow down tree rebuilding.

## 5.2 Experiments

Below we describe our experimental setup and metrics, and then discuss the experiments in detail.

### 5.2.1 Methodology

We perform all our experiments using the ns2 network simulator [4]. We extend ns2 with the STANDARD mobility protocol and the PROXY, DELAYED, and PREFETCH-DELAYED protocols presented in Section 4.

Designing realistic experiments to evaluate pub/sub systems is difficult. The developing publish/subscribe field has yet to produce realistic values for many important parameters discussed in Section 5.1. Moreover, it is difficult to obtain real-world traces for these tests. Cellular phone companies, for example, are understandably hesitant to divulge information on the mobility or disconnection patterns of their users. We have tried to provide reasonable parameters where possible, and random values for parameters that are highly application specific.

The experiments simulate a city-wide pub/sub scenario, with a network of 64 brokers distributed across a city. These brokers are the leaves of a tree of height 4 and degree 4. While our protocols work in general graph topologies, we feel a tree topology is common in metropolitan area networks. Conceptually, the first few leaf brokers are located at the west end of the city, the next few are downtown and the last few in the east end. Each broker services a 0.5km range, so the 64 brokers service a 32km wide city. Clients always connect to leaf brokers. All nodes communicate over TCP. A reliable transport is required because our protocols maintain hard state and cannot tolerate message losses. The brokers communicate with each other over a 10Mbps link with 2ms latency. The clients have a 128kbps link to the brokers.

Each publisher is assigned a random unique publication and an advertisement that is identical to its publication. (Unique advertisements are justifiable in content-based pub/sub because advertisements can be very expressive and customized for each publisher). Each subscriber randomly subscribes to one of the publications assigned to a publisher (there is no use simulating subscribers that are not interested in anything published since they have almost no impact on the system). Unless otherwise stated, there are 50 publishers and 500 subscribers in the system.

The publishers randomly move at speeds of 5km/h (walking), 50km/h (city driving) or 100km/h (highway driving). Random mobility is a conservative decision; a more structured mobility pattern will tend to create hotspots in the network, making the STANDARD mobility algorithm perform worse, and make the benefits of the optimizations more pronounced. Publishers connect and disconnect to adjacent brokers as they move, and the disconnection time when moving between brokers is three seconds. Note that this disconnection time is independent of the mobility speed. We keep

Parameter	Value
Broker topology	Tree of height 4 and degree 4
Number of brokers	85 (64 leaves)
Bandwidth (broker-broker)	10Mbps
Latency (broker-broker)	2ms
Bandwidth (client-broker)	128kbps
Latency (client-broker)	2ms
Broker placement	0.5km wide broker service area
Publisher speed	$\text{rand}(5 \frac{km}{h}, 50 \frac{km}{h}, 100 \frac{km}{h})$
Publisher direction	$\text{rand}(\text{east}, \text{west})$
Subscriber speed	0km/h
Disconnection time	3s
Number of publishers	50
Number of subscribers	500
Publisher advertisement	$\text{rand}(1, \text{number of publishers})$
Subscriber subscription	$\text{rand}(1, \text{number of publishers})$
Proxies (PROXY)	5
Unad delay (DELAYED and PREFETCH-DELAYED)	10s

Table 4: Common experimental parameters

the subscribers stationary in order to isolate the effects of publisher mobility. All clients connect to a random leaf broker during a warm-up phase, during which statistics are not counted. Subscribers subscribe immediately after connecting to a broker and never unsubscribe.

For the PROXY protocol, the five closest brokers to the broker to which a publisher first connects are assigned as its proxies. For the DELAYED protocol, the previous broker tears down the tree 10s after the publisher disconnects. For the PREFETCHING and PREFETCH-DELAYED protocols, we assume perfect mobility prediction. The STANDARD protocol does not depend on any parameters.

These parameters are summarized in Table 4.

### 5.2.2 Routing Computation Models

We model a pub/sub router (see Section 2.2) using the FAST local matching algorithm for performing routing computations as having 2ms publication matching time and 2ms subscription insertion time. Consequently, the total publication time is 2ms, the total subscription processing time is 4ms (covering + insertion), and the total advertisement processing time is 2ms (intersection). These are conservative estimates based on the results in [11].

For a COMPLEX-based router, we proceed in the same way, the only difference being the publication matching time, which is modeled as  $\frac{\text{totalSubscriptions}}{1000} + 40ms$ . For all the experiments, *totalSubscriptions* is 100,000, unless stated otherwise. Again, this is a conservative model derived from [20].

We *ignore* any other processing delays that may come from various sources (e.g., network protocol stack, operating system, etc.).

### 5.2.3 Metrics

We measure the cost of supporting mobile publishers in terms of the effects on the network and the clients. The network effect is measured as message load introduced by tree rebuilding. Each hop a message travels is counted. The effect on the user is measured as the delivery rate of publications, computed as the ratio of actual deliveries of publications to the expected number of deliveries.

We also introduce another metric that measures both the effect on the network and user: the tree rebuilding speed.

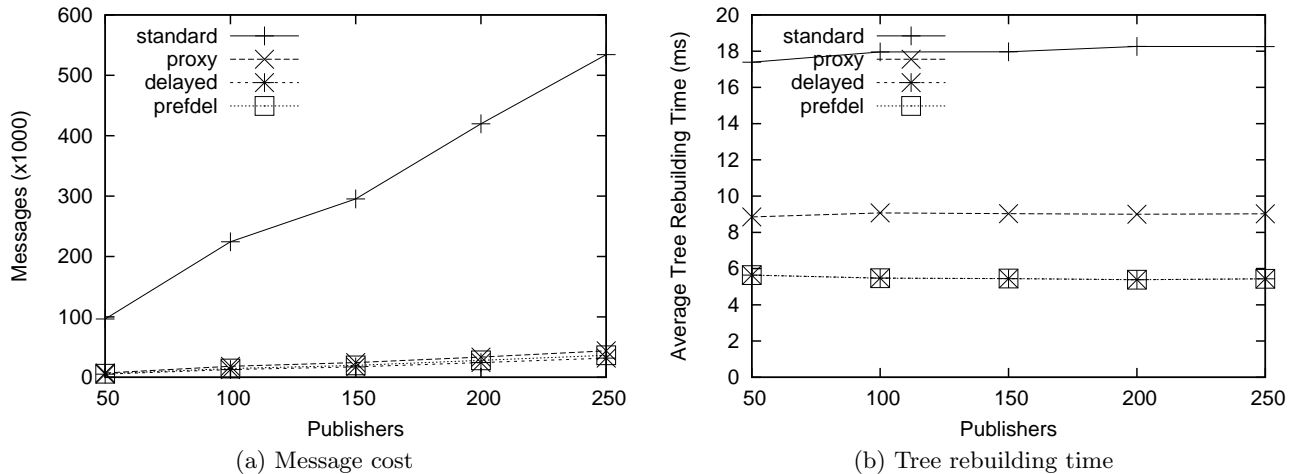


Figure 4: Publisher scalability without routing computations

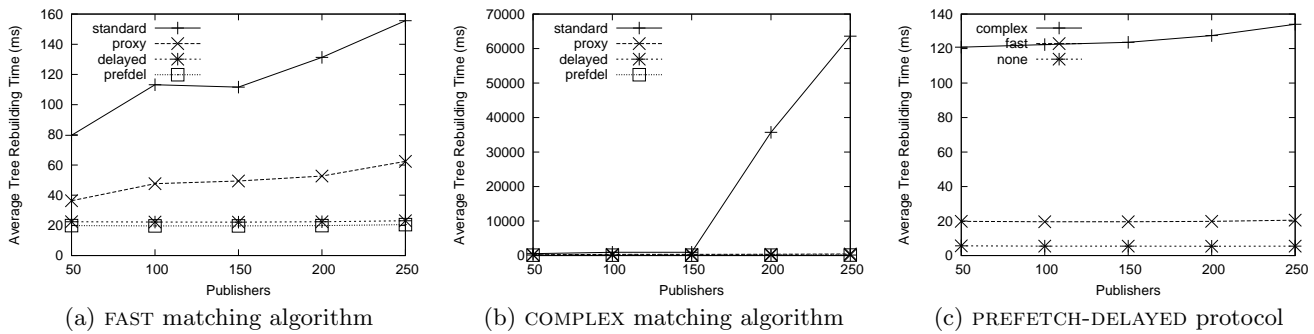


Figure 5: Publisher scalability with routing computations

A fast tree reconstruction allows a publisher to send events sooner after moving into a broker, and indicates a lower network cost in terms of number of messages.

One technique to measure tree reconstruction time is to send a sequence of probe publications soon after reconnection and count the delivery of these probes. The faster the tree is rebuilt, the sooner events get delivered to all interested subscribers. This technique for measuring tree rebuilding has the disadvantage of having a granularity equal to the frequency of the probes, and introduces network traffic that itself could affect the tree rebuilding time. Note that attempts to decrease the network cost of the probe publications by reducing the probe rate adversely impacts the granularity of the tree rebuilding time measurement, while increasing the probe rate increases the network impact.

In this paper, we measure tree rebuilding time offline in our simulation. Tree construction begins when the publisher performs a MOVEIN operation on a broker and sends an advertisement message to the broker. As described in Section 4, as this advertisement propagates through the network, it may induce subscriptions to be sent (as part of the multicast tree reconstruction). Tree construction completes when the advertisement message has stopped propagating through the network and the last subscription message induced by the advertisement is processed. This technique for measuring tree construction imposes no network cost and

accurately measures tree construction time. The tree tear-down time is measured in a similar fashion by examining unadvertisements and unsubscriptions.

#### 5.2.4 Publisher Scalability

In this experiment we evaluate the scalability of the protocols with an increasing number of publishers (50, 100, 150, 200, 250). Note that an increase in the number of mobile publishers also increases the aggregate mobility in the network.

Figure 4(a) shows the tree building message cost for the four mobility protocols without modeling routing computations. The cost of tree rebuilding grows approximately linearly. However, there is a substantial difference in the cost of the protocols. The STANDARD protocol has more than ten times the message cost of the PROXY, DELAYED, and PREFETCH-DELAYED protocols. STANDARD performs poorly because every MOVEOUT (MOVEIN) causes the entire advertisement and multicast trees for the moving publisher to be torn down (rebuilt). While not shown here, the same is true for PREFETCHING but with an additional cost of having the old broker inform the new broker to start rebuilding the tree. Therefore PREFETCHING is not building the new tree fast enough to graft onto the old tree that is being torn down. Since PREFETCHING does not offer any benefits that are not present in PREFETCH-DELAYED, we do not present

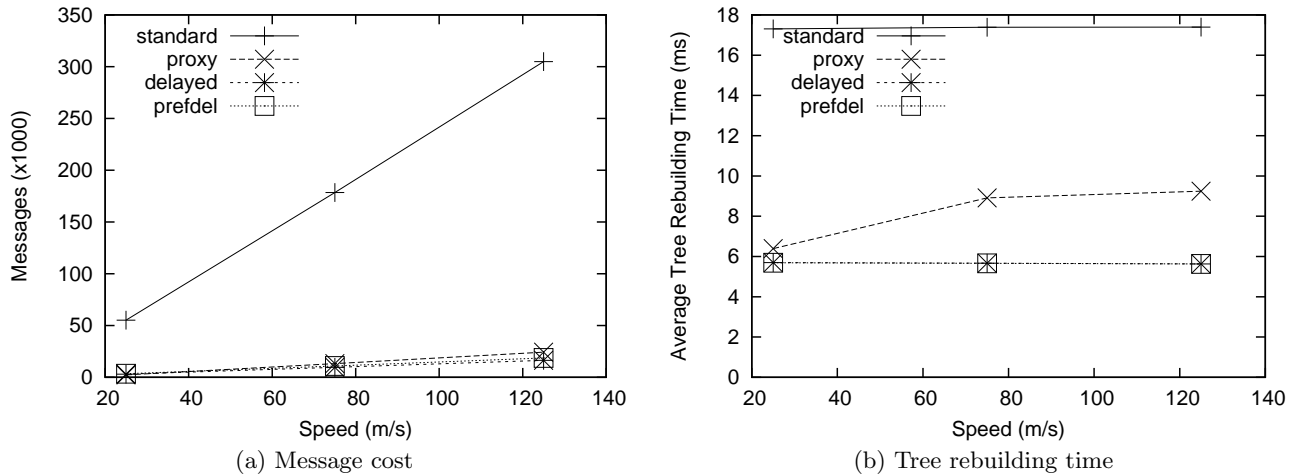


Figure 6: Publisher speed of movement without routing computations

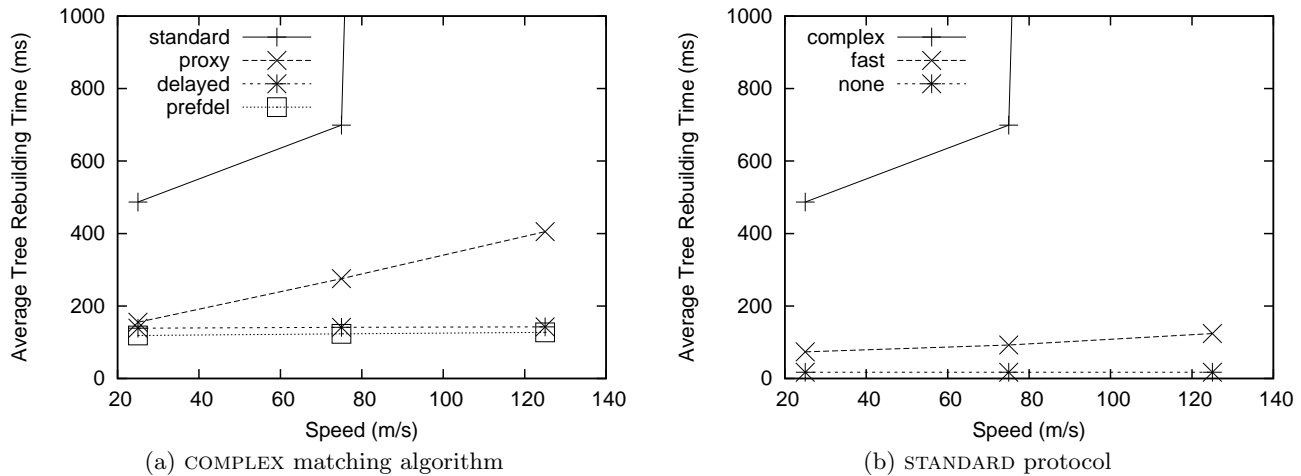


Figure 7: Publisher speed of movement with routing computations

results for the PREFETCHING protocol in this paper in order to simplify the exposition.

On the other hand, the PROXY, DELAYED and PREFETCH-DELAYED protocols all graft onto existing trees. The reason DELAYED performs better than PROXY is because in DELAYED the old tree is rooted at a nearby broker (recall that the publishers move along adjacent brokers), so the distance an advertisement must travel to graft onto an existing tree is short. In PROXY, the distance to the old tree depends on the position of the publisher relative to its fixed proxies. PREFETCH-DELAYED has the same message cost as DELAYED. The only difference between the two is that PREFETCH-DELAYED performs tree reconstruction earlier—during the publisher’s disconnection period.

Figure 4(b) shows the average time to rebuild trees for the protocols without routing computations. The STANDARD protocol suffers the longest tree construction time because it performs a complete tree teardown and reconstruction on a MOVEOUT and MOVEIN, respectively. The PROXY protocol has shorter tree rebuilding time since the tree reconstruction stops when it meets one of the existing trees maintained

by the proxies. The DELAYED protocol also only performs partial tree reconstruction, but improves on PROXY since the old tree in the DELAYED protocol is rooted at an adjacent leaf broker (since publishers move along adjacent brokers), while the trees maintained by PROXY may be nearby or distant from the publisher. Note that while the DELAYED and PREFETCH-DELAYED protocols have the same tree reconstruction time, the latter has the advantage of hiding construction time during publisher disconnection periods, but of course requires mobility prediction.

It should be noted that the good performance of the DELAYED protocol only requires that the unadvertisement delay (10s here) be larger than the disconnection time of the publisher. If this is not the case, DELAYED would have the tree rebuilding time of the STANDARD protocol, and additional message cost of the STANDARD protocol.

The tree rebuilding times shown in Figure 4(b) indicate that the mobility protocols scale very well; they are hardly affected by the number of mobile publishers. However, this conclusion is incorrect, since, as we will show, routing computations are not taken into consideration. Figure 5(a) dis-

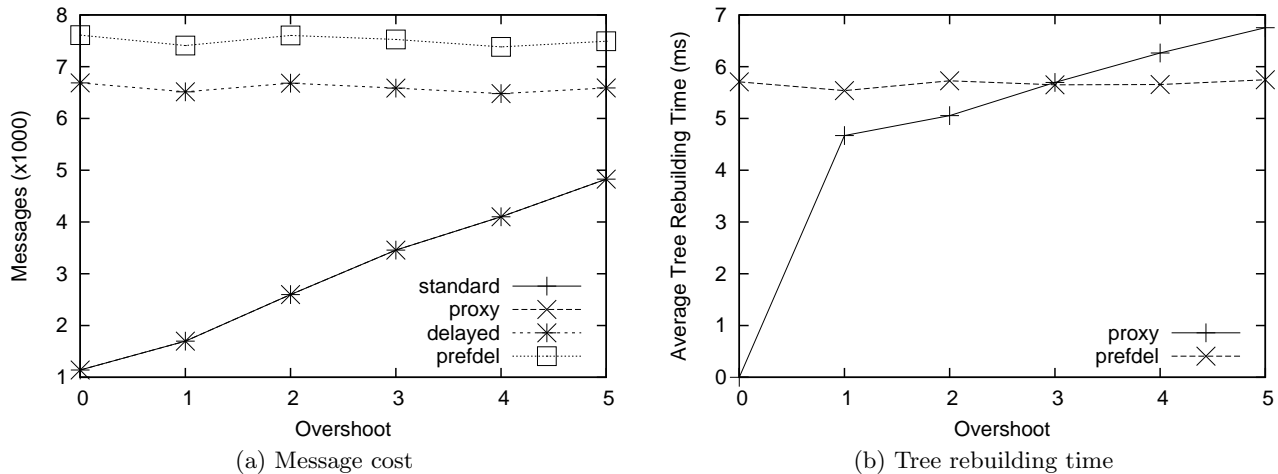


Figure 8: Publisher proxy sensitivity

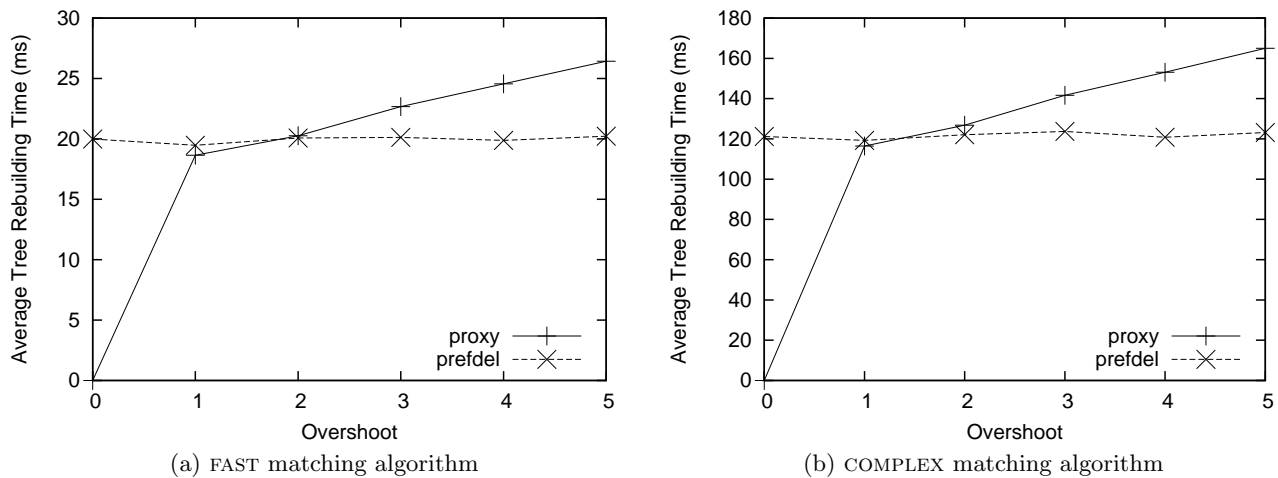


Figure 9: Publisher proxy sensitivity with routing computations

plays the tree rebuilding time for the same experiment but with the modeling of route computations using the FAST matching algorithm. The first observation is that tree rebuilding time has increased by an order of magnitude. Furthermore, the scalability trend of the STANDARD protocol is now vastly different. The STANDARD protocol is no longer unaffected by the number of mobile publishers. However, we see that our DELAYED and PREFETCH-DELAYED mobility protocols maintain good scalability and are not affected by the number of publishers.

Figure 5(b) shows the tree rebuilding time for the same experiment but using the COMPLEX route computation model. Here we see that after 150 publishers, the performance of the STANDARD protocol suffers drastically. With 250 publishers, the average tree rebuilding time for the STANDARD protocol is more than 60 seconds. This is because the COMPLEX algorithm imposes a processing time that cannot keep up with the number of messages in the STANDARD mobility protocol.

The above result is important because it shows that routing computations can have a profound impact on the per-

formance characteristics of mobility protocols. Existing distributed pub/sub research has ignored the effects of routing computations with the assumption that the network is the bottleneck. However, the above result illustrates this is a misconception. We think the the impact of routing computations on distributed pub/sub protocols will have an even bigger impact in the future as more expressive matching algorithms are developed and demanded by users.

Figure 5(c) illustrates this more clearly. Here we show the tree rebuilding time for our best mobility protocol, PREFETCH-DELAYED, with various route computation algorithms. We see that even for this mobility protocol, the impact of the routing computations is significant.

### 5.2.5 Publisher Mobility Speed

In this experiment, we measure the scalability of the protocols with increasing publisher mobility speed. The speed of the publishers range from 25km/h to 125km/h. Figure 6(a) shows that the faster speed leads to more frequent tree rebuilding and hence a larger tree rebuilding message load on the network. This trend is true for all the mobility

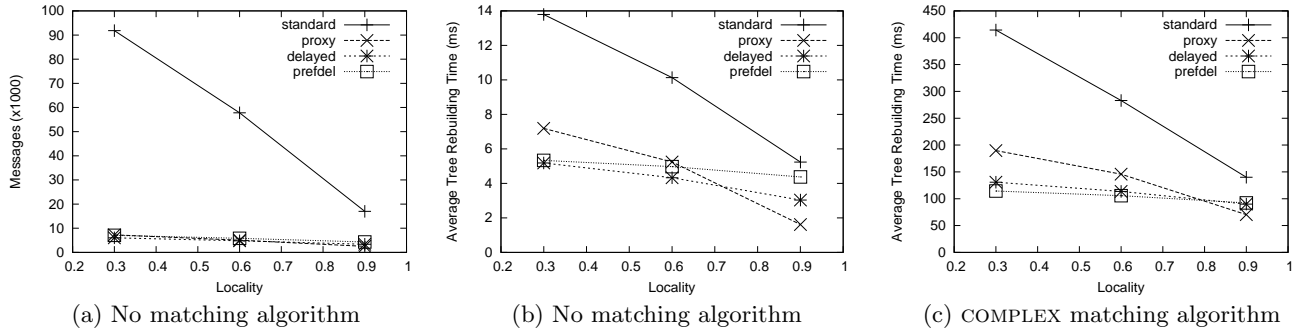


Figure 10: Publication locality with routing computations

protocols, but our optimized protocols reduce the message load by an order of magnitude from the STANDARD mobility protocol.

Figure 6(b) shows that our mobility protocols improve the tree rebuilding time. While all four protocols seem impervious to mobility speed, Figure 7(a) shows that this is not so when the COMPLEX algorithm is used for route computations. The importance of modeling routing computations is made clear in Figure 7(b) where the tree rebuilding time for the STANDARD mobility protocol is plotted with different algorithms; the STANDARD protocol seems to scale well until routing computations are taken into account.

### 5.2.6 Proxy Locality

In this experiment we test how sensitive the PROXY protocol is to choosing a good set of proxy brokers. Initially, publishers only move among their five adjacent proxy brokers. This may be the case of a police officer patrolling a few city blocks repeatedly. (The police officer may be publishing status reports, calls for backup, or accident report information). Then we vary how much the publisher may overshoot this proxy set. An overshoot of  $\delta$  means that the publisher may move  $\delta$  brokers past its proxy set.

As expected, Figures 8(a) and 8(b) show that the tree rebuilding cost and time, respectively, grow as the publisher moves farther away from its proxy set. In particular, when  $\delta = 3$ , the proxy protocol's tree rebuilding time becomes as expensive as that of the PREFETCH-DELAYED protocol. Interestingly, when routing computations are taken into account, the same trends are seen but the point at which the proxy protocol becomes worse than the PREFETCH-DELAYED protocol changes. With the FAST algorithm in Figure 9(a), PREFETCH-DELAYED begins to outperform PROXY when  $\delta = 2$ . Moreover, when the COMPLEX algorithm is used, in Figure 9(b), PREFETCH-DELAYED performs similar to PROXY even with  $\delta = 1$ . Therefore, the negative impact on PROXY of a publisher moving outside its proxy set is greater (relative to the other mobility protocols) with more expressive subscription languages as routing computations take more time. The figures above indirectly show the effect of the number of proxies; increasing the number of proxies will lower the average overshoot, and have a roughly linear performance impact.

### 5.2.7 Publication Locality

In this experiment we vary the degree of similarity of publications in the system. We achieve  $x\%$  locality by having

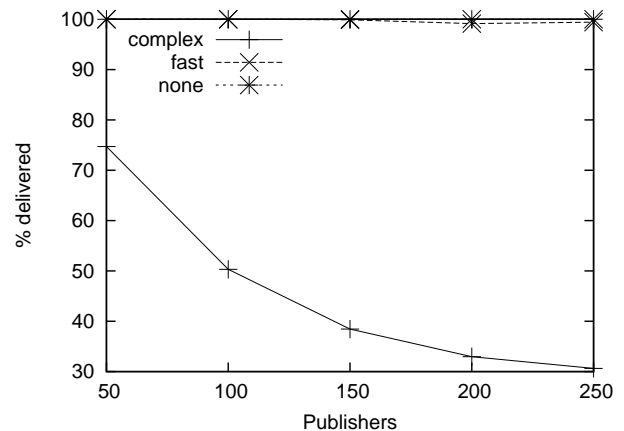


Figure 11: Data Delivery ratio

$x\%$  of the publishers publish the same publication and the remaining publish different and unique publications.

The results in Figures 10(a) and 10(b) indicate that with enough similarity between publications (90%), the STANDARD protocol's tree rebuilding message cost and time, respectively, approach that of our optimizations.

An interesting observation from Figure 10(b) is that the PROXY protocol gains the most from increased locality. In fact, with 90% locality, the PROXY protocol outperforms all the others. This is because PROXY has a side effect of magnifying the perceived locality in the network. Since each publisher has five proxy brokers, when there are two publishers with the same advertisement, there are not two but ten identical multicast trees maintained in the network.

While the use of the COMPLEX algorithm in Figure 10(c) does not affect the trends seen in Figure 10(b), it does affect the degree of locality required before PROXY outperforms the other mobility protocols. That is, as subscription languages become more expressive and expensive to process, the benefits of publication locality on PROXY will diminish.

### 5.2.8 Publisher Scalability With Publications

This experiment is similar to the publisher scalability experiment in Section 5.2.4, but we introduce publications into the network. Publishers publish an event every two seconds while they are connected to a broker. Figure 11 shows the ratio of publications that are successfully delivered to interested subscribers. The results are shown for the STANDARD

mobility protocol with different algorithms performing routing computations. The important point in this figure is that while the STANDARD protocol seems to provide 100% delivery even with increasing publishers, the introduction of the more expressive COMPLEX algorithm severely degrades the delivery rate.

Therefore, we emphasize again that routing computations cannot be ignored in the evaluation of distributed pub/sub protocols. Modeling these algorithms in our simulations leads to different conclusions on the performance of the distributed protocols.

## 6. RELATED WORK

To the best of our knowledge we are the first to look at the effects of routing computations on pub/sub middleware which supports mobile sources.

The publish/subscribe matching problem has been investigated extensively [11, 1, 9]. This work concentrated on devising efficient algorithms for matching in centralized publish/subscribe systems. Scalability concerns prompted researchers to look at distributed pub/sub protocols [6, 8, 3]. Common to all of them is the use of a network of brokers which rely on multicast protocols for efficient dissemination of events. These approaches, however, do not address the issue of mobile publishers, and they ignore the route computation costs.

Some distributed pub/sub systems [6, 8, 23] have added extensions to support mobile subscribers. The problem with mobile subscribers is the efficient storage and replay of publications missed by a subscriber while disconnected. We, on the other hand, examine extensions for supporting mobile publishers. Mobile publishing is a different problem stemming from the asymmetry of publishers and subscribers in a pub/sub system. We propose and evaluate a set of optimizations that reduce the overhead associated with supporting mobile publishers.

Mobile IP [18] uses the concept of a home agent to handle mobile clients. Each IP client has a home agent associated with it. When some node wants to communicate with this client, it contacts the node directly. When the node is roaming, the home agent (transparently) redirects the connection to the new location of the mobile client. This is an approach that may help address mobile publisher costs by maintaining a tree at the home agent and eliminate tree reconstruction altogether. However it has been shown [5] that such an approach in a pub/sub system causes excessive traffic, because the gains of a multicast tree are lost to the unicast traffic from the mobile client to the home agent.

Picco *et al.* [22] consider having an overlay broker network, where any broker can connect to any other broker. In such as network, every client (publisher/subscriber) is also considered to be a broker. The authors then examine the effects of broker mobility on the network performance. We take a different approach, in that we consider a fixed broker topology where the light-weight clients, such as would be found on portable devices like PDAs and mobile phones are connecting to the broker network. Mobile clients, roam in their environment (e.g., a city) and they connect to the broker network via the broker that is closest to their current location. We feel that this scenario is representative of the existing cellular networks or of the 802.11 networked hot spots commonly found today in cafes, cafeterias, schools, universities and shopping malls.

This work—referred to as Mobile-ToPSS [5, 17]—is part of the ToPSS (Toronto Publish/Subscribe System) research projects [13, 14, 19, 20, 21, 16].

## 7. CONCLUSIONS AND FUTURE WORK

The decoupling properties of the pub/sub model makes it well suited for applications with mobile users. We feel that the number of mobile information producers will increase in the future. However, there has been no evaluation of mobile publishers in pub/sub systems that take into account routing computations. This is important because mobility breaks a fundamental pub/sub assumption (that the impact of the advertisements on the system is very low). Plus, we show that, even with a small number of mobile sources, routing computations can significantly affect the performance of the content-based routing network.

Our evaluation supports these claims: the current STANDARD mobility protocol causes excessive state reconstruction traffic which easily exceeds processing capacity of the network. In other words, the routers spend most of the time processing state maintenance data rather than routing user data. The PREFETCH-DELAYED and PROXY protocols perform well in terms of network load and user perceived performance. Usually PREFETCH-DELAYED is preferable because, while it performs only slightly better than PROXY, it does not require the additional task of assigning proxies to a publisher. However, when future mobility patterns are not known, but it is known that a publisher moves within a confined range, PROXY is the best choice. When nothing is known about mobility patterns, the DELAYED protocol is recommended. We cannot recommend the STANDARD protocol in any case. We also showed that all four mobility protocols improve their tree rebuilding time with increasing similarity of publications in the system, with the PROXY and STANDARD protocols gaining the most, since the DELAYED and PREFETCH-DELAYED protocols already perform very well. Furthermore, with sufficient similarity of publications, the STANDARD and PROXY protocols perform similar to the best protocols (DELAYED and PREFETCH-DELAYED).

The evaluation also demonstrated the importance of modeling computation time in distributed pub/sub protocols. Routing computation cannot be ignored as a lower order effect as is standard in the literature [6, 8, 3, 15, 5]. In our experiments, routing computation sometimes cause the time to rebuild trees to increase by an order of magnitude. In addition, the computation time affects not just the scale but the trends of the results and cause completely different conclusions to be reached. Protocols that seem to scale with increasing publisher mobility no longer do so when routing computations are considered. Moreover, using an expressive subscription and publication representation increases route computation times to the extent that the STANDARD protocol's performance collapses.

There is much more important work to be done in this area. We plan to refine the routing computation models with emerging algorithms that evaluate covering and intersection performance [13]. We would like to obtain real-world usage patterns and scenarios. This includes communication link speeds, mobility patterns (how fast, and where clients move), the publication rate, the distribution of subscriptions, the locality of clients (how close publishers and subscribers are to each other), and publication locality. We would also like to vary more parameters (such as the num-

ber of proxies for the PROXY protocol), and develop better mobility protocols that can more effectively deal with the processing delays introduced by using very expressive subscription languages. In addition, we hope to study the impact when both publishers and subscribers are mobile. We feel there is much interesting research to be performed in this area.

## 8. 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 *Symposium on Principles of Distributed Computing*, pages 53–61, 1999.
- [2] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *International Conference on Very Large Databases (VLDB)*, 2000.
- [3] G. Banavar, T. D. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *International Conference on Distributed Computing Systems (ICDCS)*, 1999.
- [4] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, P. H. A. Helmy, S. Mc-Canne, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. *IEEE Computer*, 33:59–67, May 2000.
- [5] I. Burcea, H.-A. Jacobsen, E. de Lara, V. Muthusamy, and M. Petrovic. Disconnected Operation in Publish/Subscribe Middleware. In *International Conference on Mobile Data Management (MDM)*, 2004.
- [6] 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.
- [7] A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *Proceedings of ACM SIGCOMM*, pages 163–174, Karlsruhe, Germany, Aug. 2003.
- [8] G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9), 2001.
- [9] Y. Diao and M. Franklin. Query processing for high-volume xml message brokering. In *International Conference on Very Large Databases (VLDB)*, Berlin, Germany, September 2003.
- [10] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, 2003.
- [11] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *Proceedings of ACM SIGMOD*, 2001.
- [12] H. K. Y. Leung. Subject space: A state-persistent model for publish/subscribe systems. In *Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research*, page 7. IBM Press, 2002.
- [13] G. Li, S. Hou, and H.-A. Jacobsen. A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. In *International Conference on Distributed Computing Systems (ICDCS)*, Columbus, Ohio, 2005.
- [14] H. Liu and H.-A. Jacobsen. Modeling Uncertainties in Publish/Subscribe System. In *International Conference on Data Engineering (ICDE)*, 2004.
- [15] G. Muhl, L. Fiege, F. C. Gartner, and A. Buchmann. Evaluating advanced routing algorithms for content-based publish/subscribe systems. In *MASCOTS*, page 167. IEEE Computer Society, 2002.
- [16] V. Muthusamy and H.-A. Jacobsen. Small-scale peer-to-peer publish/subscribe. In *P2P Knowledge Management Workshop at MobiQuitous*, July 2005.
- [17] V. Muthusamy, M. Petrovic, D. Gao, and H.-A. Jacobsen. Publisher mobility in distributed publish/subscribe systems. In *Distributed Event-based Systems Workshop (DEBS) at ICDCS*, June 2005.
- [18] C. E. Perkins and D. B. Johnson. Mobility support in IPv6. In *ACM Conference of Mobile Computing and Networking (MOBICOM)*, 1996.
- [19] M. Petrovic, I. Burcea, and H.-A. Jacobsen. S-ToPSS – a semantic publish/subscribe system. In *International Conference on Very Large Databases (VLDB)*, Berlin, Germany, September 2003.
- [20] M. Petrovic, H. Liu, and H.-A. Jacobsen. G-ToPSS – fast filtering of graph-based metadata. In *International World Wide Web Conference (WWW)*, Chiba, Japan, May 2005.
- [21] M. Petrovic, V. Muthusam, and H.-A. Jacobsen. Content-based routing in mobile ad hoc networks. In *MobiQuitous*, July 2005.
- [22] G. P. Picco, G. Cugola, and A. L. Murphy. Efficient content-based event dispatching in presence of topological reconfiguration. In *International Conference on Distributed Computing Systems (ICDCS)*, 2003.
- [23] P. Sutton, R. Arkins, and B. Segall. Supporting disconnectedness - transparent information delivery for mobile and invisible computing. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, 2001.
- [24] Talarian Inc. Publish-subscribe middleware helps direct traffic of Olympic proportions. [http://messageq.ebizq.net/communications\\_middleware/talarian\\_2.html](http://messageq.ebizq.net/communications_middleware/talarian_2.html).