Disconnected Operation in Publish/Subscribe Middleware

Ioana Burcea[†], Hans-Arno Jacobsen[†‡], Eyal de Lara[‡], Vinod Muthusamy[†], Milenko Petrovic[†]
[†]Department of Electrical and Computer Engineering
[‡]Department of Computer Science
University of Toronto
{ioana,jacobsen,vinod,petrovi}@eecg.toronto.edu
delara@cs.toronto.edu

Abstract

The decoupling of producers and consumers in time and space in the publish/subscribe paradigm lends itself well to the support of mobile users who roam about the environment and have intermittent network connectivity. This paper identifies the factors that affect the performance of a distributed publish/subscribe architecture supporting mobility; formalizes mobility algorithms for distributed publish/subscribe systems and develops and evaluates optimizations that reduce the costs associated with supporting mobility in publish/subscribe systems. In our analysis, we focus on the "unicast" traffic generated to support mobile users, as opposed to the regular "multicast" traffic used for event dissemination to stationary clients. We find that the network capacity must be doubled to handle the extra load of just 10% of mobile users.

1. Introduction

Computing devices are becoming more pervasive and our dependence on the information delivered through these devices is increasing. Due to these trends, users expect access to information on multiple devices at various geographic locations at any time. For example, a user might receive stock quotes, press releases, audio and video footage, and other kinds of alerts delivered to her laptop at the office, her PDA while shopping, or her desktop at home. In some applications, users expect that data disseminated while they are disconnected, for example when commuting between locations, is reliably stored and delivered upon reconnection. Users may disconnect when network connectivity is absent or to conserve battery life. Therefore, support for disconnected operation is essential for information dissemination applications that supports mobile users.

Information dissemination applications have been effectively modeled with publish/subscribe style interactions [10, 19], where publishers act as information providers, subscribers act as information consumers, and a broker mechanism routes relevant publications (events) to interested subscribers.

Publish/subscribe systems have a number of desirable characteristics for mobile information dissemination applications. First, they can efficiently filter and disseminate large amounts of data to large number of users. Second, 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 publish/subscribe paradigm naturally supports disconnected operation.

In this paper, we show that existing proposals [7, 11] for supporting disconnected operation, which are based on state-transfer protocols implemented using unicast messages, may result in drastic increases in the load on a network where events are disseminated in a multicast fashion to stationary users. We show that even a small percentage of mobile users in the system can significantly increase the traffic in the network. In our evaluations we see that mobility can increase the average network traffic by up to 100%.

The three most important contributions we make in this paper are as follows:

1. We identify and classify the factors that affect the performance of a publish/subscribe system that supports client mobility.
2. We formalize mobility algorithms for distributed publish/subscribe systems. We use an analytical model to reason about the network cost of supporting mobility.
3. We develop and experimentally evaluate optimizations that reduce the costs associated with disconnected operation. While we prototype these algorithms on top of the Toronto Publish/Subscribe System (ToPSS) system [15, 17], all the optimizations we propose are applicable to any type of publish/subscribe system.

The rest of the paper is organized as follows. Section 2 provides a brief overview of the publish/subscribe paradigm. Section 3 describes the algorithm most commonly used to support disconnected operation in publish/subscribe systems, and develops a number of optimizations to this algorithm. Section 4 develops an analytical model for reasoning about the network cost of supporting mobile clients. Section 5 presents an experimental evaluation of the mobility algorithm and the proposed optimizations. 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

The publish/subscribe paradigm is effective at supporting information dissemination applications [1, 2, 4, 11, 12, 13]. Clients in a publish/subscribe systems are autonomous components that exchange information by publishing events and by subscribing to events of interest. Clients that produce information are referred to as *publishers*, while clients that consume information are referred to as *subscribers*[1]. Publishers generate messages (publications or events) to inform the external world that a certain condition has occurred. Subscribers, which express their interest in events by means of subscriptions, are then notified of the occurrence of these events. The central component of a publish/subscribe system is the *event broker*, which records all subscriptions in the system. When a certain event is published, the event broker matches it against all subscriptions. On a match, the event broker notifies the corresponding subscriber.

Publish/subscribe systems can be based on the notion of *topics* (or *subjects*), *types*, or *content*. In topic-based publish/subscribe, clients can subscribe to several topics and receive notifications about all events 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 event types instead of topics for matching. *Content-based* publish/subscribe systems improve the expressiveness of subscriptions by allowing subscriptions to contain complex queries on the event content.

The broker represents the most important component in the system. It has to perform the matching between the incoming events and subscriptions in the system and it also has to send the subscribers all the events for which they have expressed interest. It is important to note that messages from the publishers (events) do not contain any address; instead, they are routed through the system based on their content (for a content-based system). The broker architecture can be centralized or distributed.

In a distributed broker architecture, one of the most important problems is the routing of publications to interested subscribers based on the content of the publication. There are several routing algorithms proposed in the literature [9, 11, 4]. The key point of all these algorithms is that, based on the subscriptions available in the system, the algorithms build a multicast tree such that when an event enters the system, it is sent along the multicast tree to all interested subscribers, rather than sending it individually to all interested subscribers.

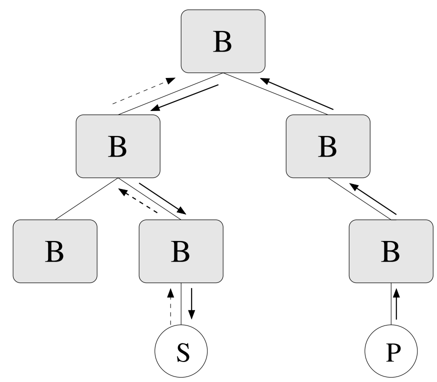

Figure 1. Routing in a hierarchical broker architecture

In a hierarchical distributed broker architecture, the routing problem can be addressed as follows. Subscriptions always propagate upward to the root broker. Each broker on the path from the subscriber to the root stores the subscriptions together with the node identifier of the interested children (see Figure 1, where dashed arrows represent subscriptions, and solid arrows represent events). An event enters the system when the publisher publishes the event to the broker it is connected to. Upon receiving an event, a broker forwards it to its parent and, based on matches in its subscription table, also sends the event to any interested children. Therefore, events are always propagated upward to the root, but downward only towards interested subscribers.

Subscription covering [9] can optimize the routing algorithm presented above. Given two subscriptions s_1 and s_2, s_1 covers s_2 if and only if all the events that match s_2 also match s_1. In other words, if we denote with E_1 and E_2 the set of events that match subscription s_1 and s_2, respectively, then $E_2 \subseteq E_1$. When a broker B receives a subscription s, it will send it to its parent if and only if it has not previously sent another subscription s', that covers s, to its par-

1 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

ent. Broker B will receive all events that match s, since it receives all events 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.

One problem with the scheme presented above is that the root broker of the hierarchical topology can become a bottleneck for the system: all events and subscriptions reach the root broker. Advertisements [9] address this by requiring publishers to announce the set of publications they are going to publish. Advertisements look like subscriptions, but play a different role in the system. For example, if a publisher produces only events about products with brand "IBM" and prices greater than 500, it can issue the following advertisement: *(brand = "IBM", price $\geq$ 500)*. In other words, each attribute-value pair from the publication is "covered" by a predicate in the advertisement. We say that a subscription intersects with an advertisement if there is at least one event that matches both the subscription and the advertisement. In general, using advertisements in a hierarchical architecture changes the routing scheme as follows. The advertisements are sent up to the root, and subscriptions go up to the root and down to the publishers that issue advertisements that intersect the subscriptions. When an event is published, it is forwarded only to those brokers that previously sent it subscriptions that match the event. Thus, events are never unnecessarily propagated up the tree. Usually, the number of events in the system is much larger than that of subscriptions and advertisements, thus one expects advertisements to reduce the overall network traffic. Advertisements are most effective when subscribers interested in events from a publisher are localized in the network.

3. Disconnected Operations

To the best of our knowledge, Cugola et al. [11] were the first to support mobility in publish/subscribe systems. They introduce the "movein" and "moveout" operations that offer subscribers the ability to disconnect from and reconnect to the system. In this section, we first describe the mobility algorithm proposed by Cugola et al. [11], which we refer to as the `standard` algorithm. We then present a set of optimizations to this algorithm. Since we only study the mobility of subscribers in this paper, we use the term client to only mean subscribers from now on.

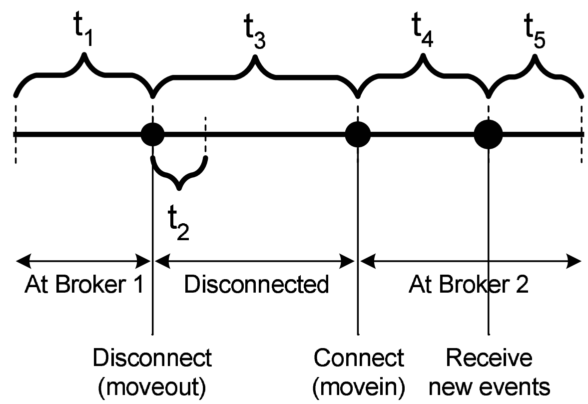

Figure 2. Disconnected operation timeline

3.1. Standard Algorithm

Figure 2 illustrates a timeline of a client going into a period of disconnection and reconnecting to a different broker. During period t_1, the client is connected to Broker 1 and can receive events. At the end of period t_1, the client disconnects from Broker 1 and reconnects after period t_3 to Broker 2. Period t_2 is used by an optimization described in Section 3.2. Period t_4 is required to complete the reconnection phase, which involves retrieving and playing back events that the client missed while disconnected. Finally, during period t_5 the client receives newly published events as in period t_1.

The objective of the algorithm is to reconfigure the event multicasting tree to account for client mobility. We assume the client always reconnects to its physically closest broker. When a client disconnects from Broker 1, the broker begins to locally store the events that the client would have received if it had been connected. If the client reconnects to Broker 1, then the stored events are simply replayed to the client during period t_4. The more interesting case is when the client reconnects to some other broker, say Broker 2. The following steps take place during period t_4:

1. Upon reconnection, the client notifies Broker 2 that it was previously connected to Broker 1.

2. Broker 2 retrieves the subscriptions associated with the client from Broker 1.

3. Broker 2 subscribes to these subscriptions, and then sends a $REQUNSUB$ message to Broker 1 requesting it to unsubscribe.

4. Broker 2 stores in a local queue new events it receives for the client.

5. Broker 1 forwards stored events to Broker 2.

6. All the state has now been transferred from Broker 1 to Broker 2. Broker 2 now replays both the events received from Broker 1, and the new events stored in its local queue, to the client.

In Step 6, duplicates in the set of events transfered from Broker 1, and those in Broker 2's local queue are removed. This assumes duplicate events can be distinguished, with, for example, a publisher-specific event sequence numbers.

In Step 3, to guarantee that no events are lost, we require that the common ancestor of Brokers 1 and 2 in the broker hierarchy sees the subscriptions from Broker 2 before the unsubscriptions from Broker 1. To simplify this step, we assume that the overlay broker hierarchy is consistent with the underlying network hierarchy such that the shortest path between any two brokers in the overlay topology is the shortest path in the underlying topology. We also assume that messages between brokers are received in the order they were sent, which is reasonable as we expect TCP connections between brokers. Since Broker 2 sends the subscriptions followed by $REQUNSUB$, the common ancestor sees them in this order, and since the common ancestor is in the shortest path between Brokers 1 and 2 in the overlay topology, Broker 1 sees $REQUNSUB$ after the common ancestor, by our assumption. Thus, our requirement is met, and no events are lost. More elaborate schemes [7] are possible in cases where our assumption is not valid. However, more complex protocols likely increase the state transfer costs, and thus only worsen the overhead costs we quantify in Section 5.

All the messages exchanged between the two brokers represent unicast messages. As we will see in the experimental section, the unicast messages contribute significantly to system overhead. Also, it is possible that cost of reconfiguring the multicast tree may be greater than the gains from having events take the shortest path.

3.2. Optimizations

3.2.1. Prefetching The `prefetching` algorithm exploits knowledge of future mobility patterns. It is similar to the `standard` algorithm except that Steps 2-5 (the transfer of subscriptions and stored events) occur during period t_2. In period t_4 only Step 6 (the replay of stored events) takes place. `Prefetching` gains in two ways. First, the latency of the state transfer from Broker 1 to Broker 2 is hidden from the user since is occurs while the user is disconnected. This results in a shorter t_4 period. Secondly, since state transfer occurs early, there are very few stored events that Broker 1 needs to forward. Therefore there are fewer messages transferred overall.

The effectiveness of `prefetching` depends on the successful prediction of the client's destination. To improve the likelihood of success, Broker 1 can predict the *set* of brokers that are likely to be the next destination of the mobile client. Broker 1 can make this prediction based, for example, on statistics of the mobility patterns of its clients. If the client reconnects to one of the predicted brokers, that particular broker replays all stored events, and informs the other brokers to discard the stored events and subscriptions for that client. If the client reconnects to a completely different broker, the new broker can contact the closest broker from the set of prefetched brokers, and the state transfer happens as in the `standard` algorithm. We do not evaluate prefetching to multiple brokers in this paper.

3.2.2. Logging The `logging` algorithm exploits subscription locality (a measure of the similarity of subscriptions) in the system. Here, all brokers maintain a log of recently received events. When the mobile client reconnects, Broker 2 scans its log for events of interest to the mobile client. Any relevant events found in the log do not need to be transferred from Broker 1.

The state transfer is similar to the `standard` algorithm. After Step 2, however, Broker 1 sends a message with the IDs of its stored events to Broker 2. Broker 2 checks for these events in its log and sends any matched IDs to Broker 1 instructing it to not send these events during Step 5. By only transferring, in Step 5, those events not already at Broker 2, the period t_4 can be shorter than in the `standard` algorithm. However, if no such events exist, the wasted overhead of sending event IDs can make period t_4 longer. Thus, in terms of both message and latency costs, `logging` can perform better or worse than `standard` depending on the movement scenario and subscription locality.

`Logging` requires that events have system-wide unique IDs. This can be easily achieved if we consider that each publisher in the system has a unique ID and that events are ordered within the publisher. Thus, the ID for the event is composed from the ID of the publisher that sent it and its local ID given by the order number.

3.2.3. Home-Broker In this algorithm each client is assigned a home broker. On a `movein`, a client reconnects to its physically closest broker. The client, however, reconnects logically to its home broker. Subscriptions remain on the home broker, which keeps receiving events through the regular multicast mechanism. The home broker then forwards these events to the client using unicast messages.

`Home-broker` gains by not migrating subscriptions and rebuilding the multicast tree, at the expense of not sending events through the shortest path. Note that there is unicast traffic even when the client is connected.

3.2.4. Subscriptions-on-device Subscription migration is an important component of state transfer. If the mobile device has enough resources, it locally stores the subscriptions, it can directly send the subscriptions to the new broker upon reconnection. This way, the old broker does not need to transfer the subscriptions to the new broker. The applicability of this method depends on the number of subscriptions that a client has, the resources of the mobile device, and whether the user wishes to use than one device.

3.2.5. Discussion It should be noted that these optimizations make minimal assumptions about the underlying system. They can be used with any type of distributed publish/subscribe system. Moreover, the optimizations can be combined. For example, subscriptions-on-the-device can be combined with `prefetching` or `logging`.

4. Analysis of Optimizations

In this section we develop an analytical model to reason about the number of unicast messages required to support a mobile client under the `standard` mobility algorithm introduced in Section 3.1. We then use this model to discuss the effects that the optimizations proposed in Section 3.2 have on the message overhead. Our analysis is independent of the underlying network topology that connects brokers and the matching algorithm used by brokers.

To a first approximation, the total number of messages required to support a reconnecting mobile client is equal to the sum of two quantities: the cost to migrate subscriptions and the cost to forward queued events. Equation 1 reflects this relationship.

$$\begin{aligned} total\_msg\_cost \quad = \quad & subs\_msg\_cost + \\ & event\_msg\_cost \end{aligned} \tag{1}$$

Equation 2 models the message cost to migrate subscriptions, which is a linear function of the number of subscriptions the mobile client has. For each subscription (s), three messages are transmitted: one message describing the subscription is sent from the old to the new broker, one subscription message is sent by the new broker to the network of brokers, and one unsubscription message is sent by the old broker to the network of brokers. (Each of these messages can actually take multiple overlay hops.)

Some implementations may require an additional message to ensure that the subscription migration is atomic and that all events are reliably delivered to the client. The extra message, sent from the new to the old broker, guarantees that the old broker does not stop logging (and forwarding) events before the new broker has a chance to register for event delivery. On the other hand, for some systems it may be possible to save one message per subscription by storing subscription information in the mobile client itself. Upon reconnection, subscriptions are sent to the new broker from the subscriber instead of the old broker.

Covering optimizations (as described in Section 2) may reduce the cost of migrating subscriptions by obviating the need for brokers to send messages to subscribe and unsubscribe to events. Equation 2 incorporates the effects of covering optimizations by calling on the binary function *coverage*, which takes as arguments a subscription (s) and a broker (old, new), and returns 0 if the subscription is covered by some other subscription at the broker (eliminating the need to send a message), and 1 otherwise. There can be multiple implementations of the *coverage* function; ranging from implementations that closely mimic specific covering algorithms, to those based on simple probabilistic models.

$$\begin{aligned} subs\_msg\_cost \quad = \quad & \textstyle\sum_{s=1}^{n} (1 + 1 * coverage(s, old) \\ & 1 * coverage(s, new)) \end{aligned} \tag{2}$$

The message cost to forward stored events is a function of the length of the disconnection period and the sum of the publication rates of the events to which the mobile client subscribes. Equation 3 captures this relationship. The disconnection time for the `standard` protocol is shown in Equation 4. The terms for Equation 4 are derived from the diagram shown in Figure 2, where t_3 represents the interval between the time the mobile client disconnects from the old broker and the time it reconnects to the new broker, and t_4 is the time it takes to migrate subscriptions and forward queued events from the old to the new broker.

$$event\_msg\_cost = \sum_{e=1}^{n} rate(e) * disconnection\_time \tag{3}$$

$$disconnection\_time = t_3 + t_4 \tag{4}$$

4.1. Reducing Message Cost

We now consider the effects of the optimizations proposed in Section 3.2 on the above equations. Two of the optimizations (`prefetching` and `home-broker`) represent opposite extremes in the eagerness with which subscriptions migrate between brokers, while the third optimization (`logging`) reduces the message count by adding a negative term to Equation 3.

`Prefetching` and `home-broker` change the publish/subscribe system's perception of the length of time the client remains disconnected (the $disconnection\_time$ term in Equation 3). The system *perceives* a mobile client as reconnected once subscriptions have been migrated to the new location, and events can be delivered to the client using the standard event delivery mechanisms (i.e., multicasting). Conversely, the publish/subscribe system *perceives* a mobile client as disconnected as long as events are being stored or forwarded for this client by the broker holding the client's subscriptions. Therefore, the *perceived* length of the disconnection period can differ (depending on the mobility algorithm) from the actual time span the mobile client spends physically disconnected from the network.

`Prefetching` reduces the number of stored events that need to be propagated by eagerly migrating subscriptions to a predicted destination broker. By migrating subscriptions early, `prefetching` makes the disconnection interval perceived by the publish/subscribe system equal to just t_2 in Equation 4. In reality, the mobile client remains disconnected until the end of t_3, but the publish/subscribe system becomes aware of the mobile client migration, and starts delivering events (by means of multicasting) to the new broker, by the end of period t_2.

While successful prefetching can significantly reduce the message cost to support a mobile client, a failure in predicting the mobile client's destination increases the overall number of messages sent by the system. A failed prefetch incurs (in vain) the cost of transferring all subscriptions, as well as all events logged in period t_2. Moreover, subscriptions and events logged during t_3 and t_4 have to be transferred from the miss-predicted broker to the broker to which the mobile client actually reconnects. To increase the probability of predicting the correct destination, it is possible to migrate the subscriptions to several target brokers. While the cost of migrating the subscriptions grows linearly with the number of predicted broker destinations, this cost can still be significantly smaller than the costs related to miss prediction. Moreover the extra cost to the publish/subscribe network of replicating subscriptions in different brokers is limited as it would likely result in small (if any) increases in the multicast traffic used to propagate events.

`Logging` reduces the number of stored events that need to be forwarded between brokers by exploiting subscription locality in the system. (Locality refers to the correlation of subscriptions at different parts of the network.) By logging recently seen events, it is possible to obviate the need to forward events that the mobile client shares with other clients serviced by the new broker. We account for the reduction in messages resulting from logging by adding a negative term to the event message cost (see Equation 5). The number of logged events is hard to model analytically as it depends on several factors including the size of the event log, the event arrival rate, and the event locality.

$$event\_msg\_cost = \begin{array}{l} \sum_{e=1}^{n} rate(e) \\ *disconnection\_time \\ -logged\_events \end{array} \tag{5}$$

The `home-broker` approach eliminates subscription migration between brokers, so $subs\_msg\_cost$ becomes zero. `Home-broker` makes $disconnection\_time$ equal to t_3. However, `home-broker` increases the perceived disconnection time by extending this period to include all time the mobile client is not directly connected to its home broker. Note that as long as the client is connected through a broker other than its home broker, events must be forwarded from the home broker to the client. `Home-broker` can be beneficial when the event rate is very low and the rate at which the client roams between brokers is high enough that messages needed for rebuilding the multicast tree outnumber those messages needed to forward events.

4.2. Limitations

While the model described here helps in understanding the costs and benefits of the various mobility algorithms, it is not sufficient to predict results. For example, the terms $coverage$ and $logged\_events$ in Equations 2 and 5, respectively, are difficult to derive analytically. In addition, the model only quantifies the number of messages that must be sent, not the number of hops these messages travel. The latter is a more accurate measure of network load and depends on the mobility patterns of clients. Due to these limitations of the analytical model, we carry out several simulations in the next section and analyze these results.

5. Evaluation

In this section we first discuss the factors that affect the evaluation of a distributed publish/subscribe system that supports mobility. We then present an experimental evaluation of the algorithms introduced in Section 3.

5.1. Parameters

There are several factors that must be considered when designing or evaluating a distributed publish/subscribe system supporting mobility. We classify these factors into three categories: network, mobility, and application. Note that several of these factors are also issues in publish/subscribe systems that do not support disconnected operation.

Network characteristics are those concerned with the network infrastructure:

- **Bandwidth and latency** of the links in the network. This includes the links between brokers and those between brokers and clients. Faster links can perform state transfer 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 taller the topology, the longer events potentially take to get up to the root and down to the leaves. In contrast, a wider topology increases the load on each broker since it serves a greater number of neighbor (descendant) brokers.
- **Number of brokers** in the system. 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 state transfer 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 disconnections result in higher unicast state transfer cost.
- **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` optimization. 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:

- **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. An increase in the number of subscribers, would increase aggregate mobility, that is, the total number of state transfers in the system, and can lead to an increase or decrease in subscriber and interest locality.
- **Publishing rate**. A higher publication rate increases the regular publish/subscribe multicast traffic, as well as the number of events that must be stored during a disconnection period and then unicast during the state transfer protocol upon reconnection.
- **Specificity of subscriptions**. A very general subscription will increase the number of events that must be delivered to the subscribing client.
- **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. Our `logging` approach, for example, benefits when clients at the same broker have similar subscriptions.
- **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 during the state transfer protocol.

5.2. Experiments

We developed three mobility scenarios: commute, pervasive, and random. The commute scenario depicts users commuting from work to their homes. It is characterized by medium to large disconnection periods. The pervasive scenario depicts users who experience short disconnection periods while moving. The random scenario reflects average subscriber behavior as all parameters are selected uniformly from their respective value ranges. Below we describe our experimental setup and metrics, and then discuss the experiments in detail.

5.2.1. Methodology We performed all our experiments using the ns2 network simulator [5]. We extended ns2 with the `standard` mobility algorithm described in Section 3.1 and the three optimizations presented in Section 3.2.

Designing realistic experiments to evaluate publish/subscribe systems is hard. 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. Despite these obstacles, we have tried to design scenarios we feel are representative of applications for which these systems would be deployed. We have attempted to choose sensible parameter values in each scenario.

We simulate a network with 64 brokers distributed across a city. These brokers are the leaves of a tree of height 4 and degree 4. Conceptually, the first few leaf brokers are located

Parameter	Value
num_brokers	85
num_leaf_brokers	64
broker_height	4
broker_degree	4
broker_log_size	1000 events
num_publishers (p_i)	100
event(p_i)	rand(1,27)
rate(p_i)	1 every minute
num_subscribers (s_i)	200, 400, 600, 800

Table 2. Common experimental parameters

Parameter	Value
num_publishers	100
broker(p_i)	rand(10 downtown brokers)
subscription(s_i)	rand(1, 40)
startbroker(s_i)	rand(20 downtown brokers)
moveout(s_i)	rand(4pm, 6pm)
endbroker(s_i)	for 40% of s_i: rand(30 downtown brokers) for 60% of s_i: rand(remaining brokers)
movein(s_i)	for 40% of s_i: moveout(s_i) + rand(15min, 45min) for 60% of s_i: moveout(s_i) + rand(45min, 90min)

Table 3. Commute scenario parameters

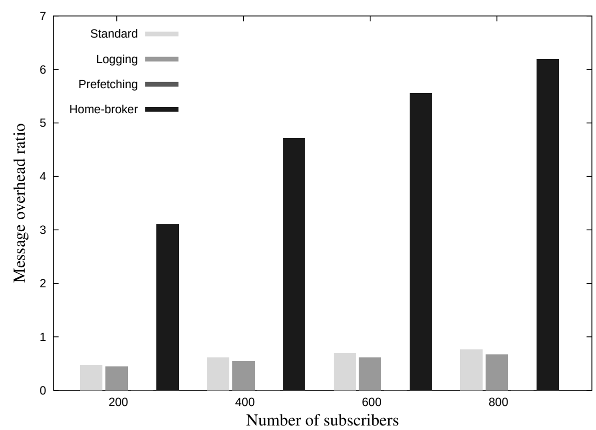

Figure 3. Avg. overhead (Commute scenario)

at the west end of the city, the next few in 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. Each publisher is randomly assigned one of 27 unique events, which it publishes once every minute. Likewise, each subscriber randomly subscribes to one of 40 subscriptions at the beginning of the experiment. The first 27 of these subscriptions exactly match one of the 27 publications. Of the remaining subscriptions, 9 match 3 publications each, 3 match 9 publications each, and one matches all publications. These last 13 subscriptions can be used to evaluate the covering optimization. The relatively small number of distinct subscriptions is sufficient for these experiments. We are not concerned with local matching time so a large number of complex subscriptions is not necessary. Furthermore, these subscriptions are sufficient to vary the subscription locality in different parts of the network. These parameters are summarized in Table 2. The clients' mobility patterns vary with each scenario and are described in the respective sections.

We measure the cost of supporting disconnected operation in terms of the message overhead introduced by state transfer. We consider regular messages to be those delivered using multicast, and the overhead is the unicast messages used to migrate subscriptions and forward events.

5.2.2. Commute Scenario In this scenario we simulate the evening commute home. Users are connected to the company network while at work, disconnect for their commute home, and reconnect at home. Publishers are randomly placed in one of the 10 inner (downtown) brokers. Each subscriber is assigned one of the 40 subscriptions and subscribes to this at the beginning of the experiment. Subscribers then disconnect and start their commute sometime between 4:00pm and 6:00pm, and reconnect at home after some time. Some subscribers live in the city center and arrive home sooner than those that live in the outskirts. The parameters are summarized in Table 3.

Figure 3 shows how `standard`, `prefetching`, `logging`, and `home-broker` compare in terms of the state transfer message overhead. The `prefetching` results assume perfect prediction. For `logging` we use a log size of 1,000 events. The overhead is expressed as the ratio of state transfer (unicast) vs. regular (multicast) messages over the length of a simulation run.

`Home-broker`'s overhead is by far the highest, and increases with the subscriber population. We explain this poor performance in the analysis for Figure 5. Among the remaining approaches, `standard` performs the worst, and also increases with subscriber population. An important result here is the almost 80% overhead of `standard` with 800 clients. This means that introducing mobility to a network built with enough capacity to service a non-mobile publish/subscribe system will require almost doubling the network capacity. `Prefetching`, by transferring state early, has almost no overhead in this scenario, because the periods of disconnection are large and the traffic required to migrate subscriptions is much smaller than the event traffic during the disconnection period. `Logging` improves on

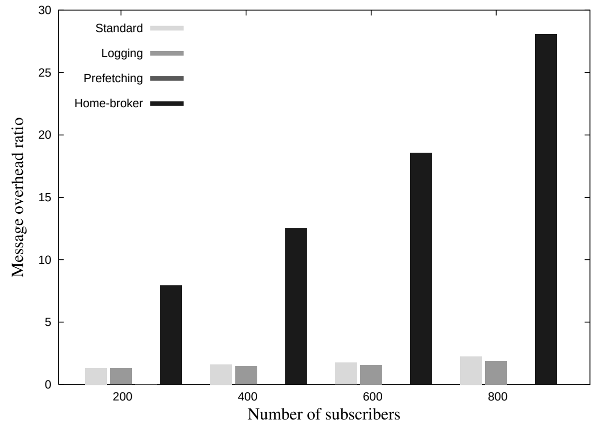

Figure 4. Peak overhead (Commute scenario)

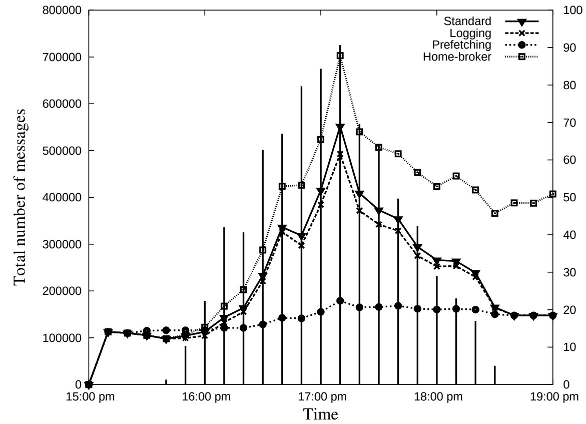

Figure 5. Total cost (Commute scenario)

`standard` by partially replacing unicast state transfer with multicast by leveraging logging and subscription locality.

In Figure 4 we show the peak overhead instead of the average overhead as in Figure 3. In the commute scenario the peak overhead occurs between 5:00pm and 5:30pm, which is when the number of concurrent state transfers is highest. As expected, the relative performance of each approach is comparable to the average case. But it is interesting that during the peak time, the message overhead almost triples as compared to the average case, and more than quadruples in the case of `home-broker`. Looking at absolute values, `standard` incurs a 230% overhead for the population of 800 clients. This means that the situation is worse than we indicated earlier; introducing mobility in this scenario requires network administrators to more than triple the network capacity to handle peak usage periods. Also, handling mobility at the network layer (which `home-broker` tries to emulate), leads to more than 25 times the peak load with 800 clients—clearly an infeasible solution.

In this scenario, for the `standard` and `logging` approaches, we found that the portion of the unicast state transfer protocol devoted to subscription migration is less than 1%, so most of the overhead is due to the transfer of stored events. The `prefetching` overhead is almost fully due to subscription migration and some constant overhead of the state transfer protocol. Note that there are no mobility induced subscriptions in the `home-broker` approach.

The commute scenario has up to 10% of subscribers concurrently performing state transfer. Thus we see that serving even only a fraction of disconnected users can increase the network cost considerably. Consequently, supporting disconnected users will require provisioning much higher network bandwidth compared to what is required for supporting only connected users.

In Figure 5 we plot the message cost over time with 800 subscribers. We see that the message cost of `standard` and `logging` closely tracks the number of concurrent clients executing state transfers. This figure helps explain the dismal `home-broker` performance. After all clients have reconnected, the other three approaches have identical costs, since they rebuild the multicast tree and events are multicast to clients. With `home-broker`, however, events are still unicast from the home-broker to the client. This large and sustained unicast, which persists even after reconnection is why `home-broker` performs poorly.

5.2.3. Random Scenario In this scenario, each subscriber is randomly assigned to one of the 27 subscriptions each of which exactly matches one of the 27 publications. Each subscriber alternates between periods of connection and disconnection, with each period varying uniformly between 10 and 30 minutes. While disconnected, the subscriber moves at a randomly chosen speed of 5km/h (walking or cycling), 50km/h (city driving), or 100km/h (highway driving). The disconnection period and speed together with a randomly chosen direction of movement (east or west) determine the broker that the subscriber reconnects to.

In Figure 6 we see similar trends and relative performance of the four approaches. `Standard` with 800 clients has virtually 100% overhead. The small `prefetching` overhead is due to the protocol messages for switching from the old to the new location and transfer of any events that arrive while switching over. Since the switch-over usually happens immediately after disconnection, the number of events that need to be exchanged is usually very small. The other three algorithms have considerably more overhead. We observe that the `logging` overhead increases marginally with larger population. Larger populations have more users interested in the same content and `logging` can take advantage of that, whereas the other approaches cannot. With 800 clients, the `logging` overhead is less

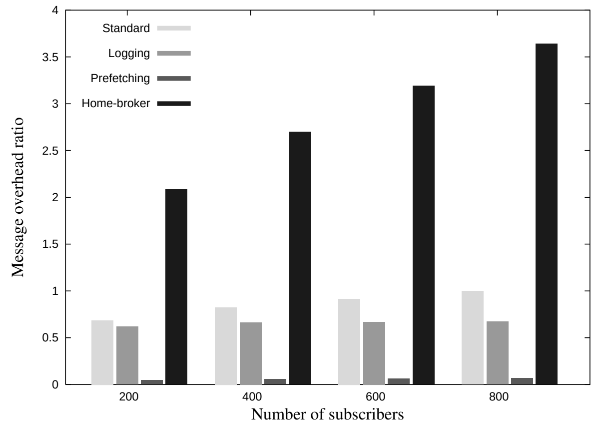

Figure 6. Avg. overhead (Random scenario)

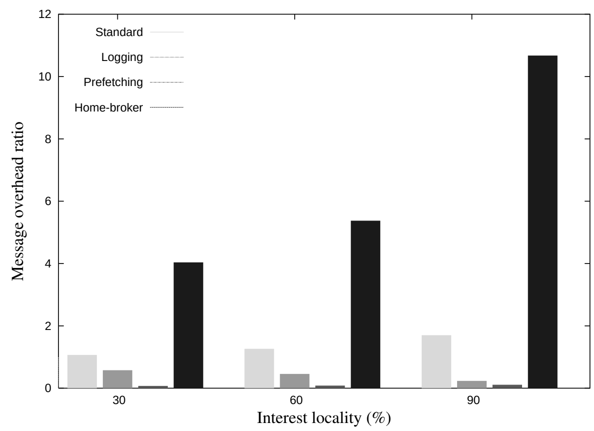

Figure 7. Effect of locality (Random scenario)

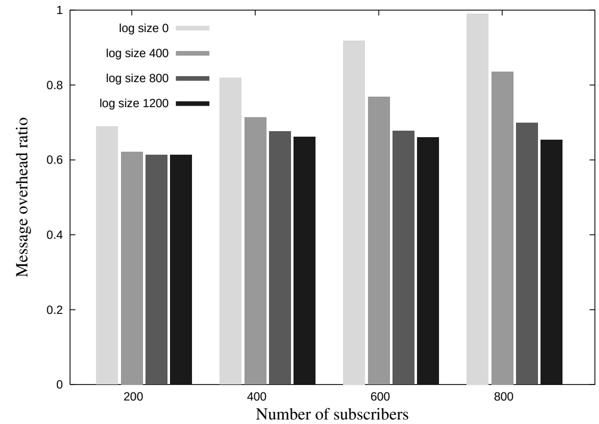

Figure 8. Effect of log size (Random scenario)

than 70% that of `standard`.

We try to improve on this by varying the interest locality for a population of 800 users. To achieve $x\%$ locality, $x\%$ of the subscribers have identical interest (subscription) and the remaining have a disjoint and random interest. With increasing locality it becomes increasingly advantageous to log events since more users can benefit. We note in Figure 7 that unlike the other approaches, the message cost for `logging` decreases with increasing interest locality. We also see that that with appropriate subscription locality, `logging` matches the performance of `prefetching`.

The log size determines the effectiveness of `logging`. In Figure 8 we show the overhead of `logging` as we vary the log size. We observe that when the log size is increased from 800 to 1200 events, the `logging` overhead decreases only marginally regardless of the population size, so the `logging` performance in previous figures (which used 1,000 events) can not be improved much.

Similar to the commute scenario, here subscription migration represents about 2% of the unicast state transfer protocol traffic in the `standard` and `logging` approaches. Again, the overhead of `home-broker` is completely due to stored and forwarded events, and stored events have a negligible impact on the `prefetching` overhead.

5.2.4. Pervasive Scenario At the other end of the spectrum of disconnected operation, the pervasive scenario reflects an environment where devices are able to maintain connectivity as they roam. Here, disconnection periods are very short but more frequent.

The mobility patterns in this scenario are almost identical to the random scenario. In the random scenario, a subscriber disconnects from the source broker and reconnects to the destination broker when traveling between these brokers. In this scenario, the subscriber remains connected while traveling, by connecting in turn to the intermediate brokers between the source and destination brokers. This is akin to the handoffs between cells in a cellular network.

In Figure 9 we compare the four approaches in the pervasive scenario. We see that `home-broker` approach still has the highest overhead, but the other approaches all perform equally well. Short disconnection periods result in little state at any broker, and thus `standard` and `logging` perform comparably well to `prefetching`. Only subscriptions and unsubscriptions contribute to the overhead in these three approaches. The overhead of `home-broker` is completely due to stored and forwarded events.

In this scenario, we found that subscription migration accounts for about 60% of the unicast state transfer protocol traffic in the `standard` and `logging` approaches. This percentage is even higher for `prefetching`. This implies that a subscription-on-device approach would be beneficial

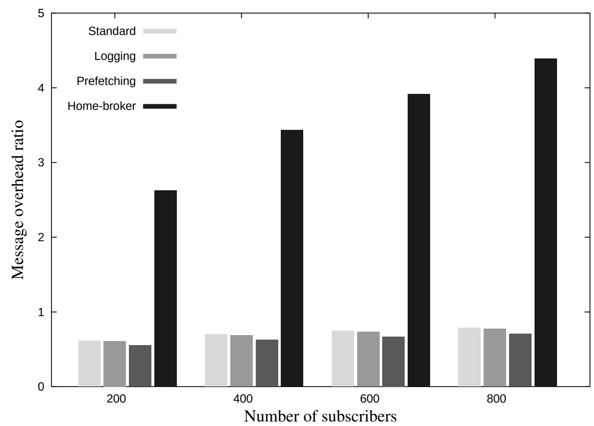

Figure 9. Avg. overhead (Pervasive scenario)

in this scenario. However, one can argue that such an approach diminishes usability. Notably, it discourages the user from using more than one device; storing subscriptions at the brokers relieves the user from having to keep subscriptions consistent across all devices.

5.2.5. Advertisements and Coverings Advertisements and covering are optimizations used to reduce subscription traffic in distributed publish/subscribe systems (see Section 2). Because mobility results in additional subscriptions and unsubscriptions as the users connect to and disconnect from the network, we expected advertisements and covering to substantially reducing message overhead. We found this not to be the case. This is due to the lack of locality in our scenarios, and the fact that mobility costs are dominated by the forwarding of stored events in the commute and random scenarios. For example, in the random scenario with 800 clients (see Figure 6), subscriptions and unsubscriptions represent only 2% of traffic of forwarding stored events. While covering and advertisements do reduce some of the overhead, their overall effect is negligible.

6. Related Work

The work in this paper is part of the Toronto Publish/Subscribe System (ToPSS) project of the Middleware Systems Research Group (MSRG). In addition to the Mobile-ToPSS research presented in this paper, other related projects in the MSRG include research into local matching in publish/subscribe systems [3], Approximate-ToPSS [15], Semantic-ToPSS [17], Location-ToPSS [6], and P2P-ToPSS [20].

The publish/subscribe matching problem has been investigated extensively [13, 1, 12]. This work concentrated on devising efficient algorithms for matching in centralized publish/subscribe systems. Scalability concerns prompted researchers to look at distributed publish/subscribe [9, 11, 4]. 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, focus on scenarios applicable to non-mobile clients.

SIENA [7, 8], JEDI [11], and ELVIN [18] have added extensions to support disconnected operation. However, these approaches lack substantial quantitative analysis to evaluate the large scale effects of mobility. In contrast to the quantitative evaluation we present in this paper, evaluations of these systems focus either on the portability and flexibility of the system, or present experimental results for small networks of brokers (e.g., 3 brokers). Moreover, we proposed and evaluated a set of optimizations that reduce the overhead associated with supporting mobile clients.

Mobile IP [16] 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 approach is equivalent to the `home-broker` optimization presented in Section 3.2.3.

Multicasting support for mobile clients in a general context has been an active area of research [14]. Existing approaches usually assume that mobile users retain network connectivity while they move. Our work, on the other hand, assumes that users can be temporarily disconnected while they move between brokers. The longer disconnection periods result in more state information that needs to be transfered between brokers.

7. Conclusions and Future Work

The scale of unicast traffic arising from the mobility of subscribers in the publish/subscribe system has not been previously studied. We presented an evaluation of the network cost associated with supporting mobile clients in a distributed publish/subscribe system. Our experiments reveal that the unicast traffic used to support mobile clients, on average, can double the load on the network even when only up to 10% of clients are mobile. During the peak times, the load spikes are triple the non-mobility load. In the scenarios we evaluated, the mobility costs are dominated by the forwarding of stored events, while the message overhead incurred by subscriptions and unsubscriptions is small.

We described and evaluated several optimizations that reduce the cost of supporting mobile subscribers. `Prefetching` can virtually eliminate the overhead, `logging` reduced overhead to 70% of `standard` in the random scenario, and we saw that with sufficient locality, `logging` can approach the performance of the ideal `prefetching` optimization. We also showed that handling mobility at the network layer (as emulated by `home-broker`) is likely infeasible as it needs more than a 25 times capacity increase to handle peak loads.

There is much work to be done in this area. While researching this topic, we quickly became aware that there are countless parameters to consider. We were unable to find "realistic" values for these parameters. Consequently, one significant area for future work is to obtain real-world usage patterns and scenarios. This includes communication link speeds, mobility patterns (how fast, and where client move), the number of brokers and clients, the publication rate, the distribution of events and subscriptions, the locality of clients (how close publishers and subscribers are to each other), subscription locality, and publication locality. Acquiring such data, however, might be difficult until the distributed publish/subscribe paradigm is adopted by industry. Thus, in the short term, we plan to devise and evaluate scenarios that are inherently different from the commute scenario, including ones where subscribers' interests change over time and publishers are mobile.

In continuing research, we wish to examine not just message costs but also how increased unicast traffic affects message delivery latencies. Another area for future work is to develop a protocol to perform state transfer in a multicast fashion. An extension to the `logging` approach where intermediate brokers would cache the events transferred during state transfer, might be able to achieve this. We would also like to investigate the covering and advertisements optimizations further.

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 *Proceedings of the 26th VLDB Conference*, 2000.

[3] G. Ashayer, H. Leung, and H.-A. Jacobsen. Predicate matching and subscription matching in publish/subscribe systems. In *DEBS'02 Workshop at ICDCS'02 (DEBS'02)*, Vienna, Austria, 2002.

[4] 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*, 1999.

[5] 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.

[6] I. Burcea and H.-A. Jacobson. L-ToPSS - push-oriented location based services. In *Proceedings of the 2003 Workshop on Technologies for E-Services*, Lecture Notes in Computer Science. Springer, 2003.

[7] M. Caporuscio, A. Carzaniga, and A. L. Wolf. Design and evaluation of a support service for mobile and wireless publish/subscribe applications. Technical Report CU-CS-944-03, Department of Computer Science and University of Colorado, Jan. 2003.

[8] M. Caporuscio, P. Inverardi, and P. Pelliccione. Formal analysis of clients mobility in the Siena publish/subscribe middleware. Technical report, Department of Computer Science and University of L'Aquila, Oct. 2002.

[9] 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.

[10] A. Carzaniga and A. L. Wolf. Content-based networking: A new communication infrastructure. In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, number 2538 in Lecture Notes in Computer Science, pages 59–68, Scottsdale, Arizona, Oct. 2001. Springer-Verlag.

[11] 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.

[12] Y. Diao, P. Fischer, M. Franklin, and R. To. Yfilter: Efficient and scalable filtering of XML documents. In *Proceedings of ICDE2002*, 2002.

[13] 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 *SIGMOD Conference*, 2001.

[14] K. Keeton, B. A. Mah, S. Seshan, R. H. Katz, and D. Ferrari. Providing connection-oriented network services to mobile hosts. In *Proceedings USENIX Symposium on Mobile & Location-Independent Computing*, pages 83–102, August 1993.

[15] H. Liu and H.-A. Jacobsen. A-ToPSS - a publish/subscribe system supporting approximate matching. In *Very Large Databases (VLDB'02)*, University of Toronto, August 2002.

[16] C. E. Perkins and D. B. Johnson. Mobility support in IPv6. In *Proceedings of 2nd ACM Conference of Mobile Computing and Networking*, 1996.

[17] M. Petrovic, I. Burcea, and H.-A. Jacobsen. S-ToPSS - a semantic publish/subscribe system. In *Very Large Databases (VLDB'03)*, Berlin, Germany, September 2003.

[18] P. Sutton, R. Arkins, and B. Segall. Supporting disconnectedness - transparent information delivery for mobile and invisible computing. In *CCGrid 2001 IEEE International Symposium on Cluster Computing and the Grid*, 2001.

[19] Talarian Inc. Publish-subscribe middleware helps direct traffic of Olympic proportions. http://messageq.ebizq.net/communications_middleware/talarian_2.html.

[20] D. Tam, R. Azimi, and H.-A. Jacobsen. Building content-based publish/subscribe systems with distributed hash tables. In *International Workshop On Databases, Information Systems and Peer-to-Peer Computing*, Berlin, Germany, September 2003.