Design and Evaluation of a Support Service for Mobile, Wireless Publish/Subscribe Applications

Mauro Caporuscio, Antonio Carzaniga, and Alexander L. Wolf, *Member*, *IEEE Computer Society*

Abstract—This paper presents the design and evaluation of a support service for mobile, wireless clients of a distributed publish/subscribe system. A distributed publish/subscribe system is a networked communication infrastructure where messages are published by senders and then delivered to the receivers whose subscriptions match the messages. Communication therefore does not involve the use of explicit addresses, but rather emerges from the dynamic arrangement of publishers and subscribers. Such a communication mechanism is an ideal platform for a variety of internet applications, including multiparty messaging, personal information management, information sharing, online news distribution, service discovery, and electronic auctions. Our goal is to support such applications on mobile, wireless host devices in such a way that the applications can, if they chose, be oblivious to the mobility and intermittent connectivity of their hosts as they move from one publish/subscribe access point to another. In this paper, we describe a generic, value-added service that can be used in conjunction with publish/subscribe systems to achieve these goals. We detail the implementation of the service and present the results of our evaluation of the service in both wireline and emulated wireless environments.

Index Terms—Publish/subscribe, mobility, wireless networks.

1 Introduction

A publish/subscribe system is a middleware communication service that delivers messages from a sender to one or more receivers using the preferences expressed by those receivers, rather than relying on an explicit destination address set by the sender. Specifically, a sender *publishes* messages, while receivers *subscribe* for messages that are of interest to them; the system is responsible for delivering published messages to matching subscribers. Examples of internet applications that can use a publish/subscribe system are multiparty messaging, personal information management, information sharing, online news distribution, service discovery, and electronic auctions.

A publish/subscribe system can be implemented by a centralized server or by a network of message routers. In the first case, every client application—acting as a publisher, a subscriber, or both—uses the system by connecting to a single server, which then acts simply as a switch.

In the second case, illustrated in Fig. 1, clients connect to one of several distributed access points. The access points are themselves interconnected through message routers that cooperate to form a distributed, coherent communication service.

In this paper, we focus on this latter type of publish/subscribe implementation. In particular, our goal is to support mobile client applications, that is, applications that move from one access point to another during their execution. We do this by implementing a *mobility support service* that transparently manages active subscriptions and incoming messages when an application detaches from one access point until it reattaches at another.

In order to better scope our work, it is useful to distinguish two classes of mobile publish/subscribe applications. The first class consists of applications running on a mobile host such as a notebook computer, a PDA, or a cellular phone. As the host moves (along with the person using the host), the applications may access the network from various locations through the publish/subscribe system. The host may also be disconnected for some significant amount of time while moving from one access point to the other. The duration of these black-out periods and the physical and topological distance between access points, may vary from local-area networks to wide-area networks, depending on the specific network technology in use, and obviously depending on the specific movements of the user. The second class of mobile applications consists of those that, using some form of mobile-code technology [11], [24], migrate autonomously from one host to another. During their life time, these applications (often referred to as "mobile agents") may execute on several hosts, and on any given host they may bind to and access the publish/subscribe system.

Although both classes of applications may benefit from using our mobility support service, the work described in this paper focuses primarily on the first class of applications, that is, on applications resident on the device carried by a mobile user. Our choice of scope is based on two practical observations. First and foremost, these applications are by far the most prevalent. Second, their mobility

- *M. Caporuscio is with the Dipartimento di Informatica, Universitá degli Studi dell'Aquila, Via Vetoio 1, I-67100 L'Aquila, Italy. E-mail: caporusc@di.univaq.it.*
- *A. Carzaniga and A.L. Wolf are with the Department of Computer Science, University of Colorado at Boulder, 430 UCB, Boulder, CO 80309-0430. E-mail: {carzanig, alw}@cs.colorado.edu.*

Manuscript received 31 Dec. 2002; revised 6 July 2003; accepted 5 Aug. 2003. Recommended for acceptance by Markku Oivo and M. Morisio. For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 118782.

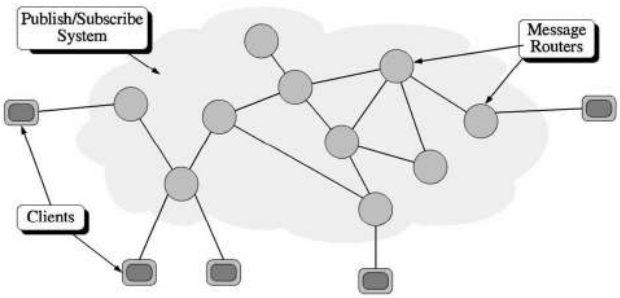

Fig. 1. Distributed publish/subscribe system.

needs are naturally limited to the mobility of the person using them and, therefore, occur at a manageable time scale.

The contributions of this paper are a detailed description of a portable, adaptive mobility support service for distributed publish/subscribe systems and an extensive evaluation of that service. The design is based on a client proxy that acts as an interface to the publish/subscribe system while the client is disconnected and that switches subscriptions and messages from one access point to the other when the client reconnects to the network. In addition to a basic buffering function, the proxy implements synchronization mechanisms designed to reduce the loss or duplication of information during the switch-over operation. These mechanisms are portable (i.e., easily reimplemented for use on another platform) in the sense that they do not rely on internal operations of any specific publish/subscribe system. Nevertheless, they are also adaptive in the sense that they implicitly exploit certain features of the underlying system opportunistically.

At a high level, our evaluation has three components. In the first part, we implemented the mobility support service on top of three different distributed publish/subscribe systems. The goal of this part of the evaluation was a feasibility demonstration of the portability of the service architecture. (A secondary outcome is a revealing survey, summarized in Table 1 of Section 2, of the architecture and availability of several publish/subscribe systems.) In the second and third parts of the evaluation, we analyzed the performance of the three implementations in a variety of scenarios. Specifically, the goal of the second part was to compare the behavior of the implementations on a wireline network and on a GPRS [1] wireless network. This evaluation is particularly relevant to the environment targeted by our work. The goal of the third part was to assess the effectiveness of the synchronization mechanisms across the various implementations and under different workloads of subscriptions and publications.

Overall, our results indicate that the mobility service architecture we propose is portable, and that its optional synchronization mechanisms adapt nicely to a significant sample of implementation features of the target publish/subscribe system.

The paper is organized as follows: Section 2 discusses related work, including the results of our survey of publish/subscribe systems. Section 3 details the architecture and implementation of the mobility support service. Section 4 reports on the performance evaluation. Finally, Section 5 summarizes our experience and discusses future work.

TABLE 1
Survey of Publish/Subscribe Systems

System Name	Architecture	Availability	Reference
Elvin	Distributed	Available	[20]
Herald	Distributed	Not Available	[3]
Hermes	Distributed	Not Available	[17]
JEDI	Distributed	Not Available	[9]
JMS FioranoMQ	Distributed	Available	
JMS JBossMQ	Not Specified	Not Available	
JMS Joram	Distributed	Available	
JMS OpenJMS	Centralized Distributed	Available Not Available	
CORBA OmniNotify	Centralized	Available	
CORBA OpenORB	Centralized	Available	
Siena	Distributed	Available	[6]
STEAM	Distributed	Not Available	[15]

2 Related Work

The work presented in this paper is rooted in our ongoing study of distributed publish/subscribe systems [6]. The idea of a publish/subscribe system is quite mature, and a fair number and variety of publish/subscribe systems have been proposed and described, including research prototypes [2], [9], [17], [20], commercial products [23], and attempts at standardization [16], [21], [22]. In recent years, researchers have focused on techniques to implement such systems with scalable, distributed architectures. However, despite this interest in distributed architectures, the issue of mobility or relocation of clients has not been explored in great detail.

The general problem of dealing with mobile clients is not specific to publish/subscribe systems. In fact, the designer of almost any distributed system must deal with duplicate or lost information when allowing clients to move or simply to switch their point of contact. A common case is that of a person changing addresses (e-mail or even postal address). In this situation, a common approach would be to announce the change of address to every known sender and to maintain both addresses for some period of time to catch messages inadvertently or unknowingly sent to the old address, possibly redirecting their senders to the new address. A more formalized, but conceptually identical, mechanism is used to implement network-level mobility support in mobile IP [4]. Our approach differs from this class of solutions because we do not maintain a fixed proxy, but rather combine a place-holder proxy with a reconfiguration of the publish/subscribe system to adapt to the new location.

To our knowledge, the first publish/subscribe system to propose explicit support for mobile clients was JEDI [9]. JEDI defines two functions—*move-out* and *move-in*—that a client can use to explicitly detach from the publish/subscribe network and to reconnect to it, possibly at a different location. In our mobility support service, we have adopted the same move-out/move-in interface. As for the implementation of the move-out and move-in functions, the authors of JEDI briefly discuss

how a client can move by detaching, serializing its state, and reconnecting. However, they do not detail the effect of a movement on the publish/subscribe system. Also, they recognize the synchronization problems involved in the movement, but explicitly avoid exploring those issues.

In the context of the Session Initiation Protocol (SIP) [19], an event notification protocol is specified [18] that is intentionally restricted. This protocol is based on common publish/subscribe interface concepts and tailored to SIP-specific tasks. The specification recognizes the need for migrating subscriptions to, for example, balance the overall notification load. However, both the mechanisms and algorithms necessary to implement such a migration, as well as the policies that determine when the migration is triggered, are left to the designers of individual implementations of the protocol.

Other researchers have studied the problem of mobility in the context of publish/subscribe systems. Among these studies, some are concerned with the general requirements posed by collaborative applications and mobile clients over the publish/subscribe system. The scope of this type of work is therefore broader than the one we describe in this paper and deals with other publish/subscribe interface design issues, such as the data definition for publications and the corresponding query language used to define subscriptions. One such study is that of Fenner et al. [10].

In other cases, researchers have focused on the specific problem of enhancing connectivity with mobile clients, which is also the focus of this paper, and have proposed techniques to solve this problem. The work of Huang and Garcia-Molina [12] belongs to this category. They propose a general framework, summarizing previous research of their own and of others, to adapt centralized and distributed publish/subscribe systems to a mobile environment. Although the proposed adaptation strategies are, at a high level, similar to the ones we present in this paper, our work differs in two ways from that of Huang and Garcia-Molina. First, their general approach is to modify the publish/subscribe system to adapt its functionality to the mobile environment, while our goal is to implement a mobility support service that is independent of the underlying publish/subscribe system. Second, they do not present an implementation of their techniques nor any empirical analysis of their performance, while a large part of our work focuses on experimentation with existing publish/subscribe systems.

To allow us to experiment with our mobility support service, we needed to obtain several representative publish/subscribe systems. We performed an extensive survey of the literature asking two basic questions: First, does the system provide a truly distributed implementation consisting of multiple, interconnected elements or just a single, centralized server? Second, is the implementation available for download and use? Table 1 summarizes our findings.[1]

1. Systems listed as unavailable may become available in the future. Further information on these systems is available at the following web sites: Elvin: http://elvin.dstc.edu.au/; FioranoMQ: http://www.fiorano.com/products/fmq/; JbossMQ: http://www.jboss.org/; Joram: http://www.objectweb.org/joram/; OmniNotify: http://www.research.att.com/~ready/omniNotify/; OpenJMS: http://openjms.sourceforge.net/; OpenORB: http://openorb.sourceforge.net/; Siena: http://www.cs.colorado.edu/serl/siena/.

Due to limited space, we describe only the systems we used in our experiments: Elvin, FioranoMQ, and Siena. The others are described elsewhere [5].

Elvin is an event notification system developed by the Distributed Systems Technology Center (DSTC) at the University of Queensland [20]. Its main features are quenching, server discovery, server clustering, and federation. Quenching allows a publisher to receive information about subscriber requests so that the publisher can limit its output to only those events that are of interest to at least one subscriber. Clustering and federation are the two distribution mechanisms used by Elvin. Servers collocated on a local-area network can be grouped into a *cluster*. Clusters appear as a single access point to clients, and use multiple servers to perform some load balancing. Clusters and servers can also connect to each other across a wide-area network to form a *federation*. The interconnection of individual servers and clusters in a federation must be acyclic. Routing within a federation is statically configured, meaning that every component of a federation must be statically configured for sharing specific publications and subscriptions.

The Java Message Service (JMS) [22] is an API specification defined by Sun that allows applications to create, send, receive, and read messages. FioranoMQ is an implementation of JMS. It implements the entire JMS API, and provides proprietary features for management, data storage, and load balancing. FioranoMQ is designed to have a centralized architecture as well as a distributed architecture made up of a federation of servers. A federation is built by using an additional entity called a *repeater*. The FioranoMQ repeater is a dedicated JMS client that acts as a gateway between servers.

Siena [6] is a distributed publish/subscribe system developed at the University of Colorado. Siena's goal is to maximize both the expressiveness of the subscription filtering language and the scalability of the implementation. In the simplest setting, clients may connect to a single Siena server. However, servers can also dynamically connect to other servers to form a networked implementation. Two implementations of Siena are currently available, one using a hierarchical network of servers, and the other using an acyclic network. Interconnected Siena servers maintain dynamic routing information for publish/subscribe data by propagating subscriptions to other servers, as necessary.

3 The Mobility Service

The general functional goal for our mobility service is to support the movement of clients between access points of a distributed publish/subscribe system. Our service does this by managing subscriptions and publications on behalf of a client application, both while the client is disconnected and during the switch-over phase. The service attempts to minimize duplication and, most importantly, loss of information. Our nonfunctional goals are portability to different publish/subscribe platforms, and adaptability to the characteristics of the given publish/subscribe system implementation, as well as to the characteristics of the underlying communication network.

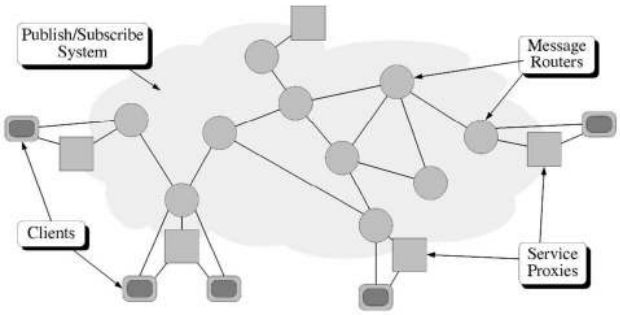

Fig. 2. Mobility service proxies.

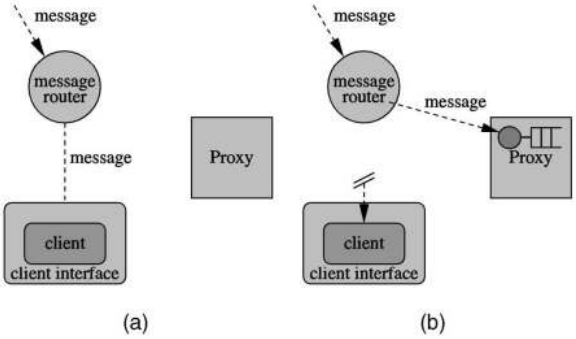

Fig. 3. Connected and disconnected client states.

Our design makes a few, relatively weak assumptions about the target publish/subscribe system. We assume a service API consisting of a general *subscribe* function and a general *publish* function. Notice that the subscribe function is stateful, meaning that it leaves behind state and modifies the behavior of the system. We are limiting the service to managing subscriptions only and, therefore, we are assuming that mobile publishers will themselves manage outgoing messages before, during, and after migration. This can be easily done using a simple local buffer in the client. We do not make any assumptions about the data model and filtering language of the publish/subscribe system, but we do require that subscriptions and publications can be "serialized" for storage. We also do not make any assumptions about the internal routing algorithm used by the publish/subscribe system.

The mobility service is implemented by *mobility service proxies*. Mobility service proxies are independent, stationary components that run at the access points of the publish/subscribe system (see Fig. 2). In addition to the proxy, mobile clients use a *mobility service client library* (or *client library* for short), linked with the client application. The client library wraps the target publish/subscribe API, mediating some of the requests made to the publish/subscribe system, and interacting with the mobility proxies during the move-out and move-in functions. Notice that, while the proxy is largely independent from the target publish/subscribe system, the client library is tightly coupled with the target publish/subscribe API. Whenever the language binding permits, the client library should be implemented as a derivative of the original target API by adding the move-out and move-in functions, and by overriding the subscribe function.

The basic operation of the mobility service is quite intuitive. During normal (connected) operations, the client publishes and receives messages directly to and from the publish/subscribe system. However, while the client is connected, the subscribe operation is mediated by the client library, which maintains a local copy of the client's subscriptions. Before detaching, the client calls the move-out function on its local mobility interface. The move-out function causes the client library to transfer its stored subscriptions to its mobility proxy. The proxy proceeds to subscribe using all the client's subscriptions and to store all the incoming messages in a dedicated buffer.

Fig. 3a shows a connected client, while Fig. 3b shows a disconnected client. When the client reaches its destination, it uses the move-in function to instruct its mobility interface to contact a local proxy (termed a "move-in proxy"), passing it the address of the remote proxy from which it detached (termed a "move-out proxy"). At this point, the local move-in proxy and the remote move-out proxy engage in a protocol that results in the transfer of all the subscriptions and all the buffered messages from the remote site to the local site and then onto the buffer maintained by the client library.

Notice that the client itself does not need to be aware of the move-out/move-in procedure. In fact, some monitoring process running on the mobile host, such as a power management daemon or a network connectivity daemon, might trigger both the move-out and move-in functions on behalf of all client applications running on that host. The same monitor could also take responsibility for configuring the mobility libraries by discovering the address of a mobility proxy in the vicinity of the mobile host.

Below, we detail the architecture of the proxy and the synchronization operations of the mobility service.

3.1 Mobility Service Proxy

The mobility service proxy is a fixed component running on various sites throughout the publish/subscribe network. One can think of mobility service proxies as local service stations for mobile clients. A mobility service proxy can serve multiple mobile clients. Here, however, we only discuss the proxy from the viewpoint of an individual client. The mobility proxies of a distributed publish/subscribe system are independent from each other and do not form permanent connections. The only connections between proxies are established as a consequence of a move-in function.

The internal architecture of a mobility service proxy is illustrated in Fig. 4. The proxy consists of a *proxy core*, which implements the core functionalities of the proxy, including the buffering for received messages and the synchronization services, plus three interface components: the *client interface*, the *proxy interface*, and the *service interface*, which manage the interaction with the client, with the other proxy (move-in proxy or move-out proxy, as the case may be), and with the publish/subscribe system, respectively.

3.2 Basic Mobility Services

When a client invokes the move-out function, the client's mobility interface invokes the move-out function on the mobility proxy, passing a unique client identifier, a list of

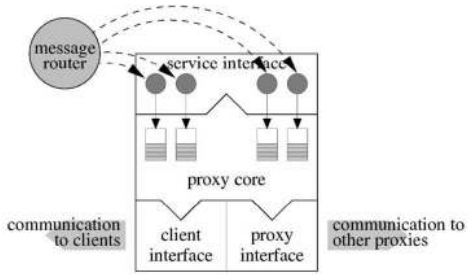

Fig. 4. Internal architecture of a proxy.

subscriptions, and an optional quality-of-service specification (discussed in Section 3.3). The mobility proxy executes the move-out function by creating a *client handler* with the given identifier and subscribing that client handler using the subscriptions passed by the client library. When the proxy is finished subscribing, it immediately sends an acknowledgment back to the client library. Upon receiving the acknowledgment from the proxy, the client library detaches the client from the publish/subscribe system and returns from the move-out function. The client library may choose to detach the client either by suspending its connection or by unsubscribing the client completely, depending on whether or not the publish/subscribe system supports a suspend function.

When a client invokes the move-in function, the client library may reconnect to the same proxy, which represents a situation in which the client has not moved, but was simply disconnected. In this case, the client library invokes an abbreviated move-in function that does not involve the transfer of messages from one proxy to another. The client library just restores the client's subscriptions, either by resuming the previously suspended connection or by reissuing all the client's subscriptions and absorbs the buffered messages. In particular, using the unique client identifier passed as a parameter to the move-in function, the proxy retrieves the handler associated with that identifier and transfers to the client library all the buffered messages. Finally, the proxy unsubscribes and destroys the client handler.

In the situation where the client does indeed move from one access point to another, the client library will connect to a different proxy and invoke the full move-in function. Of course, the full move-in function is a bit more complex. It uses the unique client identifier, the address of the move-out proxy, and an optional quality-of-service specification as parameters and proceeds in the following steps illustrated in Fig. 5.

1. The client library activates a local merge queue and starts up a receiver process to manage the queue.
2. The client library restores all the client's subscriptions by subscribing the receiver process at the new access point. At this time, some messages matching the client's subscriptions may be delivered directly by the router to the receiver process, which stores them in the merge queue.

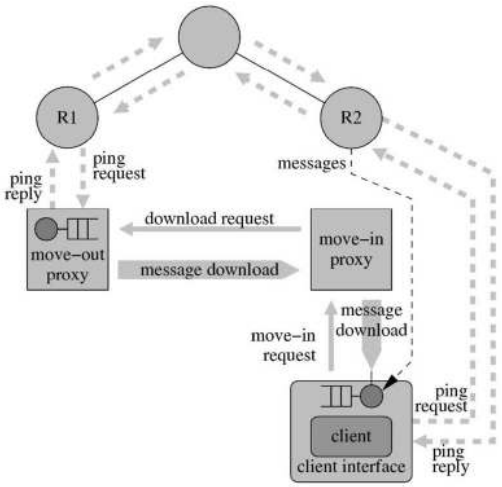

Fig. 5. Full move-in function.

3. The client library sends a *message download request* to the move-in proxy, specifying the address of the move-out proxy and the client identifier.
4. The move-in proxy contacts the move-out proxy and downloads all the messages buffered by the client proxy associated with the given identifier. The move-out proxy then unsubscribes its client handler, sends all the buffered messages to the move-in proxy, and destroys the handler. The move-in proxy forwards every downloaded message to the client library, which stores them in the merge queue.
5. After the client library has received all the messages downloaded from the move-out proxy, the client library redirects the client's subscriptions from the receiver process to the actual client, and destroys the receiver process.
6. Finally, the client library merges the messages from the merge queue, passes them to the client, and destroys the merge queue.

The steps involving the "ping" requests and replies shown in the figure are discussed in Section 3.3.

Some publish/subscribe systems provide a switch function that can be used to redirect messages from one access point to another. In cases in which this feature is absent, two main strategies are available to the client library when switching from the receiver process to the actual client. In one case, the client library may choose to implement the receiver process as a wrapper to the client, and therefore to always have the wrapper intercept received messages and pass them over to the actual client. In the other case, the client library would have to implement another local, and therefore simpler, switch-over process by subscribing the actual client and unsubscribing the receiver process. Both solutions have some clear advantages and disadvantages.

The mobility service allocates and maintains resources for each disconnected client and must therefore manage these resources carefully. Consider for example a situation in which a disconnected client either fails to reconnect or receives an excessive number of messages. In the former case, the service must determine when to reclaim the

resources, while in the latter case the service must decide how to limit the use of the resources. These types of decisions should be based on configurable quality-of-service parameters, either fixed or negotiated between the client and the service at move-out time. We note that a number of well-known time-out and quota mechanisms can be applied to this resource management problem. However, the current implementation of the mobility service does not address this issue.

3.3 Synchronization Options

It is easy to see that both the move-out and move-in functions are characterized by some phases in which messages may be lost or duplicated. Specifically, messages may be duplicated in the following phases:

- during the move-out function, after the proxy has subscribed its local client proxy, and before the actual client has detached since the same message may be delivered to both the client and the client proxy;
- during the move-in function, after the client library has set up the receiver process, and before the move-out proxy detaches its client proxy since the same event may be delivered to both the client proxy and the receiver process.

Messages may be lost in the following phases:

- during the move-out function, after the actual client has detached and before the subscriptions of the client proxy become effective;
- during the move-in function, after the move-out proxy detaches its client proxy, and before the subscriptions of the receiver process become effective.

The black-out periods may or may not occur, depending on the semantics of the subscribe function implemented by the target publish/subscribe system. For example, in some publish/subscribe systems, subscriptions are activated by established "routing" paths using subscription information (e.g., in Siena [6], [7], [8]) and, therefore, the implementation may return from the subscribe function before all the necessary routing information is in place and stable. This issue is discussed further in Section 4.

To avoid duplications, the mobility service may attempt to exclude duplicates from the merge queue. However, this idea has two major problems. First, the merge algorithm would require a comparison operator for message objects that the publish/subscribe system may not provide and whose semantics may not be completely clear. Second, and probably more importantly, a merge algorithm would also require that messages be somehow unambiguously identified so that the merge operation could distinguish a pair of syntactically identical, but distinct messages, from a pair of messages that are copies of the same original message. For these reasons, we have decided not to remove duplicates in our merge operation.

Avoiding losses is, to a large extent, possible and our mobility service provides two classes of mechanisms. They are applied to both the move-out and move-in functions and are designed to assure some synchronization between the publish/subscribe system and the mobility service so as to avoid black-out periods. In particular, when switching a set of subscriptions from point A to point B, the main idea behind these mechanisms is to make sure that the subscriptions at B are active before proceeding to remove the subscriptions at A.

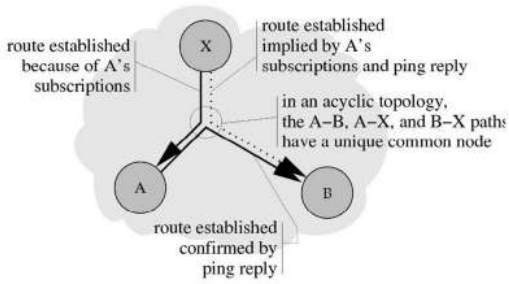

Fig. 6. Ping reply within an acyclic topology.

The first mechanism is based on a "ping" message sent from B to A and back using the publish/subscribe system itself, as illustrated in Fig. 5. The mechanism works as follows: During the move-out function, move-out proxy A will subscribe for a "ping request from B." The client library B will then subscribe for a "ping response from A" and will start publishing a series of "ping request from B" messages at regular intervals. Upon receiving a ping message from B, A will publish a set of ping responses, one of which will eventually reach B. It is only after receiving the response from A that the client library issues the download request that will eventually cause A to remove its subscriptions.

We use the fact that B has received A's ping response as an indication that the necessary routing information is in place. In particular, the receipt of a ping response implies that the publish/subscribe system has established a route for the ping response message along the path between A and B. This, in turn, is a good indication that routes are also in place for every other subscription (previously) issued by B from any other publisher X to B. The mechanism is guaranteed to be effective for acyclic topologies, where the routing information propagating from B will certainly reach some point Y between B and A (possibly A or B themselves) that is already connected to X. This must be true since the same set of subscriptions issued by B are already active between X and A, as illustrated in Fig. 6.

For other topologies, specifically ones in which there are multiple paths between message routers, this mechanism is not guaranteed to be effective since routing information for a ping response message may well reach its destination A before the routing information for the subscriptions issued by B reaches every other message router. Note, however, that none of the publish/subscribe systems we surveyed supports such a topology.

The second mechanism consists of a simple delay. The idea is to wait for some configurable amount of time before removing the subscriptions for proxy A. Besides its simplicity, this method has the additional benefit of not adding traffic to a possibly congested system. The obvious disadvantage is that, unlike the ping protocol, it is not adaptive and, therefore, can only provide a limited assurance of consistency.

The relative performance of the two mechanisms is evaluated in the next section.

4 Evaluation

The objective of our work is to create a value-added, portable, and adaptive mobility support service for the clients of a distributed publish/subscribe system running on mobile, wireless devices. This objective defines, more or less explicitly, a framework for evaluating the work. In particular, the goals of the evaluation study are the following:

- *To assess the benefits of using the mobility support service.* To satisfy this goal, we ask ourselves how well a client application would perform under various workloads with and without the support service.
- *To assess the portability of the architecture and service implementation.* In this case, the question we ask is how easy it is to retarget the implementation of the service to a new publish/subscribe system.
- *To assess the adaptability of the service and, particularly, its synchronization mechanisms.* This evaluation goal leads to a comparative analysis of the performance of the synchronization mechanisms across a valid sample of publish/subscribe implementation features.
- *To confirm the validity of the service design on the network substrates used by target applications.* This demands a performance evaluation across heterogeneous networks, ranging from fast and reliable wireline LANs to much slower and less reliable wireless links.

In order to answer the questions listed above, we implemented the mobility support service on top of three distributed publish/subscribe systems and then experimented with the different implementations using a simple mobile client application under various workloads and network configurations. For each experiment, we recorded a log of events, including every publication, every received message, and the client's movement operations. From these logs, we were able to collect aggregate metrics, such as the number of lost or duplicate messages and any delay resulting from a move-in function. We also plotted the logs in a style of graph that gives visual clues for many of the metrics.

The remainder of this section details the experimental setup as well as the most prominent results of our analysis.

4.1 Experimental Setup

We implemented the mobility support service for Elvin, FioranoMQ, and Siena (see Section 2). Although this is a somewhat limited set of target systems, we believe that they represent a valid sampling of features and interface types, tempered by what was available to us for experimentation.

In our experiments, we used a fixed publish/subscribe overlay network of three nodes, a single stationary publisher client, and a single mobile subscriber client. Again, this is a limited configuration, but still rich enough to illuminate significant differences among the test subjects.

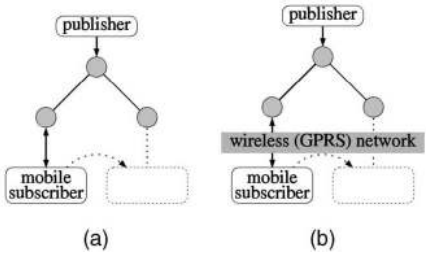

Fig. 7. Experimental network setup.

We mapped this configuration onto two types of network infrastructures. In both mappings, the whole publish/subscribe system and the publisher node reside on a wireline, low-latency, high-bandwidth network. In the first configuration (Fig. 7a), the mobile client also connects to the publish/subscribe system through wireline links, while in the second configuration (Fig. 7b), the mobile client connects to the publish/subscribe system through a simulated GPRS network. Specifically, we set up the first configuration using four computers running the publish/subscribe system and its clients, connected using a LAN. (The publisher client runs on the same host with its publish/subscribe server.) For the second configuration, we used the same four computers, with three of them connected over a LAN. The fourth runs the mobile client application connected through an IP network tunneled through a wireless network simulator called Seawind [13], [14].

Seawind emulates the behavior of a GPRS network according to its configuration parameters, thereby introducing characteristic delays, errors, and packet loss. The parameters we used in our experiments are listed in Table 2 and correspond to a class-6 mobile station with two and three timeslots in uplink and downlink, respectively, over a GPRS network with coding scheme CS-2 [1].

In all our experiments, the subscriber issues s subscriptions, while the publisher publishes a series of publications at a constant rate of p publications per second. We ran experiments with $s = 1, 2, 5, 10, 20, 50, 100$ and $p = 1, 2, 5, 10$. Subscriptions and publications are such that every publication matches all subscriptions. Although this might seem an extreme and unrealistic case for a single publisher and a single subscriber, it fits our goal of testing the mobility support service under a steady flow of messages. Every experiment presented here consists of a single migration of the mobile subscriber client, for a fixed disconnected interval of $\Delta = 10$ seconds. This, too, is perhaps an extreme case, corresponding for example to a person moving around a building, using a PDA connected to a highly distributed publish/subscribe system through a multipoint wireless network.

As it turns out, it is the combination of high numbers of subscriptions (s) and the total number of messages buffered during a disconnection period ($\Delta \times p$) that stresses the mobility support service. So, in that respect, our chosen parameters generate a fairly heavy load on the service.

TABLE 2
Seawind GPRS Parameters

Base-Rate	13400 bps (uplink/downlink)	Capacity of an individual channel
Max-Rate	2 uplink, 3 downlink	Maximum number of channels per user
Available-Rate	1–2 uplink, 1–3 downlink	Available channels
Error-Type	UNIT	Error probability applies to packets
Error-Probability	10^{-3} (uplink/downlink)	Error probability
Error-Handling	DELAY_ITERATE (uplink/downlink)	Link-level error-handling (GPRS network retransmits broken packets)
Error-Delay-function	30–50ms (uplink/downlink)	Delay used in retransmission function

4.2 Overall Overhead and End-to-End Benefit

Before we explore the detailed results of our experiments, we briefly review the overall impact of adding the mobility support service in terms of end-to-end delivery benefit during a client's movement. This brief presentation also introduces the main type of graph we use to display the behavior of a mobile publish/subscribe application.

Fig. 8 shows a first result indicating the general benefit of the mobility support service—that is, to guarantee an uninterrupted flow of messages, as seen from the viewpoint of a client application. More detailed results are given in Section 4.3. The left-side graph shows the case of a mobile client executing without the benefit of mobility support, while the right-side graph shows the same client using the mobility support service. Both graphs show results for the Siena implementation over the simulated wireless GPRS network, in a scenario of 50 subscriptions and a publication rate of two publications per second.

Let us explain how the graphs in Fig. 8 display the results. The figure shows two graphs that we call *departure/arrival traces*. The departure time is when a message is published, while the arrival time is when the message is actually received by the subscriber. Every point corresponds to a message received by the mobile client. Points are connected by lines representing the sequential ordering of messages as they are published by the publisher. Missing segments correspond to messages published by the publisher, but never delivered to the mobile subscriber. The two vertical dashed lines represent, respectively, the time when the subscriber invokes the move-out function and the time when the subscriber invokes the move-in function.

In the case without mobility support (left-side graph of Fig. 8), the vertical dashed lines simply represent the times when the client detaches from one access point and reattaches to the new one. Notice that all messages during this interval are lost. In the case of mobility support (right-side graph of Fig. 8), the vertical dashed lines represent not only the detach/reattach times, but also when the major processing provided by the service begins, especially message buffering and message merging. Notice that nearly all published messages eventually arrive at the client, but only after some amount of delay caused by the processing associated with the move-in function. Furthermore, notice that there can be a delay between when the client requests to be detached and when the client actually stops receiving messages. This delay is due to the processing associated with the move-out function.

The departure/arrival trace gives a good visual sense for the behavior of the mobile publish/subscribe application. Below, we use this representation to display more detailed results concerning the mobility support service.

4.3 Wireline Network

The next set of experiments are based on a wireline network configuration. We show a series of results for the mobile application, first without any mobility support, then with the basic mobility support service and, finally, combining the basic service with various synchronization mechanisms.

Fig. 9a shows the departure/arrival traces of the mobile client for the Elvin, FioranoMQ, and Siena implementations, but without using our mobility support service. As mentioned above, the messages published during the disconnection period are lost. Notice also that with Elvin some messages are also lost during a short transition period immediately after the mobile client reattaches (i.e., after the move-in function is performed). This behavior is due to the fact that the client library reissues all the subscriptions (100 of them in this experiment) upon reattaching; during this period of time, the publish/subscribe system and the underlying network

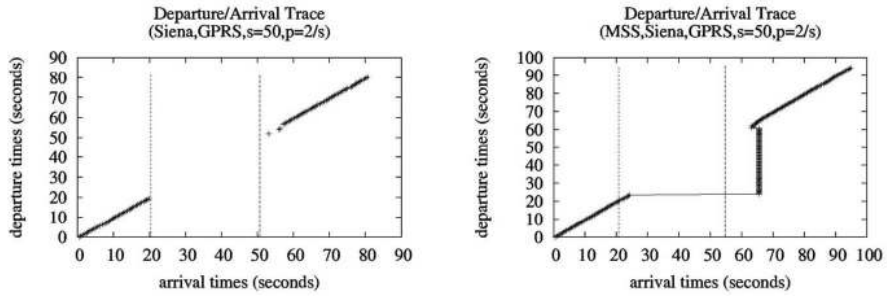

Fig. 8. End-to-end effect of mobility support.

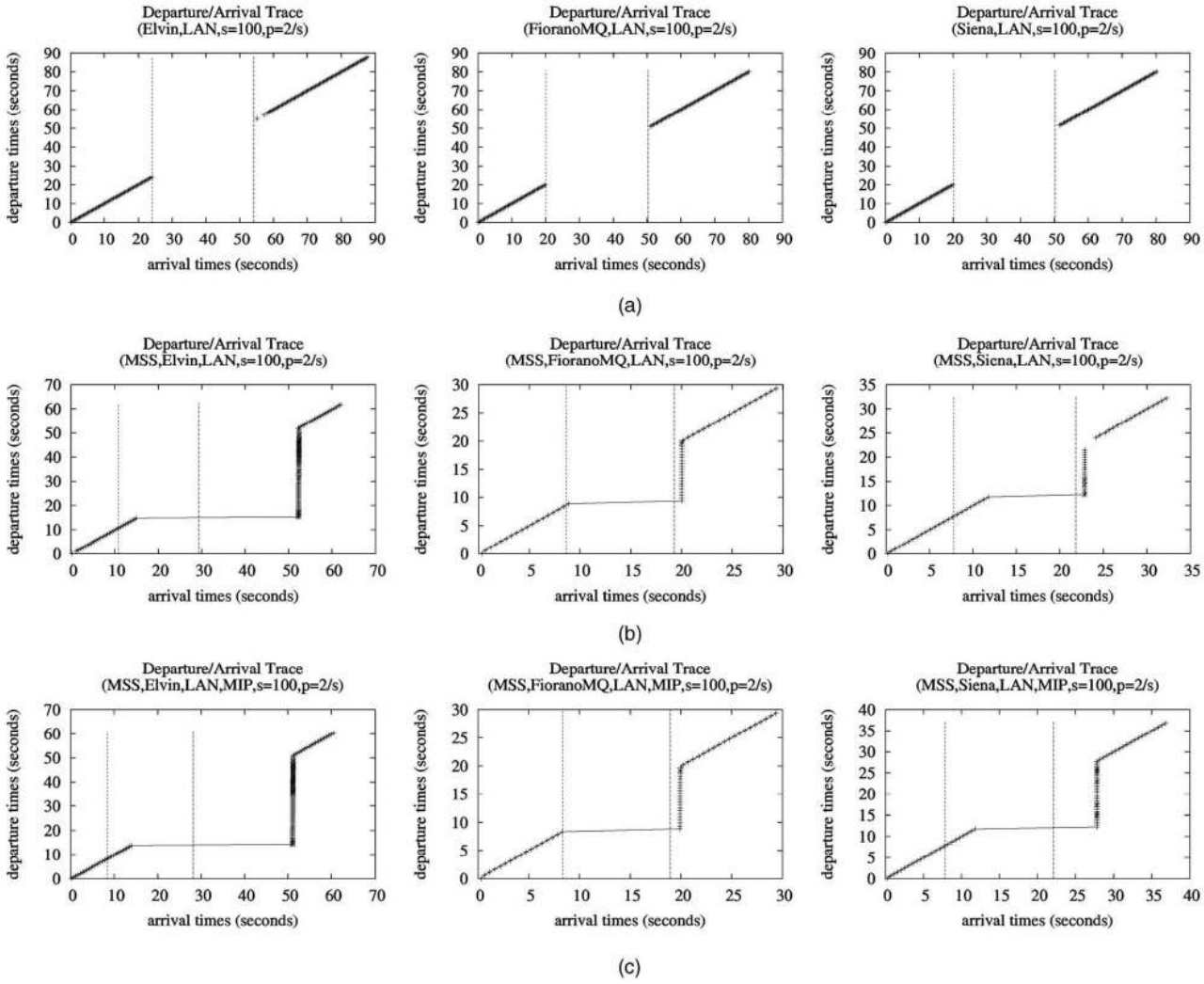

Fig. 9. No mobility support (a), basic mobility support (b), and mobility support with ping option upon reattach (c).

infrastructure are busy handling the subscriptions and dropping published messages. We did not notice this behavior in experiments with fewer subscriptions.

The traces of Fig. 9b represent the behavior of the application using the basic mobility service implemented on top of Elvin, FioranoMQ, and Siena. The nearly horizontal line indicates the period of time during which the mobile client is not receiving any messages, namely, the interval between the last received message after the move-out function is invoked and the first received message after the move-in function is invoked. The almost vertical sequence of message arrivals, which occurs some time after the client reattaches, indicates delivery of the buffered messages.

The graphs highlight two important quantities. The first is the interval between the detach time and the time the last message is received before the client becomes completely disconnected. This delay represents the move-out processing time, which is essentially the time it takes for the mobility proxy to subscribe on behalf of the client. The second quantity is the corresponding delay after the reattach time, which represents the move-in processing time, and amounts to the duration of Steps 2 and 4 of the move-in procedure presented in Section 3.2. In our experiments, with 100 subscriptions, the service shows better performance on FioranoMQ, than on the other two systems. On both Elvin and Siena, the application incurs a significant delay after the move-out function is invoked. Moreover, as mentioned above, on Elvin the service also suffers a performance degradation with the move-in function. Again, these delays are due to the fact that the system is busy processing subscriptions. Experiments using only 10 subscriptions (presented elsewhere due to space limitations [5]) indicate much better performance.

The latency incurred by the move-in function introduces a critical section in the use of the mobility service. A highly mobile application might want to move in at a site and immediately detach from it, before the move-in function has terminated. We have not studied the implications of such mobility patterns, and these behaviors are therefore not supported by the current implementation. The implementation is, however, robust with respect to these behaviors, in the sense that the move-in function is synchronous, and does not allow the client to call the move-out function at the same time.

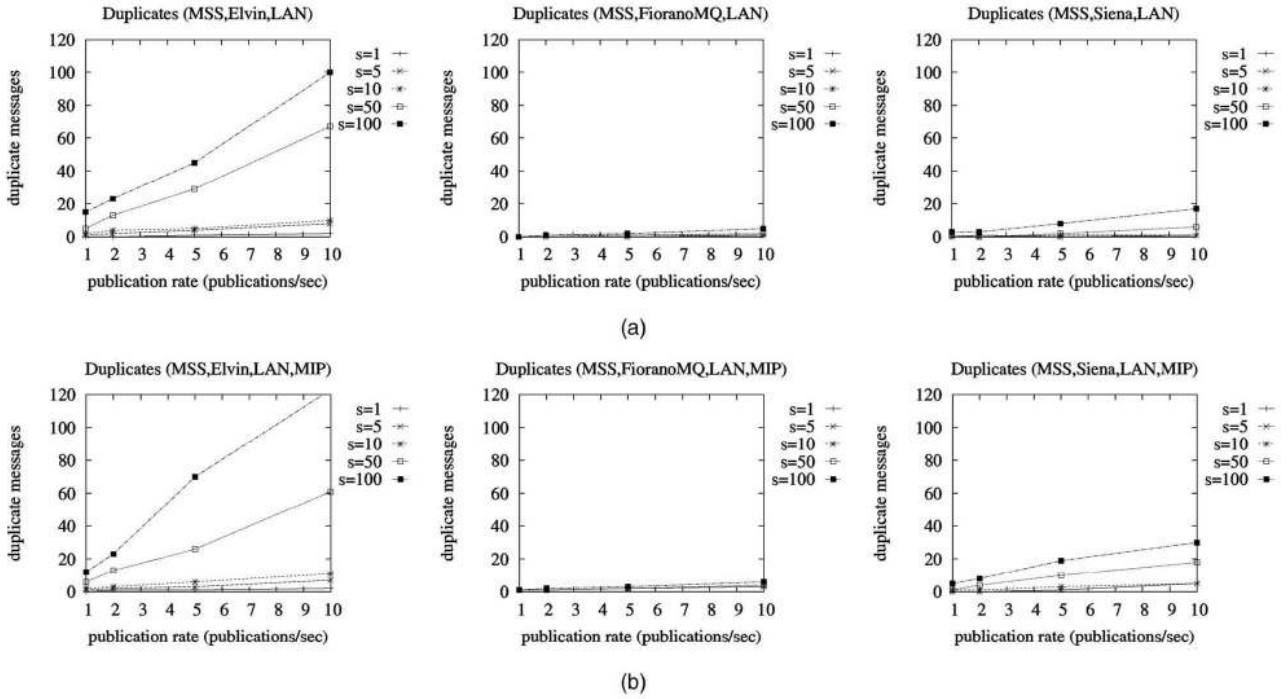

Fig. 10. Duplicate messages upon reattach without ping (a) and with ping (b).

Another phenomenon evident in Fig. 9b is how the Siena implementation loses some message after the move-in function. As explained in Section 3.3, messages are lost because the publish/subscribe system may return from the subscribe operation before all the necessary routing information has been propagated. Siena is the only system of the three that suffers from this problem because it is the only one that employs a truly distributed message routing algorithm. While the propagation of subscriptions is a prominent and documented feature of Siena, neither Elvin nor FioranoMQ specify how they treat subscriptions (and publications) when used in a distributed federation. Specific experiments conducted using simple network monitors indicate to us that they maintain subscriptions local to their access point and that they broadcast publications among those access points. Thus, the designers of Elvin and FioranoMQ evidently chose to incur the cost of flooding the network with publications, rather than trying to optimize message traffic through a subscription-based routing protocol. This approach shows benefits when a mobile client reattaches, but is quite costly at all other times.

The precise purpose of the optional synchronization mechanisms described in Section 3.3 is to reduce message losses during the move-out and move-in functions. Therefore, we have run all the previous experiments with the synchronization options turned on. Fig. 9c shows the traces for Elvin, FioranoMQ, and Siena, running the optional ping during the move-in function for the same scenarios of Fig. 9b. A comparison of these two series of graphs confirms that the ping option is an effective synchronization mechanism. In fact, we notice that Elvin and FioranoMQ have essentially the same behavior with or without the ping option, incurring the same delay. On the other hand, Siena incurs a slightly greater delay with the ping option. This delay is the price paid for the reduction of losses. The increase in reliability is evidenced by the fact that the departure/arrival trace of the experiment with ping does not show holes, which are instead visible in the traces of the same experiment without ping.

In practice, this means that the ping option is effective in reducing lost messages caused by a delay in propagating subscriptions, as in Siena. Moreover, this is achieved without penalizing implementations that do not experience such delays because they keep subscriptions local and broadcast their messages, as in Elvin and FioranoMQ.

This fundamental observation about the ping option leads to two further questions. How does the simpler delay option compare with the ping option? Also, since we can hypothesize an inverse correlation between the number of lost messages and the number of duplicate messages, how does the ping option affect the number of duplicate messages? The experiments we performed with the delay option provide an initial answer to the first question, giving us confirmation that the delay option is effective [5]. The second question instead requires us to look at this experimental data from a different viewpoint. In fact, although the departure/arrival traces give us some immediate visual clues, they do not show the precise count of duplicate messages, nor do they clearly show the precise amount of delay and the total number of lost messages.

Fig. 10 shows measurements of duplicate messages during the move-in function. These measurements are given as a function of the number of subscriptions and of the publication rate. The top row shows the basic mobility service, while the bottom row shows the experiments that use the ping option. For convenience in comparing the results pairwise, the graphs are set with the same vertical range for each system. These graphs give use two

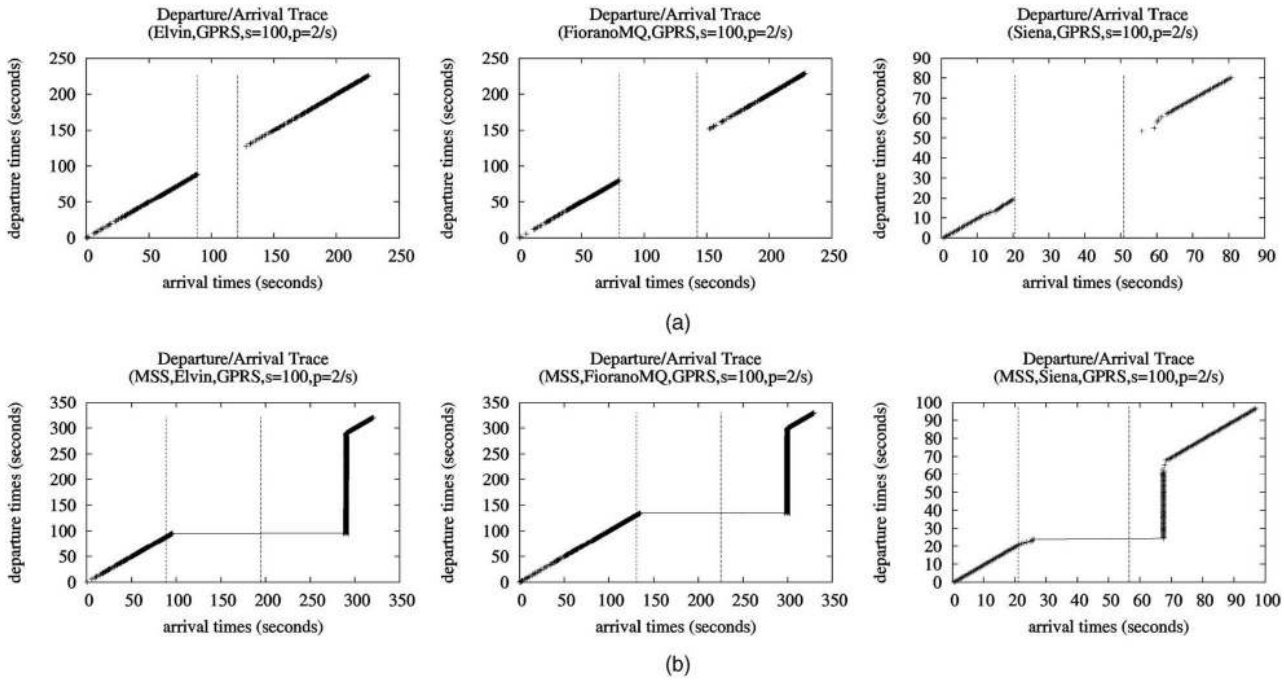

Fig. 11. No mobility support (a) and basic mobility support (b) over the wireless infrastructure.

confirmations. First, once again, we note that the ping option has only a very limited cost in terms of added duplicates on Elvin and FioranoMQ, which in fact do not need that option. Conversely, it increases the number of duplicates on Siena (in the worst case by 45 percent), which benefits from the ping option. The second and also intuitive confirmation is that the number of duplicates is proportional to the number of subscriptions and to the publication rate.

4.4 Wireless Network

In order to analyze the behavior of the mobility support service over a wireless network infrastructure, we conducted a second set of experiments, replicating all the combinations of publish/subscribe parameters and mobility support services explored in the wireline case. As mentioned above, the configuration for these experiments consists of a mixed network that includes a simulated GPRS link between the mobile client and the publish/subscribe system. Here, we report the interesting differences we found between the wireline and wireless configurations.

Fig. 11a shows the departure/arrival traces of the mobile client over a simulated GPRS network without the mobility support service. Notice how these traces show remarkably different behaviors than the corresponding cases on the wireline network (see Fig. 9). The presence of the slow, unreliable link between the client and the publish/subscribe system amplifies the loss of messages after the client reattaches to the network. Notice, too, that a similar effect is visible at the beginning of the trace, when the client attaches to the network for the first time. Finally, notice the difference between Elvin and FioranoMQ on the one hand, and Siena on the other. Elvin and FioranoMQ are noticeably slower than Siena in this case. We have not performed additional experiments to analyze this difference. However, we believe that the reason for this difference is that Elvin and FioranoMQ adopt communication primitives that are less bandwidth efficient and that favor reliability over speed, whereas Siena adopts a bandwidth-efficient "best effort" approach.

The presence of the GPRS link has an impact, as one would expect, on the use of the mobility support service. However, in the cases of Elvin and Siena, the performance of the mobility support service is comparable to the corresponding cases on a wireline network. Fig. 11b shows traces that use mobility support without synchronization options with 100 subscriptions. Notice that, comparing Fig. 11b to Fig. 9b, the only implementation that shows a noticeable performance degradation is FioranoMQ. Again, we believe that this difference is due to the particular choice of communication mechanism adopted for FioranoMQ.

As we did in the wireline case, we tested the effectiveness of the ping option in combination with the GPRS network. The results of these experiments are reported in Table 3. Once again, the experiments show that only Siena loses messages during the move-in function and, therefore, that Siena is the only system that needs the ping option. Nonetheless, the experiments also confirm the adaptive behavior of the ping option that causes only minor performance degradation to Elvin and FioranoMQ.

Another conclusion we can draw by examining the data of Table 3 is that the mobility service suffers a significant slowdown during the move-in function using Elvin and FioranoMQ over GPRS. Although we have performed only a few experiments to investigate this behavior in detail, we believe that it is due to the choice of communication protocol. In our experiments, we determined that both Elvin and FioranoMQ use one-time TCP connections to exchange messages and subscriptions, while Siena uses "keep-alive" connectors that attempt to reuse the same TCP connection

TABLE 3
Summary of Delay, Losses, and Duplicates upon Reattach over GPRS for $s = 10, 50, 100$ and $p = 2/sec$

	s	delay		losses		duplicates	
		basic	ping	basic	ping	basic	ping
Elvin	10	13551	17151	0	0	18	22
	50	48608	52525	0	0	76	88
	100	95031	98798	0	0	148	159
FioranoMQ	10	13932	15747	0	0	11	18
	50	40544	42215	0	0	43	50
	100	74773	77304	0	0	90	90
Siena	10	8294	11751	0	0	7	12
	50	10984	23585	2	0	4	17
	100	10841	39608	10	0	4	30

to pass multiple messages and/or subscriptions between clients and message routers (and between message routers).[2] As it turns out, this simple optimization of the communication within the publish/subscribe system yields a significant performance improvement over slow and unreliable communication channels.

4.5 Portability

Clearly, providing a comprehensive assessment of the portability of the mobility support service is a very difficult task. Nonetheless, our experience in successfully implementing the service on three, rather different publish/subscribe systems, and obtaining comparable, feature-independent behaviors, is an initial confirmation that the service is indeed portable.

We believe that the primary reason for the portability of the mobility support service is its simplicity. We do not have strong data to confirm our hypothesis, and a more thorough analysis is well beyond the scope of this paper. What we can show is a simple measurement (Table 4) of lines of code for the core mobility service (proxy and library) and for its specializations for Elvin, FioranoMQ, and Siena.

5 Conclusions and Future Work

In this paper, we presented the architecture of a mobility support service for distributed publish/subscribe systems. The specific goal of this service is to provide added-value support to the mobile clients of a publish/subscribe system. In simple terms, the support we offer consists of buffering incoming publications and in handling the switch-over of subscriptions from one publish/subscribe access point to another. With this service, we also implemented two synchronization options designed to reduce the loss of publications during the switch-over process. Both the basic service and its options are intended to be independent of the target publish/subscribe service. Yet they are also designed to adapt to the numerous implementation features present in different target publish/subscribe systems.

In addition to the design of the service, we presented the results of an extensive evaluation of its implementation. For this evaluation, we implemented the mobility service architecture on three distributed publish/subscribe systems and ran over 2,000 experiments for each one of the three implementations, under a variety of workloads and network configurations. The experimental data we have collected confirmes the validity of the general service architecture as well as the adaptive nature of its optional synchronization devices.

2. Elvin also features an optional communication mechanism based on persistent connections.

TABLE 4
Mobility Support Service Implementation Sizes

core service	639
Elvin interface	797
FioranoMQ interface	936
Siena interface	755

This work is part of our on-going research into advanced publish/subscribe systems and content-based networking. In this larger context, it is our first step in considering the issue of client mobility. We can think of two main extension paths. One direction is toward a service with more options and more configurability. An example of such a useful configuration feature would be a mechanism to selectively exclude subscriptions or publications from the mobility support. The other research direction is concerned with a tighter integration with the inner routing mechanisms that characterize truly distributed publish/subscribe systems. Such a study would analyze and possibly optimize the process of adjusting the routing information within the publish/subscribe system in response to, or in anticipation of, client migrations.

Acknowledgments

This work was supported in part by the US National Science Foundation, Defense Advanced Research Projects Agency, Air Force Research Laboratory, Space and Naval Warfare System Center, and Army Research Office under agreement numbers ANI-0240412, F30602-01-1-0503, F30602-00-2-0608, N66001-00-1-8945, and DAAD19-01-1-0484. The US government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government. The work of M. Caporuscio was additionally supported in part by the MIUR National Research Project SAHARA.

References

[1] 3rd Generation Partnership Project—Technical Specification Group, Digital Cellular Telecomm. System (Phase 2+); General Packet Radio Service (GPRS) Service Description; Services and System Aspects; Stage 2, Jan. 2002.

[2] G. Banavar, T.D. Chandra, B. Mukherjee, J. Nagarajarao, R.E. Strom, and D.C. Sturman, "An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems," *Proc. 19th IEEE Int'l Conf. Distributed Computing Systems (ICDCS '99),* pp. 262-272, May 1999.

[3] L.F. Cabrera, M.B. Jones, and M. Theimer, "Herald: Achieving a Global Event Notification Service," *Proc. Eighth Workshop Hot Topics in Operating Systems,* May 2001.

[4] P.R. Calhoun and C.E. Perkins, "Mobile IP Network Access Identifier Extension for IPv4," RFC 2794, Mar. 2000.

[5] M. Caporuscio, A. Carzaniga, and A.L. Wolf, "Design and Evaluation of a Support Service for Mobile, Wireless Publish/Subscribe Applications," Technical Report CU-CS-944-03, Dept. of Computer Science, Univ. of Colorado, Jan. 2003.
[6] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service," *ACM Trans. Computer Systems,* vol. 19, no. 3, pp. 332-383, Aug. 2001.
[7] A. Carzaniga and A.L. Wolf, "Content-Based Networking: A New Communication Infrastructure," *Proc. US Nat'l Science Foundation Workshop an Infrastructure for Mobile and Wireless Systems,* Oct. 2001.
[8] A. Carzaniga and A.L. Wolf, "Forwarding in a Content-Based Network," *Proc. 2003 Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM),* pp. 163-174, Aug. 2003.
[9] G. Cugola, E. DiNitto, and A. Fuggetta, "The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS," *IEEE Trans. Software Eng.,* vol. 27, no. 9, pp. 827-850, Sept. 2001.
[10] P. Fenkam, E. Kirda, S. Dustdar, H. Gall, and G. Reif, "Evaluation of a Publish/Subscribe System for Collaborative and Mobile Working," *Proc. 11th IEEE Int'l Workshops Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE),* June 2002.
[11] A. Fuggetta, G. Picco, and G. Vigna, "Understanding Code Mobility," *IEEE Trans. Software Eng.,* vol. 24, no. 5, pp. 342-361, May 1998.
[12] Y. Huang and H. Garcia-Molina, "Publish/Subscribe in a Mobile Enviroment," *Proc. Second ACM Int'l Workshop Data Eng. for Wireless and Mobile Access,* pp. 27-34, 2001.
[13] M. Kojo, A. Gurtov, J. Manner, P. Sarolahti, T. Alanko, and K. Raatikainen, "Seawind: A Wireless Network Emulator," *Proc. 11th GI/ITG Conf. Measuring, Modelling and Evaluation of Computer and Comm. Systems (MMB '01),* Sept. 2001.
[14] M. Kojo, A. Gurtov, J. Manner, P. Sarolahti, and K. Raatikainen, *Seawind v3. 0 User Manual.* Univ. of Helsinky, Finland, Sept. 2001.
[15] R. Meier and V. Cahill, "Steam: Event-Based Middleware for Wireless Ad Hoc Networks," *Proc. First Int'l Workshop Distributed Event-Based Systems (DEBS),* July 2002.
[16] Object Management Group, *Notification Service.* Aug. 1999.
[17] P. Pietzuch and J. Bacon, "Hermes: A Distributed Event-Based Middleware Architecture," *Proc. First Int'l Workshop Distributed Event-Based Systems (DEBS),* pp. 611-618, June 2002.
[18] A. Roach, "Session Initiation Protocol (SIP)-Specific Event Notification," RFC 3265, June 2002.
[19] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, June 2002.
[20] B. Segall and D. Arnold, "Elvin has Left the Building: A Publish/Subscribe Notification Service with Quenching," *Proc. Australian UNIX and Open Systems User Group Conf. (AUUG '97),* pp. 243-255, Sept. 1997.
[21] *Java Distributed Event Specification.* Mountain View, Calif.: Sun Microsystems, Inc., 1998.
[22] *Java Message Service.* Mountain View, Calif.: Sun Microsystems, Inc., Nov. 1999.
[23] *TIBR+: A WAN Router for Global Data Distribution.* Palo Alto, Calif.: TIBCO Inc., 1996.
[24] D. Wong, N. Paciorek, and D. Moore, "Java-Based Mobile Agents," *Comm. ACM,* pp. 92-102, 1999.

Mauro Caporuscio received the laurea degree in computer science from the University of L'Aquila, Italy. In 2002, he was a professional research assistant in the Software Engineering Research Laboratory at the Department of Computer Science at the University of Colorado at Boulder. He is currently a PhD student and a member of the Software Engineering and Architecture Group at the University of L'Aquila. His research interests mainly include formal methods for software validation, software architecture, distributed component-based systems, and physical and logical mobility.

Antonio Carzaniga received the laurea degree in electronic engineering and the PhD degree in computer science from Politecnico di Milano, Italy. He is an assistant research professor in the Department of Computer Science at the University of Colorado at Boulder. His current research interests are in the design and engineering of advanced communication systems and highly distributed software systems, with a special emphasis on content-based networking and publish/subscribe systems. He has also worked in the areas of distributed software configuration management, code mobility, and software process environments.

Alexander L. Wolf received the PhD degree in computer science from the University of Massachusetts at Amherst. He is a professor in the Department of Computer Science at the University of Colorado at Boulder. Previously, he was at AT&T Bell Laboratories in Murray Hill, New Jersey. His research interests are in the discovery of principles and development of technologies to support the engineering of large, complex software systems. He has published papers in the areas of configuration management, software architecture, distributed systems, networking, and security. Dr. Wolf served as program cochair of the 2000 International Conference on Software Engineering (ICSE 2000), is currently serving as the chair of the ACM Special Interest Group in Software Engineering (SIGSOFT), and is on the editorial board of the *ACM Transactions on Software Engineering and Methodology* (TOSEM). He is a member of the IEEE Computer Society.