A Pro-Active Mobility Management Scheme for Publish/Subscribe Middleware Systems

By

Abdulbaset A. Gaddah

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Ottawa-Carleton Institute of Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada

December 4th, 2008

The undersigned recommend to the Faculty of Graduate Studies and Research acceptance of the thesis

A Pro-Active Mobility Management Scheme for Publish/Subscribe Middleware Systems

Submitted by:

Abdulbaset A. Gaddah

in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Dr. Victor C. Aitken
Chair, Systems and Computer Engineering Department

Dr. Ioanis Nikolaidis
External Examiner

Dr. Thomas Kunz
Thesis Supervisor

Carleton University

December 2008

Abstract

The publish/subscribe (pub/sub) interaction paradigm has recently played a central role in the development of a large number of information dissemination applications such as stock trading, traffic information, news tickers, and electronic auctions. Its success is mainly attributed to its capability to decouple interacting participants, which makes it a good candidate for the development of such applications in mobile wireless environments that are characterized by frequent and unpredictable disconnections of participants due to wireless channel impairments or user mobility. Most current pub/sub middleware systems are optimized for fixed networks (i.e., users do not roam and the infrastructure itself is fixed). Therefore, add-on protocols are needed to extend such systems to cope with the challenges imposed by user mobility.

This thesis presents a novel and efficient mobile management scheme that is based on a pro-active caching approach (i.e., context transfer/caching occurs prior to the subscriber's movement) to extend current pub/sub systems to support mobility. This approach is based on the use of a data structure called *neighbor graph*, which dynamically captures the *set* of next potential brokers to ensure that subscriber context remains always one hop (broker) ahead of its current broker. The proposed approach employs "dummy" subscribers that automate the task of context caching and removal at immediate neighboring brokers on behalf of the actual moving subscribers. We have extended a JMS-based pub/sub system with our pro-active caching approach and observed the incurred overhead of the approach. This is achieved by comparing the end-to-end latency of message delivery as well as the latency of message routing with and without enabling our pro-active caching approach.

We have comprehensively evaluated the effectiveness of our proposed approach through testbed experiments, comparing it to the state-of-the-art solutions, *durable subscription-based* and *reactive*, proposed in the literature. The experimental results show that our pro-active approach reduces the message loss by more than 50% and message duplication to zero, compared to durable subscription-based approaches. The results also indicate that our approach experiences much lower handoff latency compared to reactive approaches. Overall, our proposed approach achieves superior performance across a range of scenarios. We conclude our work by discussing a modeling approach that can be used to extrapolate the performance of our approach in a near-size environment (in terms of broker and/or subscriber population) to our experimental environment.

Dedication

To my parents and wife

Acknowledgement

Million thanks to Dr. Thomas Kunz, my parents, and wife

Table of Contents

List of Figure

List of Table

List of Acronyms

API	Application Programmer Interface
CEA	Cambridge Event Architecture
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CTMC	Continuous Time Markov Chain
DCOM	Distributed Component Object Model
DSs	Dispatching Servers
IDL	Interface Definition Language
IEEE	Institute of Electrical & Electronics Engineers
IP	Internet Protocol
Java RMI	Java Remote Method Invocation
Java SDK	Java Standard Development Kit
JEDI	Java Event-Based Distributed Infrastructure
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JVM	Java Virtual Machine
LRU	Least Recently Used
MAC	Medium Access Control
MANETs	Mobile Ad-Hoc Networks
Message ID	Message Identification Number

NBR	Neighbor Mobility Pattern
NG	Neighbor Graph
ORB	Object Request Broker
Pub/Sub	Publish/Subscribe
P2P	Peer-to-Peer
QoS	Quality of Service
RND	Random Mobility Pattern
RMI	Remote Method Invocation
SIENA	Scalable Internet Event Notification Architectures
STEAM	Scalable Timed Event And Mobility
TCP	Transmission Control Protocol
ToPSS	Toronto Pub/Sub System
WLAN	Wireless Local Area Network
XML	eXtensible Markup Language

CHAPTER 1

INTRODUCTION

1.1 Background

The availability of portable devices and wireless network technologies has increased our dependence on receiving information and services through them. Users expect access to different information and services while they are roaming. Nevertheless, the limited and dynamic resources of the mobile computing paradigm impose several challenges that complicate the development of mobile information dissemination systems. Users with portable, wirelessly connected devices may frequently get disconnected from the network due to poor network connectivity, running out of battery, or when commuting between locations. They typically expect that data disseminated while they are disconnected will be available and can be delivered to them upon their reconnection. These constraints highlight the demand for middleware infrastructure, based on a flexible communication model, to meet the dynamic and decoupled nature of the mobile computing paradigm and facilitate the development of innovative applications.

While the traditional request/reply middleware systems like CORBA, DCOM, and Java RMI have proved their suitability in fixed networks, it becomes apparent that such systems are not adequate to seamlessly address the requirements of mobile computing systems [1][2]. Most of these existing systems are based on synchronous and point-to-point communication models that mainly impose a tight coupling between the sender and receiver participants. In such a classical *pull-based* model, a pull-consumer, requesting instantaneous updates of information, would require to continuously poll the information sink (the server), thereby resulting in server resource contention and network overload and congestion. Moreover, the high dynamicity of information suppliers is not supported in pure pull-based approaches, as new suppliers can only be discovered by exhaustively searching the network. This can be very costly in a large-scale network and is almost impossible in the mobile wireless context where a permanent network connection is not always guaranteed. Finally, traditional middleware platforms were designed with the assumptions that the terminals are powerful, stationary and will be permanently connected. These assumptions are unrealistic in mobile wireless domains as the mobile units are often resource poor, travel across different access points, and inter-communicate through extremely fluctuated connectivity. Therefore, this drives the need for an information dissemination model that can support the dynamicity of the mobile computing paradigm.

The publish/subscribe (or *pub/sub*) paradigm is largely recognized as one of the most effective way to model information dissemination applications [3], where *publishers* are event producers, *subscribers* are event consumers, and *brokers* are event dispatchers. The decoupling of publishers and subscribers in time, space, and flow in the pub/sub paradigm makes it highly scalable and flexible by hiding all explicit dependencies between the interacting parties. Time, space, and flow decoupling allow the publisher and subscriber parties to communicate without being connected simultaneously, being aware of each other, and being blocked while producing or consuming events. Pub/sub systems can also efficiently filter and disseminate a significant amount of data to a large number of subscribers. Hence, the loose coupling inherent in pub/sub-based middleware systems along with their inherently asynchronous and anonymous features make them a good candidate for supporting mobile wireless settings in a natural manner.

Given the potential of the pub/sub paradigm, the past few years have witnessed the development of a tremendous number of pub/sub middleware systems that vary along many directions. Most of them have focused on event dissemination in fixed networks (i.e., clients do not roam and the infrastructure itself is fixed). Some of these directions have covered fundamental issues: the expressiveness of the subscription language, routing, and filtering techniques. Others have considered and proposed some approaches to implement pub/sub systems with flexible and scalable, distributed architectures. While extensive research on such systems has been conducted [4][5][6][7][8][9], both in industry and in academia, for fixed networks, comparatively little research has recently focused on extending these systems to mobile, wireless domains [10][11]. The main interest of our research is to extend current pub/sub systems to better support subscriber's mobility. The extension is described in the context of Java Message Service (JMS) [12][13], one of the most widely accepted messaging system standards. The JMS semantics and features are briefly described in Chapter 4.

This chapter is organized as follows. Section 1.2 motivates this work and defines the research problem addressed in this thesis. Section 1.3 lists the contributions of this thesis. Section 1.4 provides a roadmap for the remaining chapters.

1.2 Pub/Sub Systems and Mobility: Problem Definition

There is currently a great deal of interest in pub/sub systems due to their high flexibility and scalability in dynamic distributed environments. As discussed earlier, the semantic characteristics of pub/sub systems make them a desirable choice for mobile information dissemination applications. First, the interacting parties can operate in the system without being aware of each other. Second, they can always be plugged in and out of the system without impacting each other directly. Third, the pub/sub paradigm is better adapted than the traditional point-to-point paradigm to cope with unannounced disconnected operation, which characterizes mobile wireless environments. Most existing pub/sub systems are optimized for fixed networks and have not considered the issues imposed by subscriber mobility. They assume (1) permanent network connectivity and that (2) information publishers and subscribers are stationary. Hence, this indicates a pressing need to extend pub/sub systems to the mobile wireless domain, which is the area of interest of this work.

Although the lower layers (such as link-layer and IP-layer) are conceptually the "right layers" to express the context of mobility support, the lower-layer mobility mechanisms have not been widely accepted and deployed for a variety of reasons: protocol stack modification, infrastructure change, considerable changes in the mobile host's kernel, and inherent operational complexity. The application-layer mobility protocols on the other hand can easily remove the major drawbacks of the lower-layer solutions and provide better mobility solution for the next-generation heterogeneous networks. This motivates our choice to solve the mobility problem at the application-layer (a detailed discussion about the motivation of our choice is presented in Section 4.3).

1.2.1 Mobility Aspects

Mobility, as described in [14] falls in two different and orthogonal aspects refereed to as code (or *logical*) and host (or *physical*) mobility: *Code mobility* refers to the migration of a code fragment (or *mobile agent*) autonomously from one host to another across the networks. *Host mobility* refers to the movement of mobile terminals from one access point to another. The two aspects of mobility are different in terms of how their mobility is handled. Although both mobility aspects are interesting and challenging, our research work has considered only the second aspect of mobility for two reasons. First, current mobile information dissemination applications are mainly designed to reside permanently on mobile hosts. Second, host mobility represents the majority of mobility scenarios.

1.2.2 Pub/Sub System in Wireless LANs: Mobility Issues

As indicated earlier, the broker topology of a pub/sub system can be either centralized or distributed. To reflect mobility scenario and to meet scalability aspects, this research focuses on a pub/sub system that is deployed as a distributed network of brokers. Figure 1.1 shows the architecture of such a system in a mobile wireless LAN network.

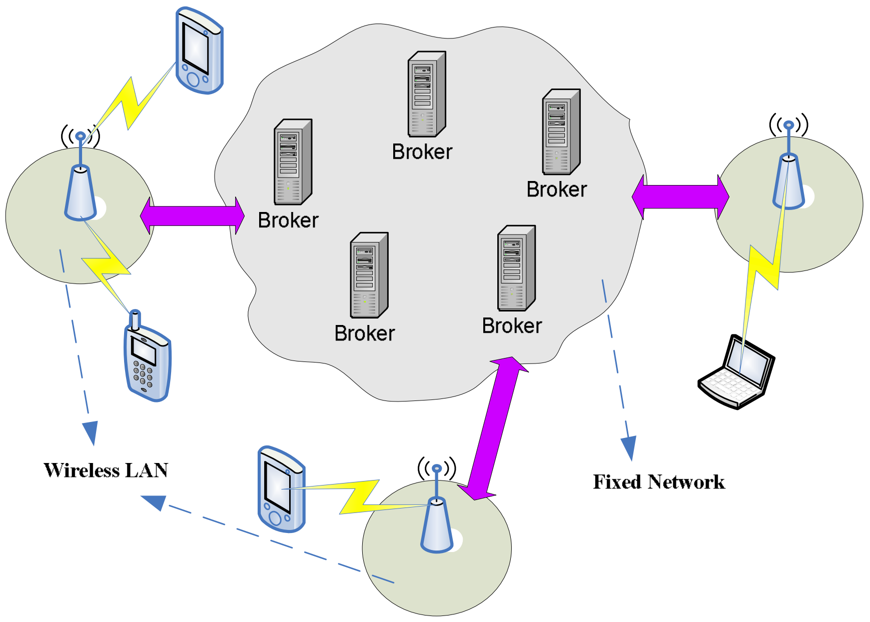


Figure 1.1: A distributed pub/sub system in WLAN network

A mobile client that can be either a producer or a consumer of messages or both connects to one of several distributed brokers through a wireless access point. The wireless access points form the boundary of the distributed communication service and maintain connections to the clients. The message brokers run on stationary machines that are located within the fixed network infrastructure. They are interconnected through a set of routers to form a distributed communication service. As our research interest is limited to manage subscriber mobility, we only focus on the mobility challenges from this prospect. Publisher mobility, discussed in [15][16] is beyond the scope of this research.

The vision of being connected to the same broker all the time is not valid any longer. Mobile subscribers may disconnect from one broker and reconnect to another broker while they are roaming. Due to the *handoff* procedure, subscribers may go through a temporary blackout period. There is also a possibility for the subscribers to get frequently disconnected from the network due to the absence of network connectivity or running out of battery. During such blackout periods, subscribers will loss their ability for receiving messages. This may result in missing some or all of their messages. Hence, mobility and temporary disconnections are the major problems that need to be managed by a pub/sub system. Limited bandwidth environments can be another challenging issue that may prevent or delay message delivery. Broker performance is also a major concern when extending pub/sub systems to the mobile wireless domain. The broker may become a performance bottleneck due to the extra load imposed by dealing with the above issues.

1.2.3 A Scenario for Mobility Challenges

In this research, we focus on the use of the durable subscription model [12][13] adopted by JMS-based pub/sub systems as it represents a good candidate for supporting disconnected operation. Durable subscriptions allow the brokers to track inactive subscribers and deliver their messages when they become active again. However, durable subscriptions, in the absence of a mobility management scheme, can face several challenges in mobile wireless environments as described in the following scenario. A timeline of a durable subscriber that goes through an interval of disconnecting and reconnecting to a different broker is shown in Figure 1.2.

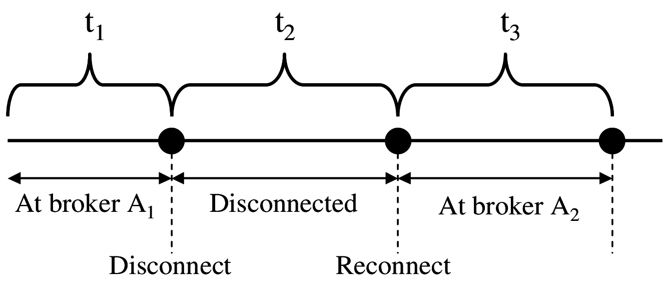


Figure 1.2: Disconnected operation timeline

During t_1 interval, the subscriber is connected to broker A_1 and can receive messages that match its subscription(s). By the end of the t_1 interval, the subscriber disconnects from broker A_1 for some reasons and reconnects to broker A_2 after interval t_2. During this time, the broker A_1 will keep locally buffering the messages that the subscriber would have received if it had been connected. When the subscriber reconnects to the broker A_2, it needs to resubmit the same copy of its subscription(s) to A_2 in order to receive messages. At the beginning of the t_3 interval, the broker A_2 starts routing messages to the subscriber based on the submitted subscription(s). Broker A_2 does not have any previous knowledge about the former subscription(s) that had been attached to the old broker. Also, broker A_2 cannot retrieve the former subscription(s) since it has no information about the location of the old broker. As a result, the subscriber will end up losing all the messages that were generated during t_2.

In the previous scenario, as the subscriber migration is transparent to the system, broker A_1 will have inactive subscription(s) that initiate the buffering activity. Such activity can happen quite often in the mobile environment as the mobile subscribers frequently move from one broker to another without removing their subscriptions. Even worse, they may leave the original broker and never come back again and hence the broker perpetually keeps buffering the messages. Moreover, the mobile subscribers may occasionally lose their network connectivity due to poor wireless connections, triggering buffering activity. Thus, each broker may end up having a large number of inactive subscriptions and has to track their corresponding messages. As buffering messages for disconnected subscribers puts a substantial overhead on the broker, the overall system performance may gradually degrade to the point of failure.

Let us consider the scenario when the disconnected subscriber reconnects to the original broker (broker A_1) after it has been connected to a different broker (broker A_2). By default, broker A_1 will stop the buffering activity for that subscriber and start routing the buffered messages to the subscriber. Since the broker is not aware of the messages that have been delivered by the other broker, the subscriber may receive duplicated messages, which may be problematic for some applications. Also, such duplications will flood the wireless channel and consume a significant amount of the channel bandwidth. The results of a pure durable subscription-based scheme, presented in Chapter 5, demonstrate the negative impact of such mobility scenarios on the system performance.

1.3 Thesis Contributions and Publications

This thesis presents the results of a broad-range study on research problems related to extending current pub/sub middleware systems to support subscriber mobility. The first contribution is a general survey of the state-of-the-art of research in pub/sub middleware domain. The review includes a general description of a representative set of pub/sub systems found in the literature and a deep investigation of the internal mechanism of the proposed solutions for extending pub/sub systems to support mobility, presenting some of their drawbacks. As part of our preliminary work, we have investigated and analyzed the behavior of pub/sub primitives and their reliability cost in a non-mobile, wireless environment. In particular, we focused on two subscription models supported by JMS-based pub/sub systems, *nondurable* and *durable,* that respectively provide low and high levels of reliability. The reliability cost is evaluated and compared with baseline data collected on a local-area, wired network. The study gave us valuable insights into the sensitivity of performance to the primitives supporting high reliability of message delivery. The results of our previous/preliminary work are presented in [17][18]. Others [19][20] have evaluated different JMS-based pub/sub systems and showed consistent results. The other contributions of this thesis are as follows:

A Pro-Active Context Distribution and Caching Scheme

This thesis presents a comprehensive and efficient mobility management scheme to extend current pub/sub middleware systems to operate in the mobile wireless environments. The main objective of the proposed scheme is to guarantee that all the published messages are successfully delivered to all interested subscribers in their publishing order regardless of the current location (broker) of the mobile subscribers. The proposed mobile management scheme is based on a pro-active caching approach that is initiated whenever a mobile subscriber hands off to a new broker. The pro-active approach depends on the use of a data structure, called neighbor graph, which is used to predict the set of next potential brokers where the subscriber context should be transferred prior to the subscriber's movement. The neighbor graph is automatically created and regularly updated to eliminate outlier neighbors. Whenever the mobile subscriber disconnects from its original broker due to its mobility or poor network connectivity, the set of next potential brokers are notified and requested to buffer the subscriber messages, using a subscriber's previously transferred context. The pro-active approach employs dummy (or *virtual*) subscribers that buffer messages on behalf of the actual moving subscribers. Accordingly, subscriber messages will be always available for the mobile subscriber at its next potential location.

Analytical Model for Messaging Cost

An analytical study has been carried out that captures the messaging cost of the proposed approach and the alternative solutions found in the pub/sub literature as it is of key interest to understand the overhead imposed by the proposed approach. Through this study, we present the messaging cost ratio between the different solutions and show the scenarios when the pro-active approach would have smaller messaging cost than the other solutions.

Implementation and Experimental Work

We show a prototype implementation of the proposed pro-active solution and the reactive solution and provide a detailed description to their core components. We then present our experience in evaluating the effectiveness of the proposed approach using a simplified but reasonable experimental testbed that is sufficient for the purpose of this research work: exploring the effectiveness of our solution and compare it to the alternative solutions. An extensive number of experiments are conducted to study the performance of our proposed approach under different workload conditions and mobility models, comparing it to the state-of-the-art solutions found in the literature. The experimental results show that our approach reduces the message loss by more than 50% and message duplication to zero, compared to durable subscription-based approaches. The results also indicate that our approach experiences much lower handoff latency compared to reactive approaches. The proposed approach overall shows superior performance across a range of scenarios.

Analytical Models for Extrapolating the Pro-Active Performance

An analytical model is developed to extrapolate the performance of our proposed pro-active scheme in a near-size environment (in terms of broker and/or subscriber population) to our experimental environment, using performance data obtained from our experiments. We first use the analytical model to derive the expected number of subscribers for a given broker topology, mobility model, and overall subscriber population, based on continuous-time Markov chains (CTMC). Next, we present how to extract performance-related data through curve-fitting from our testbed results. These curves relate the average number of subscribers with a performance metric of interest, here the per-broker throughput. The approach can be generalized to other performance metrics. Using these two steps, we can then extrapolate the throughput of the pro-active approach in a near-size environment to our experimental environment. This is an essential step as we were not able to achieve this experimentally due to the limitations of our experimental testbed.

Parts of this thesis appeared in the following publications:

1. T. Kunz, A. Gaddah, and L. Li, "Mobility support in a P2P system for publish/subscribe applications", to appear in Mobile Peer-to-Peer Computing for Next Generation Distributed Environments: Advancing Conceptual and Algorithmic Applications, edited by Boon-Chong Seet, to be published by IGI Global (USA), 28 pp in ms, accepted September 2008.

2. L. Li, A. Gaddah ,and T. Kunz, "Mobility Support in a Tactical P2P Publish/Subscribe Overlay", to appear in Proceedings of the 27th International Conference for Military Communication, (MILCOM2008), San Diego, CA, USA, November 2008.

3. A. Gaddah and T. Kunz, "Subscriber Mobility in Pub/Sub Systems: Pro-active vs. Reactive Handoffs", to appear in Proceedings of the 4th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2008), Avignon, France, October 2008.

4. A. Gaddah and T. Kunz, "A Pro-active Mobility Extension for Pub/Sub Systems", Proceedings of the First International Conference on Mobile Wireless Middleware, Operating Systems, and Applications, Innsbruck, Austria, February 2008.

5. A. Gaddah and T. Kunz, "Performance of Pub/Sub Systems in Wired/Wireless Networks", in Proceedings of 64th IEEE Vehicular Technology Conference (VTC'06), Montreal, Canada, pages 1-5, September 2006.

6. T. Kunz and A. Gaddah, "Adaptive Mobile Applications", in Encyclopedia of Information Science and Technology, edited by Mehdi Khosrow-Pour, pp. 47-52, Idea Group Publishing 2005, ISBN 1-59140-553-X. Second edition published September 2008, ISBN 978-1-60566-026-4.

7. A. Gaddah and T. Kunz, "Evaluating the Impact of Application Design Factors on Performance in Publish/Subscribe Systems over Wireline and Wireless Networks", Technical Report SCE-05-14, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, August 2005.

8. A. Gaddah and T. Kunz, "Does Modern Middleware Address Mobile Computing Requirements?", in Proceedings of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics, Orlando, USA, Vol. 5, pp. 493-499, July 2004.

9. A. Gaddah and T. Kunz, "Why Current Middleware Fails for Mobile Peer-to-Peer Computing", NATO IST-030/RTG-012 Workshop on the Role of Middleware in Systems Functioning over Mobile Wireless Networks, Wachtberg, Germany, August 2003.

10. A. Gaddah and T. Kunz, "A Survey of Middleware Paradigms for Mobile Computing", Technical Report SCE-03-16, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, July 2003.

1.4 Structure of the Thesis

This thesis is organized as follows.

Chapter 2: provides a general overview of pub/sub systems and a more detailed review of a representative set of these systems found in the literature. In particular, it discusses the most well-known approaches that have been proposed to extend the pub/sub systems to the mobile wireless environment. This chapter concludes by discussing the drawback of these approaches.

Chapter 3 provides a detailed description of our proposed approach that aims to extend current pub/sub systems to operate on mobile wireless environments.

Chapter 4 provides an introduction to JMS, describes the experimental setup, and presents the prototype implementation of our proposed solution.

Chapter 5 describes the evaluation results of our proposed solution and the alternative solutions along with detailed analysis and comparison.

Chapter 6 presents an analytical model that can be used to extrapolate the performance of the proposed approach and a comparative study for the analytical and experimental results.

Chapter 7 summarizes the achieved work and offers a list of topics for the future work.

C H A P T E R   2

PUBLISH/SUBSCRIBE OVERVIEW AND RELATED WORK

2.1 Introduction

This chapter presents a general overview of pub/sub systems and a more detailed review of a representative set of these systems found in the general literature. The last few years have witnessed the development of a tremendous number of pub/sub systems that are mainly designed for event dissemination in fixed environments. Many research communities have proposed and described techniques to develop such systems with a flexible and scalable, distributed architecture. They have also addressed some fundamental issues related to the expressiveness of the subscription language, event routing, and filtering techniques. While the existing pub/sub systems are well established in the fixed domain, they have been recently considered for the mobile wireless context. Unfortunately, mobility support has not yet been investigated in much detail in such systems. A narrow spectrum of research has recently been conducted to extend pub/sub systems to mobile environments, using different methodologies. For simplicity, the reviewed systems are classified into two categories: *non-mobile* and *mobile* pub/sub systems. We have covered the most popular systems that belong to these categories. However, our main attention is given to the later systems as our research focus is devoted to systems that recently have been extended to support subscriber mobility. We discuss their characteristics and limitations in terms of their mobility extensions.

This chapter is organized as follows. Section 2.2 provides a general overview of pub/sub systems. Section 2.3 describes the research efforts related to the pub/sub systems optimized for the fixed environments. Section 2.4 presents different mobility support extensions that are proposed to extend pub/sub systems to mobile wireless environments. Section 2.5 gives a summary of this related work.

2.2 Pub/Sub Systems: An Overview

The general objective of any pub/sub system is to disseminate information from senders to interested receivers, in an anonymous, decoupled way. Such a common general behavior is implemented with different flavors in actual systems known in the pub/sub literature. In this section, we provide a general overview of pub/sub systems, by first describing the participants to the system and their roles and then highlighting the various semantical features (or *properties*) supported by pub/sub systems.

2.2.1 System Components

A generic pub/sub-based system (or *Event Notification Service*) consists of a set of broker (or *dispatcher*) nodes organized into an overlay network that forwards messages from the publishers to the interested subscribers. The clients of this system are classified based on their roles into *publishers*, which are information producers, and *subscribers*, which are information consumers. The interaction takes place by passing messages (or *events*) from publishers to interested subscribers through the broker nodes. Brokers coordinate themselves in order to route information to all interested subscriber clients. Publishers notify the outside world about the occurrence of certain events. Subscribers that are interested in receiving particular sets of messages can express their interest by means of *subscriptions*. Upon receiving a new message, the broker node matches the message against all the subscriptions and then forwards it to all interested subscribers. Messages are forwarded to the subscribers in an asynchronous manner. Thus, the architecture of a pub/sub system depends basically on a mediated node that manages subscriptions as well as the delivery of messages and acknowledgements. Figure 2.1 illustrates the basic components of a pub/sub system.

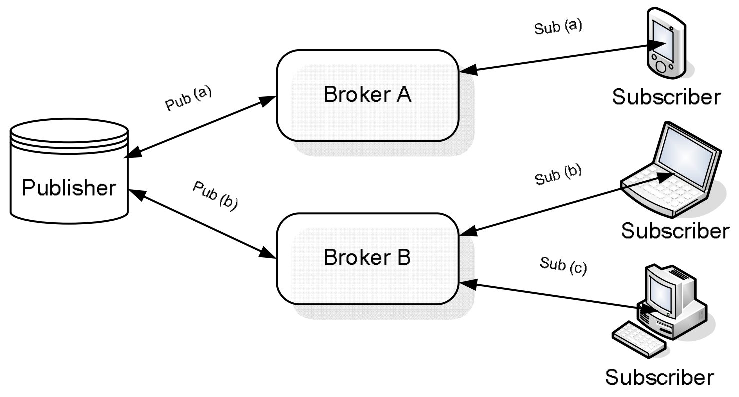


Figure 2.1: Components of a pub/sub system

2.2.2 The Basic Communication Model

The strength of the pub/sub communication model lies in the full decoupling of message publishers from subscribers in time, space, and flow [21]. The communication model is decoupled in time, because simultaneous interactions between publishers and subscribers is not required; decoupled in space, because publishers and subscribers do not need to know the identity of each other; decoupled in flow, because publishers are not blocked while publishing messages and subscribers can asynchronously receive messages while performing some concurrent activity. The decoupling of message publishers and subscribers makes pub/sub systems highly scalable and flexible by removing all explicit dependencies between the interacting participants. Indeed, it makes the resulting communication model well adapted to highly dynamic distributed environments that are asynchronous by nature, such as mobile computing environments [22].

2.2.3 Subscription models

Pub/sub systems are based on two different subscription models: topic-based (or *subject-based*) and content-based (or *property-based*). In the topic-based model, subscribers may register to one or more topics and thus receive all the messages delivered to those topics. Subscribers that share the same topic will receive a copy of each message within that topic. Although topic-based subscriptions are simple and easy to implement, they represent a static scheme that offers only limited expressiveness. The content-based model on the other hand extends the notion of topics by using a subscription scheme that is based on the actual content of the desired messages. Such a scheme allows subscribers to assign certain queries on the message properties as part of their subscriptions. Hence, subscribers are able to receive selective sets of messages published on a particular topic. It should be noted that messages do not rely on an explicit destination address set by the publishers. Instead, they are routed to the end destination based on their content. While a content-based model allows subscribers to select messages of interest, an efficient and scalable filtering mechanism is still an open issue.

Besides the topic-based and content-based models, JMS-based pub/sub systems support nondurable and durable subscription models. Nondurable subscriptions allow subscribers to consume messages as long as they are active. If a nondurable subscriber disconnects from the network, it will then miss all the published messages during the period of its inactivation. Nondurable subscriptions maintain low levels of reliability as the JMS broker does not keep records of inactive subscribers. At the same time, they impose lower overheads as the published messages are not buffered persistently for inactive subscribers. Durable subscriptions provide high levels of reliability at the cost of higher overhead. If a durable subscriber becomes inactive for a certain period of time, the JMS broker retains all the messages for the subscriber until it reactivates and consumes them. Therefore, durable subscriptions naturally support disconnected operations in a mobile environment.

2.2.4 Quality of Service (QoS)

The degrees of the QoS offered by the broker vary in different pub/sub-based systems. The most common QoS features supported in such middleware systems are briefly described. Persistence generally guarantees that messages will not be lost upon failure of the messaging system. Such messages are logged in an external storage until it is confirmed that they are consumed successfully. Priorities are offered to ensure that messages with higher priorities get processed before other messages. The priority levels and the way they are applied differ from one messaging system to another. Transactions are a set of operations grouped together into atomic blocks that are either entirely committed or safely rolled back. If any of the operations in a block fails the whole transaction will be re-processed.

2.2.5 The Broker Architecture

The broker architecture can be either centralized or distributed. A centralized architecture consists of a single broker entity that connects several publisher and subscriber clients. This central entity is potentially a performance bottleneck and a single point of failure. This affects system scalability and limits the use of centralized architectures to small-scale deployments. In a distributed architecture, a set of distributed brokers collaborate in collecting subscriptions and forwarding messages to the interested subscribers. Publishers and subscribers are not attached to a single broker entity; instead, they are distributed over several interconnected brokers. This can potentially reduce the network load and improve system scalability. The distributed brokers can be organized in several topologies that differ in terms of their strategies in routing subscriptions and messages. Two different broker topologies are presented in Figures 2.2 and 2.3.

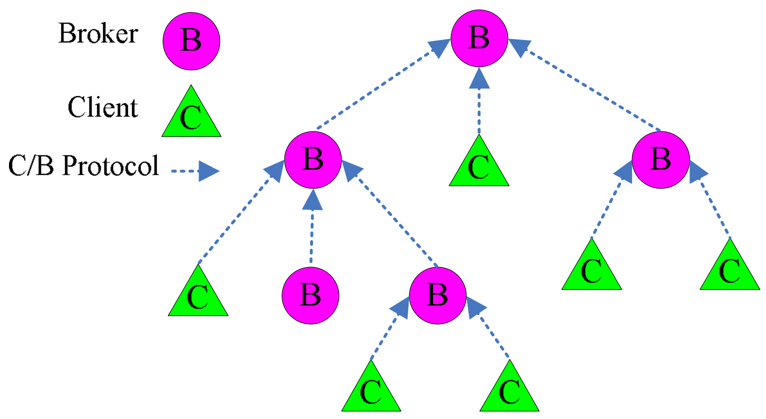


Figure 2.2: Hierarchical broker topology

In a hierarchical (or *multicast*) topology, the brokers are organized in a forwarding tree that has a root broker and several downward brokers. Excluding the root broker, each broker is considered as a client to the broker at the upward level of the hierarchy. Subscribers may connect to any broker regardless of the location of the corresponding publishers in the hierarchy. Whenever a new subscription is received, the broker forwards it upward to the root broker. Each broker on the way from the subscriber to the root broker stores a copy of the subscription. When a message is received by a broker, it is forwarded to the broker's parent. The message is also matched against all the stored subscriptions. This includes any subscriptions from downstream brokers. The broker forwards the message to any interested children (subscriber/broker) only if the matching result is true. Thus, messages are always forwarded upward to the root broker, and downward towards any interested subscribers. In this topology, each broker node is a critical point of failure. Also, parent brokers are potentially overloaded as they perform extra work for their children.

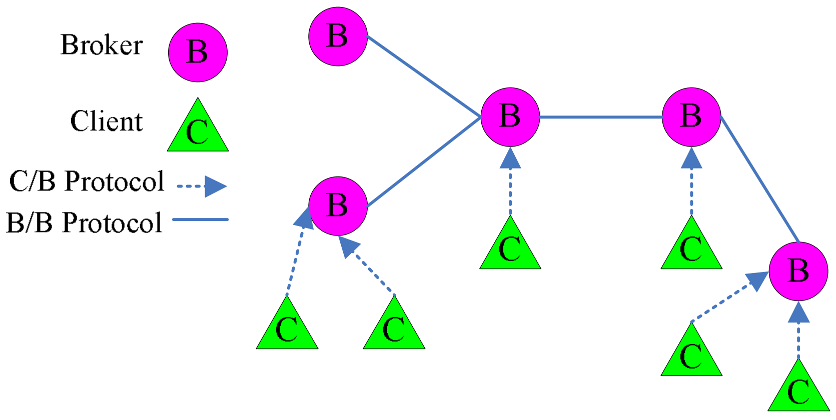


Figure 2.3: Peer-to-peer broker topology

A peer-to-peer (or *broadcast*) topology consists of a set of brokers that are connected in the form of symmetrical peers. Their communication model supports a bi-directional flow of subscriptions and messages. Each broker is responsible for a subset of subscriptions. A publisher sends a message to any broker that it is connected to. That broker than becomes responsible for propagating the message to all other brokers in the topology. When a new message enters the system, each broker checks the message against its own subscriptions and forwards it as necessary. It is apparent that the matching and forwarding overhead is reduced compared to the previous topology. This is because each broker needs to match messages against a portion of subscriptions. In this topology, the network will be flooded by the generated messages since they travel to all brokers.

2.3 Non-Mobile Pub/Sub Systems

This section describes the research efforts related to pub/sub systems optimized for the fixed environments. In such systems, subscribers are assumed to be stationary and the infrastructure itself is fixed. As a result, many research activities are aimed to enhance the functionality and performance of these systems in the fixed-domain. They have suggested several techniques that address some fundamental issues related to, but not limited to, system scalability, event matching, event routing, subscription expressiveness, and event distribution in a large-scale system. Next, we briefly describe some of these systems.

Hermes [23] is a distributed pub/sub system that consists of two main components, *event clients* and *event brokers*. The event clients represent both *event publishers/subscribers* and interact with the event brokers by using asynchronous message-passing (XML). The event brokers contain the entire implementation of the services that are used by the event clients. The main role of the brokers is to route events based on their content to all interested subscribers. To reduce network traffic, event filtering is done at the event publishers (*source-side filtering*).

Herald [24] is considered as an event notification service that is deployed as a self-configuration federation of peers. It consists of publishers, subscribers, and rendezvous points. Rendezvous points act as service access points for clients, allowing the clients connected to the same access point to communicate with each other. Herald does not support event filtration and it is not clear whether or how multiple rendezvous points can be interconnected to form a distributed implementation.

CORBA Notification Service [25] is a standard for the implementation of an event service built on top of CORBA ORB. The standard defines the IDL interfaces for three types of components that are involved in an event-based interaction. These are the event supplier, the event consumer, and the event channels. Event suppliers and consumers can be directly connected or mediated by an event channel that allows multiple suppliers to interact with multiple consumers asynchronously. The CORBA Notification Service supports a record-based event model. The channel can either be aware of the structure of the events (typed approach) or not (untyped approach). The CORBA-compliant event channels that are currently available on the market mostly present a centralized architecture.

The Toronto Pub/Sub System (ToPSS) [26] focuses on developing an efficient content-based pub/sub system for high speed event notification. The matching engine kernel is the core of the ToPSS system. The kernel implements a minimal predicate language on top of which different higher-level subscription languages are modeled. A-ToPSS [27] project introduces the notion of approximate matching that extends the kernel's predicate language. The semantics of this matching is mainly based on some probability and fuzzy set theories that measure the *possibility* and *necessity* with which the publication satisfies the expectation expressed by a subscription.

Gryphon [28] is a distributed messaging system that focuses on defining efficient algorithms to match events against content-based subscriptions and limiting the network traffic concerning event delivery. The system supports a record-based event model and a compound expression-based subscription approach. Upon receiving an event, each dispatcher executes a matching algorithm to determine the set of neighbors (other dispatchers or application agents) that are interested in receiving the events. Currently, there is no attention paid to how subscriptions are distributed to all event dispatchers. It is assumed that dispatchers are somehow informed of the subscriptions issued by all the connected active objects.

TIB/Rendezvous [29] is a commercial infrastructure used to create and maintain large, distributed, event-based applications. It offers several features including reliable and scalable distribution of events. TIB/Rendezvous uses a three-level hierarchical event dispatcher. Each node runs a TIB/Rendezvous daemon that is in charge of filtering events for the agents running in that node. The TIB/Rendezvous daemons interact with each other by means of broadcast messages. TIB/Rendezvous exploits subject-based addressing and does not offer full addressability of content-based messaging. Each event has an associated subject that plays the role of a special field.

2.4 Mobile Pub/Sub Systems

This section describes different mobility extensions proposed to extend pub/sub systems to mobile wireless environments. These extensions are based on techniques that differ in their methodologies as well as the environment in which they operate. The proposed extensions can be classified into three categories: reactive (or *fetching*), pro-active (or *prefetching*), and durable subscription-based (or *logging*) extensions. In reactive extensions, mobility is supported by fetching subscriber messages from the old broker just after the subscriber reconnects to the new broker. In pro-active extensions, mobility support is achieved by transferring and caching subscriber context just after disconnected operation takes place. In durable subscription-based extensions, all the published messages are logged at every broker visited by the subscriber, irrespective of its current active subscriptions. In the following subsections, we present the most significant systems with different mobility extensions that belong to one of these three classes.

2.4.1 Reactive Mobility Extensions

To the best of our knowledge, **JEDI** (Java Event-Based Distributed Infrastructure) [10] is the first proposed pub/sub system that natively supports subscriber mobility. The JEDI system exploits a set of dispatching servers (DSs), organized in a tree structure to simplify event routing. To support mobility, JEDI provides two operations: *moveIn* and *moveOut.* A subscriber may temporarily disconnect from its event dispatcher by invoking the moveOut operation, change its location, and reconnect possibly to a different DS by invoking the moveIn operation. The DS maintains temporary storage of events during the disconnection period and forwards the stored events to the interested subscribers when they reconnect again. Since the subscriber may reconnect to a different DS, the new DS needs to directly engage with the old one to retrieve all the subscriptions and events for that subscriber.

A drawback of this work is that JEDI cannot dynamically adapt the routing strategy to changes in the pattern of communication which results from mobility. JEDI DSs are specifically connected in a rather fixed tree (new DSs can only be added as leafs of the tree) and events are routed along this tree to travel from publishers to subscribers. The system is not capable of dynamically reconfiguring the topology of the dispatching tree to cope with changes in the networking environment (e.g. link breaks) or to changes in the overall workload (e.g. adding or removing DSs at runtime). Also, the tree-topology of JEDI is not scalable to a high number of DSs since the root node can become a critical bottleneck [30]. In [16], the authors argue that the cost of reconfiguring the multicast tree may be greater than the gains from having events take the shortest path. In addition, mobility support is not transparent to the applications, and even unrealistic as subscribers often can only react after having been relocated.

Cugola et al. [31] define an extension to JEDI to adapt the routing strategy to the changes in the communication pattern introduced by subscriber mobility. They propose a leadership election and group management protocol to use a different and dynamically built dispatching tree. The idea of their implementation model is based on existing multicast IP algorithms. When a dispatcher, named D, receives a subscription from a component, it checks if such a subscription has been already received. If this is not the case, it updates its internal tables and broadcasts the subscription to all other dispatchers. Upon receiving the subscription, each dispatcher updates its tables by storing the subscription information and a reference to D. Any equivalent subscription to the one propagated by D is not broadcast by any dispatcher in the hierarchy. Instead, it is delivered directly to D. D has implicitly become the leader of a group of subscribers. It maintains the access of other subscribers to the group and the distribution of group members in a tree. When an event is published, if the publisher is not part of the corresponding group of subscribers, the event is directly delivered to the group leader. In turn, the leader broadcasts it to all its neighbors in the dispatching tree. If the publisher belongs to the subscriber group, its dispatcher can immediately initiate the propagation of the event along the dispatching tree.

The potential downside of this approach is that leadership clashes may occur during the group startup as two (or more) equivalent subscriptions are independently broadcasted by different dispatchers in the graph. Due to subscriber mobility, the system needs to update the routing information on the dispatching tree. In the high mobility scenarios, this update can be very costly from the performance perspectives. The authors have assumed that components always disconnect in an announced way, which is not the most common scenario in mobile environments.

SIENA (Scalable Internet Event Notification Architectures) messaging system has been recently extended to support client mobility [32][33]. SIENA initially aims to maximize expressiveness of the filtration mechanism without sacrificing scalability of the routing mechanism. It is implemented as a distributed network of brokers that can be organized either in a hierarchical or a peer-to-peer topology. Each SIENA broker maintains a dynamic routing table for pub/sub data and broadcasts subscriptions to all brokers in the network. The mobility extension of SIENA is very similar to the JEDI approach. It is based on independent, fixed proxy components that run at various access points (brokers) of the pub/sub system. Each subscriber uses a local library that is linked with its applications to mediate some of the requests sent to the system, and to interact with the proxies during the handoff procedure. Each proxy temporarily buffers messages for disconnected subscribers, and transfers their subscriptions and messages when they are reconnected to a new access point (broker). Mobility service proxies use a synchronization mechanism designed to minimize the loss and duplication of messages during the handoff procedure. The old and new brokers interact directly to perform a handoff protocol that transfers the buffered messages from the old proxy to the new one, and eventually to the interested subscribers.

Although the reported experimental results have shown the applicability of this approach in supporting subscriber mobility, they are limited to the narrow evaluation of a single mobile subscriber roaming across the network. This limits the value of their results, since their approach never needs to transfer large volume of messages between brokers. The existence of proxy components in the system may affect the overall performance if they are heavily utilized. In Siena, publishers first generate advertisements (information about the type of events intend to publish) to create an event dissemination tree. The system then broadcasts these advertisements through the entire network of brokers. Such a global broadcasting may increase the system overhead and affect its performance. During the handoffs, the merge operation takes place to merge the buffered messages from the queue of the old and new brokers. Eliminating duplicated messages during this operation is discarded due to its cost and complexity. Therefore, the system suffers from the issue of duplicated messages. Also, the mobility service is manually trigged by the subscribers and involves several operations before they can be relocated. This is inapplicable in the wireless environments where the connection can be suddenly lost.

Farooq et al. [34][35] report their experience in evaluating the performance of SonicMQ [36], a commercial pub/sub system based on JMS, in a cellular environment using emulation environments. The goal of their research is to extend pub/sub systems in such a way that it can provide high performance under high frequency of handoffs and error prone wireless channels. They propose a middleware-level handoff protocol to manage subscriber mobility from one broker to the other. A component called *Handoff Manager* is located on each broker node to handle subscribers' mobility. The subscribers are responsible for locally storing the address of the last broker to which they were connected. When the subscribers disconnect from the network and reconnect again, instead of connecting directly to the broker, they first connect to the Handoff Manager. A sequence of interactions is performed by the Handoff Manager entities located on the old and new broker nodes.

Their approach is similar to the one introduced by [32][33]. The only difference is that the mobility support service (Handoff Manager) is integrated with the broker into a single entity instead of being run as a separate entity. This adds additional load on the broker and makes it awkward to support mobility under high frequency of disconnection and reconnection operations. This approach also does not hide the implementation details of mobility support from the applications, as some modifications are required at the application level. It is not clear how the Handoff Manager eliminates duplicated messages and arranges the messages in the publishing sequence. Their performance study does not show any results related to the overhead cost of such procedures. As indicated by [32], message coordination can be costly due to the high frequency of handoffs in mobile settings. Hence, their proposed solution does not guarantee the delivery of ordered and duplicate-free messages. This work partially overcomes the problem of message loss as a result of the way in which the Handoff Manager was implemented.

CEA (Cambridge Event Architecture) [37] supports asynchronous operation by means of events, event classes, and event occurrences. CEA follows a publish-register-notify paradigm with event object classes and source-side filtering based on parameter templates. It incorporates standard platform technology: IDL for publishing events and automatic stub generation for event notification. In CEA, an object has a register method in its interface, and interested parties can register interest in any event class. Moreover, access control is performed at event registration; the service does not allow a client without appropriate authority to register, and events that the service will notify are subject to restriction. When an event occurs, the service matches it against a stored template associated with each registration and each client whose template matches is notified of the event. To address the issues of mobility, CEA defines intermediate services, called event mediators. A mediator can prevent a mobile subscriber from missing events of interest while disconnected from the networked systems. The mediator registers interest with the required event sources on behalf of the mobile subscriber and buffers the event notification it receives from these sources. It also keeps track of the mobile subscriber's location and forwards the buffered events to the mobile subscriber at the new location.

CEA has mainly focused on extending widely used platforms like CORBA, RMI, or DCOM, to support asynchronous operation. The proposed extension exploits two types of event notification: direct source-to-client and mediated. One of the drawbacks of the direct notification is the lack of mobility support. Furthermore, the event source is potentially overloaded as it performs template matching along with event generation and delivery. Although the mediated approach provides support for subscriber mobility, it is not clear how the mobility support service is implemented. This work is also limited in that it does not allow multiple mediators to be interconnected in an arbitrary topology, essentially limiting the scalability of the architecture.

The **REBECA** notification service has recently been extended to support physical and logical mobility [38][39]. Physical mobility refers to the subscriber movement between different access points (brokers) in the system. While moving physically, the subscriber terminal may leave the coverage area of one access point and move into the reach of another access point. In contrast, logical mobility refers to the subscriber mobility within the coverage area of the same access point (broker). It offers a certain form of location awareness within the notification service to support location-dependent applications. The subscriber's movement is reflected in and mapped to changing subscriptions only. As the scope of our research is limited to the physical mobility, we will only discuss the mobility support related to this aspect. The basic idea of supporting physical mobility in REBECA depends on maintaining a "virtual counterpart" of a moving subscriber at the last visited broker until some broker at a new location takes over and then merges the "actual" and "virtual" subscriber such that message loss and duplication are avoided. As a subscriber detects the change of broker, it automatically re-issues a subscription along with the sequence number of the last received notification. As a result, the new broker will detect the movement of this subscriber and starts the relocation process by notifying its neighboring brokers. The new broker finds the junction of delivery paths to the new and old brokers by inspecting its routing table and its list of received advertisements, and compared it to the received subscription. It then sends a fetching request to the old broker to retrieve the subscriber messages. The old broker sends all the events buffered in the virtual counterpart of the moving subscriber starting with the sequence number provided initially by the subscriber to the new broker. Messages are routed through the junction to reach the new broker, and eventually the subscriber.

The authors do not justify why mobile subscribers cannot maintain the information about the last visited broker. Their approach depends on locating the junction broker in the broker network in order to process handoff requests. This can significantly complicate their algorithm and increase the handoff latency particularly in a large-scale network. It is most likely the junction broker gets overloaded, as all the fetched messages travel to the new broker through it. This may impact the system performance, particularly when the population of the moving subscribers increases. The proposed solution is based on a reactive manner, i.e. the handoff process occurs only after the mobile subscriber reattaches to the new broker, and hence incurs substantial performance overheads in mobile environments. There are currently no results that evaluate the performance of the physical mobility support to show its applicability and overhead cost.

Wang et al. [40] proposed a mobility management protocol for pub/sub systems, called *multi-hop handoff* (MHH) protocol, to achieve reliable and ordered delivery of messages to mobile subscribers with a minimized cost (in terms of message loss and duplication). In MHH, when a mobile subscriber disconnects from the system, the subsequently incoming messages will be buffered at the subscriber's last visited broker. Once the subscriber reconnects to a new broker, subscription and message migration are performed in parallel.

The subscriber's subscription(s) are moved hop-by-hop along the path from the last visited broker to the new broker. As for the message migration, the last visited broker forwards the locally buffered messages for the moving subscriber to the new broker, and all the brokers on the path collect the in-transit messages and directly transfer them to the new broker. Two different types of subscriber mobility are suggested, namely *proclaimed* and *silent* mobility models. In the former model, the mobile subscriber informs the system of its new destination before its movement, while in the later model, the mobile subscriber disconnects from the system without prior notice and has to keep track of the identifier of the last-visited broker. In both models, state transfer takes place after the subscriber reconnects to the new broker.

The proposed MHH protocol is similar to the one introduced by [32][33] discussed earlier. The only difference is that the message migration between the old and new broker is immediately suspended once the mobile subscriber disconnects before the handoff process ends. This is to avoid the frequent movement of undelivered messages between the brokers. In general, the suggested protocol will introduce high handoff latency as it may take a long time for the new broker to receive the subscriber's subscription(s) as well as its messages upon reconnection, particularly when the network is congested or is large. High frequency of subscriber mobility may greatly increase the overhead on the network traffic as the buffered messages for the moving subscriber will be frequently moved between different brokers. This may significantly result in furthering the handoff latency.

Hu et al. [41] addressed subscriber mobility in a distributed content-based pub/sub system. They mainly focus on the transactional semantics required by a mobile subscriber (i.e., a subscriber who wishes to disconnect from an original broker and reconnect to a new broker in the overlay as part of a transaction). They first identified the transactional semantics for a mobile subscriber and outlined the transactional concerns at various layers, focusing on the subscriber movement and routing protocol layers. We will limit our discussion to the subscriber layer movement protocol as it is within the scope of this thesis. The proposed subscriber movement protocol, that satisfies the defined transactional semantics, involves a conversation between the original and the target broker as described next. A mobile subscriber who wishes to migrate sends a MOVE message to the system indicating the target broker it wishes to moves to. The original broker in this case sends message (1) to the target broker that includes some information about the moving subscriber such as its ID and subscriptions. When the target broker receives message (1) and decides to server the subscriber, it initializes a transaction state for the subscriber and then issues message (2) that contains the same information received from the original broker. Message (2) executes the routing table reconfiguration protocol to maintain valid routing configuration states during subscriber movement. If the target broker does not accept the subscriber, it sends a reject message (3) to the original broker. In this case, the original broker will notify the system and the move request submitted by the subscriber will be aborted. When the original broker receives message (2), it stops the subscriber and sends message (4) to the target broker, containing the buffered messages for the moving subscriber. When the target broker receives message (4), it dispatches it to the new subscriber that accordingly merges the delivered messages with those stored locally at the target broker.

The proposed mobility management protocol requires the system to reconfigure the routing tables of all the brokers on the path from the original to the target broker. Such a behavior in the high mobility scenarios can be very costly from a performance perspective. In addition, message transfer between the brokers may drastically increase the load on the network, thereby degrading the overall performance. The proposed subscriber mobility protocol is mainly based on the assumption that subscribers have advance knowledge about their target brokers and move in an announced way. Such an assumption is an uncommon scenario in mobile wireless environments and hence limits the applicability of the proposed protocol to relocate subscribers' states between brokers but not supporting the physical mobility of the subscribers. Although the reported results show the overhead cost of the proposed protocol, there are currently no results that evaluate the performance of the protocol in terms of message loss/duplication during routing table reconfiguration.

Tarkoma and Kangasharju [42] discussed their experience in investigating the safety and cost of handoff protocols for pub/sub clients in content-based routing networks. The upper and lower bound costs of handoff protocols were determined for three different topologies, refereed to as *generic mobility support*, *acyclic graphs*, and *rendezvous-based* topologies. The authors also discussed the impact of completeness and incompleteness of the routing topology on the cost of handoffs. The reported formal study gives valuable insight into the engineering of efficient and mobility-safe handoff protocols. Through this study, three techniques were proposed to improve mobility support in pub/sub systems: *overlay-based* routing, *rendezvous points*, and *completeness checking*. Overlay-based addressing prevents the content-based flooding problem. It abstracts the communication used by the pub/sub system from the underlying network-level routing and enables the system to cope with network-level routing errors and node failures. Rendezvous points simplify mobility by allowing better coordination of topology updates. Completeness checking ensures that subscriptions and advertisements are fully established (complete) in the topology. This is needed to perform the covering optimization.

Although the discussed mobility protocols gives a reasonable set of engineering guidelines for the development of efficient handoff protocols in content-based pub/sub systems, they are not intended to suggest new solutions to support subscriber mobility. This is different from our work as we introduce a new approach for managing subscriber mobility. The reported results are limited to a single performance metric (i.e., message cost), which is not sufficient to completely evaluate the impact of different protocols and topologies in dealing with mobile pub/sub clients.

2.4.2 Pro-active Mobility Extensions

Burcea et al. [30] identify and classify the factors that affect the performance of a pub/sub system intended to support subscriber mobility. They also formalize a number of mobility algorithms for distributed pub/sub systems with the objective of optimizing the costs posed by disconnected operations. These mobility algorithms are referred to as *prefetching*, *logging*, *home-broker*, and *subscriptions-on-device*. They are largely different in terms of their methodology for optimizing the mobility costs. We limit our discussion here to the prefetching algorithm as it belongs to the category of pro-active extensions. Prefetching is similar to the extension proposed by [10], except that the subscriptions and events transfers occur when the subscriber disconnects from its original broker. This minimizes the latency of the state-transfer between the old and new broker and reduces the number of events that need to be transferred over the network.

The effectiveness of prefetching algorithm is based on the successful approximation of the next subscriber's destination (broker), which is not considered in their solution. Instead, a pre-defined mobility pattern is used to define target brokers for prefetching. In contrast, our pro-active approach makes use of a data structure, called neighbor graph, which automatically identifies the next potential target brokers. As a part of the prefetching algorithm, the subscriber context (subscriptions/events) is removed from its original broker once it has disconnected. Thus, it cannot support a reconnect operation to the same broker. Our pro-active approach takes into consideration this type of operation, which may occur due to the loss of wireless connectivity. Although the authors measure the performance of prefetching in terms of network load, they have not considered other performance metrics such as message loss, message duplication, and overall throughput. Furthermore, they have not evaluated the prefetching mechanism to multiple brokers. This limits the applicability of their achieved results. The proposed prefetching algorithm is not appropriate for supporting fast handoffs as the prefetching takes place just after the mobile subscriber disconnects from its original broker. Due to high network latency or very fast mobility, such a scheme may fail to support subscriber mobility as the subscriber may reach its new destination prior to the arrival of its subscriptions and events from its previous broker. In our approach, the subscriber context is always transferred prior to the subscriber movement to support fast handoffs.

Cilia et al. [43] have proposed two forms of mobility extensions that exploit per-subscriber proxies and distributed buffers to improve the bootstrapping process. The first extension uses the knowledge of future mobility patterns to harness possible subscriber movements and attach to potential future locations before the subscriber application needs the data. The second form of extension is based on distributed buffers and a mechanism that allows to access notifications from the past. We mainly focus on the former extension since it belongs to the pro-active category. The idea of such an extension is to implement per-subscriber caching at potential future locations. When the subscriber disconnects from the network, a virtual counterpart is created, establishing per-subscriber caches in the network to forward buffered events to the subscriber when it eventually arrives at its destination.

The authors do not provide any concrete results that demonstrate the applicability of their proposed extensions. They only provide a high level description of the basic architecture of their extensions. Similar to the previously discussed scheme, the mobility prediction has not been considered in the proposed solution. It is not clear how their extension deals with the propagated subscriptions after the subscriber hands off and whether subscriptions are kept locally for potential future reuse as adopted by our pro-active approach to reduce the propagation overhead. The proposed extension is also similar to the previous extension as their pre-subscription approach takes place while the mobile subscriber is disconnected. This may affect the adequacy of their approach in supporting fast handoffs as discussed earlier. The mobility extension is based on a per-subscriber caching mechanism that may overload the hosted broker. Subscribers may have similar interest in receiving certain events and hence caching events per subscriber increases the possibility of storing similar messages on the same broker. This increases the load on the broker in terms of memory usage and processing time. Our pro-active approach benefits from subscription similarity as it uses a shared buffer to cache a single copy of each message that matches similar interests of multiple subscribers. Moreover, scenarios such as permanent and long interval disconnections, which may occur due to subscriber crashes/failures, are not considered in their work. The performance of the active subscribers may get affected due to the overhead of the continuous per-subscriber caching process. In our pro-active approach, we take care of such scenarios using a timeout interval that allows our approach to reclaim the buffer space and minimize the overhead of caching process. The accuracy of their approach (in terms of message loss) depends on the interval that is required to transfer subscriber subscriptions and create per-subscriber proxies. This is because the latency of this process is not considered in the proposed extension. In contrast, our pro-active scheme takes into account such latency as it caches/transfers events, published during subscription activation process, to the next potential brokers.

2.4.3 Durable Subscription-based Mobility Extensions

JMS (Java Message Service) [12][13] is a set of APIs developed by Sun Microsystems. It aims at representing the standard, common interface for Java messaging systems. A brief overview of JMS characteristics is given in Chapter 4. To support disconnected operation, JMS supports a particular type of subscription, called durable subscription. Durable subscriptions allow subscribers to receive messages generated while they are disconnected. Messages are stored persistently and delivered to the subscribers when they become active. JMS does not specify how the messaging server is implemented (i.e., as centralized component or through a set of distributed dispatchers). We have reviewed several implementations of JMS, including OpenJMS [44], Joram [45], FiornaMQ [46], JBossMQ [47], and JavaSMQ [48]. We were not able to find complete and functional support for subscriber mobility. Durable subscriptions and persistent messages can provide a partial solution.

Podnar and Lovrek [49] proposed a mobility extension based on persistent notifications that require the delivery of valid notifications just after the activation of a new subscription in the systems. Each broker is responsible for storing the received notifications in a persistent buffer and then forwards them to interested subscribers and neighboring brokers. These notifications are removed once their validity period expires. The broker also maintains a list of valid notifications that have been sent to subscribers and brokers. When a subscriber reconnects to the system, the broker activates its subscriptions and delivers the valid stored notifications. The subscriber must provide a list of the previous received notifications to avoid duplicated notifications.

This approach clearly creates extra traffic in the broker network, and increases the usage of broker memory and processing time. Due to the costly buffering process, the overhead on the distributed brokers can be significantly increased. This may result in degrading the system's performance. Also, the number of lost notifications can increase as the buffer space drains quickly or the notifications become invalid due to a large disconnection interval. Duplicated messages can be received when the mobile subscriber reconnects to the old broker after moving to a new one. The preliminary experimental results do not reveal the performance of the proposed extension with respect to message loss and duplication.

The **Elvin** system [11] in its latest version has added extensions to support disconnected operation in mobile environments. A prototype Elvin '*proxy*' has been developed to store events persistently for disconnected subscribers and forward them upon subscriber's reconnection. The proxy model extends the standard Elvin system by introducing proxies which act as normal subscribers to the Elvin server and as a proxy server to subscribers. Subscribers thus connect directly to a proxy server rather than the Elvin server itself to receive their events. The proxy server maintains a permanent connection with the Elvin server and remains subscribed on the behalf of all subscribers. Any events forwarded by the Elvin server to the proxy server while the subscribers are disconnected are stored at the proxy side until the subscribers reconnect again. The Elvin system provides a quenching feature that allows message producers to obtain information about consumers' requests to limit their message generations to only messages of consumer interest. Elvin currently supports both local and wide area federation of Elvin servers so as to cooperate in routing events to subscribers.

The main disadvantage of this work is that the system capability is limited to supporting disconnected operation. Mobility between proxies has not been considered. If a mobile subscriber disconnects and migrates from its home proxy, it cannot reconnect to another proxy who is nearby. It is compulsory for the subscriber to reconnect to the same proxy each time. This potentially increases the network load and reduces the system performance.

STEAM (Scalable Timed Events And Mobility) [50] is a pub/sub notification service designed specifically for MANETs. It supports three different types of event filter: subject, proximity, and content filters. The combination of these filters is mainly aimed to address the problems related to highly mobile application components that communicate using wireless technology in an ad hoc mode. In STEAM, messages are only visible to the subscribers that are in the same geographical area of the publisher. In other words, STEAM offers a special form of location-aware publishing service, where information is expressed relative to the publisher location. The STEAM implementation is particularly tailored to MANETs and takes advantage of a proximity-based group communication service [51] that uses the number of hops traveled by messages at the MAC networking layer to approximate distance.

Subject and proximity filters are deployed on the producer side. The matching process adds extra load on the producer entity and may affect its performance. A single producer has to match the potentially large number of events for every subscribed consumer. Content filters are applied on the consumer side instead of deploying them on the producer side. This results in the distribution of the matching load from a single producer entity to a number of consumers. Hence, each consumer has to deal with a small number of content filters. This approach introduces extra additional overhead on the network link due to the propagation of unwanted events to consumers that are eventually discarded by the content filter. Failed and temporarily unavailable entities and connections are characteristics of wireless networks. A mechanism needs to be provided to ensure event delivery during such circumstances.

Huang and Garcia-Molina [22] have reviewed possible models that can be used to extend pub/sub systems to operate in a mobile wireless context. They present a basic model for the deployment of such systems for wireless environments and identify some of the issues that would arise during such a deployment. Huang and Garcia-Molina have also studied the problem of extending pub/sub systems to mobile ad hoc networks [52]. They proposed a greedy algorithm, SHOPPARENT, which constructs an optimal pub/sub tree for routing information from the source to all interested subscribers in a fully distributed fashion.

Although the discussed models provide a reasonable set of guidelines for extending pub/sub systems to wireless environments they are not aimed to address specific issues related to subscriber mobility. Instead, these models provide a rich abstraction for dealing with system failures, message loss, and disconnections. One of the issues in addressing these problems is finding a suitable evaluation environment, which is not defined by the authors. Regarding their work in wireless ad-hoc pub/sub systems, the proposed algorithm can deal only with certain aspects of mobility. It does not consider reliability and high mobility aspects. Their target operating environment is one with occasional reconfigurations, followed by periods of stability allowing the pub/sub tree to be constructed. Thus, nodes may occasionally miss their events, especially when they move constantly at high speeds. The authors assume in their algorithm that a node is willing and able to take on any number of children. This assumption may affect the performance of that node since too many events will pass through it. They also assume that only the root node can publish new events, which seems a limiting assumption.

2.5 Concluding Remarks

With the growing popularity of mobile devices, there is an urgent need to extend current pub/sub systems to mobile wireless settings. Although extensive work has been conducted on pub/sub systems for the fixed networks, comparatively few research has focused on these systems in the mobile wireless domains. In this chapter, we have reviewed a number of proposed solutions that aim to support subscriber mobility in pub/sub systems. We have classified the proposed mobility extensions based on the methodology of their solutions: *reactive*, *pro-active*, and *durable subscription-based*. We have also surveyed a number of representative pub/sub systems that do not support mobility. However, we believe that they offer some functionality that is relevant to the discussion presented in this chapter.

The main lessons we have learned from the reviewed work indicate that the proposed extensions to the pub/sub systems have achieved their goals with different degrees of success. However, several technological problems related to pub/sub mobility management support need to be investigated. We still miss effective strategies to design and implement an efficient mobility support which can guarantee that no events will be lost or duplicated when a subscriber hands off from one access point to the other. The current proposed extensions are not transparent to the application layer and are manually trigged by the subscribers. They are also limited in terms of their performance and efficiency. Mobility support in [11][32], for instance, relies on a central proxy that may become a performance bottleneck and induces significant network traffic due to potential triangular routing.

With respect to these issues, our work focuses on extending pub/sub systems to meet subscriber mobility aspects as a primary goal. The main objective of such an extension is not only supporting subscriber mobility in a transparent manner but that should also ensure high performance under high frequency of handoffs. We need also to evaluate the impact of the proposed solution on the system performance. To achieve this goal, we use the most widely accepted messaging system standard, Java Message Service (JMS) [12][13], as a base platform for our research activities.

As discussed previously, the pub/sub system can be implemented as a set of distributed brokers that are organized either in a hierarchical or a peer-to-peer topology. In such distributed models, performance can be strongly affected when the traffic needed for achieving mobility support or coordinating brokers becomes considerably high. In this respect, we argue that an appropriate strategy for reducing such traffic is a necessary condition to achieve the expected levels of performance. This strategy should be flexible and scalable to meet its objectives without affecting the existing features of a distributed notification service. To this extend, it is a critical issue to identify the most promising strategy that can minimize the overhead of network traffic without negatively impacting the system behavior.

Different from other proposed pro-active solutions, we make use of a data structure, called neighbor graph, which automatically captures the set of potential brokers that are most likely to be the next-hop broker of the mobile subscribers. In our approach, the subscriber context is also transferred prior to the subscriber movement (i.e., before the occurrence of disconnect operation) to support fast handoffs. Besides to handoff, our proposed pro-active approach supports a reconnect operation to the same broker, which may occur due to the loss of wireless connectivity. The propagated subscriptions are locally stored for potential future reuse and only deleted when it is necessary to reduce the propagation overhead. Our pro-active solution takes advantage of subscription similarity as it uses a shared buffer to cache a single copy of each message that matches similar interests of multiple subscribers. This can significant reduce the load on the brokers in terms of memory usage and processing time. Scenarios such as permanent and long interval disconnections, which may occur due to subscriber crashes/failures, are supported in our pro-active approach using a timeout interval that allows our approach to reclaim the buffer space and minimize the overhead of caching process. Our pro-active approach also takes into account the network latency as it caches/transfers events, published during subscription activation process, to the set of next potential brokers. This is necessary to avoid message loss that may occur due to high network latency. We have evaluated the mechanism of our pro-active approach in respect of multiple brokers and used variety of performance metrics.

C H A P T E R 3

A PRO-ACTIVE CONTEXT DISTRIBUTION APPROACH FOR SUPPORTING MOBILITY

3.1 Introduction

Little attention has been recently given to extend pub/sub systems to operate in mobile wireless environments. The most common choice for extending these systems is based on a *reactive approach* (i.e., context transfer occurs only after the mobile subscriber hands off to the new broker). This approach considerably increases the network load since the actual messages need to be transferred between the brokers to support subscriber mobility. It also imposes high handoff latency that may not be acceptable by many applications needing fast handoffs among brokers to maintain the quality of the communications. A different technique that has been proposed for mobility support is based on a *durable subscription-based* approach (i.e., every broker locally buffers all the published messages irrespective of its current active subscriptions). This may result in increasing the overall overhead of distributed brokers due to the costly buffering process, and thus degrade the system's performance. It also drains the buffer space quickly, resulting in a high message loss rate. Increasing the buffer size may reduce message loss, but on the other hand will increase message duplication and diminish the overall throughput.

In this chapter, we propose a novel and efficient approach to extend pub/sub middleware systems to support subscriber mobility and to provide fast handoffs. The core idea of this approach is largely based on a mechanism which intelligently transfers/caches subscriber context (its actual subscriptions) one hop (broker) ahead of its current broker in a *pro-active* fashion (i.e., context transfer/caching occurs prior to the subscriber movement). As it is difficult to predict the subscriber's movement, we need to identify the *set* of potential next brokers without examining the brokers' topology and manually creating the set. In this regard, we introduce a data structure, called *neighbor graph*, which forms the basis for our pro-active approach as it dynamically identifies the candidate set of brokers to which subscriber context should be pro-actively transferred and cached. Each broker over time learns about its immediate neighbors; thus, only these neighbors will receive/cache the subscriber context prior to the occurrence of handoffs. We also present an analytical model that can be used to capture the messaging cost as it is of key interest to provide insights into the overhead imposed by different solutions (our proposed solution as well as the alternative solutions proposed in the pub/sub literature) to support subscriber mobility.

This chapter is organized as follows: Section 3.2 describes our pro-active approach in detail. Section 3.3 presents a data structure, called neighbor graph, which forms the basis for our approach. Section 3.4 presents some guidelines for overhead optimizations. Section 3.6 introduces an analytical model for messaging cost. Section 3.6 concludes this chapter.

3.2 Pro-Active Approach for Context Distribution

This section describes the pro-active context distribution approach that aims to cope with subscriber mobility at significantly minimized cost in terms of message loss/duplication, processing overhead, and handoff latency. Our proposed approach achieves its objectives by transparently managing subscriptions and incoming messages, both while the subscriber is disconnected or/and during its handoff process. Subscriptions in this approach can be either in *active* or *passive* modes. Active subscriptions are the ones used for matching the published messages to the subscribers' interests and only those matched messages can be either routed to the interested subscribers or cached for future use. Passive subscriptions on the other hand do not enforce any message routing or caching and simply ignore incoming messages. Initially, a subscriber submits an active subscription to a broker to consume its messages. Then a passive copy of this subscription is propagated to all immediate neighbor brokers. Once the subscriber disconnects from the original broker, the propagated copies of the subscription become active. When the subscriber reconnects to the broker, other propagated subscriptions become passive again.

The pro-active context distribution algorithm can be decomposed into the following three phases: 1) transfer(cache) subscriptions to(at) the neighbor brokers; 2) activate(deactivate) subscriptions and cache messages locally; 3) deliver messages and reset cache. Figure 3.1 presents the pseudocode for the algorithm. Although each broker in the system locally executes a copy of this algorithm, the provided description describes the algorithm from the viewpoint of the initial broker B_j. The following notation is used in the description:

- $\boldsymbol{S}$: denotes a subscriber who is potentially mobile.
- $\boldsymbol{B_j}$: denotes the initial hosted broker for subscriber S.
- $\boldsymbol{B_i}$: denotes the next-hop broker to B_j.
- **Neighbor($\boldsymbol{B_j}$)**: denotes the set of neighbor brokers of B_j.
- **Sub($\boldsymbol{S}$)**: denotes the subscriptions related to subscriber S.
- **Msgs($\boldsymbol{S}$)**: denotes the actual messages of subscriber S.
- **Context($\boldsymbol{S}$)**: denotes the information (subscriptions/messages) of S.
- **NBR_List($\boldsymbol{B_i}$)**: denotes the neighbor list of broker B_i.
- **Timeout($\boldsymbol{S}$)**: denotes a chosen time $T_{timeout}$ for managing the context of a disconnected subscriber S. When $T_{timeout}$ expires, subscriber context is garbage collected ($T_{timeout} \geq$ the average disconnection interval at all brokers).

Pro-active Context Distribution Algorithm – executed on broker (B_j)

```
Case 1: Initial connection
        IF subscriber $S$ connects to $B_j$ THEN
           FOR all $B_i \in$ Neighbor($B_j$) DO
             Forward Sub($S$) to $B_i$
           ENDFOR
        ENDIF
Case 2: Subscriber disconnects
        IF subscriber $S$ disconnects from $B_j$ THEN
           FOR all $B_i \in$ Neighbor($B_j$) DO
             Activate Sub($S$) stored at $B_i$
             Forward Msgs($S$) to $B_i$ stored during Sub($S$) activation
           ENDFOR
        ENDIF
Case 3: Subscriber reconnects
        IF subscriber $S$ reconnects to $B_j$ THEN
           FOR all $B_i \in$ Neighbor($B_j$) DO
             Deactivate Sub($S$) stored at $B_i$
           ENDFOR
        ENDIF
Case 4: Subscriber moves out to a peer broker
        IF subscriber $S$ hands off to $B_k$ from $B_j$ THEN
           FOR all $B_i \in$ Neighbor($B_j$) and $B_i \neq B_k$ DO
             IF $B_i \notin$ NBR_List($B_k$)THEN
                Delete Context($S$) stored at $B_i$
             ELSE
                Deactivate Sub($S$) stored at $B_i$
             ENDIF
           ENDFOR
        ENDIF
Case 5: Subscriber moves in from a peer broker
        IF subscriber $S$ hands of to $B_j$ from $B_k$ THEN
           IF Sub($S$) is not in $B_j$ buffer THEN
             Obtain Context($S$) stored at $B_i$
           ENDIF
           FOR all $B_i \in$ Neighbor($B_j$) DO
             IF $B_i \notin$ NBR_List($B_k$)THEN
                Forward Sub($S$) to $B_i$
             ENDIF
           ENDFOR
        ENDIF
Case 6: Subscriber unsubscribes
        IF subscriber $S$ unsubscribes from $B_j$ THEN
           FOR all $B_i \in$ Neighbor($B_j$) DO
             Delete Context($S$) stored at $B_i$
           ENDFOR
        ENDIF
Case 7: Subscriber context obtained
        IF $B_j$ obtains Context($S$) from neighbors THEN
           Buffer Context($S$) at $B_j$
        ENDIF
Case 8: Subscriber times out
        IF $B_j$ triggers Timeout($S$) THEN
           FOR all $B_i \in$ Neighbor($B_j$) DO
             Delete Context($S$) stored at $B_i$
           ENDFOR
        ENDIF
```

Figure 3.1: The pro-active context distribution algorithm

Our designed algorithm makes a few assumptions about the target pub/sub system. The algorithm assumes a set of dedicated brokers organized in a general graph (or *peer-to-peer*) topology to form a distributed communication service. The peer brokers (connected by wireline elements and have unique identification numbers) communicate directly with each other via reliable and authenticated TCP connections to exchange messages/subscriptions. Our algorithm also assumes the adoption of the flooding strategy to steer message dissemination to the brokers. This assumption is based on papers such as [8][28] that strongly recommend this strategy in highly mobile environments as mapping content-based pub/sub systems on top of IP Multicast (topic-based) results in an explosion of multicast group. Alternative strategies, adopted by Hermes [23], Gryphon [28], and SIENA [32][33], drastically increase the load on the root broker that potentially becomes a performance bottleneck. Security has not been addressed in conjunction with our proposed pro-active approach as it is out of the scope of this thesis. It is expected that previously proposed techniques can be used also for this context. An overview of generic security techniques proposed for the pub/sub systems was introduced in [53][54]. We assume that there are no failures at the pub/sub routing layer (broker nodes and their links) since the development of fault-tolerant pub/sub protocols [55] is out of the scope of this thesis. However, we later provide a general guideline on how the proposed algorithm can cope with the failures in the broker network. Both permanent and temporary mobile subscriber crashes are allowed. Such failures are considered as special cases of disconnect operations that can be masked by the proposed algorithm. This algorithm is also based on the assumption that the brokers are distributed in the same geographical neighborhood region and subscribers may reconnect to the closest broker to their current locations. This assumption is essential to support emerging location dependent services (i.e., messages forwarded to the mobile subscribers depend on its current location) and to reduce the network overheads. In this algorithm, we assume that the subscribers always keep their original subscriptions and do not change them while disconnect from or/and reconnect to the system. Our algorithm is only limited to manage subscriber mobility, and is not concerned about publisher mobility, which has been explored by others [15][16]. Unlike subscribers, there is no specific information that the publisher would miss during the period of its disconnection.

We next give a stepwise description of the proposed algorithm. Although it is intended to serve multiple mobile subscribers that are connected to the same broker, for simplicity, we only describe the algorithm from the viewpoint of an individual subscriber.

Case 1 starts when a subscriber *S* connects to a certain broker B_j in the system. Broker B_j sends a passive copy of the subscriber's subscription *Sub(S)* to each immediate neighbor *Neighbor*(B_j). Each neighbor B_i locally stores *Sub(S)*. In the meantime, broker B_j routes the published messages to subscriber *S* throughout its active subscription.

Case 2 starts when subscriber *S* temporarily disconnects from the network due to poor network connectivity or a handoff. If broker B_j does not receive a generic ping reply from *S*, after a certain time, it will consider *S* as temporarily disconnected and thus sends an activate request to each immediate neighbor B_i. Broker B_j also needs to forward stored messages, during the subscription's activation, to its neighbors. This is necessary to avoid message loss that may occur due to the activation latency. Following the activation of *Sub(S)*, the brokers B_i will locally buffer all the incoming messages that match *Sub(S)*. It should be noted that the ID of the last message consumed by *S* (for each subscription) is enclosed with the activation request and thus only the messages with higher IDs are stored.

Similarly, broker B_j keeps buffering the messages for *S* as S may reconnect to B_j again.

Case 3 starts when subscriber *S* reconnects to the same broker B_j. This results in sending a deactivate request from B_j to its neighbors B_i, informing them to deactivate *Sub(S)* (change subscription's mode to passive), end the caching process, and clean up their local buffers. In the meantime, broker B_j delivers all buffered messages to subscriber *S*.

Case 4 starts when subscriber *S* hands off to a peer broker B_k. Broker B_k informs B_j that *S* reconnected to it. Thus, broker B_j requests its neighbors B_i either to delete the context of *S* from their buffers or deactivate *Sub(S)*, excluding broker B_k. Broker B_k is excluded because *S* takes control over *Sub(S)* and uses them to consume its messages directly from B_k. Brokers B_k and B_j exchange the list of their neighbor graphs to reduce the overhead of context transfer. Throughout these lists, broker B_j can decide which *Sub(S)* should be deleted and which *Sub(S)* should be deactivated for later use by broker B_k. Similarly, broker B_k can identify which *Sub(S)* should be forwarded to its neighbors. Neighbor brokers typically exchange their lists whenever an edge is added or deleted from their lists.

Case 5 starts when subscriber *S* hands off to broker B_j from broker B_k. Broker B_j first checks if *Context(S)* is available in its buffer. If it is not found in the buffer, then broker B_j will inform broker B_k to send the subscriber context. This may occur in two different cases: 1) broker B_k is not a neighbor of broker B_j. Thus, broker B_j has no information about the subscriber *S*. 2) subscriber *S* is the first to visit broker B_j from its neighbor B_k. If *Context(S)* is found in the buffer of broker B_j, similar actions to case 1 will be performed.

Case 6 starts when subscriber *S* removes its subscription(s) from broker B_j. As a result, broker B_j requests its neighbors B_i to delete *Context(S)* from their buffers.

Case 7 starts when broker B_j receives the subscriber context from neighbors *Neighbor*(B_j). The subscriber context (messages/subscriptions) will be stored in a persistent buffer. Note that B_j cannot receive messages without having the corresponding subscriptions for them.

Case 8 starts when subscriber *S* disconnects from broker B_j for long interval. When the disconnected time reaches a timeout interval $T_{timeout}$, broker B_j informs its neighbors B_i to delete *Context(S)* from their buffers. This is a necessary task as buffering and managing the *Context(S)* can severely affect the broker performance.

Figure 3.2 depicts a simplified finite state machine (FSM) diagram that describes the dynamic behavior of a mobile subscriber as discussed previously (see Figure 3.1).

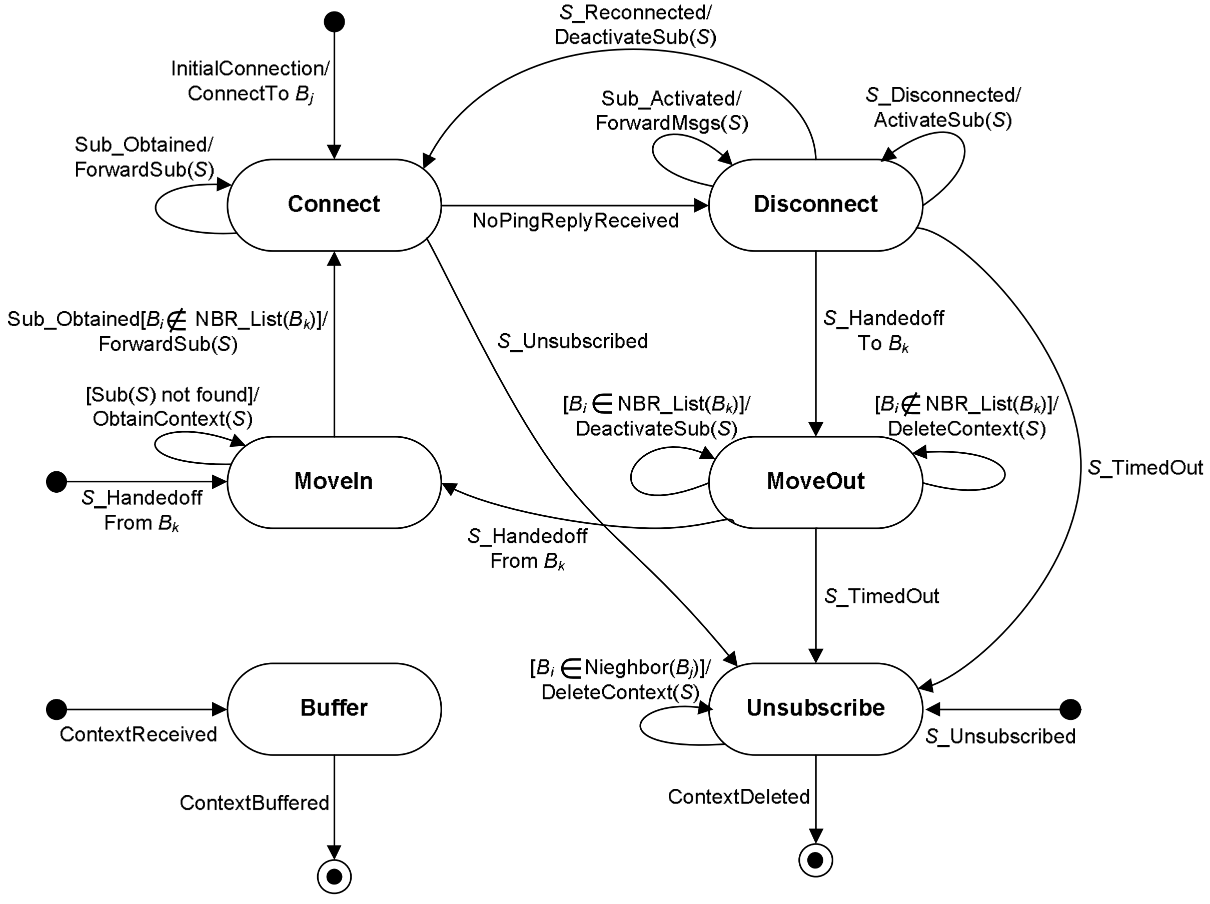

S_Reconnected/ DeactivateSub(S)
InitialConnection/ ConnectTo Bj
Sub_Activated/ ForwardMsgs(S)
S_Disconnected/ ActivateSub(S)
Sub_Obtained/ ForwardSub(S)
Connect
NoPingReplyReceived
Disconnect
Sub_Obtained[Bi ∉ NBR_List(Bk)]/ ForwardSub(S)
S_Unsubscribed
S_Handedoff To Bk
[Sub(S) not found]/ ObtainContext(S)
[Bi ∈ NBR_List(Bk)]/ DeactivateSub(S)
[Bi ∉ NBR_List(Bk)]/ DeleteContext(S)
S_TimedOut
MoveIn
MoveOut
S_Handedoff From Bk
S_Handedoff From Bk
S_TimedOut
[Bi ∈ Nieghbor(Bj)]/ DeleteContext(S)
Buffer
ContextReceived
Unsubscribe
S_Unsubscribed
ContextBuffered
ContextDeleted

Figure 3.2: FSM diagram for the pro-active context distribution algorithm

Next we describe several situations where concurrency can be an issue in the proposed pro-active context distribution algorithm and how the algorithm can cope with them. Although we avoid any concurrent access to the same subscription object(s) in our algorithm, there are several cases where such scenario may potentially occur due to network latency or/and delayed response of overloaded brokers, resulting in race condition issues. Here the race conditions appear when two concurrent operations, initiated by two different brokers, are intended to change the state of the same subscription(s) (e.g., activate, deactivate, or delete states). This can easily lead to an inconsistent state among the same subscription(s) copies stored at neighbor brokers, and hence impact the performance of the pro-active approach. Next we describe some scenarios where race conditions can possibly be raised. When a mobile subscriber S enters the *MoveOut* state, shown in Figure 3.2, broker B_j sends a deactivate request to the neighbor brokers B_i that are also neighbors of broker B_k. It may happen that subscriber S disconnects from B_k at the same time B_j issued the deactivate request. In this case, B_k also needs to send an activate request to its neighbors B_i. Due to the previously mentioned delays, concurrent operations (deactivate and activate) can be possibly applied on the same subscription(s) at neighbor brokers B_i to change their current state(s). This results in a race condition issue that may cause an inconsistent/inappropriate state among the subscriber subscription(s). Consider the case when the subscription(s) are activated to support the movement of subscriber S from B_k and then deactivated based on the request issued by B_j. Such inappropriate subscription state leads to the use of reactive method as a recovering mechanism to support subscriber mobility. To remedy the race condition issues, we have integrated the broker ID with the propagated subscription(s) to indicate which broker has the control to deactivate or delete the activated subscription(s).

The integrated ID is frequently updated using the activate request received from the most recent broker that hosted the moving subscriber. Looking at the previous scenario, B_j would not be allowed to deactivate the subscription(s) since B_k gained control of the subscription(s) just after performing the activate request. This keeps the subscription(s) in a consistent/appropriate state and eliminates the overhead of the reactive method. Similarly, race conditions may occur in the *MoveOut* state, when the subscriber *S* hands off to broker B_k. In this case, broker B_j needs to issue a delete request to remove subscriber subscription(s) from its neighbor brokers B_i (that are currently not broker's B_k neighbors) once it is informed by B_k about the subscriber's handoff. It may happen that one or more of those brokers become broker's B_k neighbors just after notifying B_j. B_k does not need to propagate a copy of the subscriber subscription(s) to the new added neighbors since it has previous knowledge (through exchanging neighbor graph lists) that there are previously propagated copies of the same subscription(s) at those neighbors. Thus, if the subscriber *S* disconnects from its current broker B_k and an activate request is sent by B_k to its neighbors, we may end up having a race condition issue due to the concurrent requests sent by B_k and B_j (activate and delete requests). The broker ID can help to control such situation. When the activate request is performed, broker B_k gains the control over the subscription(s) and thus the delete request is ignored. Note that in case the activate request could not find the subscription(s), B_k will be notified and asked to deliver the subscriber context prior to its arrival.

Similar race conditions may occur in the *unsubscribe* state shown in Figure 3.2. If a subscriber *S* hands off to broker B_k and just after the old broker B_j has been informed, the subscriber *S* has decided to unsubscribe from the system. Thus, the neighbor brokers of B_j

and B_k may receive concurrent requests (deactivate and delete). In this scenario, the delete request will fail if it gets executed before the deactivate request as broker B_j still gains the control over the active subscription(s). This leads to having a number of unnecessary subscriptions in the system. Note that the deactivate request disables the control attribute (broker ID) and hence subscription(s) can be removed. To manage this race condition, the delete operation here can have a special privilege to remove subscriptions without examining the control attribute of the subscriptions since the subscriber is leaving for good. A subscriber timing out is an alterative situation where a race condition may occur in the *unsubscribe* state. Consider the scenario when a subscriber S disconnects from broker B_j and then hands off to broker B_k. If B_j for some reasons has not been informed about the subscriber movement, it will send a timeout request to the neighbors once the timeout interval is reached, deleting the subscriber subscription(s). If subscriber S disconnects from B_k and at the same time B_j reaches timeout interval, we most probably end up with a race condition issue caused by concurrent requests (activate and delete). This race condition can be handled by using the control attribute (broker ID) as discussed earlier. Note that the delete request here does not have a special privilege to ignore the control attribute as with the previous scenario.

It is not an easy task to determine the concurrency issues as they are not always obvious. In the previous description, we provided some of the scenarios that we are aware of them. Others might be present but we believe they can be masked in similar way.

Although the pub/sub fault-tolerance is out of the scope of this thesis, we next provide a general guideline on how the proposed pro-active context distribution algorithm can cope with the failures in the broker network. As indicated previously, subscriber failures/crashes are considered as special cases of disconnect operations and hence will not be discussed here. Since we assume the links between peer brokers are reliable, we focus on broker failures and ignore other failures such as message loss. Two methods can be used to detect broker failures. The first method is based on generic ping (or *heart beat*) messages that neighbor brokers exchange. Each broker periodically sends ping messages to its immediate neighbor brokers. If a broker has not received a pig reply from a neighbor broker for a certain period of time, it assumes that the broker has failed. The second method of failure detection can be achieved during subscription propagation. If a broker cannot establish a TCP connection after a certain period of time, it assumes that the neighbor broker has failed. Although the connection between two brokers may fail, we assume here that the underlying network can fix the failed route before connection establishment times out.

When a broker detects that one of its neighbor broker has failed, it will remove the edge of that broker from its neighbor list and propagates the updated list to the rest of its neighbor brokers. As mentioned in *case 4* previously, neighbor brokers exchange their neighbor lists whenever a delete or add edge operation is performed on their lists. This prevents the broker from making the effort to propagate subscription(s) arrived after the failure detection to the failed broker(s). The sender broker also stops sending ping messages to the failed broker(s) after being detected, and accordingly reduces the load of periodically sending ping messages to the neighbor brokers. Each failed broker has to announce about its recovery to its previous neighbor brokers (stored in a persistent storage) as it completely recovers from its failure. In such a way, the previously failed brokers can be added again to the neighbor lists of their neighbor brokers and carry on with their dedicated tasks. The announcement messages also can be used to reconcile any conflict/inconsistent state that may occur among the subscriptions due to the failures. Here we assume that subscriptions are stored in a persistent database and can be accessed when the failed brokers recover. Each recovered broker needs to maintain separate lists for subscriptions received form its neighbor brokers using the unique broker ID enclosed with the subscription information. As discussed earlier, the broker ID indicates the broker who has the control attribute (such as deactivate or delete) over the subscriptions. These lists are propagated to the neighbor brokers as a part of the announcement messages. The receiver brokers (who receive these messages) decide how the subscriptions should be reconciled. This can be done by comparing the states of the received subscriptions with their identical copies stored at the receiver brokers and determining what operations (delete, activate, or deactivate) should be performed on the subscriptions stored at the recovered brokers. The recovered brokers may also need to take an action toward the subscriptions that were used by the connected subscribers before its failure. Such subscriptions need to be deleted from the recovered broker and its neighbors if the subscriptions' owners do not reconnect again. Also, if the failed broker activates some subscriptions before its failure, these subscriptions will enforce the message buffering process and cannot be timed out as the failed broker is the responsible for this action. Hence, it might be useful to allow the subscriptions propagated from the failed brokers to deactivate/delete themselves after a certain period of time. The main problems that arise in case of broker failure are the message loss and the overhead incurred by the use of the reactive method. Consider the case when a subscriber hands off to a recovered broker and the subscriber context has not been forwarded to this broker due to its failure. In this case, the reactive method is used to fetch the subscriber context. If the old broker has failed before the fetching process and other neighbors do not have a copy of the subscriber context, the subscriber will end up losing its messages.

3.3 Neighbor Graph

The neighbor graph (or *proximity graph*) is an efficient and well-known technique in many research fields (e.g., data mining, data compression, multimedia databases, and information retrieval) that typically make use of a neighbor graph for the purpose of neighborhood search. A neighbor graph is basically a geometrical structure that uses the concept of neighborhood to identify the closest items (e.g., points, nodes, objects, clusters) to another given item. Several works in connection with the notion of neighbor graph were found in the literature. In [56], the neighbor graph is utilized as a search structure to provide a faster searching method in high-dimensional data sets. In [57], the authors present a number of proximity searching algorithms using the k-nearest neighbor graph as the data structure for searching in metric spaces. In [58], an efficient method is proposed for updating a neighbor graph to improve searching in multi-dimensional spaces. In this thesis, we make use of the neighbor graph idea for a different purpose, identifying the mobility graph of mobile subscribers.

The effectiveness of our proposed pro-active caching approach largely depends on the successful approximation of the subscriber's movement between the distributed brokers. For a better chance of success, the broker B_j can approximate a set of potential brokers that are most likely to be the next-hop target of the mobile subscriber S. This approximation can be achieved, for instance, through observations of the mobility patterns of subscribers. We therefore make full use of a data structure, called *neighbor graph*, which provides the abstractions to achieve this goal. The neighboring relationship between distributed brokers can be represented by an undirected neighbor graph, which contains a number of edges (or *mobility paths*) that connect every broker to each of its neighbors. Hence, the neighbors of a given broker (or *vertex v*) in the graph correspond to the set of potential next brokers. A neighbor graph is undirected if the represented neighboring relationship among the brokers is reflective. In other words, if a mobile subscriber travels from broker *B1* to *B2* or vice versa, we then connect *B1* and *B2* with a single undirected edge. As the mobile subscribers in our experimental testbed are allowed to travel in a bidirectional way, we choose to use an undirected graph to represent the mobility paths between the brokers. A directed graph does not correctly reflect the mobility patterns (see Chapter 4) used in our experimental testbed. Under the assumption of bidirectional mobility, undirected edges allow to more pro-actively learn about neighboring relationships. For scenarios with strict unidirectional mobility patterns, a presentation based on directed edges may be more advantages, as this will reduce the size of the neighborhood and therefore the overhead imposed by our approach. The neighbor graph forms the basis for our pro-active approach and is used as a means for pre-loading the subscriber context (*subscriptions*) one hop (broker) ahead of its current broker prior to the movement of mobile subscriber.

3.3.1 Definitions

Neighboring relation: two brokers B_i and B_j can form a neighboring relation if it is possible for a subscriber *S* to reestablish its connection through a direct motion path between the physical locations of B_i and B_j. The neighboring relation between a set of distributed brokers forms the basis for the creation of the neighbor graph and depends on the distribution of the brokers in the network topology as discussed later.

Broker Neighboring Graph: an undirected graph G = (V, E), where V is the vertex set of all brokers, V = $\{B_1, B_2, ..., B_k\}$, and E is a set of unordered distinct pairs of edges, e =

$\{B_i, B_j\}$ where $B_i \neq B_j$. We say that vertices B_i and $B_j \in$ V have a neighboring relation if $\{B_i, B_j\} \in$ E. We thus define the set of all B_i neighbors in G as follows: *Neighbor*(B_i) = $\{B_{ik} : B_{ik} \in$ V, $(B_i, B_{ik}) \in$ E$\}$.

3.3.2 Constructing Neighbor Graph

The neighbor graph is utilized as a data structure for capturing the potential mobility graph of mobile subscribers. One way of constructing the graph is to allow individual subscribers to capture their own mobility graphs and offer them to the brokers upon their connections. Generating the neighbor graph in such a way has several drawbacks. Mobile subscribers presumably use portable devices (with limited capability in terms of CPU and memory) to interact with the distributed brokers in the backbone network. Hence, the task of capturing the mobility graph by subscribers adds additional load on these devices; leading to degrade their performance especially in a large-scale network where the size of the global neighbor graph is large. Each mobile subscriber needs repeatedly to submit its global graph upon its connection to the target broker. This may result in consuming a considerable amount of bandwidth and lead to congesting the wireless channel particularly with a large subscriber population. Brokers in the pro-active approach need to acquire knowledge about the local view of the complete graph (i.e., *subset* of neighbor brokers). Thus, forwarding the global view to the target brokers indeed adds substantial load on the system since some of the brokers captured in the global graph are not immediate neighbors to the target brokers. This leads to wasting the resources of some brokers that most likely will not be the next-hop brokers of the mobile subscribers. Also, every broker needs to separately deal with (e.g., store, search, and update) the graphs of individual subscribers, which may complicate and increase the overhead of processing the pro-active approach. Thus, we choose to allow individual brokers (typically runs on machines with high capabilities) to automatically build the neighbor graph that captures the local view of their immediate neighbor brokers. In addition, building subscriber-specific neighbor graphs will prevent subscribers to benefit from common movement patterns, which get reinforced as different subscribers migrate between specific brokers.

The neighbor graph can be constructed either in a static manner (i.e., manually created once and never changes over time) or in a dynamic manner (i.e., automatically generated and adaptively changes according to the mobility pattern). A static neighbor graph is problematic as it fails to adapt itself to the dynamic changes in the mobility pattern and/or broker topology. The neighbor graph also can be maintained either in a centralized manner (i.e., a single server stores the entire neighbor graph) or in a distributed manner (i.e., each broker stores a local view of the neighbor graph). A centralized neighbor graph has a scalability limitation and update difficulty. Thus, we considered a dynamic and distributed manner for generating the neighbor graph.

Several possibilities were proposed in the literature [56][57][58] for constructing neighbor graphs. One of the common techniques to various neighbor graph construction approaches is based on distances calculation between items. This is basically seeking for each item if the other items in the space are in its proximity. In this thesis, we adopt a different methodology for constructing the neighbor graph that mainly relies on subscriber mobility.

Two complementary methods can be applied by each broker to effectively learn the edges in the neighbor graph. The *first* method is to attach the address of the old broker with the reconnection request sent by the mobile subscriber to the new broker, thus establishing the neighboring relation between the two brokers. The *second* method is to use the request for context transfer received from another broker to establish the relationship. This request is usually received whenever the subscriber context is not present at that broker. Such a case may occur in two scenarios: 1) when the first subscriber moves through some motion path between two brokers; 2) when a subscriber disconnects (*voluntarily* or *involuntarily*) from the network and potentially moves to various locations to reconnect to any other broker in the coverage area. When applying these methods, some outlier edges (the ones that do not correctly model the neighboring relation) may be added to the graph. The neighbor graph may also hold some unused edges that are created through rarely used paths. The impact of the outlier and unused edges on the performance of our pro-active approach can be significant due to the additional overhead required to cache the subscriber context over time. It is thus essential to remove such edges from the graph over time. In this regard, a timestamp-based *Least Recently Used* (*LRU*) method is used to ensure the correctness and freshness of the graph. It is clear that the autonomous creation of the graph makes it self-adaptive to dynamism in the neighboring relation (e.g., adding/deleting brokers, changing network topology, changing user behavior, etc.). Each broker independently builds and locally stores a *subgraph* of the complete graph of all broker nodes. The following pseudocode is used to build the local view of the graph at each broker in a LRU manner. Here, we refer to the broker that executes the algorithm as $B_{current}$.

- *Receive a reconnection request*: If a mobile subscriber S moves to $B_{current}$ from B_i, $B_{current}$ induces the edge $\{B_i, B_{current}\}$ in its list of neighbors with a timestamp.
- *Receive a context transfer request*: If $B_{current}$ receives a context transfer request from B_i, it will add the edge $\{B_i, B_{current}\}$ to its list of neighbors with a timestamp.
- *Edge-removal*: If none of the above edge-addition operations is performed through a motion path between B_i and $B_{current}$ within a given time T, the stored edge $\{B_i, B_{current}\}$ will be removed from the list of neighbor graph. $T \geq$ the average frequency interval of edge-addition operations at all broker nodes.

Since the neighbor graph is initially an empty graph, the majority of handoffs, based on our creation algorithm, cause edge-insertion during the early age of the graph, thereby reducing its benefit in our proposed pro-active approach. Also, a mobile subscriber performing the first handoff along an edge, which is not in the graph, will miss its messages, as the graph fails to provide information about the potential next brokers. To avoid the cost of this period, the first mobile subscriber to cross over an edge will receive its context in *reactive* fashion. This will be gradually changed to *pro-active* fashion as the edges are added to the graph. To reduce the creation period, the process of edge-insertion occurs during the receipt of context transfer request by the old broker and reconnection request by the new broker. In our experiments, we have used this method to create the neighbor graph.

3.3.3 Overhead Optimization

Although the pro-active approach drastically reduces network traffic overhead (message transfer among the brokers), it introduces an additional overhead on the pub/sub network due to replicating subscriptions at different brokers. This overhead is mainly imposed by the caching process that is initiated whenever a mobile subscriber disconnects from its current broker and is linear with the number of disconnected subscribers as well as the length of their disconnection periods. This most likely results in degrading the overall performance and negatively affects the overall gain of this approach. Approximately, the total message overhead required to support a handoff operation is equal to the sum of two quantities: the cost of propagating subscriptions and caching messages. Equation 3.1 shows this relation.

$$\text{\textit{Total cost}} = \text{\textit{Propagation cost}} + \text{\textit{Caching cost}} \tag{3.1}$$

Equation 3.2 reflects the cost of propagating subscriptions, which is a linear function of the number of subscriptions (*K*) owned by a subscriber. For each subscription S_x, one copy of the subscription message is sent to *N* immediate neighboring brokers.

$$\textit{Propagation cost} = KN \tag{3.2}$$

Subscription covering [59] is a key technique to quench the subscription propagation, thereby minimizing the propagation overhead and optimizing the size of the routing tables. This results in significant improvements in the routing performance. Given two subscriptions *S1* and *S2*, we say that *S1* covers *S2* (denoted $S1 \supseteq S2$) if and only if any message matching *S2* also matches *S1*. The mechanism of subscription covering is as follows: when a subscription S_x is received by a broker *B*, it is only necessary to propagate S_x to all *Neighbor(B)* if and only if broker *B* has not previously propagated another subscription $S_x^{'}$ that covers S_x. Equation 3.3 models this using a binary function *covering* that requires two arguments (S_x and *B*) and returns either 0 (subscription S_x is covered by some other subscription $S_x^{'}$ at broker *B* and thus obviating propagation) or 1 (subscription S_x is not covered at broker *B* and propagation is necessary).

$$\textit{Optimized Propagation cost} = \sum_{S=1}^{K} N - \sum_{B=1}^{N} Covering(S_x, B) \quad <= \mathrm{KN} \tag{3.3}$$

The overhead of caching messages is a function of the publication rate P_{rate}, the length of disconnection time D_{time}, and the number of neighboring brokers *N*. Equation 3.4 depicts this relation. To reduce this overhead, the broker's degree (the number of its outward edges) can be bounded by a fixed upper bound (*M*) or ignoring rarely used edges. The caching cost can be significantly reduced if we can perfectly predict the next location of the mobile subscriber.

$$Caching\ cost = \sum_{S=1}^{K} P_{rate} \times D_{time} \times (N - \sum_{B=1}^{N} Covering(S_x, B)) \tag{3.4}$$

We next define two metrics that model the accuracy/overhead of a neighbor graph. The *false negative* of a neighbor graph, denoted as F_{neg}, is the probability that a mobile subscriber performs a handoff along an edge *e* that is not in the neighbor graph list. This probability of miss reflects the overhead of *reactively* transferring the subscriber context between the old and new brokers. Equation 3.5 models the false negative probability that occurs due to the failure of the neighbor graph to provide information about the potential next brokers during a finite interval $[t-\Delta, t]$.

$$F_{neg}(t) = \frac{Number\quad of\quad edge\quad insertion\quad [t-\Delta, t]}{Number\quad of\quad handoff\quad [t-\Delta, t]} \tag{3.5}$$

In Equation 3.5, the numerator corresponds to the number of times we add new edges in the graph list during a finite period of time $[t-\Delta, t]$ at a broker, whereas the denominator represents the number of observed handoff operations at the same broker and within the same interval. If $F_{neg}(t) > 0$, this indicates that the first handoff performed along an edge *e* has not benefited from the use of the neighbor graph. This situation can be observed during the initial phase of building the neighbor graph and thus the miss of the neighbor graph during this phase is high (i.e., $F_{neg}(t) \approx 1$). However, the probability of miss decreases gradually and gets closer to zero as the system reaches its steady state. The miss of the complete mobility pattern may become equal to zero ($F_{neg}(t) = 0$) if and only if the neighbor graph is a *subset* of the mobility pattern during the entire interval of $[t-\Delta, t]$. To reduce the miss/overhead of the neighbor graph, we use the context transfer request along with the reconnection request to add edges in the graph as discussed earlier. This will speed up the creation of the graph and benefit future mobile subscribers that handoff along the previously missed edge from the graph.

The *false positive* of a neighbor graph, denoted as F_{pos}, is the probability that we wasted resources in using the neighbor graph during a finite period $[t-\Delta, t]$. The neighbor graph may hold a number of unnecessary edges that not just waste memory but trigger unnecessary actions, thereby degrading the performance of the proposed approach. These actions include subscription propagations and message caching that increase the overhead of the pro-active approach without gaining any benefit. Equation 3.6 reflects the time ratio in which unnecessary edges can reside in the graph to the sum of the residence time of all edges in the graph.

$$F_{pos}(t) = \frac{Number \quad of \quad edge \quad removal[t-\Delta, t] \ \times \ T}{Sum \quad of \quad residence \quad time \quad of \quad all \quad e \in E} \tag{3.6}$$

In Equation 3.6, the numerator is the total number of edge-removal during an interval $[t-\Delta, t]$ multiplied by an edge-removal timeout T while the denominator is the total time of the set of all edges $e \in E$ that reside in the neighbor graph during $[t-\Delta, t]$. If $F_{pos}(t) > 0$, this indicates that there are unnecessary edges in the neighbor graph. To avoid the overhead of unnecessary edges, the edge-removal timeout T should be carefully selected. A well chosen value of T is the one that can continuously make the neighbor graph reflecting the up-to-date mobility pattern.

3.4 Messaging Cost

In this section, we present an analytical model that can be used to capture the messaging cost as it is of key interest to understand the overhead imposed by different approaches to support subscriber mobility. A simplified but reasonable mobility model is considered for this goal, as depicted in Figure 3.3, where a mobile subscriber moves between four different states, S_0, S_1, S_2, S_3:

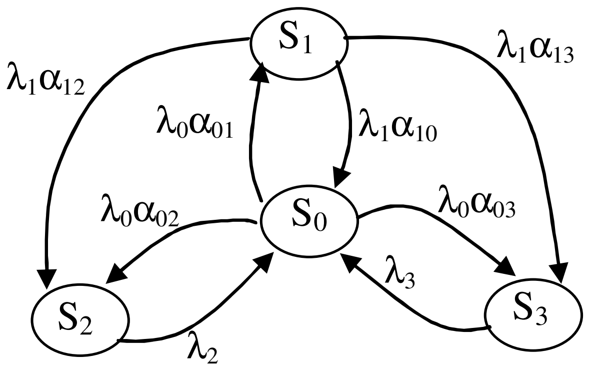


Figure 3.3: Subscriber mobility model

S₀: The subscriber connects to a broker.

S₁: The subscriber disconnects from its current broker.

S₂: The subscriber hands off to a broker that is not a neighbor of its original broker.

S₃: The subscriber hands off to a broker that is a neighbor of its original broker.

The residence time in each state S_i, $i = 0,1,2,3$ is an exponentially distributed random variable with parameter λ_i, $i = 0,1,2,3$. We further consider parameters α_{ij}, $i, j = 0,1,2,3$, which are defined as the probability of a mobile subscriber moving from the state *i* to state *j*. Thus, we have $\sum_{j=0}^{3} \alpha_{ij} = 1$. Note that $\alpha_{20} = \alpha_{30} = 1$.

Let $X(t) \in \{S_0, S_1, S_2, S_3\}$ denote a process that tracks the subscriber in the overlay network. $X(t)$ can be modeled as a continuous time Markov process. By sampling the random process $X(t)$ after time instance $t_k = k\tau$, where $\tau << (1/\lambda_i)$, for $i = 0,1,2,3$, the new sampled process $X(t_k)$ is a Markov chain [60] in the state space {0,1,2,3}, defined by the transition probability matrix:

$$P = \begin{bmatrix} 1-p_0 & \alpha_{01}p_0 & \alpha_{02}p_0 & \alpha_{03}p_0 \\ \alpha_{10}p_1 & 1-p_1 & \alpha_{12}p_1 & \alpha_{13}p_1 \\ p_2 & 0 & 1-p_2 & 0 \\ p_3 & 0 & 0 & 1-p_3 \end{bmatrix} \tag{3.7}$$

Where $p_i = 1 - e^{-\lambda_i \tau}$, for *i=0,1,2,3*. For a given network setup, with the mean residence time as $1/\lambda_i, \forall i$, and the roaming probability as $\alpha_{ij}, \forall i, \forall j$, the sampling time interval τ can be empirically obtained.

Messaging cost for managing the subscriber mobility is the metric of interest here, which is defined as the average cost per transition. This is also the average cost per round from *S₀* to *S₀*, because all the transitions leaving state *S₀* will come back to *S₀*, which can be computed as:

$$C = \sum_{\forall i} \pi_i \sum_{\forall j} \alpha_{ij} \theta_{ij} \tag{3.8}$$

Where π_i, which is also written as either π_i^r, π_i^p or π_i^d to represent the stationary distribution of the state *i* for the reactive, pro-active and durable-based schemes, respectively. θ_{ij} denotes the messaging cost for moving from state *i* to state *j*.

Case a: In the reactive mobility management scheme, there are only messaging costs when a mobile subscriber moves from state S_2 to S_0 and from S_3 to S_0. Denoting the cost for the transmission of all the buffered messages from an old broker to a new one as ξ_{m_1} and the control messages required to establish the transmission between the two as ξ_{s_1}, we have $\theta_{20} = \theta_{30} = \xi_{m\_1} + \xi_{s\_1}$. In general, ξ_{m_1} is much bigger than ξ_{s_1}.

Case b: In the pro-active mobility management scheme, messaging costs are given in matrix (3.9).

$$\Theta = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \xi_{s\_2} + \xi_n & 0 & \xi_b & \xi_b \\ \xi_{s\_2} & 0 & 0 & 0 \\ \xi_{m\_2} + \xi_{s\_2} & 0 & 0 & 0 \end{bmatrix} \tag{3.9}$$

Here ξ_n, ξ_{m_2} and ξ_{s_2} respectively represent the cost of buffering messages for the moving subscriber at the neighbor brokers, fetching messages from the old broker when the subscriber hands off to a broker that is not a neighbor of its previous broker, and sending control messages used to initiate the two previous process. The cost of ξ_n can be almost trivial in some cases. If the broker that needs to buffer the published messages for a mobile subscriber *S* who is currently disconnected from the network happens to serve other subscribers with similar interest to the same messages of *S*, the cost of ξ_n will virtually be zero. We will evaluate these cost values in more details later. When the subscriber hands off to a non-neighbor broker (moves from the state S_3 to S_0), the pro-active scheme fails to support subscriber mobility and a reactive mechanism is used instead to fetch subscriber messages from the old broker. The cost of ξ_{m_2} may in fact be different from ξ_{m_1} as the messages buffered at one of the neighboring brokers, which is closer to the current broker than the subscriber's old broker, can be fetched. In general, ξ_{s_2} is similar to ξ_{s_1}, and are much smaller than the other messaging costs: ξ_{m_1}, ξ_{m_2}, and ξ_n.

Case c: For the durable-based scheme, the messaging cost for any state transition is consistent, denoted as $\theta_{ij} = \xi_{n\_3}$, where ξ_{n_3} represents the messaging cost for all the brokers in the network that store messages for the mobile subscriber. ξ_{n_3} increases when the number of brokers or their network grows.

From Equation (3.8), we obtain the average messaging cost for the reactive C_r, pro-active C_p, and durable-based C_d schemes as follows:

$$C_r = (\xi_{m\_1} + \xi_{s\_1})(\pi_2^r + \pi_3^r) \tag{3.10}$$

$$C_p = \xi_n \pi_1^p + \xi_{s\_2}(\pi_1^p \alpha_{10} + \pi_2^p + \pi_3^p) + \xi_{m\_2}\pi_3^p \tag{3.11}$$

$$C_d = \xi_{n\_3} \tag{3.12}$$

Assume $\xi_{m\_1} = \xi_{m\_2} = \xi_m$, which is often the worst case for the pro-active approach. Since ξ_{s_1} and ξ_{s_2} (the cost of control messages) are much smaller than ξ_n and ξ_m (the cost of buffering and retrieval messages), we neglect their costs in our analytical model. Then the pro-active scheme would incur less messaging cost than the reactive and durable-based schemes only when (3.13) and (3.14) are satisfied, respectively.

$$\xi_n < (\pi_2^r + \pi_3^r - \pi_3^p)\xi_m / \pi_1^p \tag{3.13}$$

$$\xi_n < \xi_{n\_3} / (\pi_1^p + (\xi_m / \xi_n)\pi_3^p) \tag{3.14}$$

Given $\Pi = \Pi P$, where $\Pi = [\pi_0 \quad \pi_1 \quad \pi_2 \quad \pi_3]$, and $\sum_{i=0}^{3} \pi_i = 1$, the steady state probability is computed as:

$$\begin{bmatrix} \pi_0 \\ \pi_1 \\ \pi_2 \\ \pi_3 \end{bmatrix} = \begin{bmatrix} p_1 p_2 p_3 / \varphi \\ \alpha_{01} p_0 p_2 p_3 / \varphi \\ (\alpha_{02} + \alpha_{01}\alpha_{12}) p_0 p_1 p_3 / \varphi \\ (\alpha_{03} + \alpha_{01}\alpha_{13}) p_0 p_1 p_2 / \varphi \end{bmatrix} \tag{3.15}$$

Where $\varphi = (\alpha_{03} + \alpha_{01}\alpha_{13}) p_0 p_1 p_2 + (\alpha_{02} + \alpha_{01}\alpha_{12}) p_0 p_1 p_3 + p_1 p_2 p_3 + \alpha_{01} p_0 p_2 p_3$

Therefore, we can use (3.13) and (3.14) to compute the upper bound ratios *R₁* and *R₂* for ξ_n / ξ_m and ξ_n / ξ_{n_3}, respectively, as follows:

$$R_1 = (\pi_2^r + \pi_3^r - \pi_3^p) / \pi_1^p \tag{3.16}$$

$$R_2 = 1 / (\pi_1^p + (\xi_m / \xi_n)\pi_3^p) \tag{3.17}$$

Only when the ratio of messaging cost $\xi_n / \xi_m < R_1$ and $\xi_n / \xi_{n\_3} < R_2$, the average messaging cost of the pro-active scheme will be smaller than the reactive and durable-based schemes, respectively.

Table 3.1: Parameter settings for messaging cost analysis

$1/\lambda_0$	{60, 120, 300, 420, 600} (seconds)					
$1/\lambda_1$	{10, 30, 60, 90, 300} (seconds)					
$1/\lambda_2$	0.05 seconds (for durable scheme), 0.32 seconds (for pro-active scheme) and 3.29 seconds (for reactive scheme), obtained from the testbed results					
$1/\lambda_3$	0.05 seconds (for durable scheme) and 3.29 seconds (for the other two schemes)					
τ	1ms ($\tau << (1/\lambda_i)$, for $i = 0,1,2,3$)					
	α_{01}	α_{02}	α_{03}	α_{10}	α_{12}	α_{13}
Case 1	0.1	0.7	0.2	0.4	0.4	0.2
Case 2	0.2	0.4	0.4	0.1	0.7	0.2
Case 3	0.2	0.7	0.1	0.2	0.7	0.1
Case 4	0.2	0.6	0.2	0.3	0.6	0.1
Case 5	0.4	0.5	0.1	0.3	0.5	0.2

Setting the parameters for the model as illustrated in Table 3.1, we plot the upper bound of ratio R_1 for $1/\lambda_0 = 600\,\text{sec}$, as show in Figure 3.4. Varying $1/\lambda_0$ does not notably change the result.

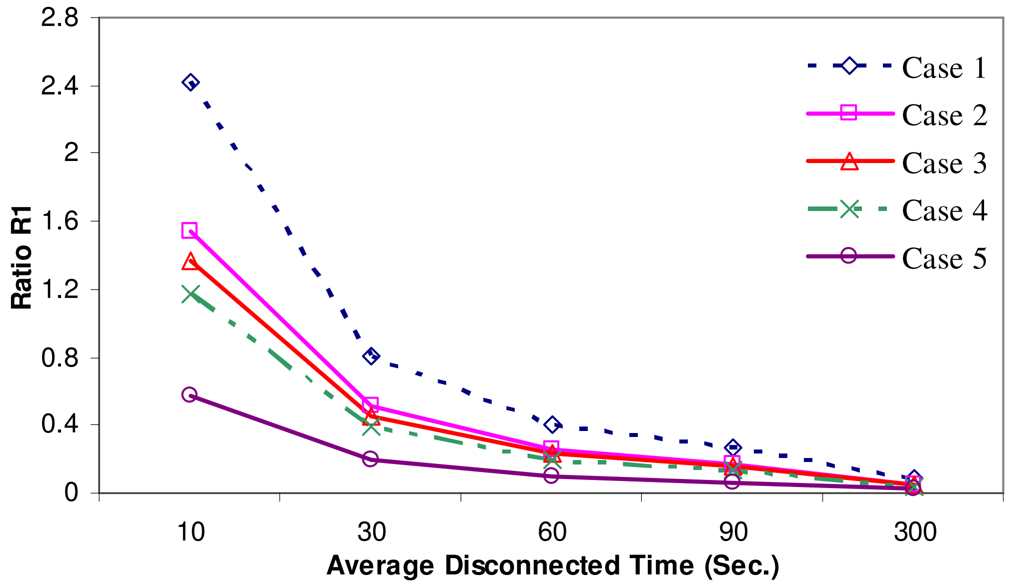


Figure 3.4: Upper bound for the messaging cost ratio R_1

It can be seen from Figure 3.4 that, for case 1, where the mobile subscriber disconnects from the broker network for an average of 30 seconds after staying connected for an average of 10 minutes, the pro-active scheme would have a smaller overall messaging cost than the reactive scheme if $\xi_n < 0.80\xi_m$; and for case 5, the pro-active scheme would incur smaller messaging overhead than the reactive scheme if $\xi_n < 0.19\xi_m$.

ξ_n is the cost incurred when the neighboring brokers of the original subscriber broker's start to buffer the messages for the moving subscriber as a pro-active action to support the subscriber movement to one of these brokers. If, on average, it takes N_1 more sends of each published message to complete this task, the bandwidth requirement of ξ_n as "send/sec" can be denoted as $\xi_n = N_1\lambda_1 M / \lambda_1$, where M is the average publishing rate of the subscribed messages. Similarly, $\xi_m = N_2\lambda_3 M / \lambda_1$ where N_2 is the average number of sends for each buffered message that needs to be fetched from the old broker to the new one. N_2 is at least 1. Taking $N_2 = 1$ to have a best case for the reactive scheme and a set of N_1, we have $\xi_n / \xi_m = N_1\lambda_1 / \lambda_3$. Referring to the parameters of $1/\lambda_1$ and $1/\lambda_3$, the resulting ξ_n / ξ_m is presented in Table 3.2.

Table 3.2: Estimated scheme messaging cost ratio

N_1	ξ_n / ξ_m for $1/\lambda_1$={10, 30, 60, 90, 300} (seconds)
1.3	{0.4277, 0.1426, 0.0713, 0.0475, 0.0143}
1.5	{0.4935, 0.1645, 0.0823, 0.0548, 0.0164}
1.7	{0.5593, 0.1864, 0.0932, 0.0621, 0.0186}
2	{0.6580, 0.2193, 0.1097, 0.0731, 0.0219}

For all the selected N_1 in Table 3.2, ξ_n / ξ_m is smaller than the upper bound *R_1* in cases 1, 2, 3, and 4 as depicted in Figure 3.4. For these cases, the pro-active scheme will have a lower overall messaging cost than the reactive scheme. If $N_1 \leq 1.7$, the pro-active scheme incurs lower overall messaging cost than the reactive scheme for all the listed cases.

Similarly, we plot the upper bound of ratio R_2 for $\xi_m/\xi_n = 1$ and $\xi_m/\xi_n = 0$, as show in Figures 3.5 and 3.6, respectively. $\xi_m/\xi_n = 1$ reflects the worse-case for the pro-active scheme as the messaging cost of fetching the buffered messages from the old broker is equal to the messaging cost of buffering messages at the neighboring brokers. $\xi_m/\xi_n = 0$ indicates the best-case for pro-active scheme as the mobile subscriber always moves between the neighboring brokers. Substituting these values in Equation 3.17, we obtain the ratio graph for the two cases using the parameters presented in Table 3.1. The plotted figures are obtained for $1/\lambda_0 = 60\,\text{sec}$ as larger values for $1/\lambda_0$ depicted much higher ratio for the durable-based scheme.

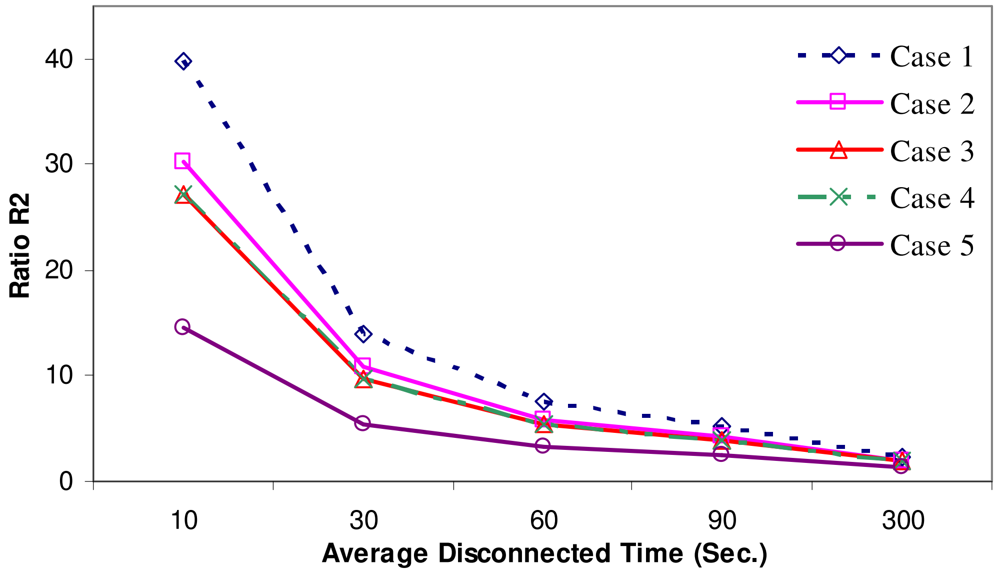


Figure 3.5: Upper bound for the messaging cost ratio R_2 with $\xi_m/\xi_n = 1$

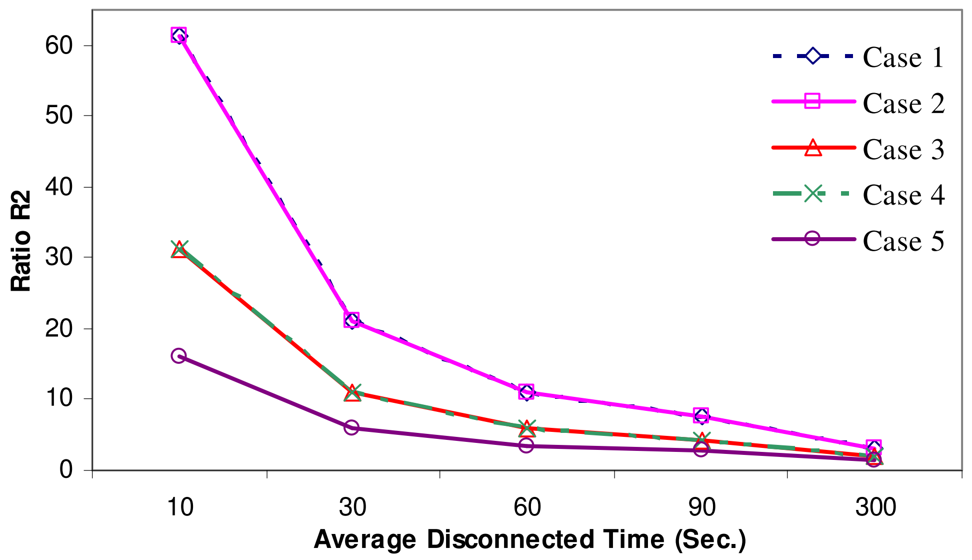


Figure 3.6: Upper bound for the messaging cost ratio *R_2* with $\xi_m / \xi_n = 0$

It can be seen from Figure 3.5 that, for case 1, where the mobile subscriber disconnects from the broker network for an average of 30 seconds after staying connected for an average of 1 minute, the pro-active scheme would have a smaller overall messaging cost than the durable-based scheme if $\xi_n < 13.95\xi_{n\_3}$; and for case 5, the pro-active scheme would incur smaller messaging cost than the durable-based scheme if $\xi_n < 5.51\xi_{n\_3}$. Similarly, Figure 3.6 shows that, for case 1, the pro-active scheme would have a smaller overall messaging cost than the durable-based scheme if $\xi_n < 21.10\xi_{n\_3}$; and for case 5, the pro-active scheme would incur smaller messaging cost than the durable-based scheme if $\xi_n < 6.02\xi_{n\_3}$.

For the durable-based scheme, one can denote $\xi_{n\_3} = N_3 M$ where N_3 is the number of extra sends to ensure that all brokers store each published message. Therefore, we have $\xi_n / \xi_{n\_3} = N1/N3$. We assume here that $N_3 < N_1$, which is a best case for the durable-based scheme. Taking $N_3 = 2$ and a set of N_1, $N_1 = 2.5,3,3.5,4$, the resulting ξ_n / ξ_{n_3} are as follows: 1.25, 1.5, 1.75, and 2. For all the selected N_1, ξ_n / ξ_{n_3} is smaller than the upper bound R_2 for $\xi_m / \xi_n = 1$ and $\xi_m / \xi_n = 0$, in almost all cases as depicted in Figure 3.5 and 3.6. For these cases, the pro-active scheme will have a lower overall messaging cost than the durable-based scheme.

The analytical model reveals the messaging cost properties in the different mobility management schemes, and shows that the pro-active scheme may potentially incur a smaller messaging cost than both the reactive and durable-based approach.

3.5 Concluding Remarks

In this chapter, we presented our proposal for extending mobility in pub/sub middleware systems. We first introduced the most common approaches that were proposed to support subscriber mobility. We then introduced our pro-active context distribution algorithm that aims to cope with subscriber mobility. This is achieved by propagating the subscriber context to all neighbor brokers ahead of the subscriber movement. We also presented the data structure, called *neighbor graph*, which forms the basis for our pro-active approach. We discussed the creation and maintenance of neighbor graphs and how the edges of the neighbor graph can be automatically learned. We analyzed the overhead imposed by the pro-active approach and the use of the neighbor graph and provided some optimization strategies that can reduce the overhead of our proposed approach. Finally, we presented an analytical model that can be used to capture the messaging cost of our proposed scheme and the alternative solutions found in the literature. The analytical results show that for a broad-range of parameters the pro-active scheme seems to incur a smaller messaging cost compared to the alternative solutions, which makes it favorable for supporting subscriber mobility. These results are promising enough to justify the actual implementation of the mobility support schemes, comparing them in an experimental testbed using a range of performance metrics.

CHAPTER 4

IMPLEMENTATION OF MOBILITY SUPPORT APPROACHES AND EXPERIMENTAL SETUP

4.1 Introduction

In this chapter, we provide a general overview of the Java Message Service (JMS) that forms the basis for implementing the mobility support solutions and present the prototype implementation of the *pro-active* and *reactive* approaches. We then describe the experimental setup used in this research to evaluate these approaches. The description provides some details about the different components in the setup including publishers, subscribers, and mobility patterns. It also motivates the selection of performance metrics and parameters used in this thesis.

This chapter is organized as follows: Section 4.2 provides a brief description of the Java Message Service (JMS) that forms the basis for our research activities. Section 4.3 presents the prototype implementation of the mobility support approaches. Section 4.4 illustrates the experimental setup. Section 4.5 summarizes this chapter.

4.2 Java Message Service (JMS): An Overview

This research is based on one of the most popular messaging system standards called Java Message Service (JMS) [12][13]. The motivations for selecting JMS as our base platform in this research come from surveying a set of representative pub/sub middleware systems. JMS is a collection of Java APIs that offer a common way for Java applications to create, send, and receive messages in a reliable manner. It supports several features that lend themselves well to the mobile, wireless environment. Next we briefly describe some of these features. Readers are refereed to [12][13] for detailed descriptions.

Although JMS supports *pub/sub* and *point-to-point* communication models, this research focuses only on the former. The pub/sub model is based on the use of *topics* to send and receive messages. Messages are delivered to topics by *publishers* and then consumed by all the registered *subscribers*. Subscribers may register to one or more topics and hence receive all the messages delivered to those topics. Subscribers that share the same topic will receive a copy of each message within that topic. They learn about the available topics through the Java Naming and Directory Interface (JNDI).

JMS adopts two subscription models, *nondurable* and *durable*. Nondurable subscriptions maintain low levels of reliability since the JMS broker does not keep records of inactive subscribers. Durable subscriptions on the other hand offer high levels of reliability at the cost of higher overhead. They instruct the JMS broker to retain all the messages for disconnected subscribers until they reconnect. This research considers the use of durable subscriptions as they naturally support disconnected operations in the mobile domains.

JMS provides two message consumption models, *synchronous* and *asynchronous*. The synchronous model is supported by explicitly invoking the *receive()* method. Once the method is invoked, the subscriber is blocked until the messages are received or the method timeout is reached. The asynchronous model is achieved by registering an event listener object with a subscriber. This object acts as an asynchronous event handler for messages and encapsulates only one method called *onMessage()*. This method contains the necessary action to be taken when the subscriber receives the messages. In this work, messages are consumed in asynchronous manner.

JMS supports content-based routing by the use of *message selectors*. With the help of message selectors, subscribers can receive a particular set of messages and thus optimize the bandwidth usage. A message selector is an object of type *String* that is used to hold conditional expressions. Subscribers can specify their message selectors as a part of their subscriptions' arguments. Hence, each message needs to be parsed and matched against the selection syntax before it can be routed. This work considers the use of this feature.

JMS messages are published either in *nonpersistent* or *persistent* mode. The nonpersistent mode maintains low levels of reliability as the JMS broker does not log the messages in a stable storage. The persistent mode on the other hand provides higher levels of reliability since all messages are logged in external storage until it is confirmed that they are consumed successfully. In this research, messages are published in a persistent mode.

JMS offers three acknowledgement modes, *DUPS_OK*, *AUTO*, and *CLIENT*. DUPS_OK minimizes the overhead on the system as it does not prevent message duplication. AUTO adds extra overhead to the system as it grantees that messages are delivered once-and-only-once. The system automatically acknowledges the receipt of a message as soon as it has been consumed. CLIENT is similar to the AUTO except that the acknowledgement has to be done manually. A subscriber acknowledges the message receipt by explicitly invoking the *acknowledge()* method. We have used the AUTO mode in this research.

4.3 Implementation

Several mobility management solutions have been proposed by the mobile computing community in the recent years. Although they share similar interests of location transparency, they are considerably different from each other due to their design and implementation choices. Existing mobility solutions can be generally classified based on their operating layers: *link layer*, *IP layer*, *transport layer,* and *application layer* [61]. As indicated in [61], most of the link-layer, IP-layer, and transport-layer solutions require considerable changes (upgrades) in the kernel-level of the mobile host's operating system and the network infrastructure. These are some of the primary reasons behind their limited acceptance and deployment. On the other hand, the application-layer mobility solutions overcome these drawbacks while providing an efficient way to make a mobile host always reachable for message dissemination. However, they incur considerable overhead in terms of the delay involved with the application level processing. Also, one of the main motivations behind the existence of the application-layer mobility solutions is the requirement for a finer level of granularity. The application-layer solutions can support host (*physical*) and code (or *logical*) mobility, whereas the lower-layer solutions are limited to host mobility. Placing mobility solutions at the application-layer can naturally provide a solution that can be used to solve the mobility management for vertical and horizontal handoff in heterogeneous wireless networks. It is also the only way to explicitly answer whether the application logic would benefit from properties such as durable/persistent messaging. In a pub/sub system, as the network backbone is composed of distributed application-level brokers, the application-layer is a natural choice to provide the most function of mobility management and minimize the changes of the end systems. Accordingly, in our work, we choose to build the proposed mobility management scheme at the application-layer so that the existing pub/sub systems can be easily extended to support subscriber mobility.

We next provide a high-level description of the prototype implementation of the *pro-active* and *reactive* approaches. Although many implementations of JMS [44][45][46][47][48] are available in the public and commercial domains, in this thesis, JavaSMQ [48] is selected as our base platform for our work activities. JavaSMQ is considered a robust, reliable, and scalable JMS implementation that achieves a competitive performance to existing pub/sub systems [12][13]. We implemented the core of both approaches in Java and extended the chosen JMS implementation accordingly. The implementation of the *durable subscription-based* approach is not described in this section since it is a built-in feature of all JMS implementations. Next we present the prototype implementation of the pro-active and reactive approaches.

4.3.1 Implementing the Pro-active Mobility Support

The core idea of implementing the *pro-active* mobility support is to replicate *dummy* (or *virtual*) subscription(s) that correspond to the moving subscriber at its prospective future locations (*brokers*). When the mobile subscriber moves out from its current location, the dummy subscriptions are activated at every neighboring location. This semantic allows the dummy subscriptions to listen and buffer messages of the moving subscriber prior to its arrival to the next potential location. Once the subscriber moves in to its new location, the buffered messages are delivered to it in an arranged order. Thus, a mobile subscriber is typically surrounded by a number of replicated dummy subscriptions that are intended to support its mobility to a particular location. Note that the replicated dummy subscriptions are frequently altered as the mobile subscriber moves in the real world. New dummy subscriptions are replicated at those locations that move within the subscriber's range and the old ones that are out of the range are garbage collected.

A prototype of the pro-active approach is implemented within an independent layer of *proxies* between the subscribers and their brokers. This layer is mainly responsible for replicating dummy subscriptions at the next future brokers to manage the messages of the moving subscribers. It also dynamically identifies the *set* of immediate brokers where these dummy subscriptions should be replicated. A single proxy process runs with each broker to manage subscriber mobility from one broker to the other. Note that the proxy layer is completely transparent to the brokers and the applications. The proxy process implements *broker* and *proxy* interface components that are respectively used to interact with the local broker process and the remote proxy processes through separate message queues. We have integrated a monitoring component with the broker process to track the subscribers' states (e.g., (re)connect, disconnect, handoff, and unsubscribe) as well as the ID of the last message consumed by the subscriber. The monitoring component regularly notifies a listening method (that is part of the *broker interface*) about the current state of each subscriber. A similar method (that is part of the *proxy interface*) is used to listen to the notifications coming from the remote proxy processes. Both methods start to listen as soon as the proxy process starts up. Every broker is configured to apply the Oldest Message Overwriting Policy when its buffer gets filled. Also, each subscriber has to keep track in a log file of the last broker to which it was connected. Figure 4.1 depicts the handoff using the pro-active approach. A stepwise description is given next to explain the operations performed during the handoff procedure.

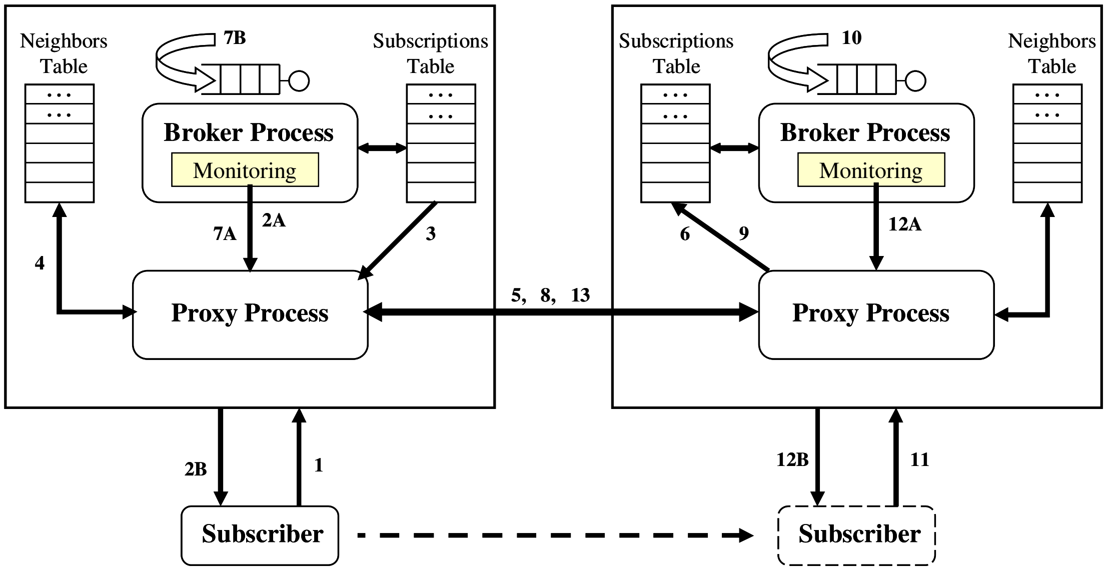


Figure 4.1: The handoff procedure with the pro-active approach

Steps 1 to 2B: When a mobile subscriber *connects* to a broker, the *monitoring* component checks the subscriber's state and reports *connect-state* to the proxy process via its *broker interface*. The monitoring component identifies this state by checking the value of the last visited broker enclosed with the reconnection request (*Null* means a new connection). In the meanwhile, messages are forwarded directly to the mobile subscriber based on its expressed interests.

Steps 3 and 4: The proxy subsequently retrieves a copy of the subscriber's subscriptions from the subscription's table and inspects the neighbor table to identify the set of neighboring brokers.

Steps 5 and 6: The proxy process, through its *proxy interface,* propagates a copy of the subscriptions to all immediate neighbors, and instructs its proxy peers to locally store the forwarded subscriptions and to set their modes to *inactive*.

Steps 7A to 10: If the broker process, after a certain time, does not receive a generic ping reply from the mobile subscriber, it will consider the subscriber as temporarily *disconnected* and accordingly the *monitoring* component will report *disconnect-state* along with the ID of the last consumed message to the proxy process. In the meanwhile, the broker process starts locally buffering messages for the disconnected subscriber. Then, the proxy process will requests its neighboring peers to *activate* the dummy subscriptions that correspond to the disconnected subscriber to buffer messages on its behalf. Following the activation request, the proxy keeps forwarding the stored messages to its neighbors until the receipt of this request is acknowledged by its peers. This is a necessary step to avoid message loss that may occur due to the activation latency. The ID of the last message consumed by the mobile subscriber (for each subscription) is enclosed with the activation request and therefore only the messages with higher IDs are stored at the next potential brokers to prevent message duplication.

Steps 11 to 12B: When the mobile subscriber *hands off* to a new broker, the *monitoring* component inspects the subscriber's state and reports *handoff-state* to the proxy process. This state is recognized by matching the current and previous broker addresses. Note that the previous broker address is sent along with the reconnection request submitted by the mobile subscriber. This is because each subscriber keeps track of the last visited broker, which is required in some cases: building the neighbor graph, applying reactive approach before the creation of the graph, and distinguishing the handoff from reconnect states. At the same time, the new broker delivers the buffered messages to the mobile subscriber in order.

Step 13: The proxy process informs the previous proxy peer that the mobile subscriber has just moved to a new broker and subsequently the previous proxy notifies its neighbor peers to *deactivate* the subscriber subscriptions and to delete its messages.

It is worth mentioning that mobile subscribers may (voluntarily/involuntarily) disconnect for a long period. This action may greatly affect the behavior of the neighboring brokers as they are required to buffer messages for disconnected subscribers, which might never show up again. Under this situation, the *monitoring* component tracks the disconnection period of each subscriber. If a timeout *T* is reached before obtaining a reconnection reply, a deletion request will be sent to all the proxy peers to delete the subscriber context. The timeout *T* should be equal to or greater than the average disconnection period at all brokers.

4.3.2 Implementing the Reactive Mobility Support

The implementation of the *reactive* mobility support is similar to the ones described in the literature [10][32][35] and shares as much code and data structures as possible with the pro-active mobility support. This provides us with a fair comparison and baseline indicator to evaluate the adequacy of our proposed approach. The main idea of the reactive approach involves uncoupling and retrieval of messages from the old broker that previously was serving the mobile subscriber. When the messages arrive at the new broker, they will be merged with the subscriber messages stored at the local buffer of the new broker, ordered, and delivered to the mobile subscriber.

A prototype of the reactive approach is implemented by running a *proxy process* with each broker entity to manage subscribers' mobility. Note that the proxy process is largely independent from the target broker. Every broker maintains a single buffer that is used to buffer the messages of all disconnected subscribers. To optimize buffer space usage, the Oldest Message Overwriting Policy is applied to control which messages should be overwritten when the buffer is filled up. Each mobile subscriber uses a *mobility service library* that is attached to the subscriber application. This library mediates the subscriber requests made to the target broker and is used to interact with the proxy process during the occurrence of connect/disconnect operations. When the mobile subscriber disconnects from its current broker, the proxy process creates a *proxy object* that takes control over the subscriber subscriptions at that broker in order to manage the subscriber messages. Once the subscriber reconnects and receives the buffered messages, the proxy object is garbage collected. Note that each subscriber keeps track in a log file of the last broker to which it was connected. This is required as the new broker needs to communicate with the old broker to fetch the subscriber messages. Figure 4.2 shows the handoff under the reactive approach. A stepwise description is given next to explain the operations performed during the handoff procedure.

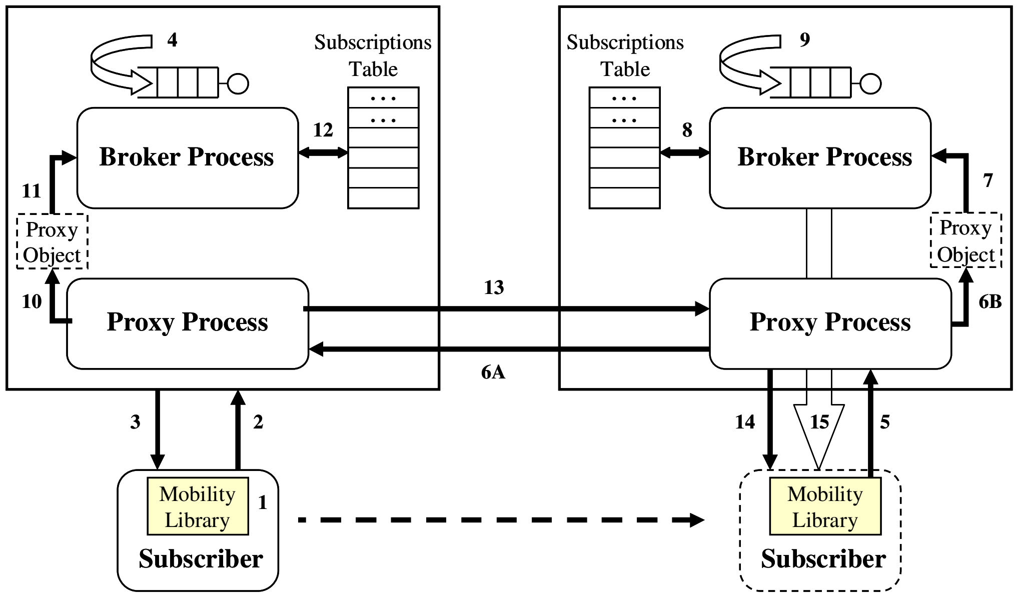


Figure 4.2: The handoff procedure with the reactive approach

Steps 1 to 3: When a mobile subscriber initially submits a *registration* request to a broker, the *mobility library* intercepts this request to get a copy of the subscriber's subscription(s) and to capture the address of the target broker. The mobility library uses a log file to store this information at the subscriber side. When the registration phase is completed, the subscriber receives its messages directly from the broker that it has registered with.

Step 4: As the mobile subscriber *disconnects* from its current broker, the *broker process* will notice this (i.e., through the use of generic ping reply) and accordingly starts buffering messages for the disconnected subscriber. As *durable subscriptions* are used to express the subscriber's interests, the proxy process is not involved in the buffering process.

Step 5: When the mobile subscriber *reconnects* to a new broker after some time, instead of connecting directly to the broker it first connects to the proxy process running at that broker. This connection is performed by the subscriber's *mobility library* that passes the subscriber's information (i.e., the address of the previous broker and the subscriber's subscriptions) along with the connect request.

Steps 6A to 9: The local proxy process, on behalf of the subscriber, requests the remote proxy process to forward all the buffered messages for that subscriber. Information about the subscriber's subscriptions is enclosed with this request. Meanwhile, the local proxy process creates a *proxy object* that subscribes with the local broker process using the received subscriptions. The broker process locally stores all the messages published during the message transfer procedure performed between the old and new brokers.

Steps 10 to 13: When the remote proxy is requested to fetch the subscriber messages, it creates a *proxy object* that takes control over the subscriber's subscriptions with the help of the forwarded information. This proxy object first deactivates the subscriptions to end the caching process and then starts forwarding the subscriber messages to the local proxy process. After all the messages are delivered to the local proxy, the proxy object deletes the subscriber's subscriptions as well as their corresponding messages.

Steps 14 and 15: The local proxy process merges the locally and remotely received messages and delivers them to the mobile subscriber in their publishing order. Finally, the local proxy process deletes the proxy object and reconnects the mobile subscriber to its new broker to directly consume messages from it. One point worth mentioning here is that the connections between the local and remote proxies are not permanent and only occur as the result of a handoff.

4.4 Experimental Setup

For our experimental study, we have selected JavaSMQ, a JMS-based pub/sub system, as our base platform (described in Section 4.3). We performed all our experiments on a dedicated network of ten Intel based Pentium 4 nodes running RedHat 9, inter-connected by a 100 Mbps switch. Six nodes were used for running six instances of the JMS broker with default configuration values. This work considers the distribution of these brokers in a simple in-building scenario as shown in Figure 4.3 (left-side). The dotted lines represent a potential path of motion whereas the square boxes show the placement of broker nodes. Figure 4.3 (right-side) shows the general view of experimental network environment. In our experimental setup, the topology of application-level messaging brokers $\{B1,\cdots,B6\}$ is modeled as an undirected general (or *peer-to-peer*) graph. Two brokers may communicate directly only if they are connected by an edge as shown in Figure 4.3.

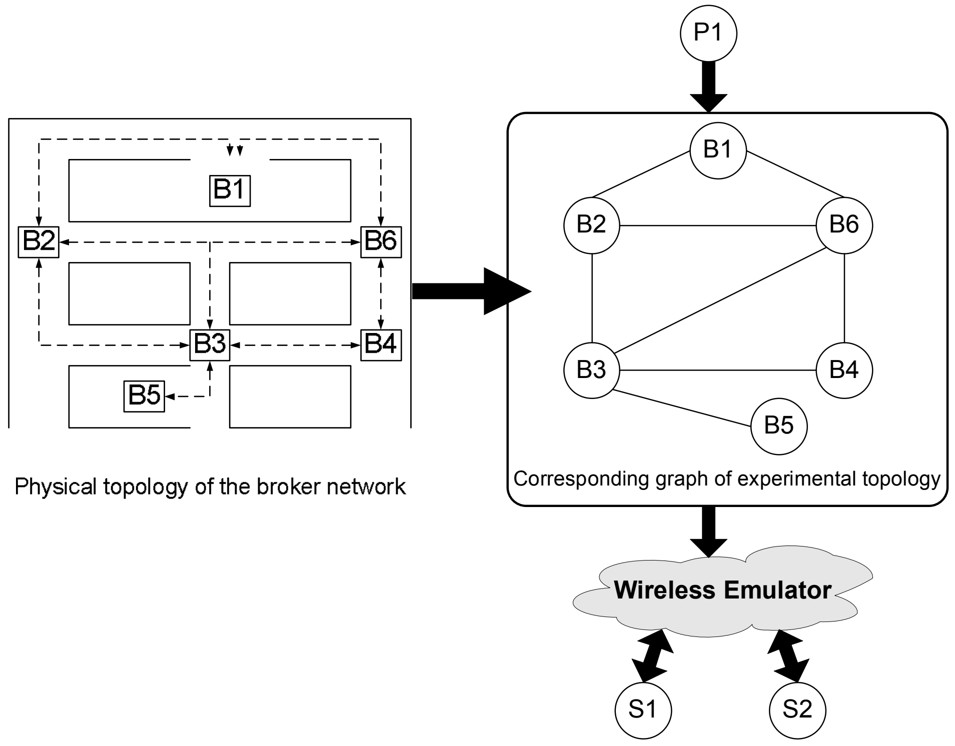

P1
B1
B2
B6
B3
B4
B5
Physical topology of the broker network
Corresponding graph of experimental topology
Wireless Emulator
S1
S2

Figure 4.3: A general view of experimental setup constituting the network under consideration.

A router node was used for running a wireless network emulator. One node was used for running a single, stationary message publisher P1. The remaining two nodes (S1 and S2) were used for running multiple subscribers. Subscribers that share the same machine run in separate threads and establish independent connections, but use the same Java Virtual Machine and JMS client library. The JVM used for running our experiments is Sun SDK 1.4.2, started with the options –Xms64m and –Xmx256m as a minimum and maximum heap size. Although this is a limited configuration, it is sufficient for the purpose of this research: evaluating different mobility support approaches.

A mobile subscriber in this setup represents an application running on a mobile terminal that transparently moves from one broker to the other. It initially registers with one of the six JMS brokers by the means of durable subscriptions. Through a mobility scenario, the mobile subscriber keeps randomly moving between the six distributed brokers during the course of the experiments. In our experiments, a maximum of two hundred subscriber threads were created and executed on two stationary machines. Subscribers are initially split evenly between the six distributed brokers. However, due to mobility, this number fluctuates over time, resulting in brokers serving a relatively large number of subscribers at times while at other times the broker may serve only a small number of subscribers.

In our setup, a single stationary publisher is used to inject messages in the broker network. Each generated message is assigned a single selector value ranging from 0 to 99. The selector values are randomly generated with uniform distribution. Similarly, each subscriber expresses its interest in receiving messages within a specific selector range that is also randomly, uniformly generated to be 1/5th of the total selector range.

All the communications between the subscribers and the brokers are tunneled through an emulated wireless channel that is created by using a network emulator called NistNet [62]. NistNet is a popular software tool that is implemented as a kernel module extension to the Linux operating system. It can be used to emulate various network environments. We used NistNet to model the characteristics of an IEEE 802.11 wireless LAN network based on a set of configuration parameters such as packet delay, packet loss, packet duplication, and network bandwidth. All these parameters were set to the most commonly used values characterizing IEEE 802.11 wireless LAN networks [63][64][65][66][67].

We developed two mobility patterns, *random* and *neighboring*, to gauge how the mobility support approaches react to dynamism in the wireless LAN environment. These patterns model the general behavior for a mobile subscriber that frequently moves around, but it also settles down for a period of time between each movement. This period should be long enough to allow the mobile subscribers to benefit from the use of mobility support approaches. Note that the considered patterns reflect a limited form of subscriber mobility. For example, we do not consider scenarios in which the subscribers move at certain speeds, to specific directions, or/and within predefined distances. The *random* pattern models subscribers that randomly select new target brokers for every move independently and uniformly over the set of all six brokers. In this pattern, we assume that subscribers are free to move directly to any broker placed in a particular geographical location. The *neighboring* pattern on the other hand reflects subscribers that move to the neighboring brokers (locations) every timeslot independently and uniformly over a set of neighboring brokers. In this pattern, restricted physical mobility is assumed due to obstructions and thus subscribers can only move to neighboring broker locations. The considered patterns reflect average subscriber behavior as the mobility parameters are selected uniformly and depict subscribers who experience short disconnection periods while roaming. Although our pro-active approach is particularly targeting neighboring mobility, we use the random mobility pattern to evaluate the behavior of our approach in the worst-case scenario. The random pattern introduces many outlier edges (i.e., ones that do not model an immediate neighbor) in the neighbor graph tables and thus incurs the overhead of using the reactive approach at the additional expense of wasting the neighbors' buffer space and processing time. Also, all the brokers in the random pattern have probabilistically the same maximum number of neighbors. This results in increasing the overhead of supporting disconnect operations at each broker.

In the considered mobility patterns, each subscriber alternates between three different mobility states: *connect*, *disconnect*, and *handoff*. While a subscriber is in the connect state, it can consume its messages from a uniformly selected broker. Each subscriber remains in this state for a randomly generated, exponentially distributed time with a mean of T_β seconds. With an equal probability, a subscriber either moves to disconnect or handoff state. The disconnect state reflects the case of signal breakdown due to poor network connectivity. A subscriber remains in this state for a randomly generated, exponentially distributed time with a mean of T_δ seconds. With a similar probability, the subscriber moves either back to the connect state and reconnects to the same broker or goes to the handoff state. The handoff state corresponds to the case when a subscriber moves out of the covered area of its previous broker. After staying in this state for a randomly generated, exponentially distributed time with a mean of T_α seconds, the subscriber moves back to the connect state and reconnects to a different broker. Subscriber mobility is described in more details in Chapter 6.

The reported results were captured from the measurement data obtained under different workloads. Each experiment was run for a duration that was long enough to reach a steady state. We ensured that the publisher and subscriber machines were not the bottlenecks in our experiments. We kept both CPU and memory utilizations at less than 65% and 38% respectively, thereby preventing publisher and subscriber bottlenecks from impacting the system performance. Each broker machine was fully dedicated to running a single instance of the JMS broker. Similarly, the CPU and memory utilizations of the broker machines were kept at less than 75% and 60% respectively in high overhead scenarios (maximum publication rate and/or large subscriber population) to prevent performance bottleneck. Before running any experiment, topic destinations and message stores were purged and reinitiated to start each test with a clean slate. Before executing each experiment, we make sure that the clocks of the publisher and subscriber machines are synchronized as it is required to calculate the end-to-end latency of message delivery. Network delays for establishing subscribers' connections are not included in our results.

4.4.1 Workload Parameters

- *Publishing rate*: the number of messages per second sent to the distributed brokers in a synchronized manner. To control the publishing rate, we used different sleep times (2, 1.5, 1, .5, and 0 seconds) between any two consecutive messages.

- *Number of subscribers*: the total number of subscribers that are served by the distributed brokers. We have varied the number of subscribers from 10 to 200, with initially evenly assigning subscribers to the six brokers.

- *Queue size*: the maximum buffer space in Mbytes that is used to temporarily store the received messages. Queue sizes of 1, 2, 3, and 4 Mbytes per-broker were used.

- *Message selector*: the selector pattern that expresses the subscriber's interest. As discussed earlier, the message selectors are randomly generated to be 1/5th of the total range of message selector values.

- *Network bandwidth*: the total available bandwidth that can be utilized to route messages from the brokers to the subscribers. Two values, 1Mbps and 11Mbps, were used to respectively reflect the effect of low and high wireless bandwidth environments.

- *Disconnect interval* (T_δ): the mean time interval during which the subscriber is disconnected from its current broker. Mean values of 12 and 24 seconds were used.

- *Connect interval* (T_β): the mean time interval during which the subscriber is connected to a particular broker. Mean values of 60 and 150 seconds were used.

- *Handoff interval* (T_α): the mean time interval during which the subscriber switches over to a different broker. Mean values of 1.5 and 3 seconds were used.

Table 4.1 summaries the workload parameters with their ranges and default values. Unless otherwise stated, experiments were conducted using the default values.

Table 4.1: The used workload parameters

Workload Parameters	Input Values	Default Values
Number of subscribers	10, 50, 100, 150, and 200	200
Sleep time	2, 1.5, 1, .5, and 0 seconds	0 seconds
Network bandwidth	1Mbps and 11Mbps	11Mbps
Queue size	1, 2, 3, and 4 Mbytes	1 Mbytes
Disconnect interval (T_δ)	12 and 24 seconds	12 seconds
Connect interval (T_β)	60 and 150 seconds	60 seconds
Handoff interval (T_α)	1.5 and 3 seconds	3 seconds
Selector pattern	1/5th of the total range	1/5th of the total range

4.4.2 Performance Measures

- *Subscriber throughput* (*Ts*): the total number of messages received per second. It is obtained by adding up the number of messages received by individual subscribers and dividing by the total duration of the experiment.

- *Broker throughput* (*Tb*): the total number of messages forwarded per second from individual brokers to a number of subscribers. It is obtained by adding up the number of messages forwarded by each broker and dividing by the total duration of the experiment.

- *Percentage of message loss* (*L*): the percentage of missed messages (due to buffer overflow and/or packet loss in wireless channel) by all the subscribers. It was obtained by calculating the difference between the total published and received messages and then dividing by the total published messages.

- *Percentage of message duplication* (*D*): the percentage of duplicated messages received by all the subscribers. It is obtained by dividing the total duplicated messages by the total received messages.
- *End-to-end latency (E)*: the time (in seconds) that it takes a message to travel from the publisher to the subscriber end. It is obtained by adding up the latency of each message and dividing the total by the total number of received messages.
- *Message processing time (M)*: the time that it takes the broker to process messages. It is obtained by adding up the processing time of each message and then dividing the total by the total number of forwarded messages.
- *Handoff latency (H):* the time (in seconds) between sending the reconnect request and receiving the first message of the subscriber at its new broker. It is obtained by adding up the latency of each handoff and dividing the total by the total number of handoffs.

4.5 Concluding Remarks

This chapter presented a general overview of some JMS features that are considered in the implementation of the mobility support approaches, referred to as *reactive* and *pro-active* approach. We also provided a high-level description of the prototype implementation of these approaches. The description includes details about the different components of each approach and their roles in achieving the ultimate objective: supporting subscriber mobility. The reactive mechanism involves message transfers from the old to the new broker during each handoff. The message transfers occur only after the mobile subscriber has reattached to the new broker. In contrast, the pro-active mechanism ensures that the subscriber context (mainly subscriptions) always remain one hop (broker) ahead of its current broker. Message transfer is not part of the pro-active mechanism since the virtual (proxy) subscribers are employed to buffer messages using the previously transferred context of the moving subscribers. We finally presented a detailed description of our experimental setup that is used to evaluate the mobility support approaches.

CHAPTER 5

PERFORMANCE EVALUATION OF MOBILITY SUPPORT APPROACHES

5.1 Introduction

This chapter provides a comprehensive experimental performance evaluation of the *pro-active* mobility support approach. To gain better insights into the effectiveness of our proposed approach, two mobility support approaches, *durable subscription-based* and *reactive*, are used as baseline solutions. We conducted an extensive number of experiments to explore the adequacy of our pro-active approach in supporting mobility and to compare its behavior to the state-of-the-art solutions. We studied the effect of subscriber mobility on message delivery using different workload parameters and with respect to several performance metrics. As described previously, the durable subscription-based approach requires each broker to continuously store all the published messages irrespective of its current active subscribers. The reactive approach involves message transfer from the old to the new broker after the handoff of a mobile subscriber. The pro-active approach ensures that subscriber messages always remain one hop (broker) ahead of its current broker.

This chapter is organized as follows: Section 5.2 presents and discusses the performance results of the mobility support approaches based on the random mobility pattern. Section 5.3 compares the performance of the mobility support approaches using the random and neighboring mobility patterns. Section 5.4 concludes this chapter.

5.2 Performance Evaluation and Comparison

In this section, we evaluate and compare the performance of the pro-active, reactive, and durable subscription-based approaches in terms of overall subscriber throughputs, end-to-end latency, handoff latency, message loss, and duplication. Unless otherwise stated, the results shown next were obtained using the random mobility pattern and the default input values listed in Table 4.1, Chapter 4. All the results presented next are the averages over 5 runs. To describe the reliability of the achieved results, we plot the 95% confidence interval on top of each data point.

5.2.1 Mobility Support Overhead

We evaluate the overhead of the *pro-active*, *reactive*, and *durable subscription-based* approaches in terms of two different metrics: the end-to-end latency (*E*) of the messages and the overall throughput (*Ts*) of the subscribers. These metrics provide a good indicator of the overhead incurred for supporting mobility from the prospective of end subscribers. To study this overhead under different workload conditions, we varied the total number of subscribers, served by the brokers, from 10 to 200 and observed the performance of the three mobility support approaches.

Figure 5.1 shows how *pro-active*, *reactive*, and *durable-based* approaches compare in terms of the end-to-end latency (*E*) and the overall throughput (*Ts*) with the increase of the subscriber population. From the graph, we can note that the *reactive* latency is by far the highest and proportionally increases with the most subscriber populations (50 to 200). A large portion of this latency is due to the message overhead (i.e., message transfer between the old and new brokers) imposed by the semantics of the *reactive* approach. During the handoffs, many messages may travel along one or more brokers before they reach their end destinations. This is because some of the subscribers may hand off prior to the completion of message transfer process. We observed that the message overhead accounts for almost (37% - 42%) of the total consumed messages (i.e., the percentage of messages consumed via state transfer protocol) in the *reactive* approach with 200 subscribers. This implies that networks designed with enough capacity to serve stationary subscribers will need to almost double their capacities. The *pro-active* approach experiences much lower latency than the *reactive* approach as it has almost no message overhead. This is mainly because almost all the messages are routed to their end destinations through their original brokers (i.e. brokers who received messages directly from the publisher). In other words, the traffic required to transfer messages to neighbor brokers in order to avoid message loss due to the latency of subscription activation request is much smaller than the traffic imposed by state transfer in the *reactive* approach. Finally, the *durable-based* approach shows the lowest latency as it has no message overhead. From the figure, the three approaches show a proportional relation between the end-to-end latency and most subscriber populations (50 to 200). This is an expected behavior as increasing the number of subscribers adds additional load on the system; thereby increasing the latency.

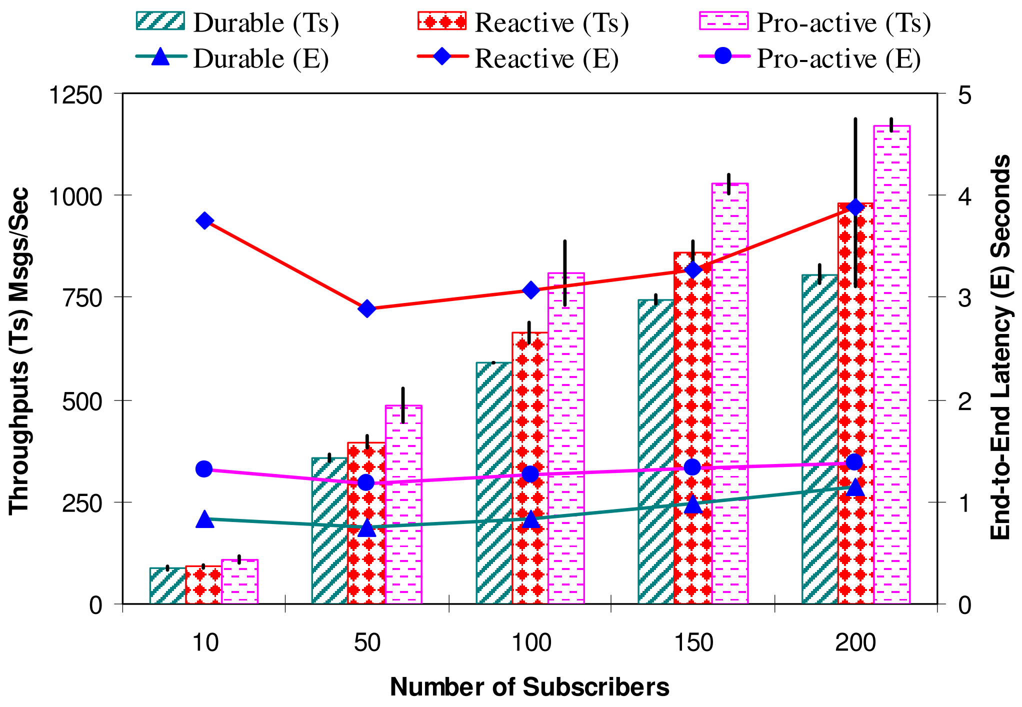


Figure 5.1: Mobility support overhead

The figure shows that the *durable-based* approach achieves the lowest throughput among the three. This is due to the overhead of the continuous message caching process adopted by this approach. *Pro-active*'s throughput, by transferring subscriber context ahead of its movement, is the highest, as it caches messages on-demand and almost has zero message overhead. The *reactive* approach falls in between the other approaches, as it has higher message overhead than the *pro-active* approach and lower message caching overhead compared to the *durable-based* approach. From the graph, we observe that the population of subscribers has a direct impact on the throughput of the three approaches. This can be noticed if we compare the reduction in the difference between any two successive data points of the *durable-based* bars. This indicates that the *durable-based* approach is more sensitive to the subscriber population among the three.

5.2.2 Handoff Latency

We evaluate the handoff latency (H) under different workload conditions. We define the handoff latency as the time, in seconds, between sending the reconnect request and receiving the first message of the corresponding subscriber at its new broker. Table 5.1 shows the averages handoff latency, in seconds, experienced by the three approaches with an increasing population of subscribers (from 10 to 200).

Table 5.1: Handoff latency of mobility support approaches

	Average handoff latency (seconds)		
Number of subscribers	Durable-based	Reactive	Pro-active
10	0.035	0.697	0.030
50	0.036	0.653	0.084
100	0.040	0.841	0.139
150	0.044	0.919	0.224
200	0.049	1.141	0.259

From the table, we can note that the handoff latency increases proportionally with the increase in the subscriber population. As expected, the subscriber population has a direct impact on the preparation time for the broker to route messages and hence increases the handoff latency. The above table also shows that the *reactive* approach experiences much higher latency compared to the *pro-active* and *durable-based* approaches. The reason for suffering this higher latency is that every handoff causes the entire messages for the moving subscriber to be read from the buffer, transferred to the new broker, merged with the messages in the new broker, and eventfully delivered to the subscriber. As a result, each subscriber has to wait for a while before it can receive the first message. On the other hand, the *pro-active* and *durable-based* approaches maintain much lower handoff latencies than the *reactive* approach since messages can be routed to the corresponding subscribers as soon as they join the new broker. The *pro-active* handoff latency is almost completely due to switching over *dummy* subscription(s) control to the mobile subscriber. When the mobile subscriber takes over the subscription(s), all neighbor brokers should be notified about the arrival of the mobile subscriber. Due to this overhead, the *durable-based* approach shows lower latency than the *pro-active* approach.

To provide better insights to the handoff latency of the three approaches under different workload conditions, the cumulative distribution graph of the handoff time observations is shown in Figures 5.2. From the figure, the *reactive* approach's latency is by far the highest among the three; almost 60% of the handoffs take more than 1.2 seconds. In contrast, almost 60% of the handoffs take less than .35 and .05 seconds with the *pro-active* and *durable-based* approaches, respectively. The observed results confirm that the *pro-active* and *durable-based* approaches can provide fast handoffs since the subscriber context is always ready at its new broker prior to its movement, but respectively require mobility prediction and large buffer space.

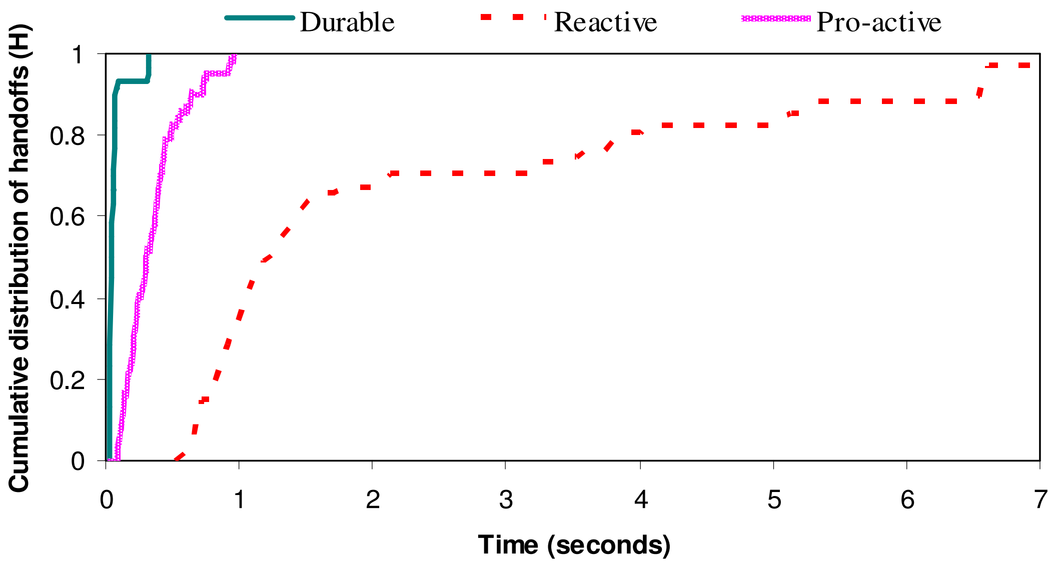


Figure 5.2: The cumulative distribution of handoff latency

5.2.3 Overall Performance

We evaluate the overall performance of the three mobility support approaches in terms of message loss, message duplication, and overall subscriber throughput. The results of these metrics are given as a function of publication rate, queue size, disconnection period, handoff frequency, and network bandwidth. This allows us to analyze conditions under which, if any, one approach can perform better than the others.

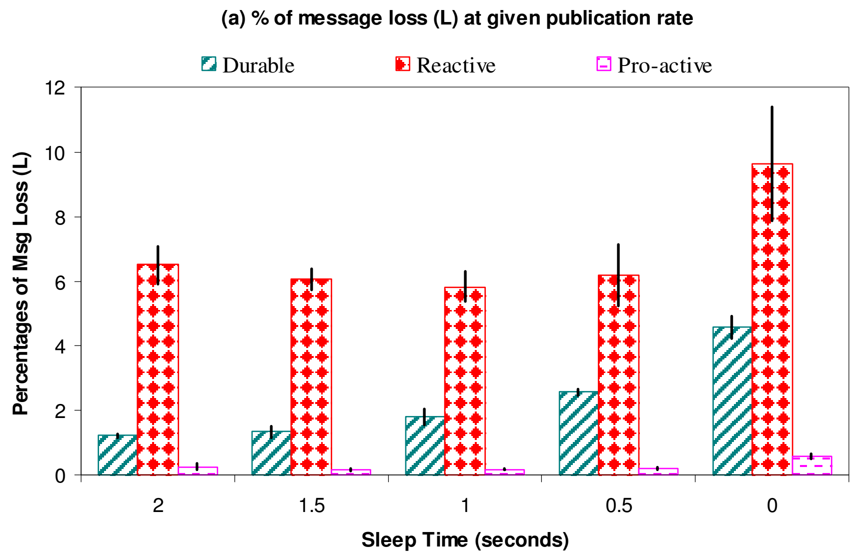
5.2.3.1 Overall performance at given publishing rates

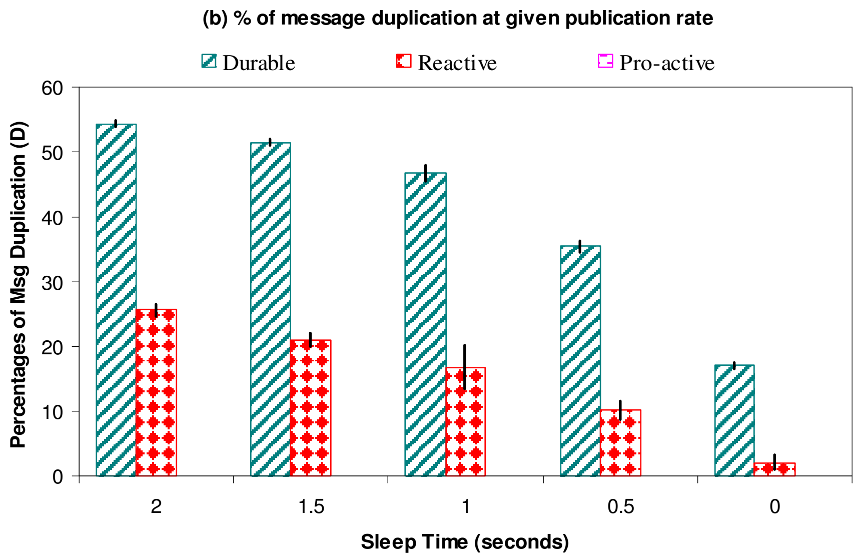
To study the effect of publication rate on the *pro-active*, *reactive*, and *durable-based* approaches, we varied the sleep time between any consecutive published messages (from 0 to 2 seconds).

Figure 5.3(a) depicts that, among the three approaches, the *reactive* approach shows the worst message loss (*L*) results that also increases with the maximum publication rate (0 sleep time). This is largely due to the overhead of message transfer between the old and new broker, which increases with maximizing the publication rate. Based on the disconnection period, the old broker may discard some messages from its buffer when it fills up. As a result, only a portion of the subscriber messages will be transferred to the new broker during the state transfer process. Furthermore, when the mobile subscriber moves to a new broker, it has to wait for a while (more than 1.2 seconds with possibility of 60%) to fetch its messages from the old broker. During this waiting time, the new broker keeps caching messages for the migrated subscriber. The caching time can be drastically increased at the new broker if the subscriber moves out prior to the arrival of its messages from the old broker. This time keeps increasing at each new broker until the messages reach the mobile subscriber. This results in a significant message loss as many messages will be discarded from the brokers' buffers due to space limitation. The message cost to forward stored messages is a function of the publication rate and disconnection period. Since the message cost increases with larger publication rates, more messages will be dropped from the buffers of the old and new broker. Among the remaining approaches, the *durable-based* approach depicts higher sensitivity to message loss and shows a linear relation with the publication rate. In this approach, message loss is fully attributed to the continuous caching of messages. This leads to overflowing the brokers' buffers and thus many messages will be discarded. Due to short disconnection periods, the *durable-based* approach depicts much lower message loss compared to the *reactive* approach. The *pro-active* approach, by storing messages on-demand (i.e., only when the subscriber disconnects) and transferring context early, shows lower message loss. Although this approach imposes extra costs in terms of wasting the neighbors' buffers and processing time, it pays off in terms of its overall performance.

From Figure 5.3(b), we note that the *durable-based* approach demonstrates the highest message duplication (D) among the three mobility support approaches. This is due to the fact that subscribers may consume identical messages from all previously visited brokers. As shown later, the *durable-based* approach is sensitive to message duplication with high frequency of handoff (default setting) since it increases the odds of consuming identical messages from different brokers. The *reactive* approach shows much lower percentages of message duplication than the *durable-based* approach as its semantics limit the odds of consuming identical messages between the old and new broker during state transfer. In addition to this, the *reactive* approach has low sensitive to message duplication with the high frequency of handoffs. The *pro-active* approach shows zero message duplication in all cases as it enforces all the neighbor brokers to cache only messages with an ID higher than the ID of the last message consumed by a disconnected subscriber.

Figure 5.3(c) shows that the *pro-active* has superior overall throughput, compared to the *durable-based* and *reactive* approaches. It is expected that the *pro-active* approach shows considerable improvement since it prevents message duplication and minimizes message loss. In the random mobility pattern, all the brokers have probabilistically the same number of neighbors. This results in increasing the cost of supporting disconnection operations at each broker. However, the overhead of this approach pays off well in various publication rates compared to the other approaches. From the same figure, we also observe that the *durable-based* and *reactive* approaches demonstrate comparable throughput results for the low publication rates. Low publication rates along with short disconnection periods result in little transfer, not incurring a very high overhead of state transfer performed at each broker in the *reactive* approach.





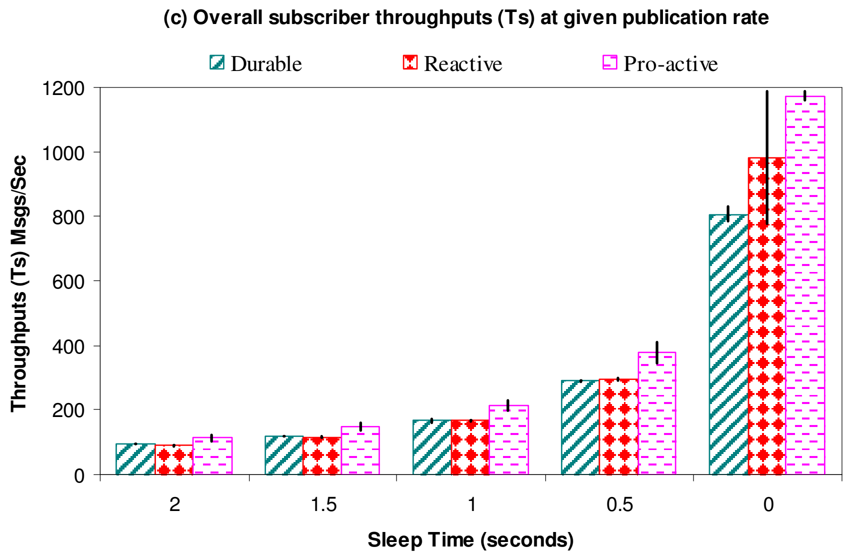


Figure 5.3: Overall performance at given publishing rates

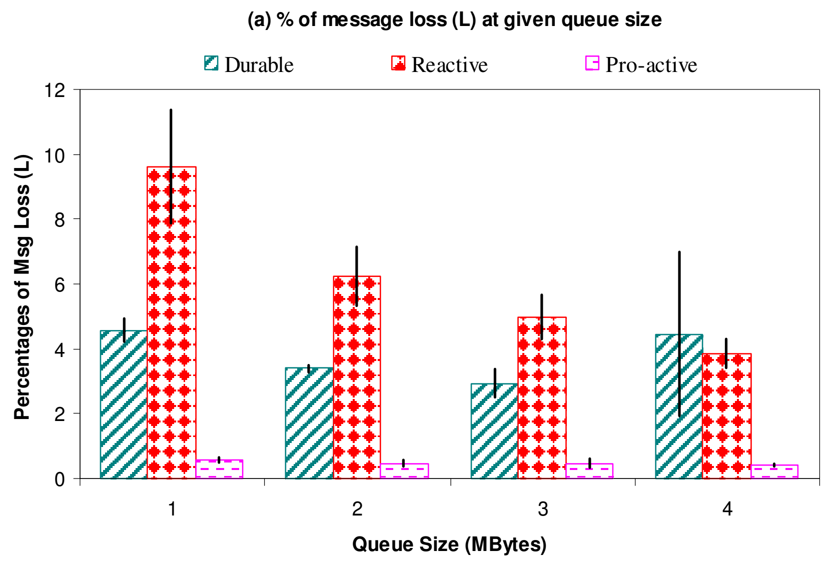
5.2.3.2 Overall performance at given queue sizes

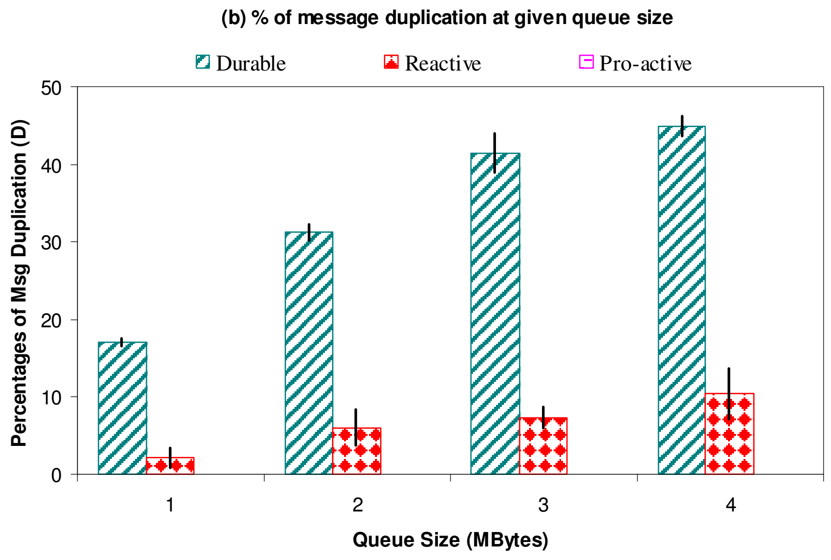
To observe the sensitivity of each mobility support approach to the available buffer space, we varied the queue size (from 1 Mbyte to 4 Mbyte) of each distributed broker in the network topology.

Figure 5.4(a) shows an inversely-proportional relationship between the queue size and the message loss rates in the three approaches. With an increase in the queue size, more messages can be accommodated and remain longer in the queue, accordingly decreasing message loss. This implies that the queue size has a direct impact on the system behavior and should be carefully selected. Increasing queue size beyond a threshold that provides zero message loss will only increase the overhead on the system without any reduction in message loss. The *reactive* approach demonstrates the worst message loss results among the three approaches due to the same reason described in Section 5.2.3.1. This indicates that the *reactive* approach is more sensitive to the queue size than the other approaches. We suspect that such sensitivity is due to the large residence interval of the messages in the queue during state transfer of migrated subscribers. Moreover, message loss may occur at the old and new brokers in every state transfer. The two remaining approaches illustrate much lower message loss with the selected queue sizes because messages are queued for much shorter durations as they are always ahead of their subscribers. In contrast to the *reactive* approach, only message loss occurred at the new broker does count during the handoffs. The *pro-active* approach demonstrates superior results in terms of message loss compared to the *durable-based* approach since it optimizes the buffer space usage by only caching messages on-demand. Therefore, the results of message loss form almost a constant relationship with the selected queue sizes.

As expected, a linear relationship is shown in Figure 5.4(b) between the queue size and the results of message duplication in the *reactive* and *durable-based* approaches. This is because large queues can store more messages and thus a larger percentage of duplicated messages can be received from multiple brokers. This clearly can be seen in the *durable-based* approach since identical messages can be consumed from all the previously visited brokers. Lower percentages are experienced in the *reactive* approach, where duplicated messages often occur between the old and new brokers. It can also be noted that the *pro-active* approach successfully eliminates message duplication in all cases.

From Figure 5.4(c), we note that the throughput of the *durable-based* and *reactive* approaches decreases gradually with the increase of the queue size. This is mainly due to the significant increase of messages duplication in both approaches with larger queue sizes. On the other hand, larger queues reduce message loss, which in turn increases the subscriber throughput. As a result, we see a slow decrease in the overall throughput of the two approaches. The *pro-active* approach shows the highest throughput results among the three approaches as it prevents message duplication and minimizes message loss. It shows almost constant throughput results with the selected queue sizes due to the elimination of message duplication. The *durable-based* approach performs the worst as it suffers from a high percentage of message duplication.





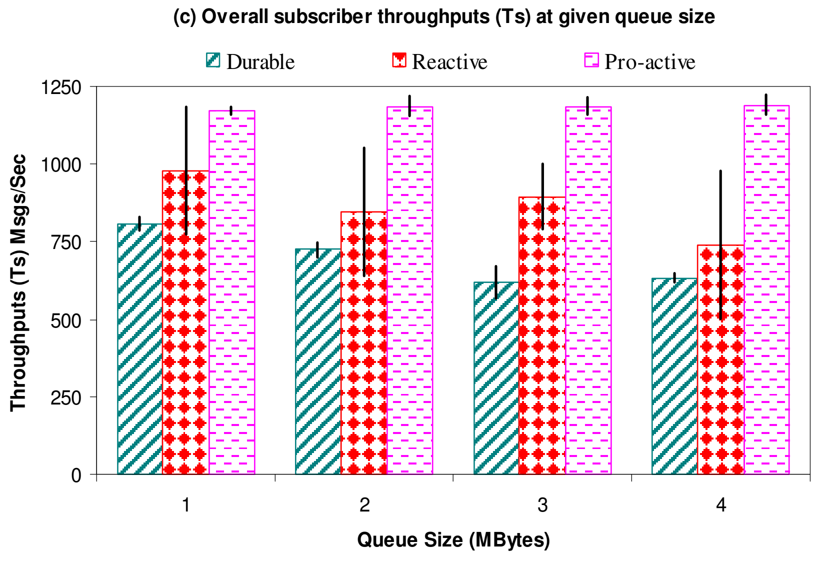
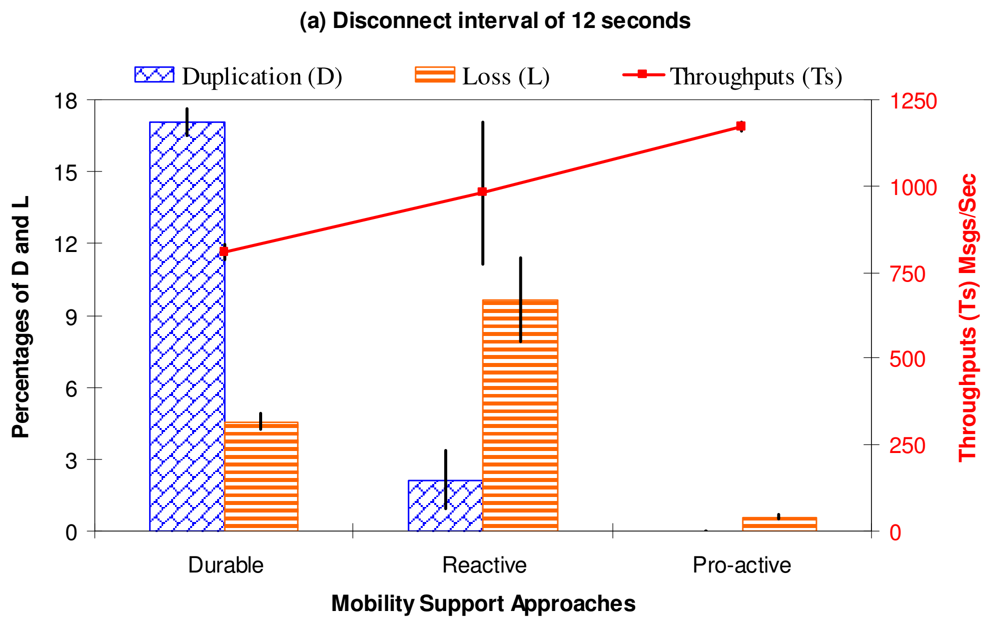


Figure 5.4: Overall performance at given queue sizes

5.2.3.3 Overall performance at given disconnection periods

To study the effect of the disconnect interval on the behavior of the *durable-based*, *reactive*, and *pro-active* approaches, we varied the mean (T_δ) of the disconnect time. Figure 5.5(a) and 5.5(b) respectively depict the reported results with means of 12 and 24 seconds.



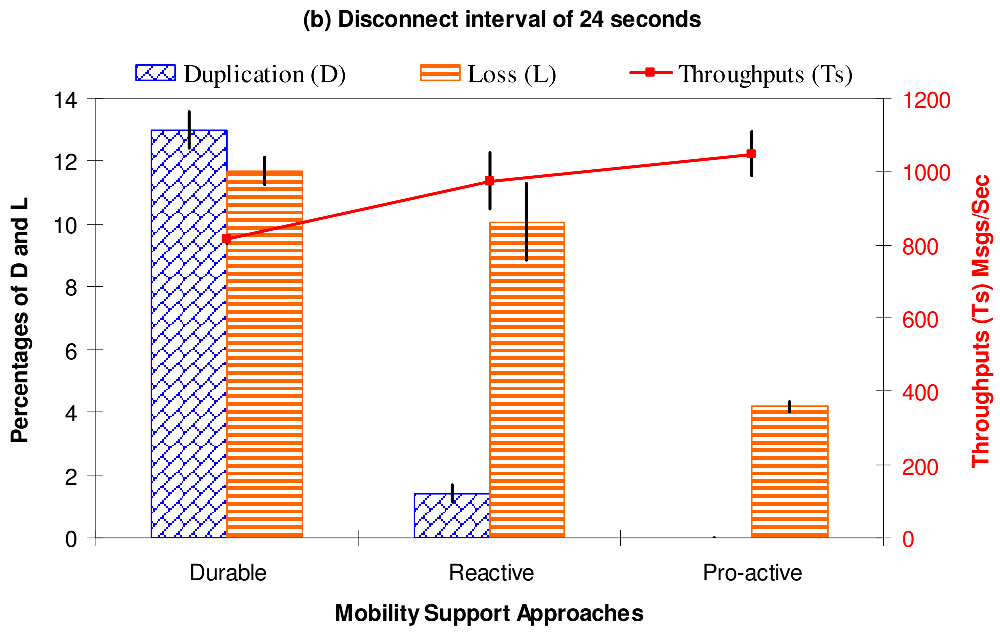
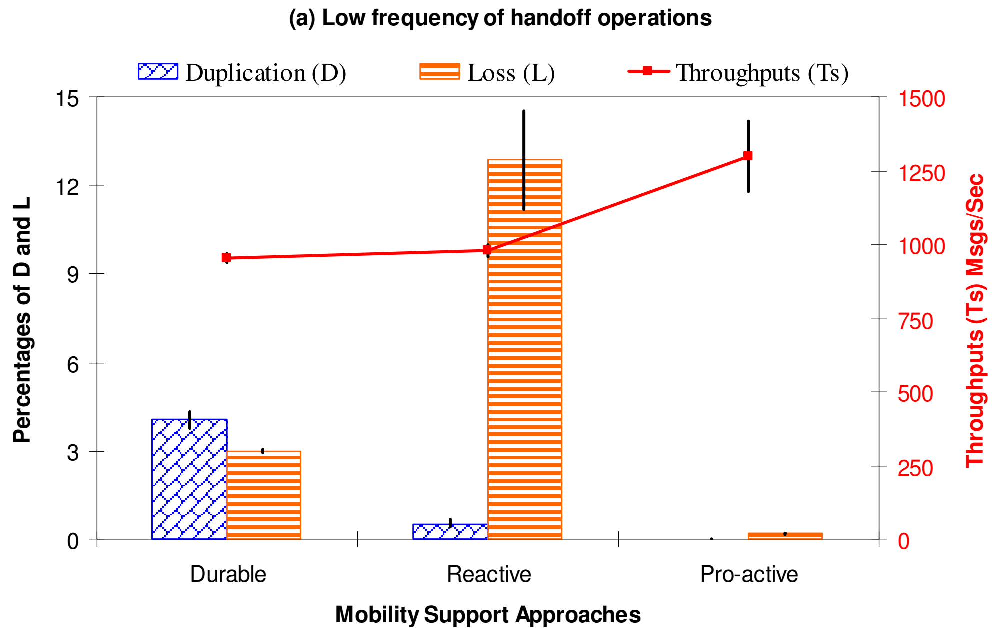


Figure 5.5: Overall performance at given disconnection periods

From the figures, we note that the three mobility support approaches show similar trends with respect to the overall throughput. With a larger disconnect interval, the throughput results tend to slightly decrease. This is attributed to the increase of the message loss rate. The percentage of message loss increases almost by a factor of three when doubling the mean disconnect time ($T_\delta = 24$) in the *durable-based* and *pro-active* approaches. This implies that these approaches are more sensitive to the disconnect interval than the *reactive* approach. This is because they rely on caching messages at multiple brokers in the network and thus quickly drain the buffer space. However, the *pro-active* approach shows lower sensitivity as it only buffers messages on-demand. We expect that the throughput of the *reactive* approach will be noticeably diminished with larger disconnect interval as more messages need to be transferred from the old to the new broker. As we found that a large portion of message loss occurs at the old broker, the overhead of state transfer is bounded by the buffer size, resulting in an insignificant decrease in the throughput results.

5.2.3.4 Overall performance at given frequency of handoffs

Figure 5.6(a) and 5.6(b) show the results of the three mobility approaches with low and high frequency of handoffs. We used the mean connect time of *60* and *150* seconds in the high and low handoff frequency levels, respectively. Hence, the number of handoffs is reduced by almost *40%* in the low frequency level compared to the high frequency level.



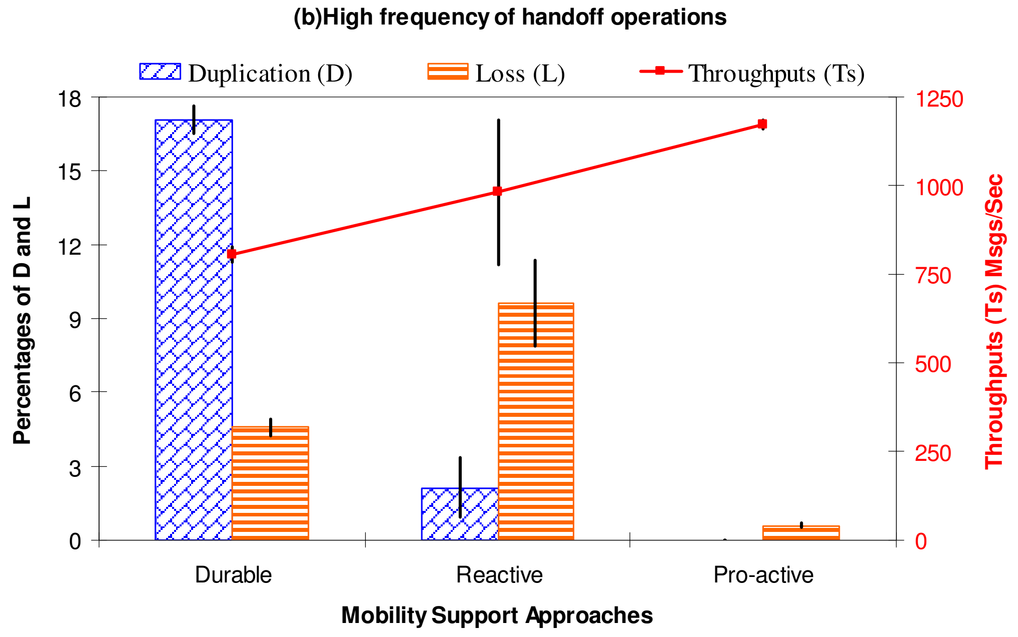


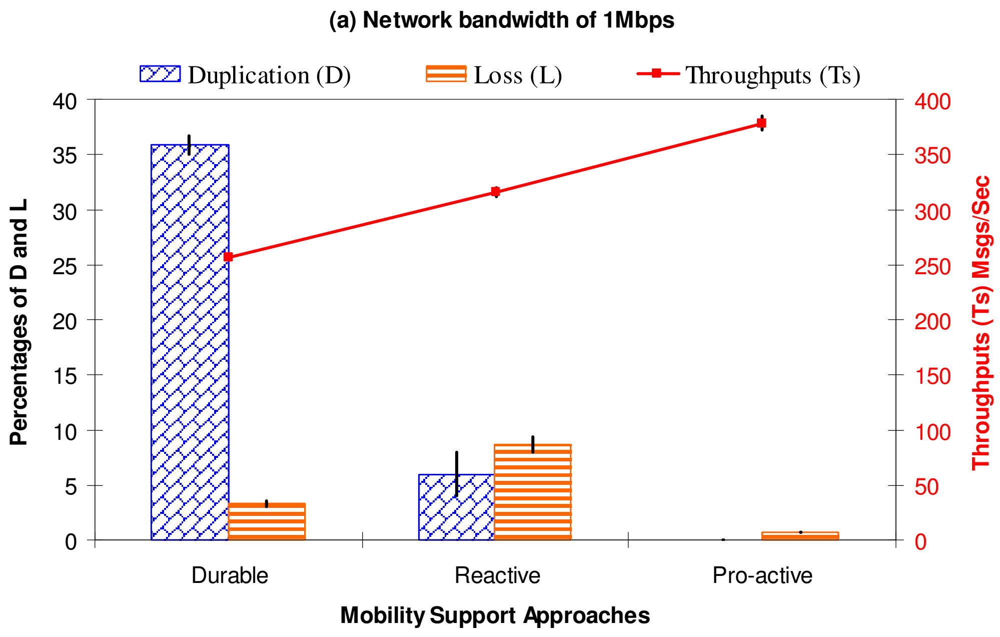
Figure 5.6: Overall performance at a given handoff frequency

From the figures, we note that the overall throughputs of the *pro-active* and *durable-based* approaches slightly increase with the low handoff frequency. This is a result of reducing the caching overhead adopted by the two approaches. This can be clearly observed in the significant reduction of the message loss and duplication in the two approaches.

Surprisingly, the overall throughput of the *reactive* approach has not improved in the low handoff scenario shown in Figure 5.6(a). In the high handoff scenario presented in Figure 5.6(b), mobile subscribers connect with each broker for a short period and may reconnect back to their old brokers before the completion of the state transfer. This results in transferring a small number of messages between the old and new broker, but occurs quite often due to the high frequency of handoffs. In the low handoff scenario, we found that most mobile subscribers migrate to new locations after the completion of the state transfer. The low handoff scenario presents the full cost of state transfer, although it occurs less frequently than the high handoff scenario. For this reason, the throughput results remain almost similar in the high and low handoff frequency scenarios. The low handoff scenario shows higher message loss as the overhead of state transfer here is high. This is due to the completion of the state transfer protocol in the most cases of the low handoff frequency. In such scenario, all the buffered messages will be transferred from the old to the new broker. Therefore, more messages will be discarded from the buffer of the new broker. In the high handoff scenario, subscribers may move back to the original broker sometimes before even the state transfer takes place due to the high load/latency of the old broker/network.

5.2.3.5 Overall performance at given network bandwidths

Two bandwidth values, 1Mbps and 11Mbps, are used to study the effect of network bandwidth on the performance of the three mobility support approaches. The bandwidth values of 1Mbps and 11Mbps respectively reflect low and high wireless bandwidth environments. Figure 5.7(a) and 5.7(b) show the overall subscriber throughputs *Ts*, the percentages of message loss *L*, and the percentages of message duplication *D* as experienced by the low and high bandwidth scenarios, respectively.



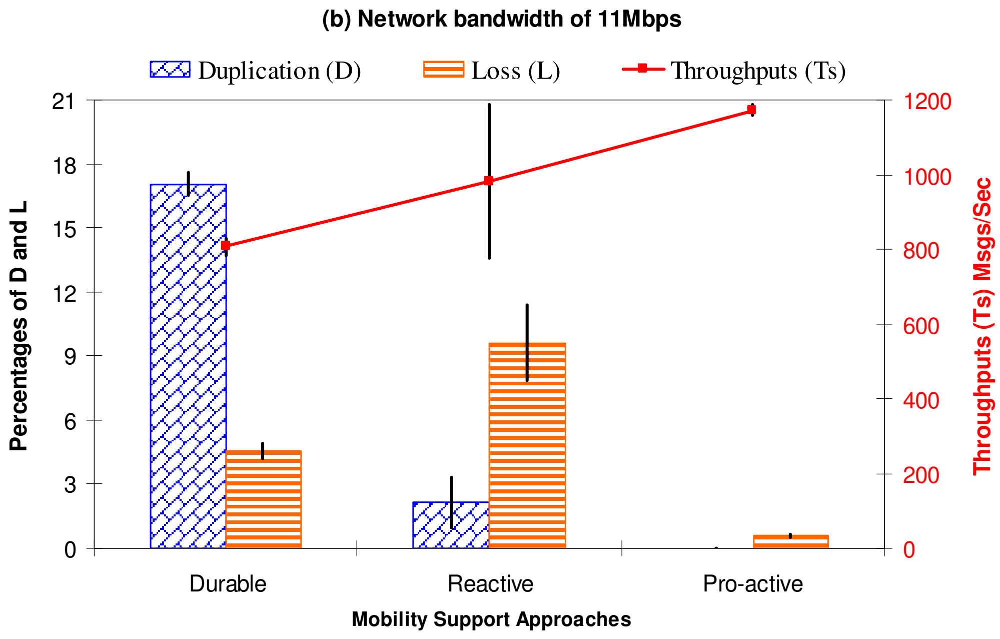


Figure 5.7: Overall performance at given bandwidths

From the graphs, we note that the three approaches show lower throughput results with the low bandwidth scenario as expected, as the wireless channel can carry only a limited number of messages per second to the subscribers. Interestingly, we noticed that although the channel was totally utilized, the total consumed messages by all the subscribers were less than the actual capacity of the channel (1Mbps). As the channel bandwidth in such scenarios becomes the bottleneck, messages will take a longer time to be received and acknowledged by the subscribers. This leads to resending the same message several times before it gets acknowledged, and hence results in delivering a noticeable amount of duplicated messages. This may explain why the percentages of message duplication are high in the low bandwidth scenario. In addition to wasting a large amount of bandwidth, redelivering duplicated messages will congest the wireless channel and thus increase the overall delay.

The figures also show that the three approaches experience message loss with the low and high bandwidth scenarios. The results in the graphs depict that the three approaches show relatively higher message loss with the high bandwidth scenario. The main reason for this is attributed to the total available bandwidth. With higher wireless bandwidth, the broker can accept and buffer more messages as the publisher and subscribers' throughputs depend on the total available bandwidth. Message discarding hence occurs more often than with the low bandwidth scenario due to the limitation of buffer space. It should be noted that a portion of the total percentage of message loss is attributed to the characterization of the wireless channel (i.e., % of packet drop) as well as the handoff procedure.

From Figure 5.7(a) and 5.7(b), we observe that the *pro-active* approach shows superior results in terms of overall throughputs, message loss, and duplication. This is a result of preventing message duplication, reducing message loss by caching messages on-demand, and hiding message overhead by propagating message ahead of the subscriber movement. The *durable-based* approach shows the worst results among the two remaining approaches as it suffers from a high percentage of message duplication, thereby degrading the overall throughputs. This is due to the overhead of caching messages continuously at each broker visited by the subscriber, irrespective of its current active subscriptions. The *reactive* approach falls in between the other approaches, as it has higher message overhead than the *pro-active* approach and lower message caching overhead compared to the *durable-based* approach. It should be noted that the *reactive* approach demonstrates much higher percentages of message loss than the other approaches in both bandwidth scenarios. This is because the overhead of state transfer does not pay off in terms of message loss with short disconnection interval (default setting 12 seconds) as depicted in Figures 5.5(a) and 5.5(b). The *durable-based* and *pro-active* approaches, on the other hand, benefit from short disconnection intervals as they store fewer messages and therefore only a small number of messages will be discarded from the buffers.

5.3 Random and Neighboring Mobility Patterns

Next we evaluate and compare the performance of the *pro-active* approach in the random (RND) and neighboring (NBR) mobility patterns in terms of message processing time (*M*) and individual broker throughput (*Tb*). Figure 5.8 shows the results of the selected metrics in the random (RND) and neighboring (NBR) mobility patterns: the left y-axis presents broker throughput (*Tb*) and the right y-axis shows message processing time (*M*). The presented results correspond to the individual brokers (B_i) used in our experimental network topology described in Figure 4.3, Chapter 4.

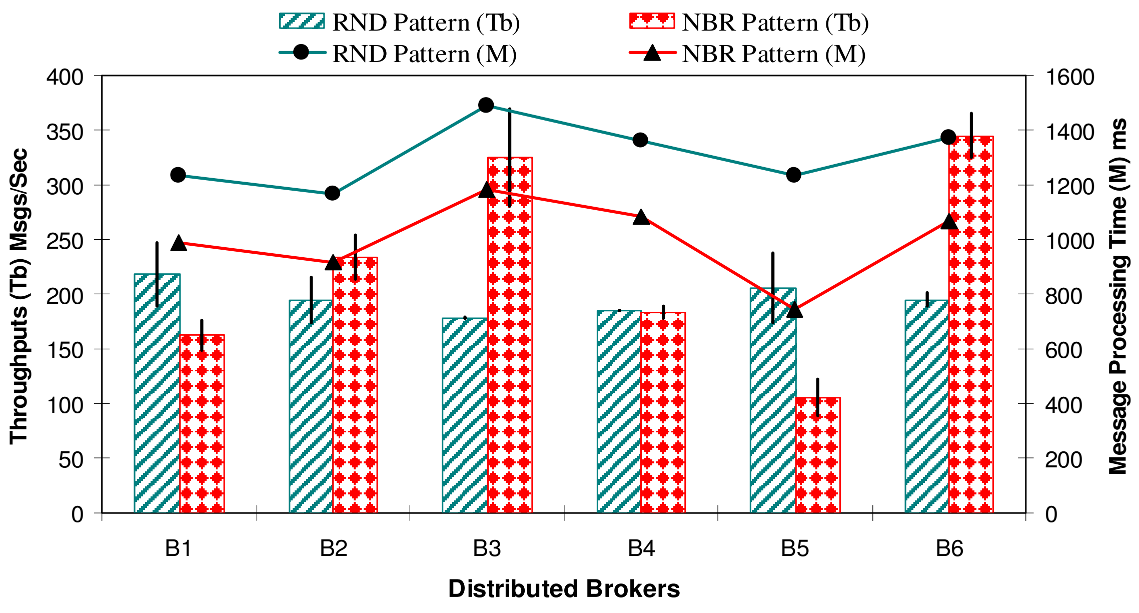


Figure 5.8: Performance of pro-active approach in RND and NBR mobility patterns

From Figure 5.8, we can observe that the *pro-active* approach shows lower message processing times in the neighboring mobility pattern with all brokers. This is a good indicator of the overhead reduction due to limiting the mobility prediction to the (true) neighbor brokers. This results in reducing the number of immediate neighbors of each broker and therefore improves the performance of the *pro-active* approach. For example, broker *B5* shows the lowest message processing time among all the brokers because it has only one neighbor broker *B3*. In the random pattern, broker *B5* shows higher processing time as it has 5 neighbor brokers and needs to buffer messages for each disconnected subscriber at one of these neighbors. We can also observe that broker *B5* depicts much lower throughput results in the neighboring pattern. Since broker *B5* has only one neighbor broker, the number of subscribers visiting broker *B5* is much less than in the case in the random pattern. Therefore, broker *B5* routes fewer messages to the overall subscribers. In contrast, brokers *B3* and *B6* have the largest number of neighbors (4 neighbors each) among all the brokers and show the highest throughput results. This is attributed to the fact that these brokers are visited by a large number of subscribers due to the neighboring mobility pattern. Thus, many messages are routed through these brokers (hot spots traffic).

The remaining section presents and compares the behavior of the three mobility support approaches using the random (RND) and neighboring (NBR) mobility patterns and the default input values listed in Table 4.1, Chapter 4. Three different performance metrics are used to evaluate the performance of the three approaches: the overall subscriber throughput (*Ts*), message loss (*L*), and message duplication (*D*). We used different subscriber populations (10, 100, and 200) to investigate the overall performance of the three approaches under different workload conditions. Tables 5.2, 5.3, and 5.4 show the obtained results of the selected metrics.

Table 5.2: Subscriber throughput in the RND and NBR mobility patterns

	Durable-based		Reactive		Pro-active	
Subscribers	RND	NBR	RND	NBR	RND	NBR
10	86.99	86.28	92.39	90.53	108.90	124.72
100	589.33	574.32	664.39	663.63	810.26	957.32
200	806.81	791.74	980.77	953.17	1172.77	1420.26

From Table 5.2, we note that the *pro-active* approach show superior throughput results in the random and neighboring mobility patterns, compared to the other approaches. As we expected, the *pro-active* approach in the neighboring mobility pattern depicts a noticeable improvement in the throughput results compared to the results achieved in the random mobility pattern. This is because each broker in this pattern has fewer neighbors compared to the scenario in the random pattern (brokers have probabilistically the same, maximum number of neighbors). As a result, the overhead (subscription propagation and message caching) of the *pro-active* approach significantly decreases and hence subscriber throughput improves. In contrast, the throughput results of the remaining approaches show a slight decrease with the various subscriber populations as shown in Table 5.2. We suspect that such behavior is due to the increased load on the central brokers (brokers that have a large number of immediate neighbors) that may become performance bottlenecks. In the *pro-active* approach, we found the overhead on the central brokers has not increased and is always less than the overhead in the random pattern, as shown in Figure 5.8.

Table 5.3: Message loss in the RND and NBR mobility patterns

	Durable-based		Reactive		Pro-active	
Subscribers	RND	NBR	RND	NBR	RND	NBR
10	4.78	3.13	3.65	5.14	2.25	1.26
100	5.14	4.06	8.92	7.03	0.86	0.72
200	4.56	3.51	9.62	9.44	0.58	0.51

Table 5.3 shows that the *pro-active*, *durable-based*, and *reactive* approaches demonstrate a slightly lower percentage of message loss in the neighboring mobility pattern. This is due to the fact that some brokers' buffers (central brokers) are heavily utilized, but not others. In the random pattern, all the brokers' buffers experience almost similar (and high) buffer utilization. An interesting observation is that the percentage of message loss of the *pro-active* approach decreases gradually with the increase of subscriber population in both mobility patterns. With larger populations, the probability of having similar interest among the subscribers increases. This leads to a significant reduction in the caching overhead of the *pro-active* approach as one copy of each message can be stored for many subscribers. As a result, message loss decreases with the increase of subscriber population as shown in the above table. The *durable-based* approach shows an approximately similar percentage of message loss with the increase in the subscriber population. Although the *durable-based* approach benefits from the similarity of interest, its caching overhead is almost constant with different population sizes. This can be attributed to the continuous caching process

adopted by this approach. In contrast, the *reactive* approach does not benefit from the similarity of interest as it does not reduce the overhead of state transfer. During the handoffs, state transfer has to be performed individually for every moving subscriber and hence its overhead increases proportionally with the subscriber population. Thus, message loss increases with the increase of the overhead imposed by subscriber population.

Table 5.4: Message duplication in the random and neighboring mobility patterns

	Durable-based		Reactive		Pro-active	
Subscribers	RND	NBR	RND	NBR	RND	NBR
10	11.35	12.10	0.80	1.52	0.0	0.0
100	12.01	11.68	1.70	0.85	0.0	0.0
200	17.05	16.97	2.13	1.90	0.0	0.0

From Table 5.4, we observe that the message duplication is slightly decreased in the neighboring (NBR) mobility pattern of the *durable-based* and *reactive* approaches. The pro-active approach shows zero message duplication in both mobility patterns as it keeps track of the last consumed message by each subscriber and only buffers messages with higher ID than the last consumed message. From the three previous tables, we can conclude that the neighboring mobility model improves the performance of the *pro-active* approach in terms of overall throughput, message loss, and duplication. The two remaining approaches have not shown a significant difference in their performance results when using either the random or the neighboring mobility pattern.

5.4 Concluding Remarks

In this chapter, we presented the implementation and evaluation of three mobility support approaches: the *reactive*, *pro-active*, and *durable-based* approaches. Each approach uses a different mechanism to achieve its goals. The reactive approach involves message transfer from the old to the new broker during each handoff. The message transfer occurs only after a mobile subscriber migrates to a new broker. In contrast, the pro-active approach ensures that subscriber context (subscriptions/messages) always remain one hop (broker) ahead of its current broker. The durable-based approach requires each broker to continuously buffer all the published messages irrespective of its currently active subscribers.

The reactive, pro-active, and durable-based approaches were evaluated and compared in terms of different performance metrics: end-to-end latency, handoff latency, message loss, message duplication, overall subscriber throughput, individual broker throughput, and message processing time. These performance metrics are evaluated using random and neighboring mobility patterns along with a set of workload parameters, such as number of subscribers, publishing rate, queue size, disconnection interval, handoff frequency, and network bandwidth, to study under which conditions one approach may perform better than another. The performance results of the three approaches were compared to gain a better insight into the behavior of each approach in supporting subscriber mobility.

Our evaluation results have shown that the reactive approach suffers from high message overhead due to state transfer. As a result, it shows much higher end-to-end latency of message delivery compared to the other approaches. The pro-active approach shows a comparable end-to-end latency to the durable-based approach since it has almost no state transfer overhead. The pro-active approach depicts superior throughput results among the three approaches as it performs state transfer before the subscriber movement and buffers messages on-demand. The durable-based approach shows the worst throughput results, among the three approaches, as it suffers from high message duplication and caching overhead. Comparing the handoff latency of the three approaches, the reactive approach shows more pronounced handoff latency than the others. This is because the subscriber messages need to be fetched from the old broker before the subscriber can consume any message at its new broker and this may take some time, particularly with a congested network or overloaded brokers. In contrast, the pro-active and durable-based approaches experience much lower handoff latency and hence they can provide fast handoff required by some applications to maintain the quality of the connections.

In general, the experimental results have shown that the pro-active approach successfully eliminates message duplication and significantly minimizes message loss compared to the other two approaches. The reactive approach shows higher sensitivity to message loss with short disconnection period. This is because the overhead of concurrent state transfer does not pay off when transferring a small number of messages. The durable-based approach results in higher message duplication as the subscribers can consume identical messages from previously visited brokers. Moreover, it is sensitive to message loss with large disconnection intervals due to the continuous message caching.

The pro-active approach benefits, in terms of subscriber throughput, from the neighboring mobility pattern due to limiting the mobility predication to neighboring brokers. This can significantly reduce the overhead of the pro-active approach and hence improve its performance. In contrast, the reactive and durable-based approaches have not shown any significant improvement when subscriber mobility is limited to the neighboring brokers. This is because the overhead of these approaches remain more or less constant.

CHAPTER 6

AN ANALYTIC MODEL FOR EXTRAPOLATING THE PERFORMANCE OF THE PRO-ACTIVE APPROACH

6.1 Introduction

In this chapter, we present an analytical model that can be used to extrapolate the performance of our proposed pro-active approach in a near-size environment (in terms of broker and/or subscriber population) to our experimental environment, using performance data obtained from our experiments. The general approach for performance approximation is as follows: most performance metrics are a function of the number of active subscribers at each broker. We thus first describe how to analytically derive the expected number of subscribers for a given broker topology, mobility model, and overall subscriber population, based on continuous-time Markov chains (CTMC). We next show how to extract performance-related data through curve-fitting from our testbed results. These curves relate the average number of subscribers with a performance metric of interest, here the per-broker throughput. The approach can be generalized to other performance metrics. Using these two steps, we can then extrapolate the throughput of the pro-active approach in a near-size environment to our experimental environment. This is an essential step as we were not able to achieve this experimentally due to the limitations of our experimental environment. Given the two sets of performance results reported in Chapter 5, we will validate our approach by deriving key parameters such as the fitted curve based on one set of experiments (using a random mobility pattern) and use it to approximate the results of the second set of experiments, using a neighboring mobility pattern.

In the next two sections, we present two Markov models that reflect the random and neighboring mobility patterns of a mobile subscriber. From the two Markov models, we calculate the expected number of subscribers at each broker. We then introduce a curve-fitting approach that is used to calculate the throughput of individual brokers using the expected number of subscribers obtained from the Markov models. We describe our analytical results and compare them against a selected set of our experimental results described in Chapter 5.

This chapter is organized as follows: Section 6.2 gives a brief description of the general model of subscriber mobility. Section 6.3 and 6.4 respectively describe continuous-time Markov chains (CTMC) for modeling the random and neighboring mobility patterns. Section 6.5 demonstrates the selected curve-fitting approach. Section 6.6 shows and compares the analytical and experimental results. Section 6.7 summarizes this chapter.

6.2 Description of Subscriber Mobility

We consider a distributed pub/sub system composed of a set of brokers $B = \{b_1, \cdots, b_N\}$ that communicate by exchanging control messages to support subscriber mobility using a pro-active approach. For convenience in demonstrating our analytical model, we assume that the distributed brokers are homogenous (i.e., all brokers are identical and have the same statistical behavior). A mobile subscriber enters to connect state S_1, shown in Figure 6.1, when it initially connects to one of the distributed broker b_i to receive its messages. The connection time is a random variable, denoted by T_C, and is assumed to have an exponential distribution with *pdf* given by $f_{T_C}(t) = \beta \exp(-\beta t)$, where β is the reciprocal of the mean of T_C, $T_\beta = E[T_C]$. The mobile subscriber remains in state S_1 for a randomly generated, exponentially distributed time with the mean of T_β seconds. For generating exponential variates of the connection time T_C, we have used the following function $T_C = \frac{-\ln(t)}{\beta}$, where t is a random variable uniformly distributed on the interval [0,1]. With an equal probability, the mobile subscriber either moves to disconnect state S_2 or handoff state S_3. Similarly, the disconnection and handoff times are also random variables, denoted by T_D and T_H, respectively. The *pdf* of T_D and T_H are given by $f_{T_D}(t) = \delta \exp(-\delta t)$ and $f_{T_H}(t) = \alpha \exp(-\alpha t)$ respectively. The mean of T_D is $T_\delta = E[T_D]$ whereas the mean of T_H is $T_\alpha = E[T_H]$. Exponential variates of T_D and T_H are generated by the same function used with T_C. The disconnect state S_2 reflects the case of signal breakdown due to poor network connectivity while the handoff state S_3 reflects the case of moving to a different coverage area. The mobile subscriber remains in state S_2 for a randomly generated, exponentially distributed time with a mean of T_δ seconds. With an equal probability, the mobile subscriber either moves back to the connect state S_1 and reconnects to the same broker or goes to the handoff state S_3. Moving from state S_2 to S_3 reflects the case when the mobile subscriber disconnects from the network voluntarily or involuntarily and finds itself in a different coverage area. In contrast, moving from state S_1 to S_3 corresponds to the case when the subscriber enters a new coverage area while he/she is roaming. After staying in state S_3 for a randomly generated, exponentially distributed time with a mean of T_α seconds, the subscriber moves to state S_1 and reconnects to a different broker. During the handoffs, the new destination (broker) is uniformly selected from the available brokers. Figure 6.1 depicts the state diagram for subscriber mobility model along with the means of the residence times (T_β, T_δ, and T_α) and the quantities of the departure rates ($\frac{1}{2T_\beta}, \frac{1}{2T_\delta}$, and $\frac{1}{T_\alpha}$) at/from each state.

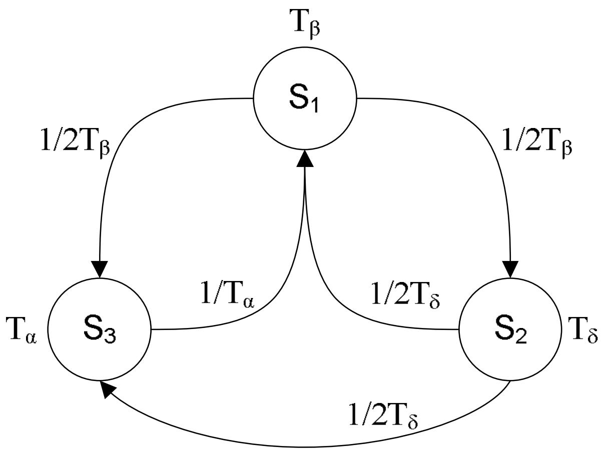


Figure 6.1: State transition diagram for subscriber mobility model

In this chapter, we use continuous-time Markov chains (CTMC) to model subscriber mobility between a set of brokers (states). This is due to the fact that we have a discrete state space of brokers (i.e., countable state space $S = \{1, \ldots, H\}$) and the *sojourn* times (holding time in one state before moving to another) are exponentially distributed. We have used CTMC to model the random and neighboring mobility patterns described in Chapter 4. The next two sections describe in detail how CTMC is used to model these two mobility patterns.

6.3 Modeling Random Mobility

In this section, we describe the random mobility pattern in which a mobile subscriber has the freedom to move to one of *N* brokers that are available in the system. The target broker is selected randomly and uniformly. Here we assume that the mobile subscriber can connect only to one broker at a time. The state space of subscriber mobility thus can be defined by the number of *connect* states (brokers) and a single *disconnect* and *handoff* state. We denoted the state with the variable *S* that takes integer values in the set $\{1,\ldots,\mathrm{H}\}$, where $\mathrm{H}=\mathrm{N}+2$. Hence, the transition states of the random mobility model can be presented by an *H*-state Markov chain, as shown in Figure 6.2.

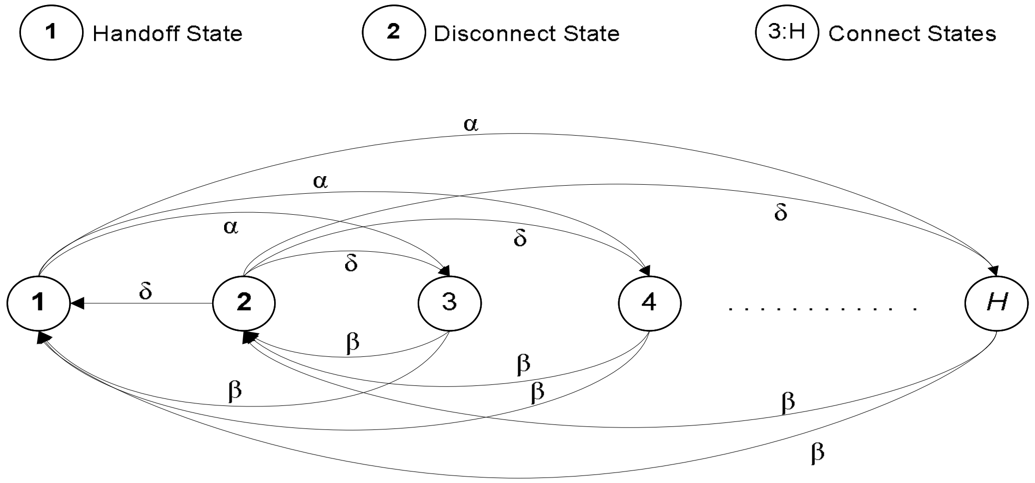


Figure 6.2: State transition diagram for the random mobility model

The aim of this model is to calculate the average number of subscribers at each state. This is required to approximate the throughput of individual brokers as described later. In Figure 6.2, states $\{1\}$and$\{2\}$ correspond to the handoff and disconnect states, respectively. The rest of the states $\{3,\cdots,H\}$ represent the connect states. Therefore, the state space *S* of the random model is given by $S=\{1,\cdots,H\}$. The arrows in the graph depict the subscriber mobility between different states while the parameters $\alpha,\delta,\text{and}\,\beta$ represent the transition (departure) rates from state $\{1\}$, $\{2\}$, and $\{3,\cdots,H\}$, respectively.

We now proceed to identify the rate matrix *M* (also know as infinitesimal generator matrix) of state space *S* according to the state transition diagram shown in Figure 6.2. Then we calculate the state probabilities $\pi_i, i=1,\cdots,H$, and accordingly the probabilistic behavior of the subscriber mobility can be completely described. Since the cardinality of S is H, M is an $H\times H$ matrix, whose entries denote the transition rates between different states. The diagonal rates of the M-matrix, $m_{i,i}$, are negative and given by $m_{i,i} = -\sum_{\substack{j=1 \\ j\neq i}}^{H} m_{i,j} \quad 1 \le i \le H$, whereas the off-diagonal rates are positive. The sum of each row is equal to zero, i.e. $\sum_{\substack{j=1 \\ j\neq i}}^{H} m_{i,j} + m_{i,i} = 0 \quad 1 \le i \le H$. The interpretation for the elements of M-matrix is as follows: $m_{i,j}$ denotes the departure rate of the mobile subscriber from state i to state j, $i, j \in S$. Then the $H \times H$ M-matrix of the corresponding random CTMC is described as

$$M = \begin{bmatrix} -N\alpha & 0 & \alpha & \alpha & \alpha & \cdots & \alpha \\ \delta & -(N+1)\delta & \delta & \delta & \delta & \cdots & \delta \\ \beta & \beta & -2\beta & 0 & 0 & \cdots & 0 \\ \beta & \beta & 0 & -2\beta & 0 & \cdots & 0 \\ \beta & \beta & 0 & 0 & -2\beta & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \beta & \beta & 0 & 0 & 0 & \cdots & -2\beta \end{bmatrix}.$$

Let $\pi_i,\ i = 1,\cdots,H$, be the stationary state probability that the mobile subscriber is in state i. Accordingly, the H-element row vector $\pi = [\pi_1 \quad \pi_2 \quad \pi_3 \quad \cdots \quad \pi_H]$ represents the H stationary state probabilities that satisfy the matrix equation

$$\pi M = 0. \tag{6.1}$$

It should be noted that the sum of stationary state probabilities is equal to one,

$$\sum_{i=1}^{H} \pi_i = 1 \tag{6.2}$$

Since the mobile subscriber moves to one of the connect states $i = 3, \cdots, H$ without any specific preferences and its mobility follows a symmetrical behavior in terms of arrival and departure, the stationary state probabilities $\pi_i,\ i = 3, \cdots, H$ are all equal. We denote the state probability of being in any one of these states by *P*. Therefore, we have $P = \pi_3 = \pi_4 = \cdots = \pi_H$, resulting in

$$\sum_{i=1}^{H} \pi_i = \pi_1 + \pi_2 + NP = 1 \tag{6.3}$$

By solving the first and second matrix equations of 6.1, we obtain the state probabilities, π_1 and π_2, of the handoff and disconnect states, respectively.

$$\pi_1 = \frac{(N+2)\beta}{(N+1)\alpha} P. \tag{6.4}$$

$$\pi_2 = \frac{N\beta}{(N+1)\delta} P. \tag{6.5}$$

By substituting π_1 and π_2 into Equation 6.3, we obtain the state probability, *P*, of being in any of the connect states *i*, $i = 1, \cdots, 6$ as follows

$$P = \left[\frac{(N+1)\alpha\delta}{N\beta\alpha + (N+2)\beta\delta + N(N+1)\alpha\delta} \right] \tag{6.6}$$

Let β be the transition (departure) rate from any connect state *i*, $i = 3, \cdots, H$, and T_β be the mean sojourn (connect) time at any connect state *i*. Thus,

$$\beta = \frac{1}{2T_\beta}. \tag{6.7}$$

Let δ be the transition (departure) rate from the disconnect state $\{2\}$, and T_δ be the mean sojourn (disconnect) time at state $\{2\}$. Thus,

$$\delta = \frac{1}{(N+1)T_\delta}. \tag{6.8}$$

Let α be the transition (departure) rate from the handoff state $\{1\}$, and T_α be the mean sojourn (handoff) time at state $\{1\}$. Thus,

$$\alpha = \frac{1}{NT_\alpha}. \tag{6.9}$$

The constant values (2, *N*+1, and *N*) shown in the denominators of the previous three equations reflect the number of possible destination states from the current state.

The expected number $E[x_i]$ of subscribers at any connect state *i*, $i = 1, \cdots, H$, follows the binomial distribution, since the presence of each subscriber at state *i* is a Bernoulli experiment with probability of success π_i and probability of failure $(1 - \pi_i)$. Let *K* be the total number of subscribers in the system. Hence, the expected number of subscribers at any connect state *i*, $i = 3, \cdots, H$, is given by

$\Pr\{x \text{ subscribers are at state } i\} = (\pi_i)^x (1 - \pi_i)^{K-x} \binom{K}{x} = P^x (1 - P)^{K-x} \binom{K}{x}$. This leads to

$$\bar{n}_i = E[x_i] = KP = K\frac{(N+1)\alpha\delta}{N\beta\alpha + (N+2)\beta\delta + N(N+1)\alpha\delta}. \tag{6.10}$$

Similarly, we can obtain the expected number of subscribers at handoff state$\{1\}$, $\overline{n_1}$, and disconnect state$\{2\}$, $\overline{n_2}$. Hence, we have

$$\overline{n_1} = E[x_1] = K\pi_1 = K\left(\frac{(N+2)\beta}{(N+1)\alpha}P\right) = K\left(\frac{(N+2)\beta\delta}{N\beta\alpha + (N+2)\beta\delta + N(N+1)\alpha\delta}\right). \tag{6.11}$$

and

$$\overline{n_2} = E[x_2] = K\pi_2 = K\left(\frac{N\beta}{(N+1)\delta}P\right) = K\left(\frac{N\beta\alpha}{N\beta\alpha + (N+2)\beta\delta + N(N+1)\alpha\delta}\right). \tag{6.12}$$

For a sanity check on the binomial calculations, we have summed up the total expected numbers of subscribers at all states, which should be equal to *K*. Hence, we have

$$\overline{n_1} + \overline{n_2} + N\overline{n_i} = K\left(\begin{array}{c}\frac{(N+2)\beta\delta}{N\beta\alpha + (N+2)\beta\delta + N(N+1)\alpha\delta} + \\ \frac{N\beta\alpha}{N\beta\alpha + (N+2)\beta\delta + N(N+1)\alpha\delta} + \\ \frac{N(N+1)\alpha\delta}{N\beta\alpha + (N+2)\beta\delta + N(N+1)\alpha\delta}\end{array}\right) = K \tag{6.13}$$

6.4 Modeling Neighboring Mobility

In this section, we describe the neighboring mobility model, which is a special case of the random mobility model, presented in Section 6.3. In this model, a mobile subscriber, based on its current state, can only move to one of its neighbor states, where the selected neighbor is chosen according to the uniform distribution. Here we also assume that the mobile subscriber can connect only to one broker (state) at a time. The neighboring model under consideration reflects the same six-broker network topology presented in Chapter 4, Figure 4.3. The restriction on the possible brokers the subscriber can move to from its current broker is modeled by substituting single disconnect and handoff states by multiple disconnect and handoff states, a pair for each broker. Hence, the state space *S* of the neighboring model includes six connect states *i*, $i = 1, \cdots, 6$, six disconnect states *j*, $j = i + 6$, and six handoff states *k*, $k = i + 12$. As a result, the state space *S* is defined by $S = \{1, \cdots, 18\}$. The transition states of the neighboring mobility model can accordingly be presented by the 18-state Markov chain shown in Figure 6.3.

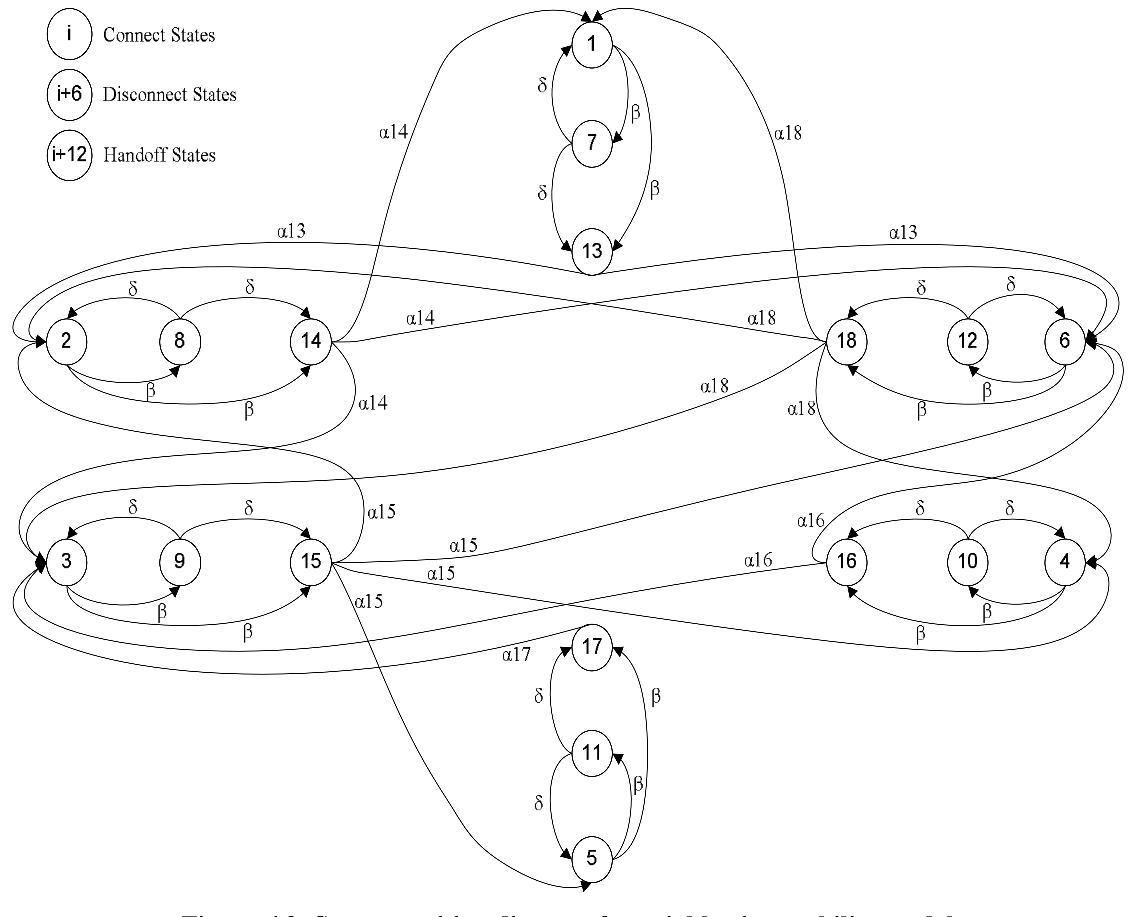


Figure 6.3: State transition diagram for neighboring mobility model

The arrows in the graph depict subscriber mobility between different states while the parameters β, δ, and $\alpha_i, i = 13,\cdots,18$ correspond to the transition (departure) rates from the connect, disconnect, and handoff states, respectively. Based on the transition state diagram shown in Figure 6.3 and the mean sojourn time at each state, we can determine the values of the parameters (β, δ, and α_i) as follows:

$$\beta = \frac{1}{2T_\beta} \tag{6.14}$$

$$\delta = \frac{1}{2T_\delta} \tag{6.15}$$

$$\alpha_i = \frac{1}{E_i T_\alpha} \tag{6.16}$$

In the above three equations, the mean sojourn time at any connect, disconnect, and handoff states is denoted by T_β, T_δ, and T_α, respectively, while the number of possible destinations from any connect or disconnect states is denoted by the constant 2. Similarly, the number of possible destinations (neighbors) from any handoff state *i*, $i = 13,\cdots,18$ is denoted by E_i.

We next determine the *M*-matrix (rate matrix) of the state space *S*. Then we can calculate the state probabilities $\pi_i, i = 1,\cdots,18$, to identify the probabilistic behavior of subscriber mobility. As the cardinality of *S* is 18, the *M*-matrix has 18×18 entries that denote the transition rates based on the state transition diagram shown in Figure 6.3. The *M*-matrix of the corresponding neighboring continuous-time Markov chains (CTMC) is given by

$$M=\begin{bmatrix}
-2\beta & 0 & 0 & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0 & 0\\
0 & -2\beta & 0 & 0 & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0\\
0 & 0 & -2\beta & 0 & 0 & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0 & 0 & \beta & 0 & 0 & 0\\
0 & 0 & 0 & -2\beta & 0 & 0 & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0 & 0 & \beta & 0 & 0\\
0 & 0 & 0 & 0 & -2\beta & 0 & 0 & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0 & 0 & \beta & 0\\
0 & 0 & 0 & 0 & 0 & -2\beta & 0 & 0 & 0 & 0 & 0 & \beta & 0 & 0 & 0 & 0 & 0 & \beta\\
\delta & 0 & 0 & 0 & 0 & 0 & -2\delta & 0 & 0 & 0 & 0 & 0 & \delta & 0 & 0 & 0 & 0 & 0\\
0 & \delta & 0 & 0 & 0 & 0 & 0 & -2\delta & 0 & 0 & 0 & 0 & 0 & \delta & 0 & 0 & 0 & 0\\
0 & 0 & \delta & 0 & 0 & 0 & 0 & 0 & -2\delta & 0 & 0 & 0 & 0 & 0 & \delta & 0 & 0 & 0\\
0 & 0 & 0 & \delta & 0 & 0 & 0 & 0 & 0 & -2\delta & 0 & 0 & 0 & 0 & 0 & \delta & 0 & 0\\
0 & 0 & 0 & 0 & \delta & 0 & 0 & 0 & 0 & 0 & -2\delta & 0 & 0 & 0 & 0 & 0 & \delta & 0\\
0 & 0 & 0 & 0 & 0 & \delta & 0 & 0 & 0 & 0 & 0 & -2\delta & 0 & 0 & 0 & 0 & 0 & \delta\\
0 & \alpha_{13} & 0 & 0 & 0 & \alpha_{13} & 0 & 0 & 0 & 0 & 0 & 0 & -2\alpha_{13} & 0 & 0 & 0 & 0 & 0\\
\alpha_{14} & 0 & \alpha_{14} & 0 & 0 & \alpha_{14} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3\alpha_{14} & 0 & 0 & 0 & 0\\
0 & \alpha_{15} & 0 & \alpha_{15} & \alpha_{15} & \alpha_{15} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4\alpha_{15} & 0 & 0 & 0\\
0 & 0 & \alpha_{16} & 0 & 0 & \alpha_{16} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2\alpha_{16} & 0 & 0\\
0 & 0 & \alpha_{17} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\alpha_{17} & 0\\
\alpha_{18} & \alpha_{18} & \alpha_{18} & \alpha_{18} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4\alpha_{18}
\end{bmatrix}$$

Let $\pi_i, i=1,\cdots,18$, be the stationary state probability that the mobile subscriber is in state *i*. As a result, the 18-element row vector $\pi=[\pi_1 \quad \pi_2 \quad \pi_3 \quad \cdots \quad \pi_{18}]$ represents the 18 stationary state probabilities that satisfy the matrix equation shown in Equation 6.1, and their summation is equal to one, i.e. $\sum_{i=1}^{18}\pi_i=1$. Note that the elements of vector π, $\pi_i, i=1,\cdots,6$, $\pi_i, i=7,\cdots,12$, and $\pi_i, i=13,\cdots,18$ are the state probabilities of the connect, disconnect, and handoff states, respectively. These state probabilities can be obtained by solving the matrix equation indicated in Equation 6.1.

After identifying the state probabilities, we can readily calculate the expected number of subscribers, $E[x_i]$, at any connect state *i*, $i=1,\cdots,18$, in a similar manner to that described in Section 6.3. Let *K* be the total number of subscribers in the system. Hence, the expected number of subscribers at any state i is given by

$\Pr\{x \text{ subscribers are at state } i\} = (\pi_i)^x (1-\pi_i)^{K-x} \begin{pmatrix} K \\ x \end{pmatrix}$. This leads to

$$\overline{n_i} = E[x_i] = K\pi_i. \tag{6.17}$$

6.5 Curve- Fitting Approach

In this section, we use a curve-fitting approach that typically creates an equation from some observed data. This approach can relate, with a percentage of error, throughput to the average number of subscribers at a broker by assigning a single function across the entire range of data. As a result, the generated function can be used to numerically extrapolate near future outcomes. There are several curve fit forms we could chose from to generate a function that provides the "best" fit (i.e., the curve with minimum error between the generated curve and date points, usually referred to as least-square error). In this research, we have chosen the polynomial curve-fitting approach since it shows the best fit, among the other generic forms we have tried, for our observed data. Using EXCEL, we fit a 3rd degree polynomial curve. Note that the observed data used for generating the curve is collected from the experiments of our general mobility model (random model) described in Chapter 5. It is also collected from all the brokers used in our experimental setup. Figure 6.4 shows the generated polynomial fitting-curve.

From the curve fitting shown in Figure 6.4, we obtain the following polynomial equation that can be used to extrapolate the throughput of individual brokers as a function of the expected number of subscribers x.

$$y = (0.0019)x^3 - (0.2256)x^2 + (12.362)x + 1.7111 \qquad (6.18)$$

The value of $R^2 = 0.9198$, depicted in the graph, is an indicator from 0 to 1 that reveals how closely the estimated values for the fitting-curve correspond to the observed data. A fitting-curve is more reliable when its R^2 (known as R-squared or the coefficient of determination) value is at or near 1.

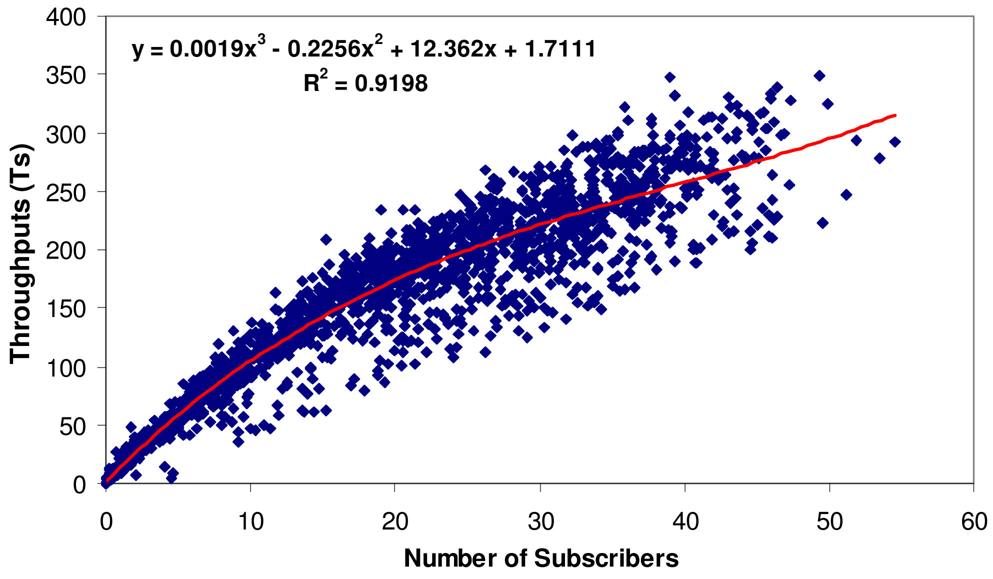


Figure 6.4: Polynomial fit

6.6 Comparative Study

In this section, we apply our approach to derive expected throughput results for each broker under both the *random* and *neighboring* mobility model, and compare them to the experimental results described in Chapter 5. The CTMC models allow us to determine the expected number $E(x)$ of subscribers at each state in both models. Then, we determine the expected throughput results via Equation 6.18 for each individual broker. For all the reported results in this section, we used fixed values ($T_\beta = 60$ seconds, $T_\delta = 12$ seconds, and $T_\alpha = 3$ seconds) for the mean sojourn (residence) time of connect, disconnect, and handoff states, respectively, in both mobility models described in this chapter. These values correspond to the used values in our experimental setup. Unless otherwise stated, the total number of subscribers (*K*) in the system was set to 200 and the total number of brokers *N* was set to 6. Therefore, the size of the state space *S* is $H = N + 2 = 8$ in the random model and 18 in the neighboring model.

6.6.1 Random Model Results

We first compare the analytical and experimental results in terms of the expected number of subscribers at each broker for a validity check of our analytical model. Using the mean sojourn (residence) times indicated earlier, we can identify the departure rate from each state. From Equations 6.7, 6.8, and 6.9, we get

$$\beta = 1/120\ ,\quad \delta = 1/84\ , \text{and} \quad \alpha = 1/18\ .$$

Now we describe the numerical solution for the set of equations (6.4, 6.5, and 6.6) using the obtained departure rates $(\beta, \delta, \text{and}\ \alpha)$. First we use Equation 6.6 to obtain the state probability of being in connect state (*P*), which is equal for all connect states as discussed previously. Hence, we have $P = 0.147679$. Similarly, we can use Equations 6.4 and 6.5 to obtain the state probability of being in the handoff and disconnect states π_1 and π_2, respectively. Thus, we have $\pi_1 = 0.025316,\ \ \pi_2 = 0.088608$. We verify that the obtained state probabilities satisfy Equation 6.3, i.e.

$$\sum_{i=1}^{8} \pi_i = \pi_1 + \pi_2 + NP = 0.025316 + 0.088608 + (6)0.147679 = 1.$$

According to Equation 6.10, the expected number of subscribers at any connect state *i*, $i = 3, \cdots, H$, will then be obtained by $\overline{n}_i = K \times 0.147679$ subscribers/broker. Figure 6.5 shows the expected number of subscribers at each broker with varying the total number of subscribers, $K = \{50,100,150,200\}$. The doted lines represent the experimental results while the sold lines correspond to the analytical results. From the graph, we note that the analytical and experimental results in most cases are close. The averages of the experimental results do fluctuate due to their random nature. From measuring a 95% confidence interval, these averages have small confidence intervals that are almost identical in length for the runs with the same number of subscribers and overlap the upper and lower bounds of the expected numbers obtained from the analytical model. This indicates that the differences between the analytical and experimental results are not statistically significant. This gives us much confidence in the use of our analytical model to determine the expected number of subscribers at each state.

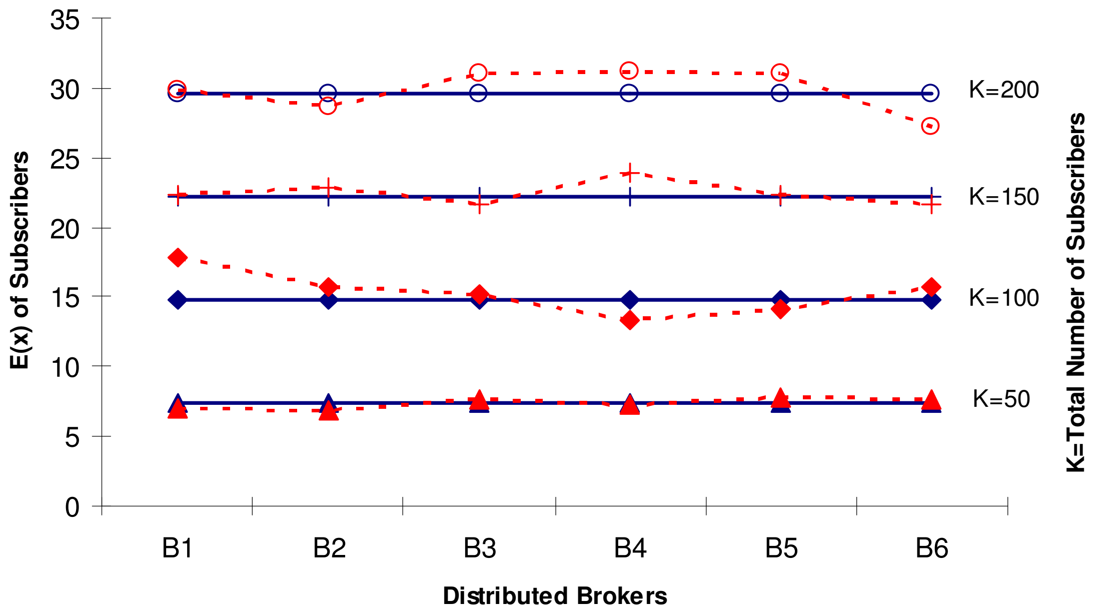


Figure 6.5: Expected number of subscribers at each broker

We next describe the numerical results obtained from the random model presented in Section 6.3. The expected number of subscribers/broker for a total subscriber population of 200 is obtained by $\overline{n}_i = 200 \times 0.147679 = 29.5358$ as indicated above. The approximated throughput of each broker is computed by substituting $\overline{n}_i$ into Equation 6.18. Thus, the throughput at each broker is given by $\mathrm{y} = 218.983 \quad \mathrm{msgs/sec}$. We plot *y* (the approximated throughput result) along with the experimental throughput results for the random model in Figure 6.6 and 6.7.

The lines across the data bars in the next figures represent the upper and lower bound of a 95% confidence interval for average throughputs. To calculate this confidence interval, we applied a different approach due to the small sample size of our experimental results. Thus, the following formula used to obtain the confidence interval of 95%, $\overline{X} \pm t_{\alpha/2} \times \frac{s}{\sqrt{n}}$, where $\overline{X}$ is the sample mean, s is the standard deviation, n is the sample size, and $t_{\alpha/2}$ is the t-value with an area of $\alpha/2$ to its right. The upper and lower bound of the analytical throughput results are captured in a different way. The normal approximation to the binomial distribution can be used to calculate the upper and lower bound of the number of subscribers that can be found at any instance with probability of 95% at each broker. Using these bounds in Equation 6.18, we obtained the upper and lower bound of the analytical throughput results. It should be noted that before applying the normal approximation, a certain condition must be met as indicated in [68] (i.e., $KP \pm 2\sqrt{KP(1-P)} \quad \in \quad [0, K]$). The normal approximation fails to provide the upper and lower bound of the expected number of subscribers if this condition has not satisfied, which is the case when *K* equals

10 and 50. This explains the reason for not plotting the upper and lower bound of the analytical throughput results with these subscriber populations in the presented graphs.

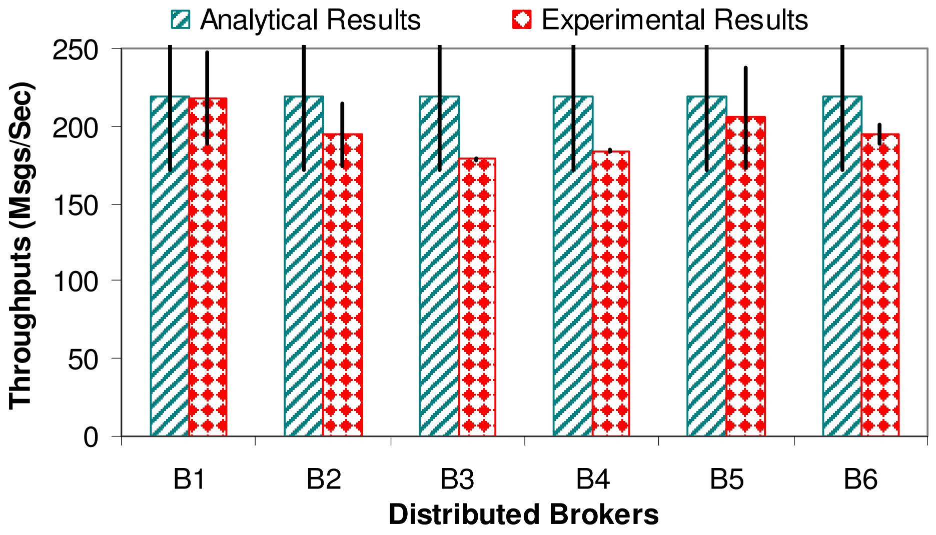


Figure 6.6: Throughput results for individual brokers

Figure 6.6 shows the analytical and experimental throughput results for individual brokers with a subscriber population of 200. From measuring a 95% confidence interval for both sets of results, we note that the confidence intervals of the experimental results overlap all the corresponding intervals of the analytical results. This indicates that the differences between the analytical and experimental results are not statistically significant. From the graph, we also note that the analytical results show relatively higher results compared to the experimental results. This can be attributed to the fact that each broker will have a large number of proxy subscribers that buffer messages on behalf of the moving subscribers. As a result, the throughput of individual brokers (especially ones run on machines with lower hardware configurations such as *B3* and *B4*) can be noticeably affected due to the overhead imposed by the proxy subscribers. Such overhead is typically higher in the random model than in the neighboring model as the later model restricts subscriber mobility between immediate neighbors, thereby having a lower number of proxy subscribers. Our analytical model can be refined in the future to take the number of proxy subscribers at each broker into consideration.

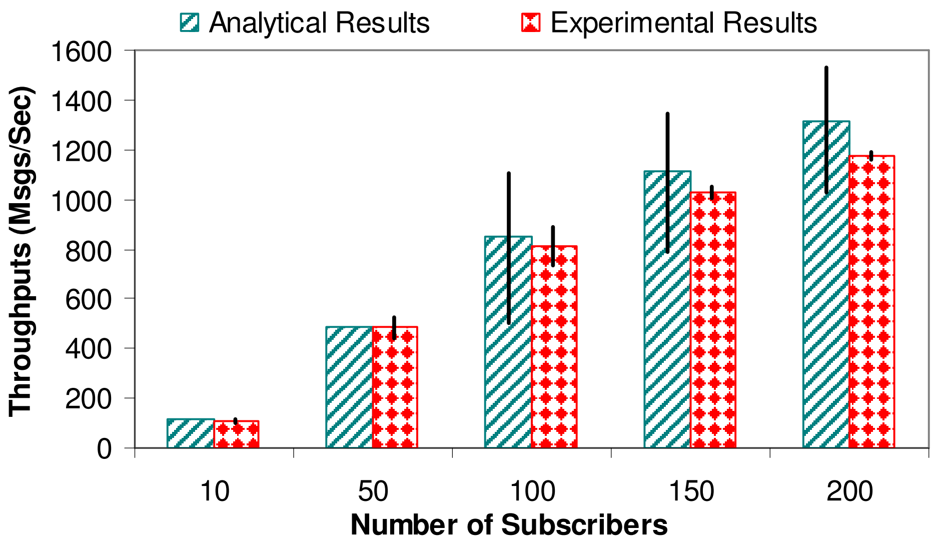


Figure 6.7: Overall throughput results with different subscriber population

Figure 6.7 illustrate the overall throughput results achieved with the increase of the subscriber population. From the graph, we note that the analytical and experimental results are very close for the small subscriber population. We also observe that the difference between the two sets of results increases with the increase of subscriber population. We conjecture that as the subscriber population increases, the overhead of proxy subscribers at each broker increases gradually. Therefore, the overall throughput results can be affected due to the overhead imposed by the proxy subscribers. This is not observed with a small population of subscriber as the impact of the proxy subscribers is much reduced.

6.6.2 Neighboring Model Results

We next compare the expected number of subscribers/broker obtained from the analytical model with the results from our testbed for the neighboring mobility model. Using the mean sojourn (residence) times indicated earlier, we can determine the departure rate from each state. From Equations 6.14, 6.15, and 6.16, we get

$$\beta = 1/120,\quad \delta = 1/24, \text{and} \quad \alpha_{13\to18} = \{1/6, 1/9, 1/12, 1/6, 1/3, 1/12\}.$$

Now we describe the numerical solution for the matrix equation indicated in 6.1 using the obtained departure rates $(\beta,\ \delta, \text{and}\ \ \alpha_{13\to18})$. To solve the matrix equation, we basically plug in the rate values shown above in the 18×18 *M*-matrix presented earlier and then find the solution for the matrix equation, which belongs to the null space of the matrix M. Solving Equation 6.1, we can obtain the state probability for the different states, and then the expected number of subscribers can be found by multiplying each state probability by the total number of subscribers *K*. Table 6.1 presents the state probabilities of the six connect states $\pi_i, i = 1,\cdots,6$ and the total state probability of disconnect and handoff states π_{DH} along with the expected number of subscribers (for a total subscriber population of *K*) at the connect states $\overline{n_i}, i = 1,\cdots,6$ and at the disconnect and handoff states $\overline{n_{DH}}$.

Table 6.1: State probabilities and *E(x)* of subscribers

State Probabilities	Expected Number of Subscribers
$\pi_1 = 0.1099$	$\overline{n}_1 = E(x_1) = K\times0.1099$
$\pi_2 = 0.1648$	$\overline{n}_2 = E(x_2) = K\times0.1648$
$\pi_3 = 0.2198$	$\overline{n}_3 = E(x_3) = K\times0.2198$
$\pi_4 = 0.1099$	$\overline{n}_4 = E(x_4) = K\times0.1099$
$\pi_5 = 0.0549$	$\overline{n}_5 = E(x_5) = K\times0.0549$
$\pi_6 = 0.2198$	$\overline{n}_6 = E(x_6) = K\times0.2198$
$\pi_{DH} = 0.1209$	$\overline{n}_{DH} = E(x_{DH}) = K\times0.1209$

We computed the expected number of subscribers/broker, as indicated in Table 6.1., for the total subscriber populations, K={50,100,150,200} and plotted the analytical and experimental results for the six brokers, as shown in Figure 6.8, to validate our analytical model. The doted lines correspond to the experimental results while the sold lines represent the analytical results. From the graph, we observe that the analytical and experimental results in most cases are close. Similar to the case of the random mobility model, the CTMC approach results in accurate values of the expected subscriber population across different brokers.

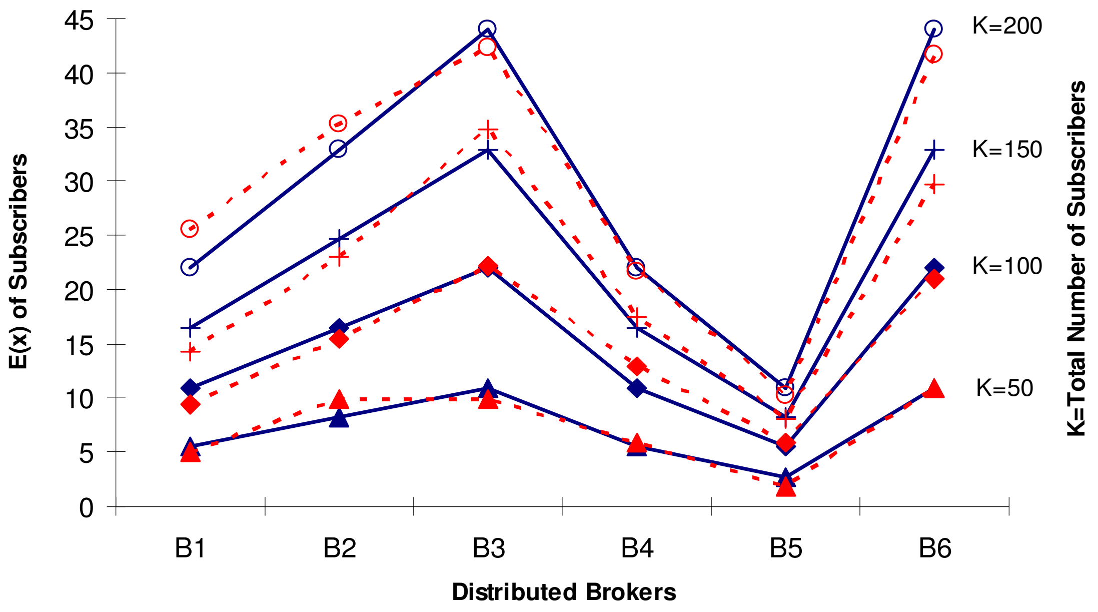


Figure 6.8: Expected number of subscribers at each broker in neighboring mobility

We now describe the numerical results obtained from the neighboring model presented in Section 6.4. The expected number of subscribers/broker for a total subscriber population of 200 is obtained by $200\times\pi$, $\pi=[\pi_1,\cdots,\pi_6,\pi_{DH}]$ as indicated in Table 6.1. Figure 6.8 shows the expected numbers for K=200. Based on the expected subscriber populations, the approximated throughput of each individual broker can now be obtained using Equation

6.18. Figure 6.9 depicts a plot of the throughput results along with the corresponding experimental results obtained for the neighboring model.

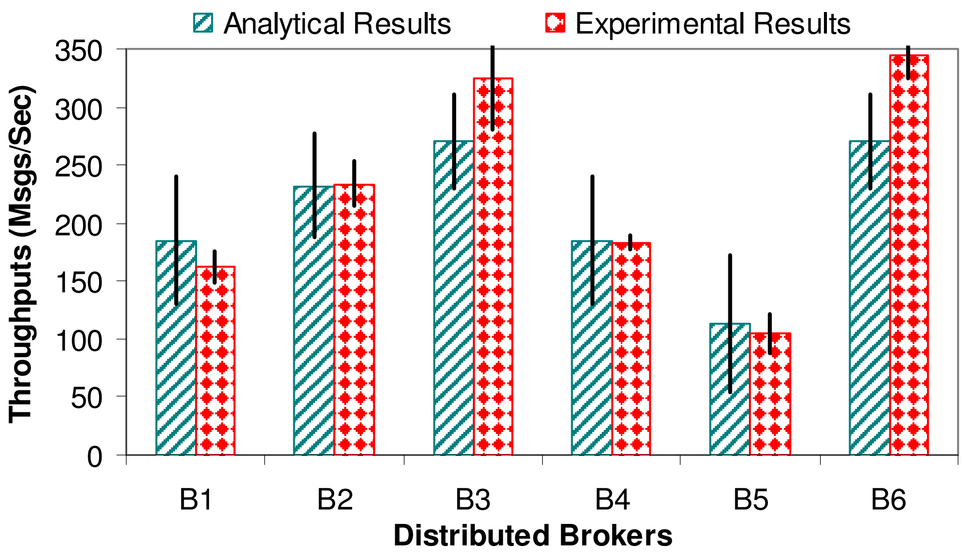


Figure 6.9: Throughput results for individual brokers

From the graph, we note that the analytical and experimental results are close for most brokers. We also observe that the analytical results show relatively lower throughput results with the central brokers, brokers with a large number of neighbors (*B3* and *B6*), compared to the experimental results. Generally, the central brokers will be visited by a larger number of subscribers than other brokers, and from the curve depicted in Figure 6.4, we observe that there is a greater variation across the experimental data points with the increase in the average number of subscribers. This can be the reason why brokers *B3* and *B6*, that serve the highest average number of subscribers, experience the largest deviation between the analytical and experimental results. Another reason can be attributed to the fact that central brokers *B3* and *B6* have a larger number of active proxy subscribers that buffer messages on behalf of the actual moving subscribers. Thus, the actual subscribers can consume more messages during their connect intervals, resulting in higher throughputs.

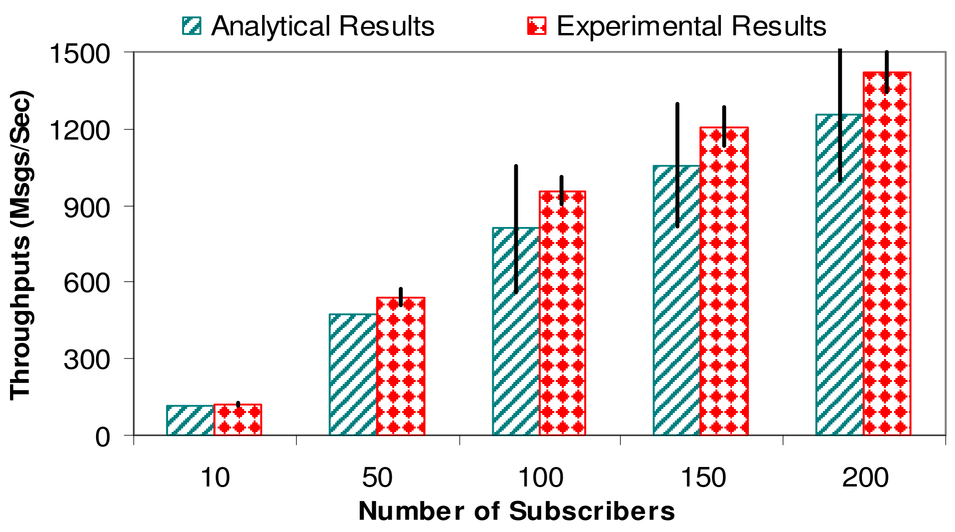


Figure 6.10: Overall throughput results with different subscriber population

Figure 6.10 shows the achieved throughput results with the increase of the subscriber population. From the graph, we note that the analytical and experimental results are very close with small subscriber populations (10 and 50) while the difference between the two sets of results increases with the increase of the subscriber population, making the analytical results relatively lower than the corresponding experimental results. In contrast to the random mobility model, the total number of proxy subscribers is much less in the neighboring mobility model due to limiting the mobility prediction to the (true) neighbor brokers. This results in reducing the overhead of the pro-active approach, accordingly improving the overall throughput results.

6.7 Concluding Remarks

The main objective in this chapter is to develop an approach that can extrapolate the performance of our proposed pro-active approach, in terms of message throughput, in a near-size environment to our experimental environment and to validate this model. To achieve this goal, we used continuous-time Markov chains (CTMC) to develop analytical models that reflect the mobility patterns under consideration, referred to as a *random* and *neighboring*, since the subscriber mobility model satisfies the characteristics of CTMC. To help understand the analytical model, we first presented a brief description of the subscriber mobility model. In Section 6.3 and 6.4, we developed the random and neighboring models using Markov chains to calculate the expected number of subscribers at each state (broker) under steady-state. Then we used a curve-fitting approach to approximate the throughput of individual brokers. Finally, we compared our analytical and experimental results. Most of the analytical and experiential results were close; however, in some cases the results were relatively different which can be attributed to the shape of the fitted curve and also due to the fact that the active proxy subscribers are not taken into consideration when developing the analytical model as discussed in Section 6.6.2. It should be noted that the analytical model developed to model the neighboring mobility is designed for a specific topology and cannot be used to approximate the performance in a different topology. Therefore, if we want to analytically investigate a larger/different topology, we will have to proceed in the same manner described in this chapter. In contrast, the analytical model developed for the general mobility model (random model) can be used to extrapolate the performance in a near-size environment (in terms of broker and/or subscriber population) to our experimental environment.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

The pub/sub paradigm is recognized as one of the most effective ways to model information dissemination applications [3]. The features of the paradigm and its peculiar advantages have been under discussion for some time and now are consolidated concepts. However, realizing the pub/sub-based systems in mobile wireless domains still remains a challenging problem, although it has been lately under investigation by many researchers [8][9][10][11]. Based on this observation, in this thesis we proposed a set of contributions aiming, on one hand, at identifying the key ideas for extending current pub/sub systems to support subscriber mobility, on the other, to suggest new problems and directions that motivate further research in the pub/sub area.

This thesis proposes a comprehensive and efficient mobile management scheme to extend current pub/sub middleware systems to operate in the mobile wireless environments. The main objective of the proposed scheme is to guarantee that all the published messages are successfully delivered to all interested subscribers in their publishing order regardless of the current location (broker) of the mobile subscribers. The proposed mobile management scheme is based on a pro-active caching approach that is initiated whenever a mobile subscriber hands off to a new broker. The pro-active approach depends on the use of a data structure, called neighbor graph, which is used to predict the set of next potential brokers where the subscriber context should be transferred prior to the subscriber's movement. The neighbor graph is automatically created and regularly updated to eliminate the outlier neighbors. Whenever the mobile subscriber disconnects from its original broker due to its mobility or poor network connectivity, the set of next potential brokers are notified and requested to buffer the subscriber messages using its previously transferred context. The pro-active approach employs virtual (or *proxy*) subscribers to cache messages on behalf of the moving subscribers. Accordingly, subscriber messages will be always ready for the mobile subscriber at its next potential location.

A general review of middleware systems literature helped in guiding this thesis research. Pub/sub (or e*vent notification service*) middleware system was selected as the general theme for the research. As part of our preliminary work, we have studied in detail the behavior of a JMS-based pub/sub system deployed in a mobile wireless environment. The study gave us insights into the characterization of the system and identified the effect of different mobility factors on the performance. These insights led to the choice of research topic and scope, namely, pro-active mobile management scheme for pub/sub systems. The effectiveness of the proposed scheme was then evaluated and compared to the alternative solutions proposed in the literature. Testbed experiments were conducted to validate and further study the performance of our proposed scheme. In additional, an analytical model was introduced to extrapolate the performance of the proposed pro-active approach in a near-size environment (in terms of broker and/or subscriber population) to our experimental environment. In the following a summary of the contributions of the thesis is given.

1. A detailed survey was conducted to define the major challenges of mobile computing systems and to evaluate the traditional and modern middleware solutions based on a set of requirements. A number of observations were made about the reviewed solutions that would be useful in designing future solutions targeted specifically for mobile computing applications (see references [1][69][70]).

2. In Chapter 1, we described the research problem in detail and identified some of the challenges that current pub/sub systems face when they are deployed in mobile wireless environments.

3. In Chapter 2, we provided a general overview of the pub/sub systems and a deep review to a representative set of these systems that motivated our selection of Java Message Service (JMS) as our base platform in this thesis work. We discussed the characteristics and limitations of these systems in terms of their proposed mobility extensions.

4. In Chapter 3, we presented the proposed pro-active solution for supporting subscriber mobility in current pub/sub systems and described its design and functionality. We also presented a data structure, called *neighbor graph*, which forms the base of the proposed approach as it is used to anticipate subscriber mobility. Next we showed some optimization techniques that can be used to reduce the overhead of the proposed approach. Finally, we presented a simplified analytical model that captures the messaging cost of the proposed approach and the alternative solutions and showed the cost ratio between them. This provides insight into the overhead imposed by the pro-active approach with respect to its counterparts. (presented in [71][72][73])

5. In Chapter 4, we briefly reviewed the advantages and features of JMS, emphasizing most on the ones used during our testbed experiments. Then we provided a high-level description of the prototype implementation of the pro-active and reactive solutions. Finally, we described in detail the setup of our experimental study.

6. In Chapter 5, we evaluated and analyzed the performance of the proposed pro-active approach using the testbed described in Chapter 4 and compared its behavior to the state-of-the-art solutions, reactive and durable subscription-based. The experimental results show that our approach reduces the message loss by more than 50% and message duplication to zero, compared to durable subscription-based approaches. The results also indicate that our approach experiences much lower handoff latency compared to reactive approaches. Overall, the proposed approach shows superior performance across a range of scenarios. (presented in [71][72][73])

7. In Chapter 6, we developed an analytical model that can be used to extrapolate the performance of the proposed pro-active approach in a near-size environment (in terms of broker and/or subscriber population) to our experimental environment. Given the two sets of performance results reported in Chapter 5 for random and neighboring mobility patterns, we validated our approach by deriving key parameters such as the fitted curve based on one set of experiments (using a random mobility pattern) and use it to approximate the results of the second set of experiments (using a neighboring mobility pattern).

7.2 Future Work

Although the experimental results shown in Chapter 5 provide confidence in the behavior and expectations of the pro-active approach, there are some other ideas for possible directions for future research that need yet to be carried out to improve the behavior of the proposed approach in supporting subscriber mobility. We conclude the thesis by proposing some of these ideas.

7.2.1 Load Balancing Among Brokers

Pub/sub systems are usually implemented as a distributed set of interconnected brokers to support large numbers of subscribers and high volumes of messages spread across a network. A subscriber uses one of the brokers in the pub/sub system to subscribe to messages of interest. The system does not provide full knowledge about the load levels of the brokers and thus subscribers may end up using overloaded brokers while there are others with light loads that can serve them better. A similar issue may occur when subscribers move from one broker to the others while they are roaming. With the absence of an efficient broker selection policy, the system may have several performance bottlenecks that can significantly degrade its overall performance.

- *Selection of initial broker*: The task of selecting an appropriate initial broker from a set of distributed brokers to better serve the subscriber is a critical issue as the messaging load between the brokers can be substantially different. This issue can also be seen in systems that use for example a server replication approach to improve system scalability. To achieve better performance, it may be necessary to select an appropriate server from a set of replicas. Hence, it is worthwhile to investigate the performance impact of this issue and define a selection policy to initially identify an appropriate broker based on predefined criteria such as the number of hosted subscribers per broker and the number of hops between a subscriber and its hosting broker. We believe that the highest priority for the selection policy should be given to the first criteria as it has a direct impact on the brokers' loads. The effect of using the selection policy should also be validated by measuring performance metrics such as system throughput, message processing time, and memory consumption. One suggested avenue for broker selection can be via a centralized proxy server that periodically collects the load information of all the brokers in the system. Subscribers initially connect to the system via the proxy server that selects and attaches them with the appropriate broker based on this load information in the proxy. Another avenue is to perform the selection process in a distributed manner. Brokers in the system periodically exchange their loading information. Instead of directly attaching to brokers, subscribers first connect to an independent proxy component that runs on each broker. Based on the selection criteria, the proxy either admits the subscribers or transfers them to an appropriate broker in the system.

- *Handing off between brokers*: Due to subscribers' mobility, subscribers may attach to different brokers in the system. Thus, some brokers at a given time may become overloaded as they serve a large number of subscribers while others have light loads. Also, mobile subscribers may often connect to certain brokers in the system due to their repetitive mobility patterns. This may result in propagating a large number of subscriptions to certain neighboring brokers that may already be overloaded. Hence, it would be useful to extend the broker selection policy to be used during handing off between brokers and propagating subscriptions to the predicted neighborhood brokers as their load sometimes become too imbalanced. It would be interesting to investigate the performance gain from integrating the selection policy with the proposed pro-active scheme in general. As discussed previously, either a centralized or distributed proxy-based solution can be used to implement broker selection. An alternative solution to the proxy-based solution would be for each broker to flood its load information into the network (limiting the scope to a relatively small number of hops) and having the subscribers select the best broker based on the retrieved information. Such an approach has several drawbacks. Subscribers are attached to the pub/sub system via a wireless network that usually has limited bandwidth. These broker advertisements require a reasonable amount of bandwidth and the wireless channel can be flooded with such advertisements and become a performance bottleneck. As a result, the overall throughput can be significantly degraded. Due to the high loss rate of wireless networks, some of the advertisements can be lost and hence subscribers may make wrong decision about the best broker. The selection process adds additional load on the subscribers' terminals that usually have limited capability.

7.2.2 Subscription Management

A key issue in large-scale pub/sub systems with a large number of mobile subscribers is how to efficiently propagate their subscriptions to the neighboring brokers. Brokers need such subscriptions in order to buffer incoming messages for disconnected subscribers. Unnecessary proliferation of subscriptions throughout the system may add significant overheads, particularly on network bandwidth requirements and serving time at the brokers. Existing systems [74][75] discuss the issue of passing subscriptions among the brokers in their systems, although they are connected via a high-speed wired links. This would be more problematic in our pro-active approach since it requires forwarding subscriptions to all neighboring brokers. Subscriptions propagation consumes some of the network bandwidth and may flood the network. This can reduce the publication rates and in turn diminish the subscribers' throughputs. Each broker maintains a list of its hosted subscriptions that is stored in memory. A large list of subscriptions will increase memory consumption, leading to slower message matching, and increased latency for message delivery. This is a serious issue because timely delivery of messages is important in many systems. The scalability issue could be viewed in terms of the number of subscribers attached to the brokers, or more critically the number of passed subscriptions in the system. This can be translated into monitoring the impact on neighboring brokers as the number of disconnected subscribers and forwarded subscriptions increases. An interesting direction for future work is to add an effective mechanism to reduce and avoid any unnecessary propagation of subscriptions. A suggested approach to achieve this is to consider covering relationships among subscriptions. If a newly admitted subscription *X* is covered by an existing subscription *Y*, then *X* is not sent to the neighboring brokers since *Y* can be used to store all messages matching *X*. Note that subscription *Y* cannot be removed unless it is not in use by any other subscriber. This decreases the number of subscriptions propagated in the system. Applying this process at every broker can result in a great reduction in the number of propagated subscriptions. Also, the numbers of remove, deactivate, and activate control messages will be reduced as well as the size of subscription lists, leading to faster delivery. Another approach is to summarize subscription descriptions and propagate compact descriptions to save network bandwidth. Every broker can use the compact information to locally create unavailable subscriptions.

7.2.3 Broker Topology

This thesis used a peer-to-peer (or *general graph*) topology of brokers, shown in Figure 2.3, Chapter 2. However, different interconnection topologies such as a hierarchical or hybrid (i.e., combination of peer-to-peer and hierarchical) topology can be used to form a distributed communication service. It is an interesting direction for future work to explore the behavior of the proposed pro-active approach in different broker topologies when the mobile subscribers connect to the distributed brokers through a dynamic mobile wireless environment with high frequency of handoffs. As shown in the analysis study conducted by [59], using a hierarchal topology of brokers can exhibit significant performance issues that may lead to different performance trade-offs with respect to the mobility management schemes,, resulting in an interesting problem to probe.

7.2.4 Publisher Mobility

Publisher mobility is another interesting direction for future work that recently attracts the attention of some researchers. Subscriber mobility is managed via some operations that incur brokers to locally store publications for the disconnected subscriber and replay them to the subscriber when it reconnects to the network. In contrast, there are no special operations to manage publisher mobility. Unlike disconnected operation with subscriber mobility, there is no specific information that the publisher would miss during the period of its disconnection. This might be the reason that no new protocols have been proposed for supporting publisher mobility.

References

[1] A. Gaddah and T. Kunz, "Does Modern Middleware Address Mobile Computing Requirements?", In Proceedings of the 8th World Multiconference on Systemics, Cybernetics and Informatics (SCI2004), Orlando, USA, July 2004,Vol. 5, pp. 493-499.

[2] D. Grigoras, "Challenges to the Design of Mobile Middleware Systems", In Proceedings of the international Symposium on Parallel Computing in Electrical Engineering (PARELEC'06), Washington, DC, IEEE Computer Society, September 2006, pp. 14-19.

[3] S. Behnel, L. Fiege, and G. Muhl, "On Quality-of-Service and Publish-Subscribe", In Proceedings of the 26th IEEE international Conference on Distributed Computing Systems Workshops (ICDCSW'06), Washington, DC, USA, July 2006, IEEE Computer Society, pp. 20-25.

[4] Y. Diao, P. Fischer, M. Franklin, and R. To, "YFilter: Efficient and Scalable Filtering of XML Documents", In Proceedings of the 18th International Conference on Data Engineering (ICDE 2002), San Jose, California, February 2002, pp. 341-342.

[5] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "SCRIBE: A Large-Scale and Decentralized Application-Level Multicast Infrastructure", IEEE Journal on Selected Areas in Communications (JSAC), Vol. 20, No. 8, October 2002, pp. 100-110.

[6] A. Carzaniga, D. Rosenblum and A. Wolf, "Achieving Expressiveness and Scalability in an Internet-Scale Event Notification Service", In Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC 2000), Portland, OR, July 2000, pp. 219-227.

[7] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith, "Efficient Filtering in Publish-Subscribe Systems Using Binary Decision Diagrams", In Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001), Toronto, Canada, May 2001, pp. 443-452.

[8] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom and D. Sturman, "Exploiting IP Multicast in Content-Based Publish/Subscribe Systems", In Proceedings of 1FIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000), New York, NY, April 2000, pp. 185-207.

[9] Y. Zhao and R. Strom, "Exploiting Event Stream Interpretation in Publish-Subscribe Systems", In Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (PODC 2001), Newport, RI, August 2001, pp. 219-228.

[10] G. Cugola, E. Nitto, and A. Fuggetta, "The JEDI Event-Based Infrastructure and its Application to the Development of the OPSS WFMS", IEEE Transactions on Software Engineering, Vol. 27, September 2001, pp. 827–850.

[11] P. Sutton, R. Arkins, and B. Segall, "Supporting Disconnectedness-Transparent Information Delivery for Mobile and Invisible Computing", In Proceedings of the 1st International Symposium on Cluster Computing and the Grid(CCGRID2001), Washington, DC, May 2001, pp. 277-285.

[12] Sun Microsystems, "Java Message Service Specification 1.1", Technical Report, Sun Microsystems, 2002.

[13] M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Stout, "Java Message Service", Sun Microsystems Inc., April 2002.

[14] P. Bracchi, and V. Cortellessa, "A Framework to Model and Analyze the Performability of Mobile Software Systems", In Proceedings of the 4th international Workshop on Software and Performance (WOSP'04), Redwood Shores, California, January 2004, pp. 243-248.

[15] V. Muthusamy, M. Petrovic, D. Gao, and H. Jacobsen, "Publisher Mobility in Distributed Publish/Subscribe Systems", In Proceedings of the Fourth international Workshop on Distributed Event-Based Systems (DEBS) (ICDCSW'05), Vol. 04, Washington, DC, IEEE Computer Society, June 2005, pp. 421-427.

[16] V. Muthusamy, M. Petrovic, and H. Jacobsen, "Effects of Routing Computations in Content-Based Routing Networks with Mobile Data Sources" In Proceedings of the 11th annual international conference on Mobile computing and networking (MobiCom'05), New York, NY, USA, August 2005, pp. 103–116.

[17] A. Gaddah and T. Kunz, "Performance of Pub/Sub Systems in Wired/Wireless Networks", In Proceedings of 64th IEEE Vehicular Technology Conference (VTC), Montreal, Canada, September 2006, pp. 1-5.

[18] A. Gaddah and T. Kunz, "Evaluating the Impact of Application Design Factors on Performance in Publish/Subscribe Systems over Wireline and Wireless Networks", Technical Report SCE-05-14, Carleton University, Ottawa, Canada, August 2005.

[19] R. Henjes, M. Menth, and C. Zepfel, "Throughput Performance of Java Messaging Services Using WebsphereMQ", In Proceedings of the 26th IEEE international Conference on Distributed Computing Systems workshops (ICDCSW'06), Washington, DC, July 2006, IEEE Computer Society, pp. 26-32.

[20] M. Menth, R. Henjes, C. Zepfel, and S. Gehrsitz, "Throughput Performance of Popular JMS Servers", SIGMETRICS Performance Evaluation Review, Vol. 34, No. 1, June 2006, pp. 367-368.

[21] P. Eugster, P. Felber, R. Guerraoui and A. Kermarrec, "The Many Faces of Publish/Subscribe", ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp. 114-131.

[22] Y. Huang and H. Garcia-Molina, "Publish/Subscribe in a Mobile Environment", Wireless Networks Journal, Special Issue on Pervasive Computing and Communications, Vol. 10, No. 6, November 2004, pp. 643-652.

[23] P. Pietzuch and J. Bacon, "Hermes: A Distributed Event-Based Middleware Architecture", In Proceedings of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02) in Conjunction with the 22nd International Conference on Distributed Computing Systems (ICDCS'02), Vienna, Austria, July 2002, pp. 611-618.

[24] L. Cabrera, M. Jones, and M. Theimer, "Herald: Achieving a Global Event Notification Service", In Proceedings of the 8th Worksop on Hot Topics in Operating Systems (HoTOS-VIII'01), Elmau, Germany, May 2001, pp. 87-94.

[25] P. Gore, R. Cytron, D. Schmidt, and C. O'Ryan, "Designing and Optimizing a Scalable CORBA Notification Service", In Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES'01), Snow Bird, Utah, US, May 2001, pp. 196-204.

[26] G. Ashayer, H. Leung, and H. Jacobson, "Predicate Matching and Subscription Matching in Publish/Subscribe Systems", In Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02), Washington, DC, July 2002, pp. 539-548.

[27] H. Liu and H. Jacobsen, "A-ToPSS – a Publish/Subscribe System Supporting Approximate Matching", In Proceedings of the 28th International Conference Very Large Databases (VLDB'02), Hong Kong, August 2002, pp. 1107-1110.

[28] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, and D. Sturman, "An Efficient Multicast Protocol for Content-Based Publish/Subscribe Systems", In Procedding of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS'99), Washington, DC, May 1999, pp. 262-272.

[29] TIBCO, "TIB/Rendezvous Concept", Available at http://www.tibco.com, Last Visited June 2007.

[30] I. Burcea, H. Jacobsen, E. Lara, V. Muthusamy, and M. Petrovic, "Disconnected Operation in Publish/Subscribe Middleware", In Proceedings of the 2004 IEEE International Conference on Mobile Data Management (MDM'04), Berkeley, California, January 2004, pp. 39-50.

[31] G. Cugola and E. Nitto, "Using a Publish/Subscribe Middleware to Support Mobile Computing", In Proceedings of the Workshop on Middleware for Mobile Computing, Heidelberg, Germany, November 2001, pp. 1-5.

[32] M. Caporuscio, A. Carzaniga, and A. Wolf, "Design and Evaluation of a Support Service for Mobile, Wireless Publish/Subscribe Applications", IEEE Transactions on Software Engineering, Vol. 29, No. 12, December 2003, pp. 1059-1071.

[33] M. Caporuscio, A. Carzaniga, and A. Wolf, "An Experience in Evaluating Publish/Subscribe Services in a Wireless Network", In Proceedings of the 3rd International Workshop on Software and Performance in Conjunction with International Symposium on Software Testing and Analysis (ISSTA'02), Rome, Italy, July 2002, pp. 128-133.

[34] U. Farooq, E. Parsons, and S. Majumdar, "Performance of Publish/Subscribe Middleware in Mobile Wireless Networks", In Proceedings of the 4th International Workshop on Software and Performance (WOSP'04), Redwood City, California, January 2004, pp. 278-289.

[35] U. Farooq, S. Majumdar, and E. Parsons, "High Performance Middleware for Mobile Wireless Networks", In Mobile Information Systems Journal, Volume 3, Issue 2, June 2007, pp. 107-132.

[36] Sonic Software Inc., "SONICMQ", Available at http://www.sonicsoftware.com, Last Visited in June 2007.

[37] J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel, and M. Spiteri, "Generic Support for Distributed Applications", IEEE Computer, Vol. 33, No. 3, March 2000, pp. 68-77.

[38] A. Zeidler and L. Fiege, "Mobility Support with REBECA", In Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCSW'03), Providence, RI, May 2003, pp. 354-361.

[39] G. Muhl, A. Ulbrich, K. Herrmann, and T. Weis, "Disseminating Information to Mobile Clients Using Publish-Subscribe", In Proceedings of the IEEE Internet Computing, Vol. 8, No. 3, June 2004, pp. 46-53.

[40] J. Wang, J. Cao, J. Li, and J. Wu, "MHH: A Novel Protocol for Mobility Management in Publish/Subscribe Systems", In Proceedings of the 2007 International Conference on Parallel Processing (ICPP'07), IEEE Computer Society, September 2007, Washington, DC, pp. 54-61.

[41] S. Hu, V. Muthusamy, G. Li, and H. Jacobsen, "Transactional Mobility in Distributed Content-Based Publish/Subscribe Systems", Technical Report, Middleware Systems Research Group, University of Toronto, April 2008.

[42] S. Tarkoma and J. Kangasharju, "On the Cost and Safety of Handoffs in Content-Based Routing Systems", Computer Networks: the International Journal of Computer and Telecommunications Networking, Vol. 51, No. 6, April 2007, pp. 1459-1482.

[43] M. Cilia, L. Fiege, C. Haul, A. Zeidler, and A. Buchmann, "Looking into the Past: Enhancing Mobile Publish/Subscribe Middleware", In Proceedings of the 2nd international Workshop on Distributed Event-Based Systems (DEBS'03), San Diego, California, June 2003, ACM Press, New York, NY, pp. 1-8.

[44] The OpenJMS Group, "OpenJMS", Available at http://openjms.sourceforge.net, Last Visited in June 2007.

[45] ObjectWeb, "Joram", Available at http://www.objectweb.org/joram, Last Visited in June 2007.

[46] Fiorano Software Inc., "FioranoMQ", Available at http://www.fiorano.com, Last Visited in June 2007.

[47] JBoss Group, "JBoss", Available at http://www.jboss.org, Last Visited in June 2007.

[48] Sun Microsystems, "Java System Message Queue", Available at http://www.sun.com/software, Last Visited in June 2007.

[49] I. Podnar and I. Lovrek, "Supporting Mobility with Persistent Notifications in Publish/Subscribe Systems", In Proceedings of the 3rd International Workshop on Distributed Event-based Systems (DEBS'04), Edinburgh, Scotland, UK, May 2004, pp. 80 -85.

[50] R. Meier and V. Cahill, "STEAM: Event-Based Middleware for Wireless Ad Hoc Networks", In Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops (ICDCSW'02), Vienna, Austria, June 2002, pp. 639-644.

[51] R. Meier and V. Cahill, "Exploiting Proximity in Event-Based Middleware for Collaborative Mobile Applications", In Proceedings of the 4th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'03), Paris, France, 2003, pp. 285-296.

[52] Y. Huang and H. Garcia-Molina, "Publish/Subscribe Tree Construction in Wireless Ad Hoc Networks", In Proceedings of the 4th International Conference on Mobile Data Management (MDM'03), Melbourne, Australia, January 2003, pp. 122-140.

[53] M. Srivatsa, and L. Liu, "Securing Publish-Subscribe Overlay Services with EventGuard", In Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05), Alexandria, VA, USA, November 2005, ACM, New York, NY, pp. 289-298.

[54] C. Wang, A. Carzaniga, D. Evans, and A. Wolf, "Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems", In Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02), Vol. 9, Washington, DC, USA, January 2002, IEEE Computer Society, pp. 303-310.

[55] G. Picco, G. Cugola, A. Murphy, "Efficient Content-Based Event Dispatching in the Presence of Topological Reconfiguration", In Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS'03), Politecnico di Milano, Italy, May 2003, pp. 234-243.

[56] O. Virmajoki and P. Franti, "Divide-and-Conquer Algorithm for Creating Neighborhood Graph for Clustering", In Proceedings of the 17th International Conference on Pattern Recognition (ICPR'04), IEEE Computer Society, Washington, DC, August 2004, pp. 264-267

[57] R. Paredes and E. Chavez, "Using the k-Nearest Neighbor Graph for Proximity Searching in Metric Spaces", In Proceedings of the 12th International Conference on String Processing and Information Retrieval (SPIRE'05), Buenos Aires, Argentine, November 2005, pp. 127-138.

[58] H. Hacid and A. Zighed, "An Effective Method for Locally Neighborhood Graphs Updating", In Proceedings of the 16th International Conference on Database and Expert Systems Applications (DEXA'05), Copenhagen, Denmark, August 2005, pp. 930-939.

[59] A. Carzaniga, D. Rosenblum and A. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service", ACM Transactions on Computer Systems, Vol. 19, No. 3, August 2001, pp. 332–383.

[60] R. Howard, "Dynamic Probabilistic Systems – Volume I: Markov Models", New York: Dover, 1971.

[61] N. Banerjee, W. Wei, and S. Das, "Mobility Support in Wireless Internet", In IEEE Wireless Communications Journal, Vol.10, No.5, October 2003, pp. 54-61.

[62] National Institute of Standards and Technology, NIST Network Emulation Tool, Available at http://snad.ncsl.nist.gov/itg/nistnet/index.html, Last visited June 2007.

[63] IEEE, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", IEEE Standard 802.11, 1999.

[64] M. Balazinska and P. Castro, "Characterizing Mobility and Network Usage in a Corporate Wireless Local-Area Network", In Proceedings of the 1st International Conference on Mobile Systems, Applications, and Services (MobiSys'03), San Francisco, California, May 2003, pp. 303-316.

[65] J. Yin, X. Wang, and D. Agrawal, "Modeling and Optimization for Wireless Local Area Network (WLAN)", Computer Communications Journal, Special Issue on Performance Issues of Wireless LANs, PANs, and Ad Hoc Networks, vol. 28, No. 10, June 2005, pp. 1204 -1213.

[66] A. Mishra, M. Shin, and W. Arbaugh, "An Empirical Analysis of the IEEE 802.11 MAC Layer Handoff Process", In ACM SIGCOMM Computer Communications Review (SIGCOMM'03), Vol. 33, No. 2, April 2003, pp. 93-102.

[67] N. Gupta and P. Kumar, "A Performance Analysis of the IEEE 802.11 Wireless LAN Medium Access Control", Communications in Information and Systems, Vol. 3, No. 4, September 2003, pp. 279-304.

[68] L. Richard, T. James, " Probability and Statistics for Engineers", Duxbury Press, April 4th 1994, ISBN 0534209645.

[69] A. Gaddah and T. Kunz, "Why Current Middleware Fails for Mobile Peer-to-peer Computing", NATO IST-030/RTG-012 Workshop on the Role of Middleware in Systems Functioning over Mobile Wireless Networks, Wachtberg, Germany, August 2003.

[70] A. Gaddah and T. Kunz, "A Survey of Middleware Paradigms for Mobile Computing", Technical Report SCE-03-16, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, July 2003.

[71] A. Gaddah and T. Kunz, "Subscriber Mobility in Pub/Sub Systems: Pro-active vs. Reactive Handoffs", to appear in Proceedings of the 4th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2008), Avignon, France, October 2008.

[72] A. Gaddah and T. Kunz, "A Pro-active Mobility Extension for Pub/Sub Systems", In Proceedings of the First International Conference on Mobile Wireless Middleware, Operating Systems, and Applications, Innsbruck, Austria, February 2008.

[73] L. Li, A. Gaddah, and T. Kunz, "Mobility Support in a Tactical P2P Publish/Subscribe Overlay", to appear in Proceedings of the 27th International Conference for Military Communication, (MILCOM2008), San Diego, CA, USA, November 2008.

[74] G. Li, S. Hou, and H. Jacobsen, "A Unified Approach to Routing, Covering and Merging in Publish/Subscribe Systems Based on Modified Binary Decision Diagrams", In Proceedings of the 25th IEEE international Conference on Distributed Computing Systems (ICDCS'05), Washington, DC, June 2005, IEEE Computer Society, pp. 447-457.

[75] Z. Shen, S. Aluru, and S. Tirthapura, "Indexing for Subscription Covering in Publish-Subscribe Systems", Technical Report TR-2005-07-2, Department of Electrical and Computer Engineering, Iowa State University, July 2005.