The Hong Kong Polytechnic University

Department of Computing

Reliable Service Discovery and Access in Pervasive Computing Environments

By

VASKAR RAYCHOUDHURY

A Thesis Submitted in Partial Fulfillment of

the Requirements for the Degree of

Doctor of Philosophy

February 2010

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

(Signature)

VASKAR RAYCHOUDHURY (Name of Student)

Abstract

Service discovery is one of the fundamental services in pervasive computing. Different services are provided by various portable devices which are interconnected in an ad hoc manner. However ensuring reliable service discovery and seamless service access is a desirable as well as challenging task in highly dynamic pervasive environments. Given the mobility and resource-constraints of pervasive devices and the unreliability of wireless connection, service unavailability can frequently occur in pervasive environments due to service provider failure, network partitioning, or service scope outage by service provider or user mobility. Due to the resource limitation of mobile nodes service discovery protocols must be message efficient which implies consumption of less bandwidth as well as lower processing overhead. Moreover, devices can suddenly fail, get disconnected or depart the network in which case, the services may become unavailable.

Existing service discovery protocols did not adequately address the issues in providing users with continuous service access at all the times in an autonomous and proactive manner. In this research we address the challenging issues and make the following original contributions in this field.

Firstly, we propose a directory community framework which works as the basis of our research in reliable service discovery and access in ad hoc networks. The directory community framework consists of a set of directory nodes along with a suite of protocols and algorithms to collaboratively provide reliable service discovery and seamless service access supports for mobile users in ad hoc networks. The directory community is constructed by dynamically electing a set of devices as directory nodes. Because of the resource constraints of mobile nodes, we choose to elect resource-rich

nodes with higher energy or computational capabilities as directories. We model the directory community formation problem as top-K weighted leader election in mobile ad hoc networks and develop a distributed algorithm to achieve the objective. Here, the weight indicates available node resources in terms of memory, processing power or energy. Our proposed directory election algorithm is scalable, reliable, message-efficient, and can handle dynamic topological changes in an efficient manner.

Secondly, using afore-mentioned directory community, we propose a quorum-based fault-tolerant service discovery protocol. The elected directory nodes are divided into multiple quorums. Services registered with a directory are replicated among its quorum members, so that, upon the failure of a directory, services can still be available. This approach guarantees network-wide service availability using the quorum intersection property and reduces replication and update costs by minimizing the quorum size.

Finally, based on the directory community, we develop a reliable and continuous service access mechanism for mobile users which works using service handoff. Service handoff provides mobile users seamless service access by proactively finding new matching services once the original service becomes unavailable. The proposed service handoff mechanism has two steps – handoff initiation and new service provider selection. Three different service handoff protocols have been designed for different situations. Our handoff protocols can reduce handoff message cost and time delay while achieving a load balance on service providers.

Publications

Journal Papers

1. **Vaskar Raychoudhury**, Jiannong Cao, and Weigang Wu, "Top K-leader Election in Mobile Ad Hoc Networks," submitted to IEEE Transactions on Computers on May 31, 2009.

2. **Vaskar Raychoudhury**, Jiannong Cao, Weigang Wu, and Steven Lai, "*K-Directory Community: Reliable Service Discovery in MANET*," an extended version of the paper published in ICDCN 2010 has been invited for submission to the fast track publication in the Journal of Pervasive and Mobile Computing (JPMC) (submitted on January 20, 2010).

Conference Papers

3. **Vaskar Raychoudhury**, Jiannong Cao, Weigang Wu, and Steven Lai, "*K-Directory Community: Reliable Service Discovery in MANET*," In Proceedings of 11th International Conference on Distributed Computing and Networking (ICDCN2010), January 3-6, 2010, Kolkata, India.

4. **Vaskar Raychoudhury**, "*Efficient and Fault Tolerant Service Discovery in MANET using Quorum-based Selective Replication*," In Proceedings of 7th Annual IEEE International Conference on Pervasive Computing and Communications (Percom 2009: Google PhD Forum), Galveston, Texas, USA, March 9-13, 2009.

5. Daqiang Zhang, Jiannong Cao, Jingyu Zhou, Minyi Guo, and **Vaskar Raychoudhury**, "*An Efficient Collaborative Filtering Approach Using*

Smoothing and Fusing," In Proceedings of the 38th International Conference on Parallel Processing (ICPP'09), September 22-25, 2009, Vienna, Austria.

6. [Invited Paper] **Vaskar Raychoudhury**, Jiannong Cao, and Weigang Wu, "*Top K-leader Election in Wireless Ad Hoc Networks*," In Proceedings of 17th International Conference on Computer Communications and Networks (ICCCN'08), August 3 - 7, 2008, St. Thomas, U.S. Virgin Islands.

7. Joanna Izabela Siebert, Jiannong Cao, Yu Zhou, Miaomiao Wang, and **Vaskar Raychoudhury**, "*Universal Adaptor: A Novel Approach to Supporting Multi-protocol Service Discovery in Pervasive Computing*," In Proceedings of International Conference on Embedded and Ubiquitous Computing (EUC'07), pp. 683-693, December, 2007, Taipei, Taiwan.

8. Miaomiao Wang, Jiannong Cao, Joanna Izabela Siebert, **Vaskar Raychoudhury**, and Jing Li, "*Ubiquitous Intelligent Object: Modeling and Applications*," In Proceedings of 3rd International Conference on Semantics, Knowledge and Grid (SKG'07), Oct. 29-31, 2007. Xian, China.

9. Yu Zhou, Jiannong Cao, **Vaskar Raychoudhury**, Joanna Izabela Siebert, and Jian Lu, "*A Middleware Support for Agent-Based Application Mobility in Pervasive Environments*," In Proceedings of the 27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07), June 25-29, 2007, Toronto, Ontario, Canada.

Acknowledgements

I wish to thank a multitude of people who have been helping me for the past few years of my PhD study. In particular, I would like to express my most sincere gratitude to my supervisor Dr. Jiannong Cao for his devoted guidance, constant encouragement, and invaluable suggestions, which helped me not only to complete this dissertation but also to prepare me for my future career. Dr. Cao is not only an outstanding researcher with broad knowledge, sharp intuition and grand vision, but also a very nice and kind person who encouraged me to face life with a positive attitude. It is mostly due to his compassionate treatment; Hong Kong became home away from home for me. He has taught me to always have high expectations and to demand more of myself. Working with him has been my invaluable and delightful experience.

I would also like to thank Dr. Weigang Wu, for his insightful and illuminative suggestions on my research. Without his acuminous insight and guidance this dissertation would never have seen the daylight. Thanks Weigang, you taught me the steps before I started walking.

Another very important person who always strives to make my research life enjoyable is my wife, Mrittika. No words are adequate to express my sincere gratitude to her, who stayed up with me nights and nights to encourage me, to check my papers and many a times just to show wordlessly that I am not alone at my endeavors. Now it is high time that I formally acknowledge her contribution.

I am largely indebted to my friends and past and present colleagues like, Ping Yu, Hui Cheng, Gang Yao, Yu Zhou, Yu Huang, Xiaopeng Fan, Daqiang Zhang, Joanna Siebert, Edwin Wei, Xin Xiao, Long Cheng, Weiping Zhu, Miaomiao Wang, Yuan Zheng and all other members in Dr. Cao's research group, for their insightful

discussions and warm friendship. Another great friend is Jack Cheng who has significant contributions in making my Hong Kong stay colorful. Thanks Jack, for everything you have done for me.

Another person who constantly inspired me to embrace the charms of academia from the onset of our acquaintances is Mr. Indrajit Dutt, currently my father-in-law. Had it not been for him, I would never have stepped out of my comfort zone to pursue research in a far away land from my home. I will forever remain indebted to him.

I would like to thank my previous roommates, Vivek Kanhangad and Manas Sarkar for putting up with me during the times when I had to work late hours. They took their turns to encourage me at my hours of despair. Staying with Mr. Manas Sarkar was wonderfully soothing and was very much like staying with my family.

Special thanks to Ms Miu Tai and Ms May Chu for their support and assistance in all administrative matters. I offer my sincere apologies to all those, who are inadvertently missed.

Last, but by no means the least, I would like to express my deepest gratitude to my parents and my sister for their love and unstinted encouragement that enabled me to complete this work.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AD: Access Delay

FT: Fraction of Time without K-leaders

F-T: Fault Tolerance

HD: Handoff Delay

HR: Hit Ratio

HSR: Handoff Success Rate

MANET: Mobile Ad hoc NETwork

NH: Number of Hops

NH/HO: Number of Hops per Handoff

NM: Number of Messages

NM/HO: Number of Message per Handoff

PvC: Pervasive Computing

PvCE: Pervasive Computing Environment

SD: Service Discovery

SDP: Service Discovery Protocol

TD: Time Delay

Chapter 1

Introduction

This chapter discusses the idea of pervasive computing systems, their characteristics and the requirements as well as challenges for service discovery applications in these types of systems. In Section 1.1 we give a classification of pervasive environments based on the available underlying infrastructure supports and then we discuss the requirements for service discovery applications for different environments in Section 1.2. We also point out the reliability concerns for service discovery in pervasive computing systems in Section 1.3 and highlight our proposed methods to address them properly in Section 1.4. Finally, Section 1.5 gives a brief outline of the dissertation.

1.1 Overview

The paradigm of distributed computing went through a sea change with the advent of portable mobile devices and wireless networks. The system support functions were developed to cope with the challenges of mobile computing. Then came the next wave of computing trend – the paradigm of pervasive computing. The primary focus of pervasive computing is human-centric. A multitude of embedded and intelligent computing devices are required to automatically detect the application requirements for

human users and to act accordingly to satisfy their need. We can describe the underlying philosophy of pervasive computing as -

Pervasive computing aims to create a smart environment with embedded and networked computing devices providing human users with convenient and seamless service access in an intelligent and invisible manner.

Applications of pervasive computing are wide-spread. There are enclosed smart environments – sometimes known as smart-spaces – examples include, smart home [46][15], smart office, smart classroom [102][88][5], smart meeting room [24], or smart museums [86][3]. Pervasive health-care [10][37][36] applications automatically monitor patient's physical conditions and provide necessary medical supports. Similarly, elderly-care [83][56] and assisted-living [73] applications keep track of the daily activities of lone older people and fetch them the required care by urgently calling ambulance or doctors when the need arises. Recently there is also much interest in developing social networking [47] applications that enable people to publish their current activity to friends and colleagues. There are also some important applications in the fields of entertainment [15][63] and logistics. Intelligent traffic management applications are being used in managing road, rail or air traffic in big cities. Altogether, pervasive computing is encompassing every aspect of human life by silently supporting computing all the time and everywhere.

The enabling technologies of pervasive computing [61] consist of sensors (e.g., UC Berkeley Motes Sensor Network Platform), Radio Frequency ID (RFID) tags, intelligent appliances, embedded processors, wearable computers, handheld computers, smart phones and many others. Tiny intelligent sensors have made it possible to deploy ubiquitous services and thus create various smart environments. RFID tags allow subtle integration of objects (e.g., commodities in a superstore or books in a library) into the

computing environment. The boundaries of pervasive computing have been further pushed by the development of new technologies as well as the extensive use of existing technologies, such as, the Internet, mobile and wireless communications, sensor networks, and the RFID technology.

Depending on the available underlying network structure, pervasive applications can be developed over wired or wireless networks. Available wireless networks can be further classified into infrastructured wireless and ad hoc or infrastructure-less wireless networks. Wired and infrastructured wireless networks have the support of a stable network backbone through which they can access computing elements in the wired infrastructure through gateways, proxies and base stations. Examples of such environment are wireless office networks based on Wi-Fi. These environments are able to leverage from the high bandwidth resource-rich wired environments. Ad hoc wireless networks, on the other hand, are composed of multiple static or mobile entities, and do not provide any support for computing elements (mobile devices) to access the wired infrastructure. The network connections with peer devices are created and broken down on-the-fly and on an as-required basis. Examples of such environments are disaster-stricken cities, under-construction sites, remote area (forest, volcano) monitoring, etc. Extensive research has been carried out for pervasive application development in infrastructured networks. In our research, we mainly focus on pervasive systems built over infrastructure-less networks, henceforth called infrastructure-less/ad hoc pervasive environments, as they offer more challenges and require novel solutions to address them.

1.2 Service Discovery in Pervasive Computing Environments – New Requirements

Design of a pervasive computing system is very much application-dependant. Also pervasive applications are mainly concerned about services instead of individual nodes

that provide them. Services can come from a single node or can be composed of a set of nodes. Different pervasive applications, such as, building smart spaces, developing assisted-living environments, health-care applications or intelligent information systems, are all service dependant. Thus, the service discovery applications are of wide importance in all types of pervasive computing systems. We define the terms - service, service discovery and service discovery protocol in the context of pervasive systems -

- **Service:** *A service is any hardware or software functionality (resources, data or computation) of a device that can be requested by other devices for usage.* For example, the media player on a mobile device can be considered as a service which provides music and movie playing functionalities for that device and can be requested by peer devices.

- **Service Discovery:** *It is a process by which any potential user (human or device) requiring a service can find services on peer devices and determines how to access or utilize the discovered services.*

- **Service Discovery Protocol (SDP):** *They are network protocols designed for automatic detection of devices and services provided by them. They aim to minimize administrative overhead of service discovery and to increase usability.*

In brief, service discovery enables devices and services to properly discover, configure, and communicate with each other with minimal or no human intervention. However, service discovery is not a novel application at all and has actively been used in traditional distributed as well as enterprise environments. Service discovery in enterprise networks are rather restricted as they consider mostly static and resource-rich computing devices connected through wired or infrastructured networks. Moreover, services in enterprise networks operate within a fixed scope, and hence, they can be

protected by firewalls and can be managed by system administrators on a centralized basis.

Pervasive computing environments, on the other hand, are far more dynamic and heterogeneous. When mingled with a purely ad hoc wireless network structure, the chances of occurrences of faults increase manifold. Moreover, due to their ad hoc nature compounded with node mobility, it is not possible to centrally control pervasive systems through system administrators. Devices must act through localized coordination, in order to discover services and use them in a reliable manner. Services in pervasive networks are not bound by any scopes and they are scattered in the ambience, this requires the capability of network-wide service discovery. All these goes to make service discovery in pervasive computing really challenging.

But the most important challenge regarding service discovery in pervasive environment lies in figuring out how to satisfy human users or to blend with their environments. Integrating people in the smart environments presents us with the following new service discovery requirements -

Autonomous and Proactive Support: Pervasive computing aims at automatically understanding user context and detecting user activity. Pervasive applications are situation-aware, i.e., they are designed to provide necessary services to users without explicit requests made by them. This is a crucial issue which demands for reliable and seamless provisioning of required services on a proactive basis.

Security and Privacy: Since, users express their preferences during service discovery, and the devices owned by different users interact with each other while discovering services, there is always a high chance of giving away user's personal information. Also, many a times, there are chances of deducing user intent based on some contextual

information which is also an equally vulnerable security issue and needs to be addressed during system design.

User Responsibility Minimization: Service discovery in pervasive computing requires some prior knowledge on the part of the users and the service providers. Most of the existing service discovery protocols consider clients (service seekers), services (networked services) and directories (centralized service information storage) in some fixed roles. People, on the other hand, can assume any role, either a service provider, or a user based on the necessities. While acting as a service user, people can discover and use services without requiring to actively manage pervasive devices. But, while working as a service provider they may need to know respective service terminologies – such as, names and attributes, in order to publish their services for general accessibility. This certainly creates extra overhead on their parts. So, the system must provide as much support as it can in order to reduce the burden on users.

Reliability and Fault Tolerance: The widespread belief is that pervasive applications are generally devised for people who are unable or less able to look after themselves, e.g., elderly people, young children, or ailing patients. Keeping it in mind, we can easily realize the sensitivity of the reliability issues of all kinds of pervasive computing applications, including service discovery. Most of the existing pervasive applications are human-centric and aim to provide people with unfaltering service support all the time. Consider cases of health-care or elderly care, which may even lead to loss of lives in case the system fails to detect the fall of an old person or a sudden deterioration of a patient's condition, and hence, does not inform the doctor. Same goes for structure health monitoring and traffic accident detection applications. Infrastructure-less environments, like disaster-stricken city can also be benefited by reliable ad hoc composition of smart devices where users need to discover life-sustaining services. So,

reliability or availability of services cannot be compromised when the involvement of human safety is concerned.

The seriousness of fault tolerance and the major failure implications of several human-centric pervasive computing applications justify further exploration of the area. The studies carried out so far are not well coordinated or in-depth. So, in this dissertation we investigate the reliability issues of service discovery and access in pervasive applications. In the following section we describe the key fault concerns of service discovery and the challenges thereof, while addressing them in infrastructure-less pervasive computing environments.

1.3 Fault Tolerant Service Discovery in Pervasive Computing

As already mentioned in the previous section, service discovery in pervasive computing introduces several critical challenges. In this section, we shall elaborate on the reliability concerns of a pervasive system. We shall discuss the fault situations that may thwart the normal service discovery operations in pervasive environments. We shall also discuss the challenges of employing traditional fault tolerance mechanisms to cope with service failures in pervasive computing environments.

We divide the faults in pervasive environments into two types – *infrastructure-related* and *software and service-related*. While the *infrastructure-related* faults cover hardware failures, such as, devices and network failures, *software and service-related* faults are concerned with failures of pervasive software and networked services.

- **Hardware-related Faults:** Pervasive computing is mostly composed of several devices characterized by low processing power, insufficient memory size and limited energy supply which are connected in an ad hoc manner with low bandwidth [7][38] and unreliable wireless networks. Due to their limited

resource supply, pervasive devices are prone to failures caused by resource outage. Energy depletion can make a device unavailable and low signal strength can lead to device disconnection, otherwise perceived as unavailability. Similar situations can arise out of the feeble wireless network connections which are intermittent and fault-prone. Moreover, device mobility can also result in device unreachability when devices go out of the wireless network range.

- **Software and Service-related Faults:** Contrary to the traditional distributed and mobile computing systems which use device-centric modeling, pervasive computing uses a service-centric system design. So, providing proper service to users based on their requirements is the key concern of pervasive computing. Currently, softwares for many pervasive devices are separately purchased from the market. They are poorly tested and not entirely reliable. So, they can crash at any instant rendering the device useless. Service failures in computing devices, however, can be caused by failure of the hosting device. Even if a service is not failed, still it can be unreachable due to network disconnection or device migration. Moreover, in a centralized directory-based service discovery system, services can be unavailable if the directory nodes fail or get disconnected. Service-related faults also comprise degradation of service quality over time. Services must continuously keep providing the same quality output and they should be self configuring and self-healing to protect against adverse environmental effects.

The different types of failures discussed previously leads to service unavailability and unreliability in pervasive computing environments. These issues must be properly addressed to ensure smooth service discovery and access operations. However, fault tolerance in pervasive computing environments, poses several challenges due to its special characteristics and design requirements.

Firstly, the dynamic nature of pervasive environment, resource-constrain of the participating devices, and the unreliability of wireless connection makes it hard to design robust fault tolerant mechanisms for them compared to the traditional distributed environment. Due to the resource-constrained nature of pervasive computing devices, it is hardly possible to use them as backup nodes as usually done in traditional distributed computing settings using static and resource-rich backup servers. Moreover, the pervasive devices used as backup nodes can fail, get disconnected or move away from the environment, in which case the replicated data is not available for recovery when faults occur.

Secondly, pervasive computing devices usually form a decentralized co-ordination which is extremely ad hoc in nature. Having any dedicated node for service information storage is unrealistic in such dynamic environments. For, the same reasons, service discovery in pervasive computing do not require collection of global knowledge. Devices can co-operate with other nearby devices in a localized manner to discover and access services as and when required. Fault tolerance mechanisms should also be developed to work through localized interaction as global infrastructure for fault tolerance is unsuitable for pervasive systems and, at the same time, incurs high energy and message costs.

Finally, pervasive environments are assumed to support users with minimal distraction and in an invisible manner. It is necessary for the system to find an alternative service provider autonomously when a device crash or disconnection renders a service inaccessible.

We have proposed a model which can address the above challenges and can support reliable service discovery in pervasive computing environments. In the next section, we shall discuss our contributions in detail.

1.4 Contribution of the Dissertation

The objective of this research is to study the fault tolerance related problems in service discovery and access in infrastructure-less pervasive computing environments and to provide innovative and cost-efficient solutions to address those problems. The dissertation makes several research contributions to achieve the afore-mentioned objective, which consists of the development of a framework for fault-tolerant service discovery and access. Based on that framework we have developed two mechanisms to support reliable service discovery and to provide seamless service access for mobile users in pervasive environments. Figure 1-1 shows how all of our works seamlessly fit into a reliable service discovery and access framework and present a coherent picture of our complete research. Here, we briefly describe our research contributions. A more detailed description of the framework will follow in Chapter 3.

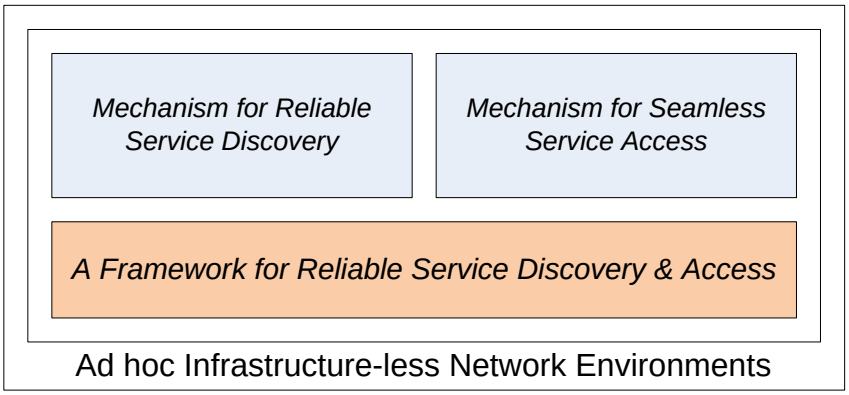


Figure 1-1: Block Diagram for Research Contributions

Due to the dynamic nature of pervasive environments and the resource-constrain of mobile nodes, reliable service discovery and access operations must be accompanied with stable directory nodes. But as we have already assumed an infrastructure-less network environment, it is hard to introduce fixed and centralize service directory. To cope with these limitations, we have proposed to elect top K-weighted nodes, from all

the nodes in the environment, to perform as directories. The node weights can refer to any required attribute of a node, e.g. remaining battery life, memory size, processing capacity, etc. Together, the selected K directory nodes are called the directory community and the weight-based election will ensure that the elected directories are more reliable and less fault-prone. Since we use this directory community as the basis for our research in fault-tolerant service discovery and access in infrastructure-less pervasive computing environments, we call this a directory community framework. The framework consists of the directory community structure along with a suite of protocols for providing reliable service discovery and seamless service access supports for mobile users in pervasive computing environments. Below we briefly describe our proposed mechanisms for reliable service discovery and access.

The reliable service discovery mechanism uses the directory community structure in order to ensure network-wide service availability in presence of directory node and service provider failures. The mechanism contains a quorum-based reliable service discovery protocol which works by dividing the directory community members into multiple quorums and then replicating the services registered with a single directory among its quorum members. This approach ensures controlled replication, minimal service discovery and update cost, and at the same time guarantees network-wide service availability by quorum intersection. Our simulation and testbed experiment results show that our protocol is fault-tolerant, message-efficient, and can cope with dynamic and frequent topological changes.

The seamless service access mechanism, also developed over the directory community, aims to provide reliable and continuous service access support for mobile users using a process called service handoff. Service handoff is required to find alternate matching services for users, in case the original service they were accessing becomes unavailable. The major concerns of service handoff are – reducing handoff frequency

and delay as well as balancing loads on resource constrained service provider nodes. We have developed three service handoff protocols for different scenarios in order to address the above issues. Simulation results show that our handoff protocols can reduce message and time costs and can achieve good load balance among different service providers.

1.5 Organization of the Dissertation

With a view to provide mobile users with reliable service discovery and access support in ad hoc pervasive computing environments, we initially propose a directory community framework. Later this framework is used to develop a fault tolerant service discovery mechanism and a reliable service access mechanism using service handoff. Below we describe the organization of the dissertation -

- **Chapter 2 (Background and Literature Review):** This chapter provides a basic introduction to the service discovery characteristics and issues in pervasive computing environments. We also study the existing research works in service discovery and analyze their advantages and disadvantages.

- **Chapter 3 (Directory Community: A Framework for Reliable Service Discovery and Access):** In this chapter we describe our directory community-framework which is developed to provide reliable service discovery and access support to mobile users in infrastructure-less pervasive computing environments. The requirements for a directory-based framework and the benefits of this framework in developing fault-tolerant service discovery and access protocols have been elaborately explained.

- **Chapter 4 (Formation of Directory Community):** This chapter discusses our underlying support structure for reliable service discovery in pervasive systems.

We propose a method to elect top K nodes as directories, based on their available resource contents. The community of K directory nodes will be later used for providing fault tolerance support in service discovery and access applications.

- **Chapter 5 (Quorum-based Reliable Service Discovery):** In this chapter we describe our proposed quorum-based reliable service discovery protocol developed over the directory community. The protocol works by forming a quorum of directory nodes and replicating service registration information among the quorum members.

- **Chapter 6 (Service Handoff Based Seamless Service Access):** In this chapter we present three different service handoff protocols which enable seamless service access for mobile users in a message and time efficient manner. The service handoff protocols are also developed over the directory community.

- **Chapter 7 (Conclusion and Future Directions):** This chapter concludes the dissertation with summary of our research works and discussions on future research directions.

Chapter 2

Background and Literature Review

This chapter discusses the background of service discovery research in general with special attention to the service discovery systems and solutions developed especially for pervasive computing systems. In Section 2.1 we discuss the basic building blocks of a service discovery system along with some essential system support components. We then briefly discuss the issues and currently available design choices to setup the background for classifying the existing service discovery protocols. We provide a classification of the existing service discovery solutions in Section 2.2. Section 2.3 and Section 2.4 describes the service discovery protocols for infrastructure-based and infrastructure-less pervasive environments, respectively and elaborates their architectural designs, fault tolerance mechanisms, and mobility management techniques. We conclude the chapter with a tabular representation of prominent service discovery protocols containing their design and operational descriptions.

2.1 General Components of a Service Discovery System

Service discovery applications are developed to enable users reuse service modules provided by other users and devices. Existing service discovery systems can be

classified according to the network type and system architecture of the service discovery system.

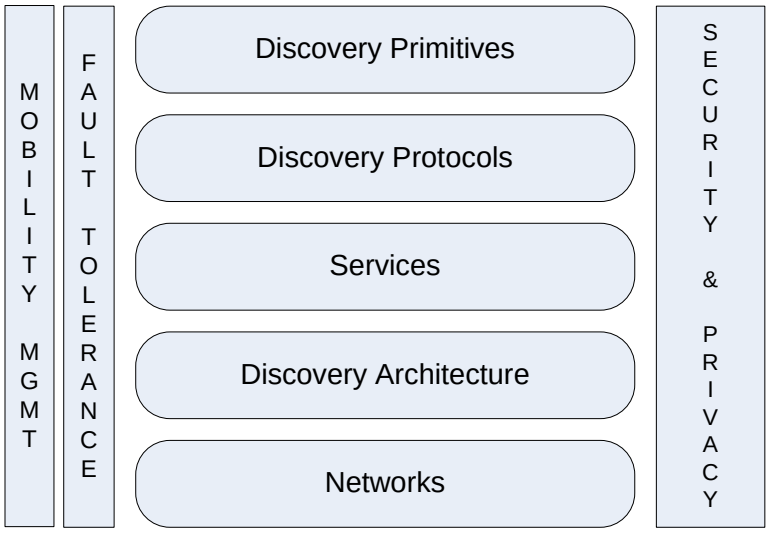


Figure 2-1**:** General Components of a Service Discovery System

In this section we shall discuss the major components of a service discovery system. As shown in Figure 2-1, the lowest layer of a service discovery system consists of the underlying network structure. Above that we have the discovery system architecture, which can be either two-party-based or three-party-based depending on the absence or presence of a service information registry or directory along with service providers and service users. The next higher layer consists of a pool of services provided by multiple devices in the network. Service discovery is facilitated by the use of various discovery protocols designed for different network structures and discovery models. Users, human or devices alike, can discover services or access them using different primitives provided by the discovery protocols. Service discovery systems also require fault tolerance supports as well as measures to protect security and privacy of the users. Due to the dynamic nature of pervasive environments, service discovery systems should also provide support to handle user and device mobility. We consider all the horizontal boxes as core components of a generic service discovery system and the vertical boxes as

essential system support service components. Different protocols may choose to implement the system support modules depending on the application and user requirements, but they are necessary for any industry-standard service discovery protocol.

We list some of the major issues (Table 2-1) and concerns that are associated with different modules of a generic service discovery system and are required to be addressed while designing such a system.

Table 2-1**:** General Issues for Service Discovery

CORE COMPONENTS	**Discovery Primitives**	Discover and Access
	Service Discovery Protocols	*Discovery*: Approach (Pull / Push), Service Information State (soft / hard), Scope, Selection Policy (Automatic / manual),
		Access: Invocation Policy, Usage Policy (Lease-based / completely released)
	Services	Naming, Attributes
	Discovery Architecture	Directory-based (Centralized/ Distributed) / Directory-less
	Network	*Connection Type*: Wired / Wireless (single / multi hop)
		Dynamics: Static or Mobile
SYSTEM SUPPORT COMPONENTS	**Fault Tolerance**	*Types of Faults*: Hardware-related (Device & Network), Software-related (Software & Service)
	Mobility Management	Network Partition, Unreachability of devices
	Security & Privacy	Securing Privacy of service provider and user

We briefly describe some of the issues before classifying the existing service discovery protocols based on their design and characteristics.

2.1.1 Network Structure

Based on the nature of the underlying network, service discovery protocols adopt different designs. Wired networks consist of resource-rich and static computing devices connected through high-bandwidth network cables. Examples of these types of systems are enterprise networks. Wireless networks, on the other hand, can be employed either in static or in mobile settings. Static wireless networks mostly contain high bandwidth network backbone infrastructure, which supports the networking need of the participating computing entities. Mobile wireless networks, however, lack any such infrastructure support. They create an ad hoc composition of multiple resource-constrained portable and handheld devices connected by unreliable and intermittent wireless connectivity. As stated in Section 1.1, wired networks and static wireless networks, which have some common properties, are together called as infrastructure-based networks. Mobile wireless environments, on the other hand, are called infrastructure-less or ad hoc and are more difficult to handle. Pervasive computing systems, though can adopt either infrastructure-based or ad hoc network environment, or a hybrid of them, they are characteristically more close to the ad hoc wireless environment due to their limited resource availability, extreme dynamism and unreliable network connections.

2.1.2 Service Discovery Architecture

Service discovery protocols adopt either a directory-based or a directory-less discovery model. For the directory-based model there is a dedicated directory node (SD) along with the service providers (SP) and service consumers or clients (SC). The directory node maintains service information and processes queries and announcements. Some directories provide additional functionality. For instance, Ninja SDS [49] directories support secure announcements and queries. The directory-based model is

more suitable for environments with hundreds or thousands of services. The directory-less model, on the other hand, has no dedicated directory. When a query arrives, every service (SP) processes it. If the service matches the query, it replies. When hearing a service announcement, a client (SC) can record service information for future use. The non-directory-based model is suitable for simple environments such as individual homes where the services are relatively few.

Directory-based systems are usually more efficient and scalable than directory-less ones. Based on the number of services and the size of the network, directory-based systems can use a single centralized directory or multiple directory nodes distributed across the network in strategically important locations. Depending on the organization of the directory nodes, directory-based models can be distinguished either as flat or as hierarchical. In a flat directory structure, directories have peer-to-peer relationships. For example, within an INS sub-domain, directories have a mesh structure: a directory exchanges information with all other directories. Salutation [86] and Jini [50] can also adopt flat structure. On the other hand, a hierarchical directory structure follows the DNS model in which information, advertisements and queries are propagated up and down through the hierarchy. Parents store information of their children and thus in this type of system, the root node may possibly become a bottleneck. Examples include Rendezvous [26] and Ninja SDS [49], both of which have a tree-like hierarchy of directories.

2.1.3 Services and Service Discovery Protocols

Services are identified by their names and discovered by matching their attributes. So, it is very important to have easily discoverable names and attributes for services. Some protocols choose user friendly service naming (e.g. Jini [50]), where as others used some standard naming template (e.g. Rendezvous [26]).

Service discovery protocols provide users the means to automatically discover networked services. Over the past few years, many organizations have designed and developed service discovery protocols. Examples in academia include the Massachusetts Institute of Technology's Intentional Naming System (INS) [1], University of California at Berkeley's Ninja Service Discovery Service (SDS) [49], and IBM Research's DEAPspace [70]. Major software vendors ship their service discovery protocols with their current operating systems— for example, Sun Microsystems' Jini Network Technology [50], Microsoft's Universal Plug and Play (UPnP) [95], and Apple's Rendezvous [26] (currently known as 'Bonjour'). Other organizations have also proposed discovery protocols standards, including Salutation Consortium's Salutation protocol [86], Internet Engineering Task Force's Service Location Protocol (SLP) [42], and Bluetooth Special Interest Group's Bluetooth SDP [14].

The methods of exchanging service discovery and registration information among clients, services, and directories are basically of two types – active / pull-based / query-based discovery and passive / lazy / push-based / announcement-based discovery. In the query-based approach, a party receives an immediate response to a query and does not need to process unrelated announcements. Multiple queries asking for the same information are answered separately. In the announcement-based approach, interested parties listen on a channel. When a service announces its availability and information, all parties hear the information. So in this approach, a client might learn that the service exists and a directory might register the service's information. Many protocols support both approaches.

2.1.4 System Support Components

After we have discussed the prime issues associated with the core components of service discovery operations in general, we also discuss the supportive operations that

are important for service discovery in real pervasive environments. The first one of this is fault-tolerance, the second one is mobility management while the third and final one is support for security and privacy.

Fault Tolerance and Mobility Support

In Section 1.3, we have described the basic fault issues for service discovery in pervasive computing. Existing service discovery protocols mostly consider crash of service providers or directory nodes which gives rise to service unavailability. This type of fault requires redundancy support to keep the operation ongoing if a directory node fails. Service unavailability can also arise due to user or device mobility and network disconnection. Mobility is a crucial issue for infrastructure-less pervasive environments and requires special attention to cope with the challenges.

When a service becomes unavailable it must be deleted from the directory or registry in order to keep the service information up-to-date and freeing the memory of stale service information. This is achieved by maintaining a soft service state which is required to be renewed at specified intervals by the service provider. In case the service is unavailable, the service state cannot be renewed, which leads to the deletion of the service from the registry. A hard service state, on the other hand, does not require to be renewed by the service providers. However, the directories must poll the service providers to find out whether the service information is up-to-date.

Another way of doing away with accumulated service usage information in service providers is by providing lease-based service. In lease-based service usage, service users must renew their lease with the service providers before the lease gets expired or the user fails or leaves the environment. Otherwise, the service providers simply delete the service states associated with the user and free the memory space. The alternative way is to explicitly release a service for a user and maintain the service execution information.

Security and Privacy Support

Security is another important issue required to be addressed for service discovery in pervasive computing. Since, users might need to interact with possibly unknown devices acting as directories and service providers in different environments, none of the parties involved is keen to take the first move for the fear of breach of privacy. So, it is important to maintain proper security and privacy at all times.

2.2 Classification of Service Discovery Protocols

Existing service discovery protocols can be grossly classified based on their underlying network structure and the discovery infrastructure built over that. In the Figure 2-2 we give a classification of existing protocols.

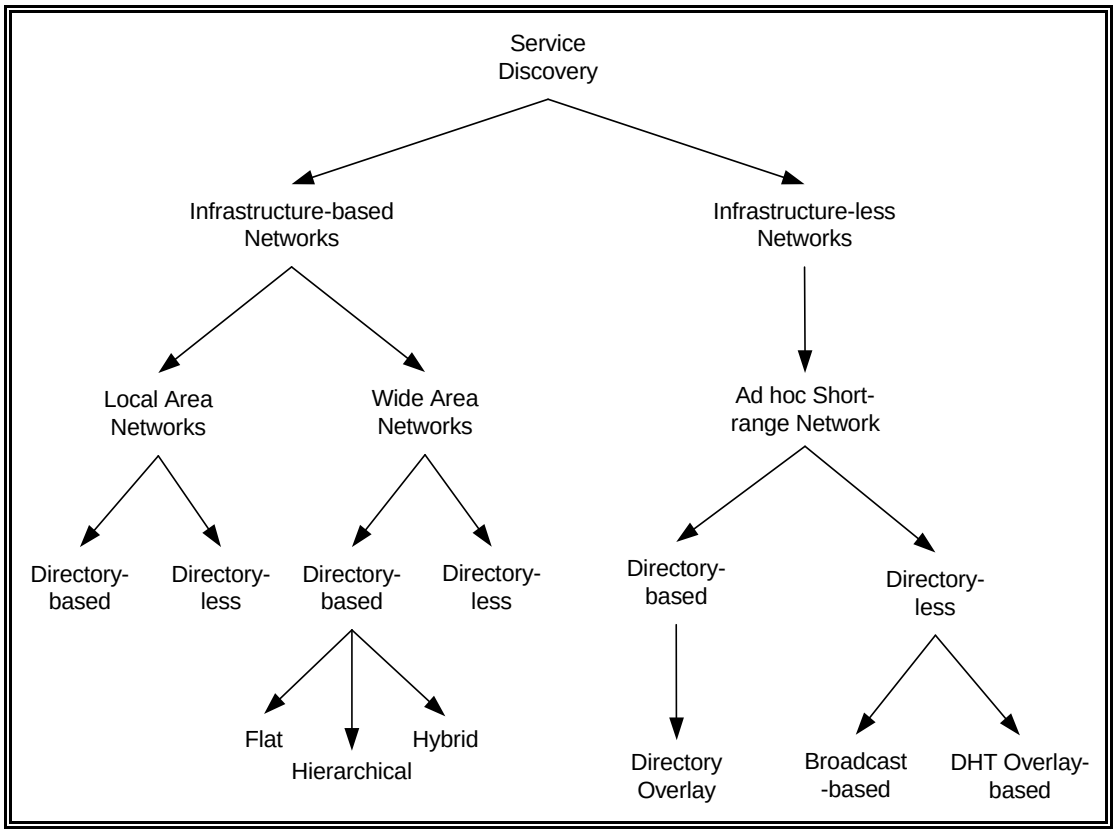


Figure 2-2**:** Classification of Existing Service Discovery Protocols

2.3 Service Discovery in Infrastructure-based Networks

Infrastructure-based networks are characteristically either wired networks or wireless networks with network backbone support. Service discovery protocols (SDP) for infrastructure-based networks have been developed either for limited area networks (LAN) or for wide area networks (WAN).

2.3.1 SDPs for Local Area Networks

As a LAN is covered by a single administrative domain, DHCP services can be used for them. Also they contain resource-rich devices which are mostly static and provide high-bandwidth network support. Enterprise environments and smart office and home environments can be considered as ideal examples of these types of systems. Some of the well-known protocols for this environment are Jini [50], UPnP [95], SLP [42], and FRODO [90].

Jini is a Java-based service discovery protocol introduced by Sun Microsystems where services are represented as Java objects. Jini adopts the directory-based discovery model where all the service information is centrally maintained in the registry or lookup servers. Universal Plug and Play or UPnP, on the other hand, is a Microsoft-initiated standard that extends the Microsoft Plug-and-Play peripheral model. UPnP uses a directory-less model and enables peer-to-peer network connectivity of intelligent appliances, wireless devices, and PCs of all form factors. The Service Location Protocol or SLP has been designed for TCP/IP networks and can choose to operate using a directory-based or a directory-less model. FRODO is a directory-based service discovery protocol for home networks where directory nodes are elected based on available resources.

Another directory-based protocol, named Salutation [86], has been developed by IBM. This can be used for any network and can perform either in P2P mode or in centralized manner. Salutation is useful to solve the problems of service discovery and utilization among a broad set of appliances and equipment and in an environment of widespread connectivity and mobility. The Salutation architecture defines an entity called the Salutation Manager (SLM) that functions as a directory of applications, services and devices, generically called "Networked Entities". The SLM allows networked entities to discover and to use the capabilities of the other networked entities.

2.3.2 SDPs for Wide Area Networks

Service discovery protocols for wide area networks have separate design concerns. These types of networks usually contain large number of devices and services that need to be managed and so, the developed service discovery protocols must be highly scalable. Also WANs do not have support for broadcast or multicast mechanisms. Moreover, to ensure network-wide service availability, they must have multiple replicas for each service. So, there must be a trade-off between the need to maintain consistency among service replicas and the generated network traffic. Also, the requested service should be available to the user within optimal time and by using minimum number of messages. Some of the well-known protocols for infrastructure-based wide area networks are SSDS [49], CSP [62], INS/Twine [8], Superstring [82], and GloServ [5]. All these protocols use a structured distribution of directory servers which stores the service information. The directory structure can be well classified into three modes - flat, hierarchical or hybrid.

INS/Twine forms a flat directory structure which is a peer-to-peer overlay network of resolvers constructed by means of distributed hash tables (DHT). INS/Twine uses the Chord [89] DHT system. Another example is One Ring Rules Them All [19] which uses

structured P2P overlays as a platform for service discovery and implements over Pastry [84] DHT system. The infrastructure proposed relies on a universal ring that all participating nodes are expected to join. The major advantage of using DHT based protocols is the efficient lookup which usually takes O(log(N)) hops, where N is the number of nodes in the overlay network. However, DHT systems do not take into account the actual physical distance between the overlay nodes and also they are costly in terms of resources. Due to this, using DHT based protocols for pervasive environments can incur higher costs. Contrary to the peer-to-peer directory overlay structure, SSDS, CSP and GloServ form a hierarchical structure of directory nodes.

With a view to tackle the disadvantages associated with either type, Superstring, combines both the peer-to-peer and hierarchical topologies and provide a hybrid model. It utilizes a flat topology to discover top-level nodes that specialize in a particular kind of service. From this top-level node, a hierarchy is created which reflects the hierarchical structure of service descriptions and helps to resolve user queries. Another research based on hybrid directory structure is the Project JXTA protocols [93] which establish a virtual network overlay on top of existing physical network infrastructure. They store the service information in rendezvous peers. This approach combines DHT methods with a random walker that checks for non-synchronized indices. In order to avoid expensive network traffic, the resolver nodes are not required to maintain a consistent distributed hash index. Instead, a limited-range walker is used to walk the rendezvous from the initial DHT target.

2.3.3 Fault Tolerance and Mobility Management Mechanisms

In infrastructured networks, the existing service discovery protocols are mostly directory-based. So, fault tolerance supports are required to safeguard the system against possible failures of directory nodes. Usual approach to achieve this is by redundancy.

For central storage, such as, FRODO, there is a backup of the directory node which takes over in case the central directory fails. For distributed storage, e.g., INS/Twine, JXTA, and One Ring Rule Them All, multiple copies of service information are maintained in different servers. When one or more fails, the rest can still answer the queries.

Service information maintenance on the face of mobility is tackled by many protocols in these types of networks. Jini uses a lease-based mechanism for service access by users. SLP and UPnP, however, enforce expiry time for service registrations and advertisements, respectively, and GloServ requires periodic renewal of service registrations, all in order to remove outdated service information. SSDS maintains service information state as soft states.

2.4 Service Discovery in Infrastructure-less Networks

Infrastructure-less networks are highly dynamic and consist of multiple resource-constrained and possibly mobile devices. They are connected through weak wireless connectivity and form a mobile ad hoc network. In these environments, we assume that the participating devices provide some services to their peer devices which can be discovered and accessed through carefully designed service discovery protocols. Depending on where the service information is maintained, the existing service discovery protocols for ad hoc environment have been classified into directory-based and directory-less models. We first discuss the existing directory-less service discovery protocols and analyze their features. Next we shall elaborate on the available directory-based protocols.

2.4.1 Directory-less SDPs

In the directory-less service discovery protocols, the service information is stored with the service providers themselves. There are two distinctly identified methods for directory-less service discovery in ad hoc networks. The first method works by broadcasting service information as well as service requests and the second approach works by building DHT-based P2P overlay [53][103][76][49][32][18] for mobile ad hoc networks (MANET). We first describe the broadcast based method followed by the DHT overlay approach.

Broadcast-based Service Discovery

Broadcast-based service discovery can adopt either push or pull model. In a push-based discovery model, service advertisements are distributed by the service providers to all the nodes in the network. A pull-based discovery model, however, necessitates a service requestor to broadcast their service request to other nodes until a matching service is found. The broadcasting nature of this model is grossly unsuitable for the mobile ad hoc networks due to their high demand of bandwidth and energy. So, these protocols can only be used in small scale networks. Some of the protocols which use broadcast policy are Bluetooth [14], DEAPspace [70], Allia [77], GSD [20], DSD [21], and Konark [47].

Among all the protocols mentioned above, Bluetooth and DEAPspace are designed for single hop ad hoc networks where the rests are for multi-hop networks. Bluetooth has been developed following a client server model whereas, DEAPspace follows a peer-to-peer architecture. DEAPspace is a push-based decentralized discovery protocol in which each node maintains information about its known services and periodically exchanges its known service list with the neighbors via broadcast. Similar policy for

broadcast is also used in Konark which allows each node to store service information from itself and other known services. However, Konark is multihop in nature and supports both pull and push based advertisement and discovery using multicast. The broadcast in DEAPspace significantly increases the message overhead and can lead to multicast storm. To cope with this problem, Konark proposes a service gossip algorithm which suppresses repeated message delivery by caching the service information and also the multicast is performed at random intervals. In GSD, every node stores service advertisement from any other node within a maximum of N hop distance, known as advertisement diameter. Allia and DSD, on the other hand, allow nodes to advertise their services only within their transmission range. In Allia, nodes which cache service advertisements form an alliance with the advertising node. Difference between Allia and DSD is that, in DSD, nodes which stores advertisements from their neighbors, can also forward the advertisement to other nodes, unlike in Allia, based on the forwarding policy.

Some other protocols in this category are Wu and Zitterbart [101], Cheng and Marsic [25] and Varshavsky et al [96]. These protocols support cross-layer types of service discovery. Wu and Zitterbart is a directory-less P2P protocol based on DSR routing protocol in which every node caches service advertisements and performs both pull and push based discovery. Cheng and Marsic, on the other hand, is based on on-demand multicast routing protocol (ODMRP) in which nodes cache service advertisements depending on their interest. Varshavsky et al proposes a protocol which has two main components - a routing protocol independent Service Discovery Library (SDL) and a Routing Layer Driver (RLD). SDL function is to store information about the service providers. RLD, which is closely coupled with the MANET routing mechanism, is used to disseminate service discovery requests and advertisements. Each node has the stack containing SDL and RLD and form a P2P networking with other nodes.

DHT Overlay-based Service Discovery

There are several MANET-oriented DHT systems [76][104][32] which integrate DHT with different ad hoc routing protocols to provide indirect routing in MANET. Ekta [76], MADPastry [104], and CrossROAD [32], each integrates Pastry [84] with DSR [51], AODV [75], and OLSR [28], respectively, to share routing information between network layer and application layer. An opposite approach is adopted by virtual ring routing (VRR) [18] which is a network-layer routing protocol inspired by overlay routing on DHTs and can significantly reduce traffic compared with broadcast-based schemes. Ekta implements a service discovery protocol and there are other DHT overlay based service discovery protocols [53][103] for MANET. DHT-based systems for service discovery, however, have certain drawbacks. Ekta and VRR construct a DHT substrate without taking into account the actual physical distance between nodes. This can cause undesirably long search latency and deterioration of success ratio of service discovery with the growing network scale. On the other hand, MADPastry proposed a clustering method which groups the overlay nodes according to their physical distance. But, MADPastry routes information using location-dependent addresses. These identifiers can change with mobility and node failures, so, it always requires mechanisms to look up the location of a node given a fixed identifier which is costly.

2.4.2 Directory-based SDPs

Given the resource-poor nature of the devices and their mobility, it is difficult to choose single centralized directory nodes. To cope with this limitation, directories are dynamically selected from mobile nodes considering their available resources, such as, processing power, memory size, battery life, or node coverage, etc. It is true that, dynamic directory assignment incurs extra overhead to the network because, directories should be selected and their identities should be informed to the rest of the network

nodes. Moreover, directory nodes must be constantly available on the face of node failures and dynamic topology changes and network partitions. Even with these difficulties, a directory-based system proves to be more scalable and fault tolerant than a directory-less one in the infrastructure-less environments. Moreover, using directories will most certainly decrease the discovery delay and will enhance load balancing among service providers, so as to reduce the load on individual services and enhance the overall service discovery performance. Examples of directory-based service discovery protocols for ad hoc wireless networks are - Service Rings [58], Lanes [59], DSDP [60], Tyan et al [94], Sailhan et al [85], Kim et al [57], and Splendor [105].

The basic approach followed by these protocols is the same. They select some nodes as directories based on some parameters and then form an overlay or backbone network connecting those nodes. Here we give a brief account of the protocols named above. Service Rings forms an overlay of nodes which are physically close and offer similar services. This structure, built over the transport layer, is called service ring. Each service ring provides a service access point (SAP), which acts like a directory and through which services provided by any of the members of a ring can be accessed. SAPs of different rings connect with each other to form a hierarchical structure. The protocol Lanes is inspired by the content addressable network (CAN) protocol, for wired P2P networks. In Lanes, nodes are grouped together to form a linear structure, called lanes. Each node in a lane contains same service information and share the same anycast address. Multiple lanes are loosely coupled together. DSDP selects certain directory nodes in the network, based on available resources, to form a dominating set, or a virtual backbone. The backbone of directory nodes is then used for both service discovery and routing. Tyan et al., proposes a protocol in which the network is divided into hexagonal grids, each having a gateway. The gateway nodes are used for routing and work as directory nodes. The connected overlay of gateways forms a virtual backbone. Kim et al,

proposed a volunteer node based protocol where volunteers are relatively stable and resource rich nodes and form an overlay structure with other volunteers. The volunteers in fact act as directory nodes.

Sailhan et al., proposed a protocol for large-scale mobile ad hoc networks in which multiple directory nodes distributed across the network interconnect to form a backbone. The directory nodes are so deployed that at least one directory is reachable in at most a fixed number of hops, H, known as the vicinity of the directory. Directories store service information available in their vicinity. Their protocol builds a hybrid network bridging mobile ad hoc networks and infrastructure-based networks, where some nodes have the same network interface, where others hold several network interfaces, and act as gateways with other networks.

2.4.3 Fault Tolerance and Mobility Management Mechanisms

Fault tolerance in infrastructure-less environments is more difficult than in infrastructured networks. The dynamic nature of the environment, resource-constrain of the participating devices, and the unreliability of wireless connection makes it hard to design robust fault tolerant mechanisms for them compared to the traditional distributed environment. Fault tolerance requires withstanding failure of directory nodes. INS [1] and LANES cope with directory failure by maintaining multiple copies of service information at different nodes. Most of the directory-based protocols replace failed directory nodes with newly selected ones.

Mobility management, however, is very challenging in ad hoc environments. Frequent node mobility renders the topology unstable and disconnections give rise to inconsistency in service information. In order to maintain consistency of service and route to service information, either a proactive or a reactive method has been adopted by

existing protocols. In a proactive approach, the participating nodes periodically exchange messages to update information. Example is, when a service provider periodically sends advertisements to update its location. A reactive method, however, updates information based on triggering of certain events. So, if a user finds out that a previously cached service is unreachable, then it seeks for new service information.

Directory-less service discovery protocols cope with node mobility by adjusting the service advertisement rate and the diameter of announcements. For example, GSD implements a small advertisement time interval for highly dynamic environments, opposed to a larger value for comparatively stable networks. The advertisement diameter (in number of hops) is also regulated depending on different mobility situations. Similarly, Allia controls the frequency of advertisements and the diameter of the alliance considering the mobility of nodes.

Most of the directory-based protocols for ad hoc service discovery require special mechanisms to maintain the directory structure – backbone or overlay. The job of these algorithms is to ensure smooth operation by handling node joining or leaving scenarios, broken connections, network partition and partition merges. Service Rings, Lanes, DSDP, Tyan et al, and Sailhan et al, all proposes similar mechanisms.

2.5 Comparison of Existing Service Discovery Protocols

In this section, we compare the major service discovery protocols in terms of their underlying network model, architectural design, communication model and adopted fault tolerance methods.

Table 2-2: Comparison of Existing Service Discovery Protocols

SDP	Network Model	Discovery Architecture	Communication Model	Fault tolerance Policy
Jini	Enterprise network	Directory-based, Centralized	Wired or infrastructured wireless	Lease mechanism for granting access to services
UPnP	Enterprise network	Directory-less, P2P	Wired or infrastructured wireless	-Expiry time for advertisements -"Device unavailable" notification
SLP	Enterprise network	Directory-based, or Directory-less	Wired or infrastructured wireless	- Lifetime for service registrations
Salutation	Any Network	Directory-based, Centralized or P2P		Periodic availability check of services
FRODO	Home network	Directory-based, Centralized	Wired or infrastructured wireless	-Central election and re-election (**FT**) -Recovering from Central/Backup failure (**FT**) -Soft state for 300D devices, periodic poll of 3D devices
SSDS	Wide-area network	Directory-based, Hierarchical	Wired or infrastructured wireless	Soft state of service announcements
CSP	Wide-area network	Directory-based, Hierarchical	Wired or infrastructured wireless	Frequent updates for intra-domain movements; no updates in the global network
INS/ Twine	Wide-area network	Overlay network of directories (a.k.a. resolvers) which form a DHT	Wired or infrastructured wireless	-Each strand is stored on multiple nodes (**FT**) -Hybrid state management scheme
Super-string	Wide-area network	Directory-based DHT overlay	Wired or infrastructured wireless	None
GloServ	Local & Wide-area network	Directory-based, Hybrid (mix of Hierarchical & flat)	Wired or infrastructured wireless	Registration is periodically renewed
Service Discovery Protocols for Ad hoc Networks				
Bluetooth	Ad hoc network	Directory-less Client-Server	Wireless Single hop, Request/ Response -based	Implicit (no caches are maintained)
DEAPspace	Wireless ad hoc network	Directory-less, Unstructured P2P	Wireless Single hop, Broadcast-based	Nodes broadcast their entire view – rapid convergence
Allia	Ad hoc network	Directory-less, Unstructured P2P	Wireless Multi-hop, Broadcast-based	Adjusting advertisement rates and alliance diameter based on mobility of nodes

GSD	Mobile ad hoc network	Directory-less, Unstructured P2P	Wireless Multi-hop, Broadcast-based	Adjusting advertisement time interval and advertisement diameter based on mobility of nodes
DSD	Mobile ad hoc network	Directory-less, Unstructured P2P	Wireless Multi-hop, Broadcast-based	Adjusting advertisement diameter and rates based on mobility of nodes
Konark	Wireless ad hoc network	Directory-less, Unstructured P2P	Wireless Multi-hop, Broadcast-based	Servers periodically announce their services and specify lifespan of advertisements
Ekta	Mobile ad hoc network	Directory-less	Wireless Multi-hop,	None
Service Rings	Mobile ad hoc network	Directory-based, Structured overlay	Wireless Multi-hop	-Periodically checking for consistency using *RingCheck* message -Algorithms for repairing broken rings, network partition and reintegration
Lanes	Mobile ad hoc network	Directory-based, Structured overlay	Wireless Multi-hop	-Proactive maintenance -Algorithms for node login/logoff, broken connections, network partition and reintegration
DSDP	Mobile ad hoc network	Directory-based, Flat structured overlay	Wireless Multi-hop	-Periodic service registration -Algorithms for backbone maintenance
Tyan et al.	Mobile ad hoc network	Directory-based, Flat structured overlay	Wireless Multi-hop	Periodic service advertisements and electing new gateway node when the existing gateway moves away
Sailhan et al.	Mobile ad hoc network	Directory-based, Hierarchical structured overlay	Wireless Multi-hop	Periodic service advertisements
Kim et al.	Mobile ad hoc network	Directory-based, Flat unstructured overlay	Wireless Multi-hop	Adjusting advertisement diameter and rates, electing new volunteers
Splendor	Mobile ad hoc network (public environment)	Directory-based	Wireless Multi-hop	-Soft state storage of service information -Hard-state storage of services represented by proxies

Chapter 3

Directory Community: A Framework for Reliable Service Discovery and Access

This chapter describes the directory community framework which refers to a group of service discovery directory nodes, called directory community, along with a suite of protocols and algorithms for reliable service discovery and access over ad hoc network environments. Section 3.1 presents a generic description of our directory community framework. In Section 3.2, the directory community creation has been discussed with necessary design principles, challenging issues and our proposed solution technique. Section 3.3 and Section 3.4 discuss the design principles and our proposed solution approaches to provide reliable service discovery and access operations using the directory community framework. Finally, Section 3.5 concludes this chapter by summarizing our contributions.

3.1 Generic Framework Structure

As already mentioned in Chapter 1, the existing fault tolerance support for service discovery operations in ad hoc infrastructure-less pervasive environments is inadequate. The primary objective of our research is to ensure reliable service discovery and seamless service access for mobile users in heterogeneous and fault-prone pervasive

computing environments. A large number of portable and embedded devices in these environments work collaboratively in a purely ad hoc manner and without any central control. This necessitates a localized and bottom-up interaction model which will achieve fault-tolerant service discovery minimizing both message and energy costs. To achieve this goal, we prefer a directory based service discovery solution to a directory-less one as the former appears to be more robust and fault tolerant than the latter. Moreover, directory-less or ad hoc service discovery protocols mostly resort to expensive message broadcasts which is not practical and not even feasible, for pervasive computing environments with large number of dynamically changing devices.

But, even if we want to use a directory-based model, still the dynamic resource-constrained and error-prone nature of pervasive environment prohibits the use of single, centralized and dedicated directory node. To cope with this limitation we propose an ingenious solution namely, the directory community framework. The directory community is composed of a set of directory nodes in an ad hoc network environment and a suite of protocols and algorithms for these directory nodes to collaboratively provide reliable service discovery and seamless service access for mobile users.

The directory community is used as a basic infrastructure over which many different service discovery related protocols have been developed. To make it more clear, the directory community provides application developers with a sense of reliability in a heavily dynamic and inherently unreliable network environment. The directory community is robust enough to carry on with fault-tolerant service discovery operations even when one or more directory nodes fail. However, proper functioning of the directory community depends on the judicious formation and timely maintenance of the structure.

The directory community framework shown in Figure 3-1 has three different layers. The lowest layer concerns about the formation of a robust directory community. After the directory community has been successfully formed, we have developed two higher layers intended to provide reliable service discovery and access supports for mobile users. However, application developers can use the directory community to develop many kinds of fault-tolerant service management protocols, including service discovery, service access, service composition, etc.

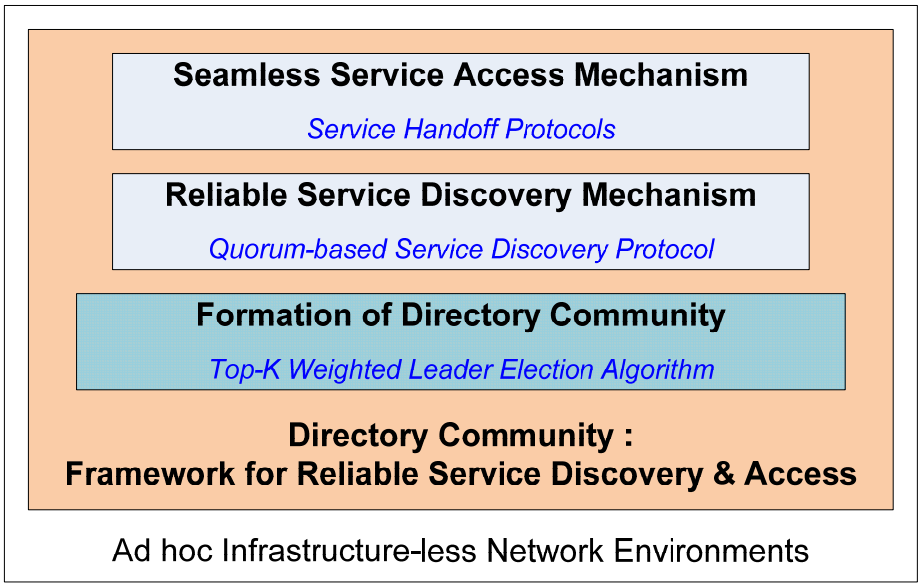


Figure 3-1: Directory Community Framework

For the directory community we have adopted a common system model. All the protocols and algorithms developed using the directory community share the same system model for their functioning. Since dynamic and ad hoc pervasive computing environment shares great similarities with mobile ad hoc networks, we have modeled the underlying network of a pervasive environment as a MANET. MANET nodes are mobile, resource-constrained and prone to disconnection.

The directory community framework is set up over a MANET that consists of a set of n (n>1) mobile nodes, each of which has a unique ID. The nodes communicate by

sending and receiving messages through wireless channels. Whether two nodes are neighbors, i.e. they are directly connected, is determined by the signal coverage range and the distance between the nodes. Each node is a router and the communication between two nodes can be multiple hops. We assume variable message delay, so the message delivery is non FIFO. A node may fail by crashing, i.e. prematurely halting, but it acts correctly until it possibly crashes. A mobile node that crashes in a run is called a faulty node; otherwise, it is correct. A node only knows the IDs of its neighboring nodes. Each node has a weight associated with it. The weight of a node indicates its ability to be chosen as a directory node and can be any performance related attribute such as the node's battery power, memory size, computational capabilities etc. Two nodes may have the same weight value. Since, two or more nodes can have identical weights, in order to create a total ordering among all the nodes, we use the <*id, wt*> tuple. So, if more than one node has the same weight, then the one with the higher id is selected over the other. E.g., if there are two nodes p and q with tuples, <1, 10> and <2, 8>, respectively, then node p is the higher weight node. But, if p and q has tuples like <1, 10> and <2, 10>, respectively, then node q is selected over p.

In the next few sections, we shall discuss the different layers of the directory community framework along with the general design principles, challenging issues and our proposed solution approaches.

3.2 Directory Community Creation

As already mentioned, directory community consists of a group of directory nodes for service discovery and access in infrastructure-less pervasive computing environments, along with a suite of protocols and algorithms to facilitate those operations in a fault-tolerant manner. Creation of directory community has to follow certain standard design principles as described below.

Firstly, the directory nodes must be relatively stable, so that, the directory community achieves extra stability compared to the unstable nature of the usual MANET nodes. As mobile ad hoc networks consist of limited-resource devices characterized by low processing power, insufficient memory size, limited energy supply and low network bandwidth [7][38], resource-based election will ensure that the directories are more reliable and less fault-prone. So, to achieve stability, directory nodes can be selected considering relatively higher processing power, memory size, or battery life, etc. Also nodes with low mobility profile can be considered.

Secondly, due to the resource constrained nature of MANET nodes, the directory community formation must be achieved at a low message expense which implies consumption of less bandwidth and low processing overhead. Message cost can be significantly reduced if directory community creation is carried out by decentralized or localized interaction of neighboring nodes. This approach will be suitable for MANET as there is no central control in this type of network.

Thirdly, the directory community must be maintained periodically in order to be up and functioning. Failed directory nodes should be replaced quickly to continue with the ongoing applications. Since, the directory community is the heart and soul of the framework, the correct functioning of the upper layer protocols will be utterly dependant on the proper maintenance of the structure. Also, the maintenance overhead should be kept to a minimum.

Finally, the size of the directory community should be enough to cater to the needs of all the nodes in the network. So, depending on the size of the network, directory community size can be varied. If more nodes join the network, new directory nodes may need to be selected to join the community. On the other hand, if the network size shrinks

considerably, some directory nodes can be dropped to minimize the directory community maintenance overhead.

However, achieving the above design principles is challenging due to many reasons. Firstly, MANET nodes are resource-constrained. So, using them as service discovery directories will quickly reduce their capacity and will require re-election of new directory nodes. Secondly, mobile ad hoc networks are extremely dynamic and have unreliable wireless connectivity. Also, nodes can fail by resource depletion or can suddenly withdraw from the network. Due to all these possibilities, MANETs can suffer from dynamic topological changes caused by frequent network partitioning as well as partition merges. In these environments, creation and timely maintenance of directory community in a message efficient manner is extremely difficult.

Addressing all the above challenges is important while designing a directory community framework for reliable service discovery operations. Directory community can be created by multiple means. Possible approaches can be like - cluster formation considering relatively resource rich nodes, or leader election to chose directory nodes with requisite properties, or some other useful algorithms.

In this research, we have opted to create directory community by electing top K highest weight nodes in a MANET, where weight can be any resource-related attribute of a node and K is a ratio of the directory nodes to the total number of network nodes. We have proposed a distributed algorithm for directory community formation which works through localized interaction of participating devices. Considering the directory nodes as leaders, we have modeled the problem as a top-K weighted leader election problem. The problem is challenging and so far, there is no K-leader election algorithm designed for mobile ad hoc networks. Details of our solution approach will be found in Chapter 4.

After the directory community is successfully formed, we start to develop different fault-tolerant service discovery and access protocols over the directory community. In the following two sub-sections we shall describe our proposed protocols for reliable service discovery and seamless service access.

3.3 Reliable Service Discovery using Directory Community

In the heavily dynamic network environments, like MANET, services can suddenly become unavailable due to many possible failure situations. Service providers may fail or service directories may fail as well. While the former can be easily addressed by finding out alternative services matching user request, the latter is much more difficult to address. Directory failures in service discovery systems require all the services registered with the failed directory to re-register with other available directories in order to publish them and become available once more. For mobile users, service unavailability can also occur if the user moves out of the scope of the service provider.

Different protocols can be developed to address above problems. In this research, however, we address the directory failure problem and investigate the means of tolerating this type of fault while using our directory community. Reliability and fault-tolerance is achieved at the cost of replication. While developing reliable service discovery protocols using directory community we need to pay heed to the following design principles.

Firstly, the MANET nodes are resource constrained. So, degree of replication must be controlled in such a way that the replication and update related costs can be minimized. Moreover, the message costs of service discovery should also be controlled. Overall, the service availability amidst directory failures must be achieved at minimum extra overhead.

Secondly, frequent node mobility can result in network partitions as well as partition merges, thereby changing the network topology and also the number of directory nodes in a single network component. So, the number of directory nodes and service replicas may need to be altered from time to time. Moreover, the service availability will be affected if the service replicas are not available due to network partitioning. So, maintaining the reliable service discovery operations on the face of dynamic topological changes requires special techniques to cope with.

Finally, scalability is also an important issue in the dynamic MANET environments. With the increase in network size and with frequent joining and leaving of nodes, new directory nodes may need to be created, new service replicas have to be generated and old service replicas must be removed or updated.

Many different fault-tolerant service discovery protocols can be designed on the directory community satisfying the above design criteria. Services can be replicated in the entire directory community, or a within a small group of directory nodes which can synchronize the service information among each other. The second approach will certainly be cheaper but could be less reliable depending on the design. So, in order to strike a balance between cost and reliability, we have proposed a quorum-based reliable service discovery protocol for MANET using the directory community. In this approach, elected directory nodes form quorums among themselves and replicate services registered with them among its quorum members. This approach ensures network-wide service availability with minimal replication. Following the quorum intersection property, we can guarantee that if a service matching user request is available, the user can certainly find the service by forwarding a request only to its quorum members. This reduces service discovery cost. However, success of this approach depends on the continuous availability of the quorum nodes. Detailed description of the quorum-based service discovery approach is in Chapter 5.

3.4 Seamless Service Access using Directory Community

Service users in pervasive computing environments usually access services while on the move. Devices, static or mobile, often limit their access scopes by physical boundary (e.g., the room in which the device is active) or network hops (for multi-hop ad hoc networks). Mobile users may frequently move out of the scope of a service provider and experience disconnection. We call this scenario as service unavailability. Continuous service access can also be hampered when the service provider suddenly crashes or moves away from the user, or when the underlying network gets partitioned. Any mechanism for providing seamless service access support must address the following concerns.

Firstly, whenever a service becomes unavailable, it must be quickly detected and all users using that service should be handed over to another service provider which is providing the same service. If no matching service provider is available, any near matches can be used provided the quality of service falls within the user specifications. Otherwise, the user has to be notified about the service unavailability.

Secondly, the service execution consistency should be maintained while a change of service provider takes place. Service execution consistency indicates a continuity of the service execution states across multiple providers. So, proper check-pointing and recovery mechanisms are required for Also, the resumption of service execution at an alternate service provider must be very smooth once the service has been handed over to a different provider.

Thirdly, any service handover mechanism for MANET environment must be scalable. With the increased network dynamics and high volume of nodes, seamless service access can be often disrupted frequently requiring service handovers. Also the service

handovers must incur minimal costs. Otherwise, the increased number of handovers will quickly diminish the network lifetime.

Finally, in a MANET environment, service provider nodes are usually resource-constrained. So, while assigning a new user to a service provider, the load on the service provider must be considered. Overloaded nodes will soon fail due to resource depletion and that will require many more service handovers.

Existing service discovery solutions in pervasive computing rarely address the afore-mentioned issues. We want to use our directory community framework to provide seamless service access support for mobile users. We propose an approach, called service handoff, by which new matching service providers are automatically selected for a user once the original service provider becomes unavailable. The objective of service handoff is to maintain service execution consistency for service users and load balance for service providers. Our proposed service handoff approach has three different protocols depending on the initiator of the handoff. We shall describe our service handoff protocols in detail in Chapter 6.

3.5 Summary

In this chapter we have introduced our directory community framework. The framework has a collection of directory nodes, carefully chosen from all the nodes in an ad hoc network considering available node resources, along with a suite of protocols that aim to provide reliable and fault-tolerant service discovery and seamless service access supports to mobile users in these environments. We have initially introduced the layered structure of the directory community framework followed by detailed description of individual layers.

Chapter 4

Formation of Directory Community

This chapter describes formation of our directory community framework which is required for reliable service discovery and access over ad hoc pervasive environments consisting of multiple portable mobile devices connected through unreliable wireless networks. Section 4.1 presents a brief overview regarding the issues which need to be addressed in order to develop the directory community for mobile ad hoc networks (MANET). Section 4.2 to 4.7 contains detailed description of our directory community formation and maintenance techniques as well as the performance measurements. Finally, Section 4.8 concludes the chapter by summarizing our main contributions.

4.1 Overview

As already mentioned in Chapter 3, we create directory community by electing top K highest resource nodes in a MANET and formulate the problem as a top K leader election problem. Typically, the leader election problem is to elect a unique node to play a particular role [67]. Many distributed algorithms in ad hoc networks, such as mutual exclusion, synchronization, concurrency control, etc, require selecting one or more nodes to act as leaders. Leader election in traditional distributed environment has been well studied. Many solutions [2][71][92][4][9][39][74][35][29][45][97] have been

proposed to solve the leader election problem in traditional distributed environment. However, existing algorithms are not suitable for weighted leader election in mobile ad hoc environment due to the following challenges.

Firstly, given the resource constrain of mobile nodes any leader election algorithm for MANET must be message efficient which implies consumption of less bandwidth as well as lower processing overhead. Secondly, due to variable number of nodes and frequent network partitioning electing a fixed number K of leaders is not suitable. Instead, we should define K as a ratio of the number of leaders to the total number of nodes in the current network component. Finally, due to the network partitions and node failures, it is impossible to elect globally unique leaders. So, we have to elect different sets of leaders for different network components and same set of leaders for two or more components merging together.

So far, there is no K-leader election algorithm designed for mobile ad hoc networks. Several leader election algorithms [44][67] have been proposed for ad hoc networks to select a single leader. These algorithms are designed to perform id-based node election and cannot be modified to perform weight-based K-leader election since two nodes may have the same weight value. More importantly, node weights change frequently depending on the operating conditions, and so the elected leaders can soon be incapable of hosting specific services. Several weight-based clustering algorithms [11][12][22] have been proposed for mobile networks, but they elect cluster heads only within single hop neighborhood and cannot be adapted for multi-hop election. Leader election algorithm proposed in [97] considers weight-based single leader election and outlines a method to extend their algorithm to elect K leaders, but they do not provide details of the suggested method which is not as message efficient as ours.

Another similar type of problem is supernode selection which involves the selection of a subset of the peers in a large scale, dynamic network to serve a distinguished role. The specially selected peers must be well-dispersed throughout the network, and must typically fulfill additional requirements such as load balance, resources, access, and fault tolerance. Supernode selection shows up in many peer-to-peer and networking applications. For example, in peer-to-peer file sharing systems, such as Kazaa [55] and Gnutella [41], protocols were developed for the designation of qualified supernodes (ultrapeers) to serve the ordinary peers for scalable content discovery. The supernode selection problem also shows up in the fields of sensor networks, ad-hoc wireless networks, and peer-based Grid computing. In ad-hoc wireless networks, connectivity under highly dynamic conditions is achieved by identifying a subset of the nodes to serve as bridging nodes. This subset is formed using a distributed dominating set protocol such as in [31][99][100] so that every node is within broadcast range of a bridging node. Within sensor networks, supernodes are selected for the purpose of data aggregation under the conditions that they are well-distributed among the sensors and also have sufficient remaining battery life [35]. However, leader election problem differs from supernode selection in that the former assumes all nodes vote (directly or indirectly) on the choice of each supernode.

Compared with the existing works summarized above, our proposed algorithm has the following distinct features. Firstly, we do not assume a unique weight value for each node. In our design, nodes can have same or different weight values. Such an assumption is reasonable and necessary for electing nodes with some desirable property, e.g. memory size. Secondly, except one-hop neighbors, we do not assume a node knows other nodes in the network. This is inspired by the observation that an ad hoc network is highly dynamic and auto-configured, which makes it infeasible for a node to know all other nodes in a network. Thirdly, we consider dynamic topology with frequent network

partitions and node failures. And finally, we want to elect K leaders in a message efficient manner. With the above features, much more challenges arise in the design of leader election algorithm and existing solutions cannot work at all.

Our proposed K-leader election algorithm is phase-based and uses a diffusing computation based approach. We first select some coordinator nodes with highest weight among their 2-hop neighbors. Those coordinator nodes then start diffusing computations to collaboratively collect the weight values of all the nodes in the environment to choose top K weighted nodes as K leaders. Our algorithm operates asynchronously in a mobile ad hoc network and can adapt to dynamic and frequent topological changes. We have also provided theoretical proof regarding the correctness of the proposed algorithm.

We have carried out extensive simulations to evaluate the performance of our proposed algorithm. Our results show that our algorithm is scalable, fault-tolerant and incurs low message cost. We have also implemented our algorithm on a wireless testbed system. The experimental results obtained are found to be in congruence with the simulation results.

4.2 Background

Although leader election is a well-known research problem in distributed computing, very little work has been carried out for weight-based leader election. We will briefly outline the available weight less and weight-based election algorithms.

Among the leader election algorithms which do not concern node weights they make some unrealistic assumptions. Several leader election algorithms [2][17][92] proposed for wired networks assume process crashes and link failures and are therefore close to our considerations. However, while [2] and [92] assumes process failures occur before

election starts, election algorithm described in [17] requires that every message be reliably broadcast to all other nodes and that the network be message order-preserving i.e., a message *m* sent by a node *i* at time *t* is received by all nodes before another message *m'* sent by node *j* at some instant $t' > t$. Such assumptions are very strong for dynamic and ad hoc mobile environments. For the algorithm proposed in [71], it is for infrastructured wireless networks where the election is in fact done by the wired part of the network using Garcia Molina's bully algorithm [40].

Many leader election algorithms have been proposed for static networks [39][74]. These algorithms construct several spanning trees with a prospective leader at the root and recursively reduce the number of spanning trees to one. However, these algorithms work only for the static topology and hence cannot be used in a mobile setting. Similarly, there have been several algorithms [4][9][35][29][45][97] for clustering and hierarchy-construction that can be adapted to perform leader election. But, these algorithms assume static networks and therefore cannot be used in, mobile settings. There are also some clustering algorithms [66][13] proposed for mobile networks. But these algorithms either assume that the nodes remain static during cluster head election [66], or they incur heavy cluster maintenance cost [13]. Moreover, they elect cluster heads only within single hop neighborhood.

Leader election algorithms for ad hoc networks are proposed in [44][67][90] and [97]. Based on the routing protocol named TORA [72], Malpani et al developed an election algorithm [67]. The algorithms presented in [67] are not weight-based leader election. As stated earlier, we are interested in a weight-based K leader election algorithm, because of the applications discussed in Section 4.1. In the algorithms in [67], a node that detects a partition in the network gets elected as the leader and a partition can be detected by any "random" node. Also, no proof of correctness of their algorithms has been provided for the case of concurrent topological changes.

There are also some algorithms which proposed weight based leader election, but they have certain limitations. Although, weight-based leader election algorithms for mobile ad hoc networks have been proposed in [44], these algorithms are unrealistic as they require nodes to meet and exchange information in order to elect a leader and are not well-suited to the applications discussed earlier. Sundramoorthy et al [90] design a leader election algorithm to solve the dynamic directory election problem in service discovery environment. Several weight-based clustering algorithms [11][12][22] have been proposed for mobile networks, but they elect cluster heads only within single hop neighborhood.

There is a weight-based single leader election algorithm [97] for mobile ad hoc networks which considers concurrent topological changes and adopts the well-known diffusing computation approach proposed by Dijkstra and Scholten [33]. The algorithm works by constructing a spanning tree of the network nodes, and then gradually shrinking the tree, by allowing every node to report their highest weight downstream node to their parent in the tree, so that, eventually, the root of the tree will know the highest weight node in the environment and select it as the leader. The authors have proposed an extension of their algorithm for electing K leaders, by which, nodes in each level select K highest weight nodes among their downstream nodes and report to the parent. They propose K as a fixed number, known a priori to all the nodes. This method may fail to guarantee always exactly K leaders for all the connected components, as discussed in Section 4.1. To resolve this issue, we, employ a ratio based approach for K leader election where K/N is a proper fraction, N being the total number of nodes in a connected network component. Moreover, in their algorithm, several nodes can start diffusing computation in response to the departure of a leader and hence, multiple diffusing computations can be in progress concurrently generating plenty of unnecessary overhead. Our algorithm stalls multiple diffusing computations by selecting a handful of

coordinator nodes, which will be allowed to carry out diffusing computations. This approach proves to be economical in terms of message cost. In the following section we formally describe the problem with the related correctness properties.

4.3 Problem Definition and Correctness Properties

Based on our motivation and requirements discussed in Section 4.1, we formally state our problem definition for electing top K weighted leaders in mobile ad hoc networks: *Given a network of mobile nodes each with some weight value, each connected component will eventually elect a set K of leaders in such a way, that the K leaders are the top K highest weight nodes in that component.*

The problem is required to satisfy the following three correctness properties:

- **Safety:** There should never be more or less than K leaders in a connected component.
- **Liveness:** Eventually a set of K leaders are elected for the current network component.
- **Agreement:** Each elected node should be among the top K weighted nodes within the current network component.

In the next section we shall describe our top-K weighted leader election algorithm in detail.

4.4 The Top K-Leader Election Algorithm

Our proposed algorithm adopts the diffusing computation [33] approach to perform K-leader election. The algorithm operates in three phases. In Phase I, one or more nodes

having highest weight among their 2-hop neighbors are voted as RED nodes. Other nodes are called WHITE nodes. Then, in Phase II, the RED nodes start the diffusing computation procedure asynchronously and build individual diffusion trees. The diffusing computation collects weight values of all the nodes in the network. To save message cost, a WHITE node is allowed to be included in the diffusion tree of only one RED node. Finally, in Phase III, all the RED nodes coordinate among themselves, and the results collected by different RED nodes are merged, in such a way, that, eventually the highest weight RED node receives the complete weight information of all the nodes in the network. It then chooses the top K highest weight nodes as leaders and informs every other node.

In this section we shall describe our algorithm in detail. A description of the data structures and message types used precedes the formal description of our algorithm. After that we discuss about the additional measures adopted by us to cope with frequent network partitions and node failures. Finally, we propose a message-efficient version of our algorithm.

4.4.1 Data Structures and Message Types

While executing our algorithm, each node *i* maintains necessary information about its state in the data structures listed in Table 4-1 and may exchange messages listed in Table 4-2.

Table 4-1: Data Structures for K Leader Election Algorithm

Variable	Meaning
id	Identifier of node *i*
wt	Weight of node *i*
color	Color of node *i* = {*WHITE, RED*}

visitor	Identifier of RED node which visits node *i*
nbr	Set of immediate (1-hop) neighbors of node *i*
pred	Predecessor of *i* in the diffusion tree
succ	Successors of *i* in the diffusion tree
in_electn	A binary variable indicating if node *i* is currently in an election or not
ldr	A binary variable indicating if node *i* is leader or not
ResultQ	List of nodes visited by node *i* (during diffusion)
RedQ	List of RED nodes known by *i* (during diffusion)
LDR	List of *K* leaders of node *i* and their weights

Table 4-2: Message Types for K Leader Election Algorithm

Message	**Purpose**
ELECT(i,wt_i)	For node *i* to exchange weight with its neighbors
VOTE(i)	For a node to vote its highest weight neighbor *i*
SEARCH(i, v)	For growing the diffusion tree initiated by RED node *i.* Where *v* is the node voted by the sender.
ACK()	For acknowledging SEARCH exchanged among a set of nodes visited by the same RED node
NACK(i, wt_i)	For acknowledging SEARCH exchanged among a set of nodes visited by different RED nodes. This message helps a node to fill in the RedQ
ATTACH(i)	For acknowledging SEARCH from a node and attaching as a child to that node – used in the optimized version of our algorithm
SIGNAL(RedQ $_i$, $ResultQ_i$)	For node *i* to send its result to its predecessor in response of the SEARCH message received
DIRECT(i, wt_i)	For RED node *i* to direct all but the highest weight RED node (*j*) in it's *RedQ* to send their result to *j*. This message is only used by the RED nodes for exchanging the RED node information
RESULT(P, $ResultQ_i$)	For RED node *i* to send its *ResultQ* to it's highest weight RED node. *P* is the set of RED nodes to which *i* sent DIRECT. This message is only used by the RED nodes to exchange results
LEADER (LDR)	For the highest weight RED node to announce the new set of K leaders to all other nodes

4.4.2 K Leader Election Algorithm

As discussed in Section 4.3, our algorithm operates in three asynchronous phases. Different nodes can be in different phases at the same time without global knowledge about phases of other nodes. An election can be triggered at a node either when the node lost connection with one or more of its leaders or the weight of one or more leaders fall below some application defined threshold.

A. Phase I

When a node detects the absence of a leader, it bootstraps for a new election process by changing its color to WHITE and setting *in_electn* to *true*. It then sends an ELECT message containing its weight to all its 1-hop neighbors.

```
/*******************************PHASE I*******************************/
//The code executed by each node, i
(1) color_i ← WHITE, pred_i ←NULL, visitor_i ←NULL,
                out_i ← 0, RedQ_i ←NULL, ResultQ_i ← NULL;
(2) send ELECT (i, wt_i) to all the neighbors in nbr_i;
(3) wait until an ELECT message is received from each neighbor;
(4) send VOTE(i) to j, where j is the the highest weight node in nbr_i;
(5) wait until a VOTE received from each node in nbr_i or a SEARCH is received;
(6) if (a VOTE received from each node in nbr_i) color_i ←RED;
```

Figure 4-1: Pseudo-code for PHASE I of K-leader Election Algorithm

When a node receives an ELECT, it also undergoes bootstrapping. After a node receives ELECT messages from all its neighbors, it sends a VOTE to the highest weight neighbor and then enters Phase II. A node must send ELECT messages to all neighbors and receive ELECT from all the neighbors before deciding on whom to VOTE. If a node receives VOTE from all neighbors, it must be highest weight among its 2-hop neighbors. It then changes its color to RED and enters Phase II as RED node. All other nodes remain WHITE. Some nodes are highest weight among 1 hop neighbors, but not among

2-hop neighbors. These nodes do not receive VOTE(s) from all the neighbors and will remain blocked in Phase I for the time being. At the end of Phase I there are one or more RED nodes in the network, but they do not know each other. And it is not necessary that all other WHITE nodes have less weight than the RED nodes. So, a second phase is necessary to collect weights of all the nodes in the network.

B. Phase II

In Phase II, each RED node starts a diffusing computation separately and unaware of the others. Each RED node then grows a diffusion tree as a *root*, by sending SEARCH to all of its 1-hop neighbors. Each node *i*, other than the root, designates the neighbor from which it first receives a SEARCH as its *predecessor (pred)* in the diffusion tree and the root as the *visitor*. Node *i* then forwards the SEARCH to all neighbors except the predecessor. Node *i*, does not, however, immediately return any acknowledgement to its predecessor and keep it blocked.

So, when two or more diffusion trees of different RED reach their boundary nodes, they stop propagating any further and start shrinking towards the root. Eventually, the boundary nodes receive either ACK or NACK from all their neighbors and send SIGNAL to their predecessors with their own id and weight stored in *ResultQ*. Each node also forwards its *RedQ* via a SIGNAL message. The predecessors, in turn, send their SIGNAL to their own predecessors, and so on, until the root node receives all pending SIGNALs. After a node sends its SIGNAL, it enters Phase III.

If a node is blocked in Phase I as it is highest among its 1- hop neighbors but not 2-hop neighbors, it will enter Phase II after receiving the first SEARCH message and carry out the task of Phase II as described above.

```
/*****************************PHASE II*****************************/
(7) if (color_l = RED) {//for node l
        pred_i ← 0;
        send SEARCH(l, v) to each node j in nbr_l; }
//This part is executed by each node i
(8) when SEARCH(l, v) is received from node j
    if (visitor_i = NULL) {   // node i has not been visited by any RED node
        visitor_i ← l;
        pred_i ← j;
        insert i into the ResultQ_i;
        succ_i ←nbr_i \{pred_i};
        send SEARCH(l, v) to nbr_k;
    }
    else {        // suppose that node i has been visited by RED node k, so, visitor_i =k
        if (k = l) send ACK( ) to j;
        else send NACK(k, wt_k) to j;
    }
    when j receives the NACK(k, wt_k) from i {
          put (k, wt_k) into RedQ_j;
          send SIGNAL(RedQ_j, ResultQ_j) to pred_j;
    }
(9) Node i waits until an ACK / NACK / SIGNAL received from each node j in succ_i {
        if (pred_i = 0) GoTo Phase III;    // this is the starting RED node
        else{
                merge RedQ_j into RedQ_i;
                merge ResultQ_j into ResultQ_i;
                send SIGNAL(RedQ_i, ResultQ_i) to pred_i;
        }
    }
```

Figure 4-2: Pseudo-code for PHASE II of K-leader Election Algorithm

When a node *i* receives SIGNAL from its successors, it appends its own id and weight to the received *ResultQ*, and also appends its $RedQ_i$ to the received *RedQ* before sending its own SIGNAL. In this way, the root node will eventually have the weight information of all the nodes in its diffusion tree and also the knowledge of the RED nodes in the adjacent diffusion trees as obtainable from the *RedQ*. Thus, at the end of Phase II, every RED node knows one or more other RED nodes and has visited a subset of nodes in the network.

C. Phase III

As mentioned earlier, when the diffusing computation is finished by a RED node *r*, it has collected information of one or more other RED nodes in its $RedQ_r$ and information of a subset of network nodes in the $ResultQ_r$. In case, there is no other RED node in the $RedQ_r$ of node *r*, except itself, node *r* will select the top K highest weight nodes from its $ResultQ_r$, as leaders, and terminate the election algorithm. Otherwise, Phase III will be triggered. In Phase III, RED nodes exchange their partial knowledge about the weight values of visited nodes and at the end, only the highest weight RED node receives the global knowledge about all the nodes in the environment. This RED node then selects the *K* highest weighted leaders.

In Phase III, there are mainly two types of messages – DIRECT and RESULT exchanged between the RED nodes. When a RED node enters Phase III, it will check the weights of all nodes in its *RedQ*, and will clearly identify the *lower weight* and *higher weight* RED nodes other than itself. Among all higher weight RED nodes, all but the highest weight RED node are called *intermediate-RED-nodes*. For the sake of clarity, we name the highest weight RED node in the *RedQ* of RED node *i*, as R_{Hi}.

A RED node *i* having no other RED node in its $RedQ_i$ except itself and its R_{Hi}, sends a RESULT to R_{Hi} with its own $ResultQ_i$. Upon receiving a RESULT from node *i*, node *j* will insert $ResultQ_i$ into $ResultQ_j$, and will delete *i* from $RedQ_j$.

In case a RED node *i* have no lower weight RED node, but one or more intermediate RED nodes, node *i* sends its R_{Hi} information to each of the intermediate nodes in its $RedQ_j$, via a DIRECT message, asking them to directly send their RESULT to R_{Hi}, without waiting for RESULT from *i*. After that, node *i* will send its own RESULT to R_{Hi}. Upon receiving a DIRECT message from node *i*, node *j* will insert R_{Hi} into $RedQ_j$, unless it is already there, and will delete *i* from $RedQ_j$.

```
/*****************************PHASE III*****************************/
//This part is executed by each RED node i
(10) if (wt_i =min(RedQ_i) ) {
        k← max(RedQ_i);
        P← RedQ_i\{k};
        send DIRECT(k, wt_k) to each node in P;
        send RESULT(|P|, ResultQ_i) to k;
    }
(11) else{
      P← RedQ_i\{k|wt_k≥wt_i};
       wait until a RESULT or DIRECT received from each j in P;
          for (a RESULT(Q, ResultQ_j) from j) {
                  merge ResultQ_j into ResultQ_i;
                  merge Q into RedQ_i;
                  delete j from RedQ_i;
           }
           for (a DIRECT(k, wt_k) from j)
                  put k into RedQ_i; delete j from RedQ_i;
           if(|RedQ_i|>1){
                  m← max(RedQ_i);
                  send DIRECT(m, wt_m) to nodes in RedQ_i\{m};
                  send RESULT(ResultQ_i) to m;
           }
          else{
                  sort the nodes in ResultQ_i based on their weights;
                  LDR ← top K of ResultQ_i;
                  send a LEADER(LDR) message to nodes in LDR;
          }
    }
```

Figure 4-3: Pseudo-code for PHASE III of K-leader Election Algorithm

Every RED node *i*, having one or more lower weight RED nodes, waits to receive RESULT or DIRECT message from all the lower weight nodes before sending its own RESULT and DIRECT messages. Then, node *j* will check whether it has any intermediate RED node. In case, it has intermediate RED nodes, it will send a DIRECT message to each of them, before sending its RESULT to R_{Hi}.

After a RED node has received, all RESULT or DIRECT messages from all lower weight RED nodes, and has no other RED node in its *RedQ* except itself, it must be the

highest weight RED node in the network component and is termed as *elector*. The elector then sorts its *ResultQ* and selects the top *K* nodes according to the weight values, as K leaders. It then sends a LEADER message containing the list of K leaders (*LDR*) to all the other RED nodes, who, in turn, will forward the message to their diffusion tree members. Any node receiving the LEADER message will set *in_electn* to *false* and will set *ldr* to *true* if it is one of the leaders.

D. Handling Delayed Messages

As the phases are asynchronous we need to introduce proper mechanisms to handle cases when a node receives a message from another node belonging to a different phase. We can associate different messages to respective phases in order to simplify our discussion. While ELECT and VOTE messages belong to Phase I, SEARCH, SIGNAL, ACK and NACK messages are exchanged in Phase II. Phase III consists of DIRECT, RESULT and LEADER messages. In case a node receives messages from lower phases, it simply ignores the message, e.g., when a node is in Phase II and it receives an ELECT message, it discards the ELECT. But when a node receives messages from higher phases, we design specific techniques to handle it. Below we describe our techniques for different phases.

Phase I: Consider node *i* is in Phase I and receives SEARCH from one or more neighbors before receiving all ELECT from the neighbors. We have two ways to handle this. One way is, Node *i* will send ATTACH to that node which sent the SEARCH message. The other way is to wait for all ELECT messages until the last of its neighbors send SEARCH. There is a tradeoff between the two approaches. When the first way may reduce the total election time, the second way may result in nodes with less number of neighbors be chosen as RED nodes and thus the RED nodes are not as resource-efficient as they are required to be for carrying out the next phases.

Similarly, consider node *i* in Phase I which has sent and received all ELECT and is waiting for VOTE, and in the mean time, it receives a SEARCH from its neighbor *j*, then it will check whom *j* voted. If *j* did not VOTE *i*, then *i* will enter Phase II.

Phase II: Suppose a RED node *i*, which is still ongoing diffusing computation, receives a RESULT or DIRECT from another RED node in Phase III. Node i will handle such messages following the mechanisms described in Phase III, but will not send any RESULT or DIRECT until it terminates diffusing computation and enters Phase III.

4.4.3 K-leader Election in Presence of Network Partition

In this section we consider concurrent topological changes. Nodes can freely move across the terrain which may result into network partitions as well as partition merges. Partitions can take place either during election or after the election.

A. Handling Network Partition

Network partitions may occur during an ongoing election and after an election has been finished. We first discuss about handling partitions during an ongoing election process which may result in one of the following three cases:

- *Case I) Network Partition in Phase I:* A node may need to wait forever to receive ELECT or VOTE messages from partitioned neighbors. We use a probe-reply mechanism to address this problem. Nodes waiting for neighbors to send messages can probe their neighbors. If no reply comes from a neighbor after probing three times, the neighbor is deleted from the neighbor list.

- *Case II) Network Partition in Phase II:* A node may need to wait for ACK, NACK or SIGNAL from partitioned children. This case can have the following two sub-cases as illustrated using Figure 4-4.

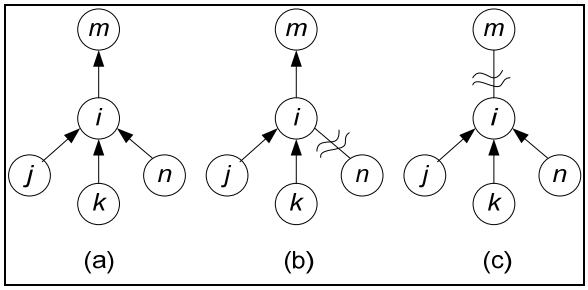


Figure 4-4: Handling Network Partition: (a) Connected Diffusing Computation Tree, (b) Partition of Node *n* from Parent Node *i*, (c) Partition of Parent *m* of Node *i*

(i) *Partition with the children:* In Figure 4-4 (a), node *n* is a child of node *i* and node *i* waits to receive SIGNAL from *n*. In Figure 4-4 (b), node *n* got partitioned and it can no longer send its SIGNAL to node *i*. If node *i* cannot detect this incident, it may keep on waiting forever. To resolve this issue, we again use the probe-reply mechanism between the parent and child nodes. Node *i* will start a timer for each neighbors after sending the SEARCH and wait for receiving ACK, NACK or SIGNAL. In case, the timer expires before the message arrives, node *i* will probe the neighbor node which failed to send SIGNAL. If three consecutive probing cannot fetch any reply, the node will be deleted from the set of successors (*succ*).

(ii) *Partition with the parent:* As illustrated in Figure 4-4 (c) where node *i* receives SIGNAL from child nodes *j*, *k*, *n*, but cannot reach its own parent node *m*. In this case also, node *i* needs to detect that *m* has been partitioned and should terminate diffusing computation.

- *Case III) Network Partition in Phase III:* RED nodes in a network component may get disconnected if a partition occurs. When one RED node does not receive a message from other RED nodes in its *RedQ*, it can probe them. Any RED node which does not reply to a probing is deleted from the *RedQ*.

Now we discuss about handling partitions that may occur after an election is over. When K leaders have been successfully elected, they will send periodic heartbeat messages to all other nodes in the network. If a network partition occurs, one or more leaders may get disconnected. Any node detecting such incident will start a probing. If consecutive three probing fails without receiving any reply, a new election will commence.

B. Handling Merging of Network Partitions

Node mobility may cause network partitions to merge at times. When two partitions come closer, they form a link to inter-connect. We need to address following cases while handling partition merges:

- *Case I) If two components each with K leaders merge:* In this case, meeting nodes of two adjacent partitions exchange their K leader information over the newly formed link. After they receive the set of 2*K leaders obtained from both the components, they sort them by weight, and select the top K leaders and then broadcast the information to other nodes.

- *Case II) If one or both the merging partitions are without K leaders and are undergoing election process*: In this case, we choose to wait until one or both of the partitions finish their election and then they will exchange their leader information over the newly formed link as discussed for the *Case I*.

4.4.4 Handling Node Failures

We consider node failure by crashing, i.e. premature halting. To handle failures, we use our probing mechanism to enable a node to detect the failure of its neighbors. We

also consider that node failure can happen anytime - while an election is ongoing or finished. We address these cases separately:-

- *Case I) Node failure during leader election:* node failure during an election can be caused by failure of either a RED or a WHITE node. First, we consider the failure of RED nodes. We adopt the backup approach to handle such failures. After a node is elected as a RED node in Phase I, it will select the highest weight node among its neighbors as a backup node, called a GREEN node. The GREEN node will take over the role of RED node in case the latter crashes. Such a mechanism is suitable and efficient considering that only few nodes are RED nodes and they are more reliable than other nodes. Now, let us discuss the failure of a WHITE node. In Phase I and II, a node will detect a crashed WHITE neighbor using probe-reply and delete it from its neighbor list. In Phase III, only RED nodes need to send and receive messages. Therefore, WHITE node failures during Phase III will not affect the algorithm performance unless the network gets partitioned.

- *Case II) Node failure after K leader election:* After the election is over, all the nodes have equal priority and there is no specific node color. This case can be handled in the same way as the partitions are handled (described at the end of Section 4.4.3).

4.4.5 Handling Node Recoveries

Our algorithm handles node recovery and new node joining in the similar way. When a failed node recovers or a new node joins the network, they first initialize the data structures described in Table 4-1. After that the recovered node is without a leader and it will initiate a new election.

Following our algorithm, after the K-leaders are elected, all nodes in the network will be designated either as leaders or as *followers*. Every follower node in a connected network component will have the same set of K leaders and every follower will know all the K leaders. Each leader will send periodic heartbeat messages to the followers. When any one of the leaders fails, one or more nodes can detect the failure and they will begin a new election starting from Phase I.

4.4.6 Optimization in Message Cost

In order to reduce the number of messages, we use a message optimization policy (*Election-Opt*) which aims to reduce the number of ACK and NACK messages. With the increase in node density, every node may have many neighbors and they need to send ACK/NACK for all the SEARCH received from the neighbors. In order to reduce the number of ACK/NACK, a node only acknowledges (by sending an ATTACH) the first SEARCH received, and can ignore the rest. To illustrate the benefit of this method we use the following example:

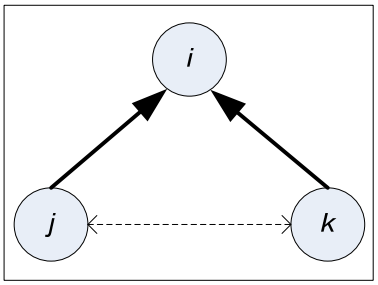


Figure 4-5: Optimized Diffusing Computation

As shown in Figure 4-5, upon receiving a SEARCH from node *i*, node *j* and *k* each sends an ATTACH requesting *i* to accept them as children. Node *i* starts a timer after sending the SEARCH and accepts all nodes as its children which sends an ATTACH before the timer expires. Nodes *j* and *k* also propagate SEARCH received from *i* to each other. Since, both of them already have node *i* as their parent, none sends ACK/NACK

to the other. Eventually, node *j* and *k* sends their SIGNAL to node *i*. In this way, nodes *j* and *k* just send two messages (one ATTACH + one SIGNAL) each and therefore the total number of messages reduce from six to four.

In the next section we shall prove the correctness of our proposed algorithm using the correctness properties mentioned in Section 4.3.1.

4.5 Correctness of the Algorithm

The proof of correctness of our K-leader election algorithm requires establishing the safety, liveness and agreement properties as described in Section 4.3.1.

Lemma 1. *After Phase I finishes, there is at least one RED node for any connected component C (V, E).*

Proof. We assume that every node has a unique id and some weight value. As described in Section 4.3.2, we ensure total ordering among all the nodes, using <*id, wt*> tuple, so that one node can be chosen as highest weight among all others.

Initially, every node sends ELECT containing its <node id, node weight>, which will be received by all the nodes in its transmission range as the links are reliable. When a node receives an ELECT from all its immediate neighbors, it finds out the highest weight neighbor and sends a VOTE to that.

Now for a node to become a RED node, it must receive a VOTE from each of its immediate neighbors. So, we have to establish that there will definitely be at least one such node which receives VOTE(s) from all its neighbors. As stated previously, there is at least one node in the environment which is highest weight among others. The highest weight node must receive VOTE(s) from all its neighbors. So, at least one node in a

connected component will receive VOTE from all its neighbors. That node will be chosen as the RED node. So, the lemma holds.

In case where node mobility may create partitions, we manage the neighbor list using a probe-reply method. So, eventually partitioned nodes will be detected and deleted from the neighbor list and only nodes in the current network component are considered for Lemma 1 as discussed above. Thus, the Lemma 1 holds for dynamic topological changes. ■

Lemma 2. *The diffusing computation started by a RED node in Phase II eventually terminates.*

Proof. The termination condition for diffusing computation is that the computation terminates when all processes are idle and all channels are empty.

In our algorithm, after a node is voted as a RED node, it starts diffusing computation and grows a tree of visited nodes rooted at it. We can have one or more RED nodes in the environment and all of them will start diffusing computation at different times. In our algorithm, any WHITE node will participate in a single diffusing computation. So, if a WHITE node has joined diffusing computation tree of one RED node, it will not accept another request from a different RED node.

We propose that, a RED node starts diffusing computation by sending a SEARCH to all its immediate neighbors. When a WHITE node receives a SEARCH directly from the root through the incoming channel, it forwards the SEARCH through all outgoing links. This phase of the diffusing computation is about the "growing" of a diffusion tree. A SEARCH may be delivered in the following three cases:

- *Case I) An unvisited WHITE node n receives a SEARCH directly from a root or via some other WHITE node*: *n* sets the node from which it receives SEARCH as its

parent and forwards the message to all the successors. Node *n* does not send any immediate reply to the parent for the time being and keeps it blocked.

- *Case II) An already visited WHITE node n receives a SEARCH from the root of its tree via some other WHITE node*: *n* acknowledges the SEARCH immediately by sending an ACK message. As the ACK will eventually reach the receiver so the sender will not remain blocked forever.

- *Case III) An already visited WHITE node receives a SEARCH from a RED node, which is NOT its root, either directly, or via some other WHITE node*: *n* acknowledges the SEARCH immediately by sending a NACK. As the NACK will eventually reach the receiver so the sender will not remain blocked forever.

For *Case I*, the parent of node of *n* remains blocked until it receives either an ACK, or a NACK or a SIGNAL from every node to which it has sent a SEARCH. This is called the "shrinking" phase of the diffusion computation tree. When a node receives replies corresponding to all the SEARCH sent, it sends a SIGNAL to the parent node. In this way, every blocked node ultimately unblocks after receiving one or more message it is waiting for. At the end, the initiating RED node eventually receives replies from all its immediate neighbors in finite time, which ensures that no node in the current diffusion computation tree remains blocked forever. Thus, all the nodes can be considered idle as all of them finished diffusing computation and the channels are also idle as there are no messages being passed through them. This is the termination condition for diffusing computation as specified before. So Lemma 2 holds.

In case where node mobility may create partitions, we can detect the partitions in finite time and update the neighbor list by deleting partitioned nodes. Thus, no node will wait forever to receive messages from partitioned nodes. So eventually the RED node

will receive SIGNAL from all its neighbors and terminate the diffusing computation in finite time. Thus, the Lemma 2 also holds for networks with dynamic changes. ■

Lemma 3. After Phase III finishes, the highest weight RED node in any connected component C (V, E) eventually receives information regarding all the nodes in the component.

Proof. After the Phase II ends, one or more RED nodes present in the network have partial information regarding the nodes visited by the diffusing computation initiated by it. There can be two cases,

- *Case I) There is only one RED node in the connected network component* – if a RED node *r* finds out that only its own id appears in the $RedQ_r$ then *r* must be the highest weight RED node in the network. So, the RED node chooses top K weighted nodes as leaders and terminates the algorithm. For this case, there will be no need to carry out Phase III.

- *Case II) There is more than one RED node in the connected network component* – if a RED node *r* finds out that its $RedQ_r$ contains nodes other than itself, then the network must have more than one RED node. Let us denote the highest weight RED node in the $RedQ_r$ of node *r* as R_{Hr}. We can have the following sub-cases for this case:

  (i) *r* has the lowest weight in $RedQ_r$: *r* sends DIRECT messages to all nodes in $RedQ_r$, except R_{Hr}, and sends RESULT to R_{Hr}.

  (ii) *r* has the intermediate weight in $RedQ_r$: *r* waits until DIRECT or RESULT from all lower weight nodes in $RedQ_r$ arrive and then sends DIRECT to the higher weight nodes in $RedQ_r$ before sending RESULT to R_{Hr}.

(iii) *r* has the highest weight in $RedQ_r$ ($r = R_{Hr}$): *r* waits for DIRECT or RESULT from lower weight nodes in $RedQ_r$ and updates the $RedQ_r$ whenever a message is received. After all the messages are received, then *r* checks the $RedQ_r$. If $r \neq R_{Hr}$, then *r* carries out step B. Otherwise, *r* terminates the algorithm as the highest weight RED node in the network and the Lemma 3 holds.

In presence of partitions, RED nodes may not receive message from other RED nodes which have been partitioned. If a RED node waits on a partitioned RED node, the routing algorithm will check the reachability of the node for which it waits. If the node is unreachable, it simply deletes the RED node from its *RedQ*. In this way, the algorithm can terminate also in presence of partitions or node failures. So, the lemma also holds for a network with dynamic changes. ■

Theorem 1 (Liveness): *The election algorithm eventually terminates electing a set of K leaders for the current network component.*

Proof. Following lemmas 1, 2 and 3 consecutively, the highest weight RED node *i* for any connected component C (V, E) eventually receives information regarding all the nodes in the component and terminates in finite time. After *i* has the complete knowledge regarding all the nodes in the network, it sorts the nodes using the total order and selects the top K highest weight nodes as a definite percentage of the total number of nodes in the network. So, liveness property holds for the algorithm. ■

Theorem 2 (Safety): *There should never be more or less than K leaders in a connected component*.

Proof. Immediately after the election algorithm terminates, the highest weight RED node selects the K highest weight nodes in the environment as leaders. So initially there

are exactly K leaders, and not more or less than K, which ensures that the safety property is satisfied.

But as we consider a dynamic topology, with node crash and migration, one or more leaders may become unavailable at some point in time which results in less than K leaders for a network component. When a node fails to receive periodic heartbeat messages from one or more leaders, it will start probing them. In case probing yields no result, the node will start a new election to elect a new set of K leaders. So, after a leader becomes unavailable and before a re-election triggers, there may be a brief spell when the network has less than K leaders. But eventually the network will elect exactly K number of leaders in finite time.

Again, for the case of partition merges (discussed in Section 4.4.3 B) if both the merging partitions have K leaders, the newly formed network component may have more or less than K nodes. We adjust the number of leaders after the merging components collaborate to select the top K leaders from the existing set of leaders. This operation will finish in finite time and eventually the new network component will have exactly K leaders as required. So, eventually, the safety property holds for the algorithm.

■

Theorem 3 (Agreement): *Each elected leader should be among the top K weighted nodes within the current network component*.

Proof. After the highest weight RED node has collected the weight information of all the nodes in its network component, it will elect the top K weighted nodes as leaders. Thus, the elected leaders will definitely be the top K nodes by weight and the agreement condition is satisfied.

But in presence of node mobility and consequent changes in network topology along with node crash, there may arise many dynamic situations. When two network components merge together, they will bring two different sets of K leaders which as a whole may not be among the K highest weight nodes in the newly combined network component. The meeting nodes then exchange the leader information of their respective components and then choose the top K weighted nodes which are highest among all the nodes in the current component. Thus, eventually it is ensured that the elected leaders are among the top K weighted nodes in the current network component. Hence the agreement condition is met. ■

In this section we have successfully proved the correctness properties of our algorithm. In the next section we shall discuss in depth the experimental evaluation results of our proposed top K weighted leader election algorithm.

4.6 Performance Evaluation

We have carried out extensive simulations to evaluate the performance of our algorithm. Both the normal and message efficient versions of our algorithm have been simulated. Moreover, to show the advantage of our algorithm, we have also extended the single leader election algorithm proposed in [97] to a K-leader election algorithm and have compared their results with ours. We follow the simulation settings from [97] for the ease of comparison.

4.6.1 Simulation Setup and Metrics

The simulation system consists of two modules: the network and the leader election algorithm. The main parameters of the simulations are shown in Table 4-3. We consider 100 nodes moving at 20m/s with a 20% node failure rate as default. Unless otherwise

specified, these default values will be used for our experiment. The network nodes are randomly scattered in a square territory. Total number of nodes is varied to examine the effect of system scale on the performance. To make the performance results in different scenarios comparable, we also scale the territory size according to the total number of nodes. For message routing, we have implemented a simple protocol based on the "least hops" policy, which is adopted in many classical routing protocols in ad hoc networks. A routing table is proactively maintained at each node.

Table 4-3: Simulation Parameters for K-leader Election Algorithm

Parameters	**Values**			
Number of nodes, (N)	50	100	150	200
Territory scale (m^2)	700	1000	1200	1400
K/n	25%			
Mean Link Delay (ms)	5			
Max Link Delay (ms)	100			
Transmission radius (m)	250			
Routing-protocol	Least hops			
Node failure rate (F_R) (in %)	10, 20, 30, 40, 50			
Mobility Model	Random Waypoint			
Max. node speed (V_{max}) (m/s)	10, 20, 30			
Min. node speed (V_{min}) (m/s)	5			
Pause time (ms)	10			
Probing time (seconds)	1			
Probing interval (seconds)	2			

The leader election algorithms are implemented as applications running on top of the network. The weight values of the nodes are assigned randomly. In case of node failures, the faulty nodes are randomly selected and a faulty node crashes in a randomly chosen time. The failure detection part is simulated based on the heartbeat-like approach.

In the simulations, we measure the performance of all the algorithms using the following metrics:

Fraction of Time without K-Leaders (FT) is the fraction of simulation time that a node is involved in an election (as indicated by *in_electn = true*). This metric is a measure of the time a node lacks K leaders. The lower the value of FT is, the higher is the efficiency of the algorithm to ensure availability of K leaders to a node.

Election-Rate (ER) is defined as the average number of elections that a node participates in per unit time (i.e. the average rate at which node *i* goes from in_electn_i = *false* to in_electn_i = *true*). Higher the election rate, higher is the time without K leaders, because a node participates in a new election only when it lacks K leaders. If our algorithm can ensure that a node retains K leaders for longer duration, then the election rate will decrease and the message overhead will also decrease which can contribute towards saving more resources for a node following the objective of our research.

Election-Time (ET) is defined as the mean time elapsed between the instant at which a node begins participating in an election process (corresponds to in_electn_i = *true* in our algorithm) and the instant at which it knows the identity of its K leaders (in_electn_i = *false*). It is a measure of how efficient the algorithm is. Less election time can ensure longer time a node has K leaders.

Message-Overhead (MO) is defined as the average number of messages sent by a node per election. Message overhead for a node is calculated dividing total number of messages sent by a node over the entire simulation period with the total number of elections the node participates in. The less is the message overhead, the more is the saving in resource for a mobile node.

4.6.2 Simulation Results and Analysis

Below we present our simulation results with analysis. We have simulated our algorithm, labeled as "*Election-Main*" and the optimized version, labeled as "*Election-*

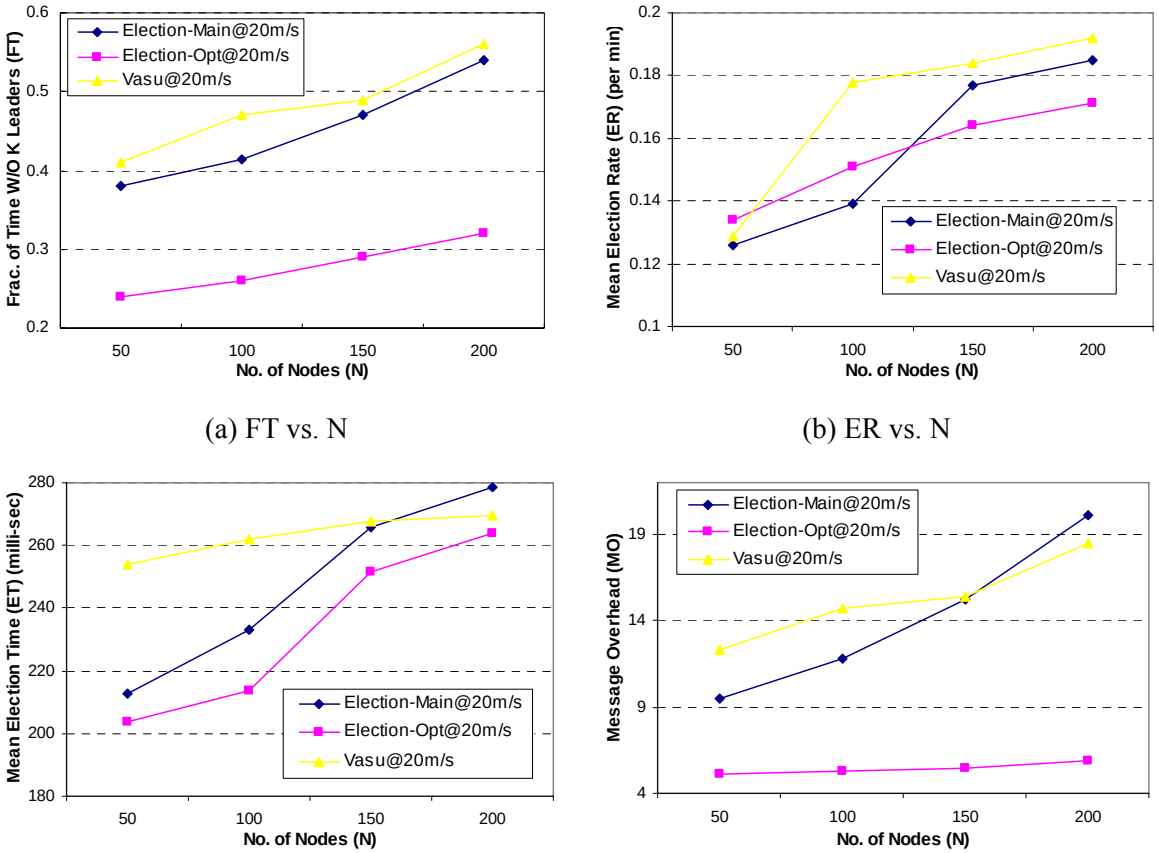
Opt". We also have simulated the K-leader election version of algorithm presented in [97], labeled as "*Vasu*". We run each simulation for 100 simulation minutes and each point is obtained by averaging over 10 different runs. We first report the general performance of our algorithm measured with respect to the four metrics and then we shall discuss the effect of mobility and node failure rates on the performance, all in separate sub sections.

A. General Performance of K-leader Election Algorithm

In this sub-section, we discuss the performance of "Election-Main", "Election-Opt" and "Vasu" under the default values of the simulation parameters.

Performance of FT: In Figure 4-6 (a), we plot the fraction of time a node is without K leaders (FT) with respect to N for "Election-Main", "Election-Opt" and "Vasu". We observe that FT for a node increases with increase in N. We choose K as one quarter of the total number of nodes in a connected component. So, for a single connected component there will be more leaders with the increase in N. With more leaders, there is a higher chance that the leaders will fail or get partitioned. This will raise the need of higher election rate (Figure 4-6 (b)) and renders the nodes without K leaders for longer time. Figure 4-6 (a) shows that "Election-Opt" ensures higher leader availability than "Vasu" and "Election-Main". "Vasu" induces multiple concurrent diffusing computations by different nodes when they detect failure or departure of a leader. High node density and a high node failure rate only increases the number of diffusing computations, which delays the election of new set of leaders ultimately increasing the FT. "Election-Main", on the other hand, decreases multiple diffusing computations by picking up few RED nodes to carry out the same, so, it can finish K leader elections quicker than "Vasu". But still, in "Election-Main" nodes havelower leader availability as each node needs to wait for all its neighbors to send ACK, NACK or SIGNAL messages.

Failure of one or more neighbors will necessitate probing the neighbor which increases the election time as well as the fraction of time without K leaders. "Election-Opt" decreases multiple ACK/NACK messages and hence the election finishes faster than "Election-Main" resulting in higher time of availability for K-leaders.



(a) FT vs. N

(b) ER vs. N

(c) ET vs. N

(d) MO vs. N

Figure 4-6: General Performance of K leader Election

To further bolster the accuracy of our simulation results, we have calculated the confidence of the FT metric. Sample means for "Election-Main", "Election-Opt" and "Vasu" for N=100 are 0.415, 0.26 and 0.47, respectively. Our measurement finds out that FT of each of the simulated protocols has a 95% confidence level with confidence intervals of (0.3990 to 0.4210), (0.2513 to 0.269919) and (0.4633 to 0.4802), respectively.

Performance of ER: In the Figure 4-6 (b), we show the mean election rate (ER) of a node with respect to increasing N, for "Election-Main", "Election-Opt" and "Vasu". As explained for Figure 4-6 (a), election rate increases with increasing number of nodes (N). In Figure 4-6 (b), election rate of all three algorithms increase with increasing N. This is because with more nodes at a high speed, the network topology changes frequently giving rise to partition with leaders. If all the K leaders are not available, a new election triggers.

Performance of ET: Figure 4-6 (c) shows the required election time of K leaders for "Election-Main", "Election-Opt" and "Vasu". We can observe that the election time increases with increase in N for all the algorithms. This is because elections can be expected to be longer when there are more nodes. "Election-Main" has highest election time at N=200, because, the higher number of ACK and NACK messages exchanged increases the election duration. We also calculate the confidence on the simulated values of ET and we found that the sample mean for N=100 for "Election-Main", "Election-Opt" and "Vasu" are 78.35326, 76.64 and 262.047, respectively. ET of each of the simulated protocols has a 95% confidence level with confidence intervals of (77.579 to 78.984), (75.379 to 77.176) and (260.772 to 264.892), respectively.

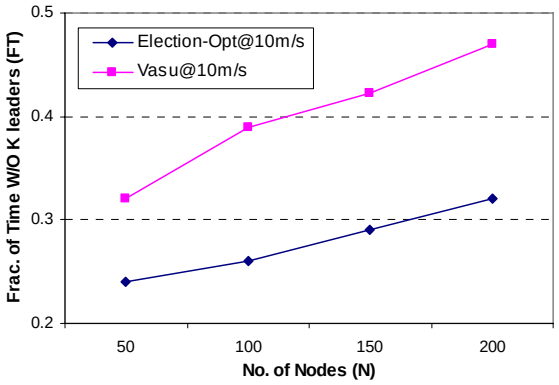
Performance of MO: Figure 4-6 (d) shows the message overhead of each node for "Election-Main", "Election-Opt" and "Vasu". We can see that "Election-Opt" is far more message efficient than "Election-Main" and "Vasu". This is attributed to the multiple concurrent diffusing computations which start on the detection of a leader failure in "Vasu". And for "Election-Main" the high message overhead is due to many ACK/NACK messages. "Election-Opt", restricts the number of diffusing computations. It also discards multiple ACK/NACK messages by using ATTACH which saves more node resources.

In general, from the above results we can realize that all the four metrics, FT, ER, ET and MO increases with the increase in the total number of nodes taking part in simulations. Comparing our algorithm with "Vasu", our algorithm consistently performs better than "Vasu".

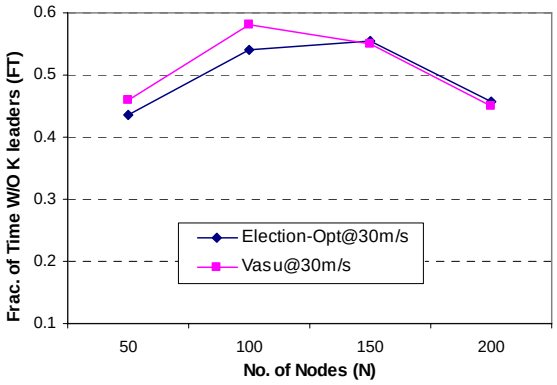
B. Effect of Node Mobility on Our Algorithm

In this sub-section we study the effect of varied node speeds on the performance of our algorithm. We vary node speeds from 10 m/s to 30 m/s and during our simulation.

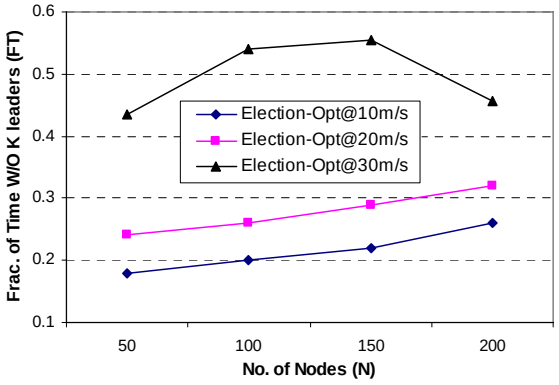
Effect of Mobility on FT: Figure 4-7 (a) and (b) compares "Election-Opt" with "Vasu" with node speeds 10m/s and 30m/s respectively.



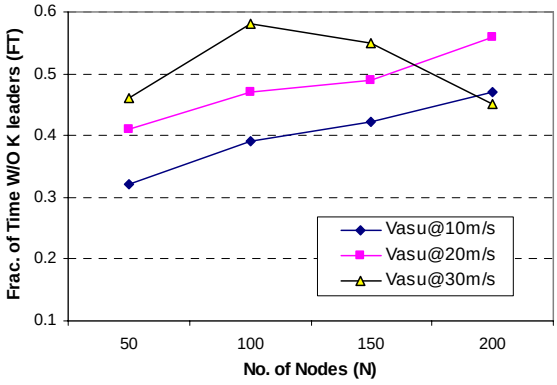
(a) FT vs. N (V_{max} = 10m/s)



(b) FT vs. N (V_{max} = 30m/s)



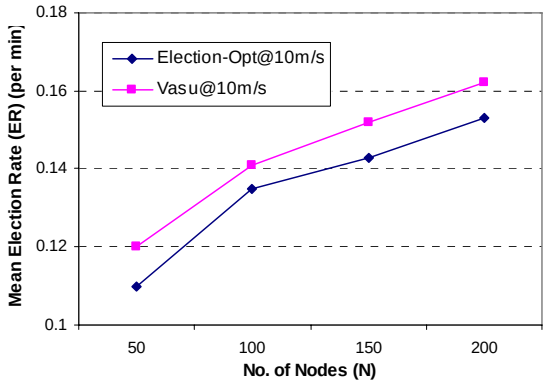
(c) FT vs. N for "Election-Opt"



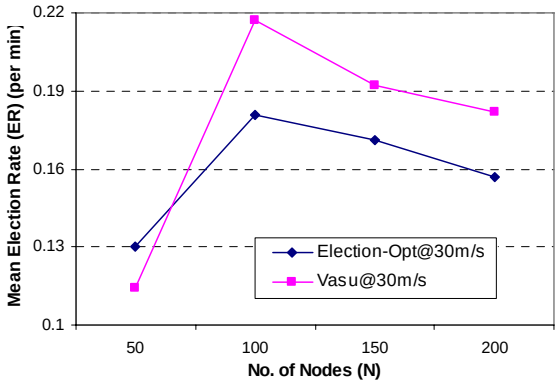
(d) FT vs. N for "Vasu"

Figure 4-7: Effect of Mobility on Fraction of Time without K-Leaders (FT)

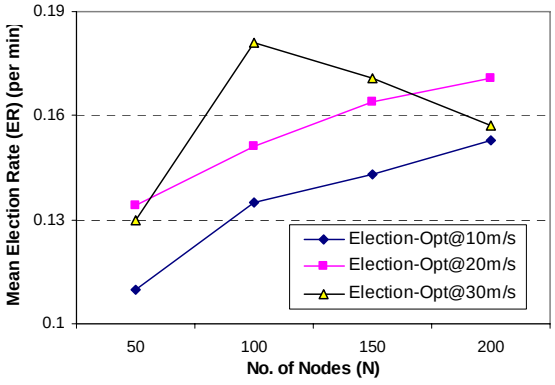
We can observe in Figure 4-7 (a) that FT increases with N at V_{max} = 10m/s. This observation is similar to that observed in Figure 4-6 (a) and can be explained similarly. An interesting fact is revealed by Figure 4-7 (b) where FT decreases with increase in N for both the curves plotted. This can be explained in the light of high node speed. With node speed as high as 30m/s, nodes get disconnected from their leaders, but they get reconnected within the probing period which results in higher time of availability of K leaders for nodes and lower election rates (Figure 4-8 (b)). We also show the variation of FT with N for different values of V_{max} for "Election-Opt" (Figure 4-7 (c)) and "Vasu" (Figure 4-7 (d)), separately. For both the cases, FT increases at lower speeds, but decreases at higher speed.



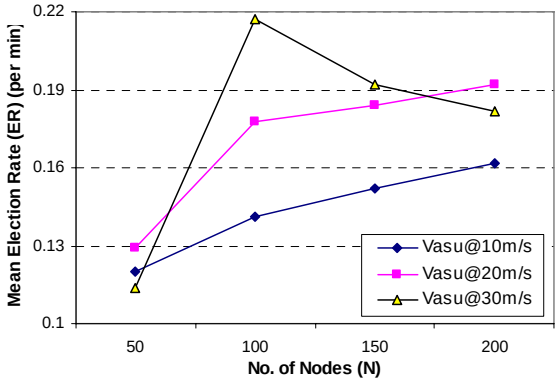
(a) ER vs. N (V_{max} = 10m/s)



(b) ER vs. N (V_{max} = 30m/s)



(c) ER vs. N for "Election-Opt"



(d) ER vs. N for "Vasu"

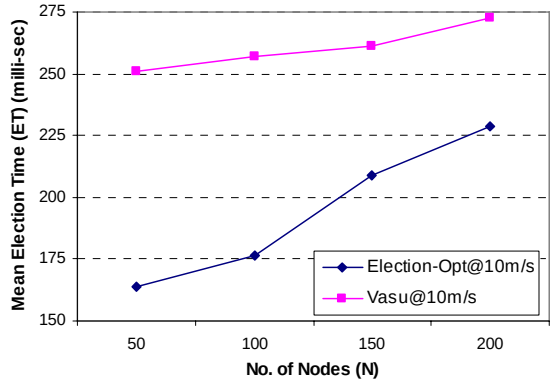
Figure 4-8: Effect of Mobility on Election Rate (ER)

Effect of Mobility on ER: Figure 4-8 (a) and (b) compares "Election-Opt" with "Vasu" at node speeds 10m/s and 30m/s respectively. We can observe in Figure 4-8 (a) that ER increases with N at V_{max} = 10m/s. This observation can be explained similarly as done for Figure 4-6 (b) that ER increases with node speed as leader nodes frequently get disconnected from others necessitating a new election. An interesting fact shown in Figure 4-8 (b) is that ER decreases with increase in N for both the algorithms. This can be explained in the light of high node speed. With node speed as high as 30m/s, nodes get disconnected from their leaders, but they get reconnected within the probing period without restarting a new election which lowers the ER. We also show the variation of ER with N for different values of V_{max} for "Election-Opt" (Figure 4-8 (c)) and "Vasu" (Figure 4-8 (d)), separately. For both the cases, ER increases at lower speeds, but decreases at higher speed.

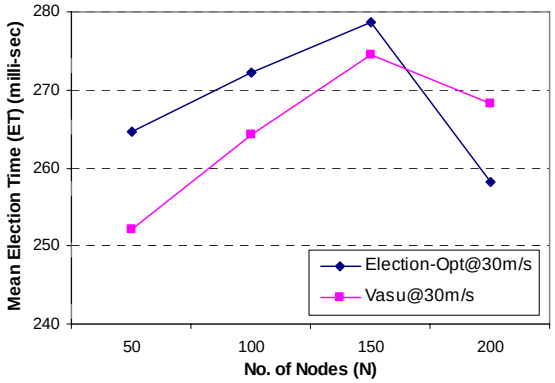
Effect of Mobility on ET: Figure 4-9 (a) and (b) depicts the change of ET with N by varying node speeds from 10m/s to 30m/s, for "Election-Opt" and "Vasu". Observation from Figure 4-9 (a) is similar to that in Figure 4-6 (c), that the ET increases with N. This is because, with more nodes in a dynamic environment, the topology frequently changes which requires probing some nodes which increases the overall election time. ET, on the other hand, decreases with any further increase in node speed (Figure 4-9 (b)). At higher node speed, such as, V_{max} = 30m/s, election rate decreases significantly as nodes get disconnected from their leaders but remains so for short duration. This decreases the probing time and ultimately the election time gets decreased.

We also show the variation of ET with N for different values of V_{max} for "Election-Opt" (Figure 4-9 (c)) and "Vasu" (Figure 4-9 (d)), separately. For both the cases, ET increases with increasing node speed. At higher speeds, link breaks occur more frequently which increases both the routing overhead (in terms of number of control

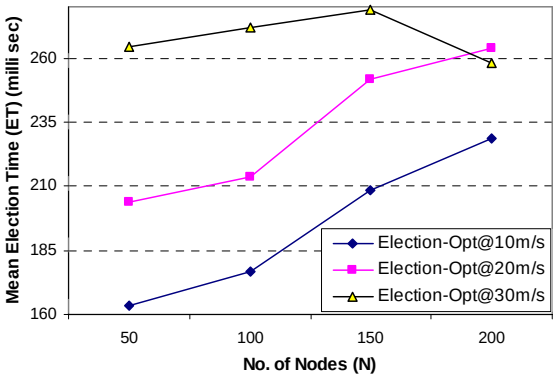
packets) as well as message delays. But the ET decreases at higher speed as the nodes remain disconnected only for very short durations. Thus, fresh election can be avoided.



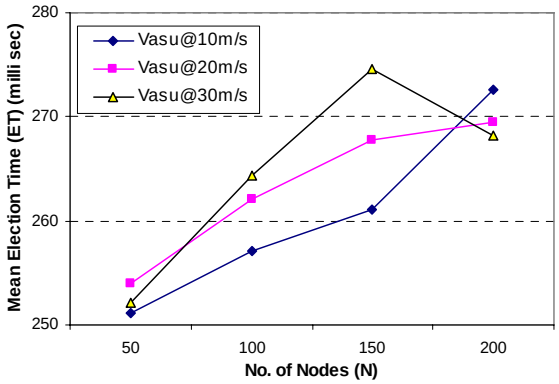
(a) ET vs. N (V_{max} = 10m/s)



(b) ET vs. N (V_{max} = 30m/s)



(c) ET vs. N for "Election-Opt"

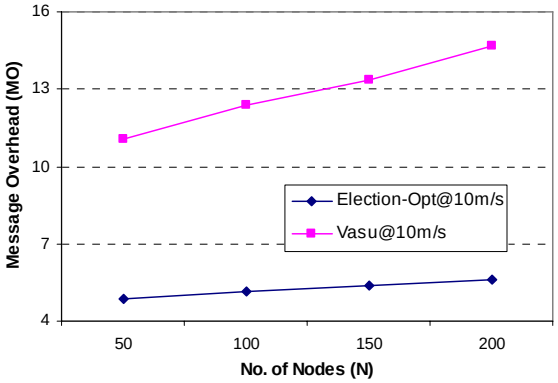


(d) ET vs. N for "Vasu"

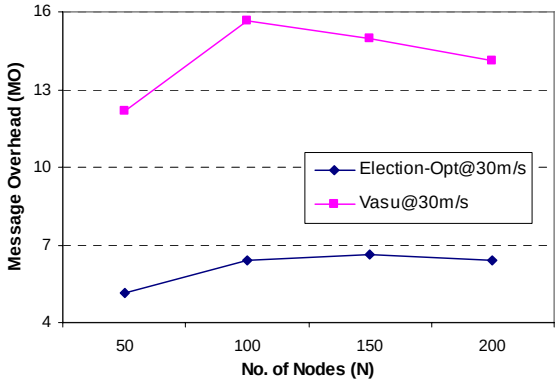
Figure 4-9: Effect of Mobility on Election Time (ET)

Effect of Mobility on MO: Figure 4-10 (a) and (b) depicts the change of MO with N by varying node speeds from 10m/s to 30m/s, for "Election-Opt" and "Vasu". As the election rate increases with N, message overhead also increases at lower speed (Figure 4-10 (a)) and similarly it decreases with further increase in node speed (Figure 4-10 (b)), as the election rate decreases. But the variation of MO for "Election-Opt" is significantly lower than that of "Vasu". For "Vasu", with the increase in ER, the number of nodes carrying out diffusing computation concurrently also increases which generates many messages. Our algorithm restricts MO by allowing only the RED nodes to carry

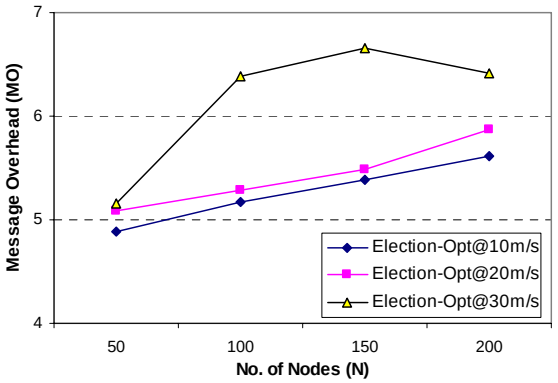
out diffusing computations. The ELECT messages generated in Phase I of our algorithm are also controlled in such a way. When a node starts Phase I and sends ELECT message to its neighbors, the receiver of ELECT message will not start a new election even if it detected the departure of a leader. This gives us an edge over "Vasu" in controlling MO and conserving energy.



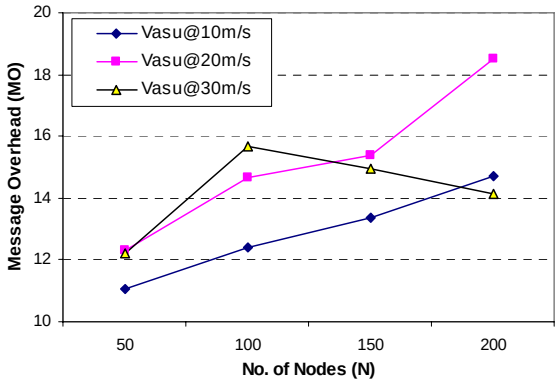
(a) MO vs. N (V_{max} = 10m/s)



(b) MO vs. N (V_{max} = 30m/s)



(c) MO vs. N for Election-Opt"



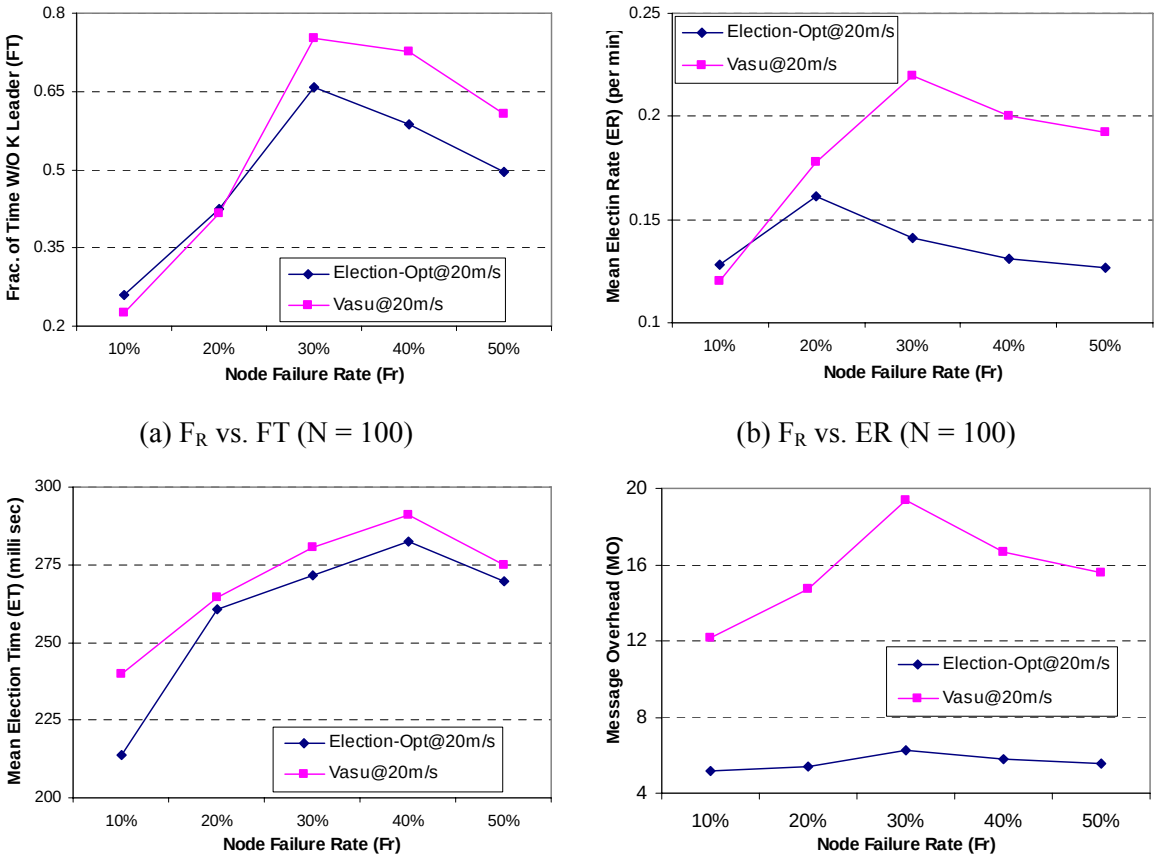
(d) MO vs. N for "Vasu"

Figure 4-10: Effect of Mobility on Message Overhead (MO)

C. Effect of Node Failure on Our Algorithm

In this sub-section, we report the results of varying node failure (F_R) rates on the performance of our algorithm. We fail a percentage, p, of the total number of nodes (N), participating in the election process in such a way that throughout the execution time, at

least as many as (p*N/100) nodes are always in the list of failed nodes. We release nodes in the head of this list if more than the expected number of nodes has been failed. The released nodes are considered as recovered nodes which takes part in K leader election as described in Section 4.4.5. Below we describe our results by varying node failure rates (F_R) from 10% to 50%. All the following experiments are executed at default values, i.e., with 100 nodes moving at a speed of 20m/s, the transmission radius being 250 m. The results have been reported in Figure 4-11 (a) to (d).



(a) F_R vs. FT (N = 100)

(b) F_R vs. ER (N = 100)

(c) F_R vs. ET (N = 100)

(d) F_R vs. MO (N = 100)

Figure 4-11: Effect of Node Failure on K-Leader Election

Effect of F_R on FT: Figure 4-11 (a) depicts the change of FT with varying node failure rates for "Election-Opt" and "Vasu". Initially FT increases with F_R and then

decreases with further increase in F_R. This can be explained by the fact that, at very high node failure rates, leader nodes frequently fail which results in low availability of leaders. We can observe that, even at high failure rates, "Election-Opt" can achieve higher leader availability compared to "Vasu".

Effect of F_R on ER: Figure 4-11 (b) depicts the change of ER with varying node failure rates for "Election-Opt" and "Vasu". The election rate increases initially and decreases with further increase in F_R. This can be explained similarly as for Figure 4-11 (a). With increasing F_R, elections are higher as leaders get unavailable frequently, but with very high F_R, the network is mostly partitioned and leader availability increases within each partition which decreases the election rate.

Effect of F_R on ET: Figure 4-11 (c) plots the change of ET with varying node failure rates for "Election-Opt" and "Vasu". The election time initially increases with F_R, as nodes get frequently partitioned which requires probing to detect failures. With further increase in F_R, however, network gets divided in small partitions with few nodes in each partition. This decreases the election time as the election needs to be carried out only within small set of participating nodes.

Effect of F_R on MO: Figure 4-11 (d) shows the change of MO with varying node failure rates for "Election-Opt" and "Vasu". Following the trend of election rate change with respect to F_R, message overhead also increases with increase in ER and decreases with decreasing ER.

We can argue from the above results that, our algorithm performs better than "Vasu" consistently and significantly.

4.7 Prototype Implementation

To further demonstrate the feasibility of our algorithm in real applications, we implement it in a testbed. We compared the performance of "Election-Opt" and "Vasu" on a sensor network testbed with 40 MicaZ [30] motes distributed over a single floor of the Man Wai building in the Hong Kong Polytechnic University.

4.7.1 Testbed Architecture

The testbed, as shown in Figure 4-12 (c), contains MicaZ [30] sensor nodes (Figure 4-12 (a)) running TinyOS [64] and a MIB600 gateway (Figure 4-12 (b)).

(a) MicaZ Node (b) MIB600 Gateway (c) Part of Our Testbed

Figure 4-12: Prototype Implementation of Top K Leader Election Algorithm

In order to make our system more flexible, we have implemented a two-layer-architecture. On the bottom, we have a topology layer which emulates physical and routing layer together and ensures end-to-end routing among network nodes. Above that layer we have the application layer which implements the weight-based K-leader election algorithm. The election algorithm gets the neighborhood information from the topology layer. We assign random numbers to different sensor nodes as weights. K is chosen as $\lceil N/4 \rceil$, where N is the total number of nodes in the network. The nodes are considered static as it is difficult to make them mobile in a testbed. We have also

implemented node failure. We arbitrarily switch off some sensor nodes during the algorithm execution and consider this as node failure. We consider 10% node failure rate.

4.7.2 Implementation

In order to implement our algorithm efficiently, we have adopted an event-driven model. We have a set of states for each node and a set of events. Based on the events we divide the protocol into multiple states. Node states are changed by triggering of an event. We have mainly three events – *send message*, *receive message* and *timeout*. Node states considered for "Election-Opt" are – *INIT*, *WAIT-for-ELECT*, *RECV-all-ELECT*, *SEND-VOTE*, *PHASE-II*, *RECV-SEARCH*, *SEND-ATTACH*, *FWD-SEARCH*, *WAIT-for-SIGNAL*, *PHASE-III*, and *FINISH*. Similarly, node states for "Vasu" are - *INIT*, *RECV-SEARCH*, *SEND-ATTACH*, *FWD-SEARCH*, *WAIT-for-SIGNAL*, *RCVD-All-SIGNAL*, and *FINISH*.

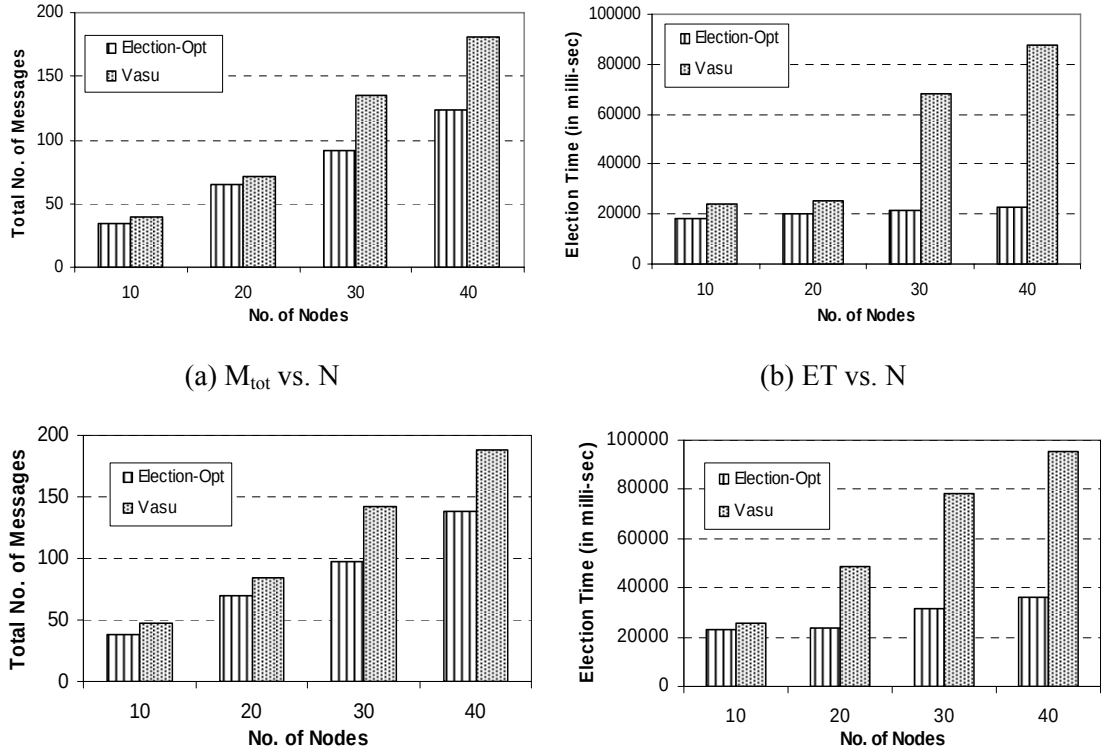
We measure the performance of our algorithm using two metrics – total number of messages exchanged for one election (M_{tot}) and total time for a single election (ET). We test the system with 10, 20, 30 and 40 nodes. We generate random topologies. The experiment is run for 10 times each with different topologies for a fixed number of nodes, and the average values are reported.

In addition to the testbed system itself, we also have a special node employed to monitor the performance of the system. Initially, when the execution starts, each node undergoes a bootstrapping phase to initialize the variables and chooses a random weight value. After that, the monitoring node broadcasts the topology information to all the nodes in the network. When a node receives the topology, it populates its neighbor list and starts the K leader election at randomly chosen time. When the execution finishes

with electing K leaders, the monitoring node sends the performance results to the computer.

4.7.3 Result Analysis

We can see from the Figure 4-13 (a) that the total number of messages for "Election-Opt" and "Vasu" increases with the increase in N. The increase in messages for "Vasu" is considerably higher than our algorithm which is due to the multiple diffusing computations proposed in their algorithm. Similar trend is shown in Figure 4-13 (b) where the total election times (ET) for both the algorithms have been plotted. "Vasu" has a very high value for ET with the increase in N pertaining to the many diffusing computation which needs to be finished before the K leaders get elected.



(a) M_{tot} vs. N

(b) ET vs. N

(c) M_{tot} vs. N (F_R = 10%)

(d) ET vs. N (F_R = 10%)

Figure 4-13: Performance Results of Prototype Implementation

In Figure 4-13 (c) and Figure 4-13 (d) we show the performance results of "Election-Opt" and "Vasu" in presence of node failures. We have implemented our mechanisms of handling node failures and the results indicate that our algorithm still performs better than "Vasu" in real scenarios where node failures may occur. But the total number of messages and the election time increases on the face of node failures. M_{tot} increases because of the multiple probing messages used and the ET is increasing as the probing time adds up to the normal election time.

The prototype implementation results clearly show the ability of our algorithm to be implemented in practical scenarios consisting of multiple resource-constrained devices.

4.8 Summary

In this chapter, we describe in depth about the formation and maintenance of directory community structure – as a basis for reliable service discovery in dynamic pervasive environments. Due to the similarity of characteristics between pervasive environments and mobile ad hoc networks, we have modeled the underlying network of a pervasive computing system as a MANET. The directory community consists of K nodes with highest available resources in the network of MANET nodes. We modeled the directory community formation problem as a K leader election problem in MANET based on available node resources. To solve this problem, we propose a ratio-based top K weighted leader election algorithm for MANET considering frequent and dynamic topological changes where node weight represents resources available at a mobile node, e.g. processing capability, battery power, etc.

Our algorithm aims to elect the top K weighted leader nodes where K is a ratio of the leaders to the total nodes in a connected network component. A diffusing computation approach is adopted to collect the weight information for electing K leaders. To reduce

message cost, we first locally choose some higher weight nodes among 2-hop neighbors, as RED nodes, to act as coordinators. Then, each RED node initiates a diffusing computation procedure to collect the weight information of other nodes collaboratively. Finally, the information collected by different RED nodes is merged together to elect the final leaders. Such an approach is useful as we can minimize the number of diffusing computations and hence can save many messages. Since, we address leader election for resource constrained nodes, a message-efficient approach will save more node resources. We also design mechanisms to handle node failures and network partitions. The simulation results and testbed experiments show that, benefiting from the use of RED nodes, our algorithm can elect weighted top K leaders with much less message cost than other algorithms.

Chapter 5
Quorum-based Reliable Service Discovery

This chapter describes our reliable service discovery mechanism using a quorum-based fault-tolerant service discovery protocol (SDP) for MANET developed over the directory community framework. Section 5.1 presents a brief overview of the issues which need to be addressed in order to develop a reliable service discovery protocol for mobile ad hoc environment. Section 5.2 introduces some preliminary operations required for our protocol. The service discovery protocol has been described in detail in Sections 5.3 to 5.4 along with the results of in depth performance analysis. Finally, Section 5.6 concludes this chapter by summarizing our contributions.

5.1 Overview

Service discovery in MANET has remained an interesting research problem for years. Existing service discovery protocols in MANET address several challenging issues, such as, building proper infrastructure to reduce discovery delay and boost discovery success, and providing support for enhancing scalability [85]. But the service availability issue has been largely ignored.

Service unavailability can arise due to the failure of service provider or directory nodes. While service provider failure is easy to address, directory failures are rather complicated to handle as it renders all services registered with the failed directory unavailable. As already mentioned in Chapter 3, we want to develop a reliable service discovery protocol using the directory community framework, and we have introduced the design issues. Directory-based service discovery protocols for MANET are more robust than the directory-less ones, though the former incurs high maintenance overhead.

Our proposed service discovery protocol [79] is fault tolerant and message-efficient where multiple directory nodes work collaboratively to enhance system robustness by increasing service availability in the system. After the directory community has been successfully formed, the elected directory nodes form quorums among themselves. In order to ensure network-wide service availability with minimal replication, each directory replicates all the services registered with it, with only its quorum members. Following the quorum intersection property, we can guarantee that if a service matching user request is available, the user can certainly find the service by forwarding a request only to its quorum members. This reduces service discovery cost. The message overhead is further checked by dividing the network into one or more tree-structured domains, thereby eliminating loops, and also by restricting broadcast and flooding. We also consider directory failure, where, a failed directory is one which either crashes or becomes unfit of hosting services due to dissipation of resources. We handle directory failure by timely replacing a failed directory with a suitable node carefully picked up using an incremental election approach. Our protocol is also able to cope with dynamic topology changes caused by frequent network partitions and node failures.

However, there are some works similar to our approach. As already mentioned in Chapter 2, Kim et al. [57] have proposed a volunteer node based service discovery protocol for MANET where volunteers are relatively stable and resource rich nodes and

form an overlay structure. The volunteers in fact act as directory nodes and they are mobile in nature. Due to the close similarity of our protocol with this approach, we have decided to compare the performance of our protocol with this one. This approach, however, has a couple of limitations which makes it more costly than our protocol, as we shall see later. Firstly, the overlay may develop loops and cycles, which increases service discovery cost. Secondly, the volunteer advertisement broadcast can significantly increase the traffic. There is also a quorum-based service matching policy [6] which is somehow related to our solution but has many differences. The proposed approach is for wireless mesh networks. Mesh nodes are static and are not constrained by energy, so they do not face the usual problems of a dynamic MANET environment. The authors also did not consider fault tolerance issues and did not evaluate their approach by simulations or testbed experiments.

We have carried out extensive simulations to evaluate the performance of our proposed protocol, and we present our results with in-depth analysis. Our results show that our protocol is scalable and fault-tolerant. Also, it can guarantee service availability with low message overhead. Moreover, our protocol can handle dynamic and concurrent topological changes. We have also implemented our protocol on a wireless testbed system. The experimental results obtained are found to be in congruence with the simulation results. The testbed experiments prove that our protocol can be useful in practical scenarios comprising an ad hoc composition of multiple low resource devices.

5.2 Protocol Preliminaries

As mentioned previously, our service discovery protocol works by using the directory community framework. Here we briefly recapitulate the directory community formation approach. This approach also divides the entire network into multiple tree-structured domains. We utilize these domains in our service discovery protocol.

5.2.1 Directory Community Formation and Domain Construction

Directory community is created by electing top K weighted directory nodes where weight is any resource-related attribute of a node. The problem is formulated as a top K leader election [78] problem in MANET. The algorithm operates in three phases and uses diffusing computations [33].

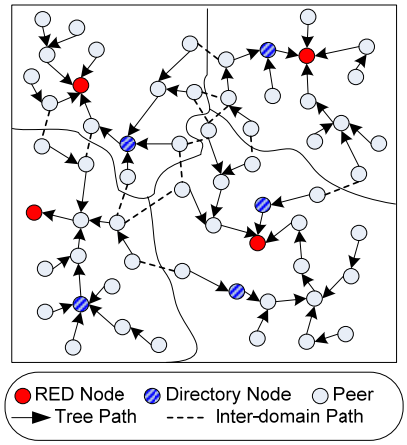


Figure 5-1: Service Discovery Architecture

In Phase I, one or more nodes having the highest weight among their 2-hop neighbors are voted as RED nodes. Other nodes are called WHITE nodes. Then, in Phase II, the RED nodes start the diffusing computation procedure asynchronously in order to collect weight values of other nodes in the network. Each RED node builds a diffusion tree (Figure 5-1) and at the end of Phase II, each RED node has weight information of the nodes in its diffusion tree and knows about other neighboring RED nodes with whom its own diffusing computation tree meets. So, after Phase II, the entire network is divided into several tree structured domains, termed as domain trees, with a RED node as the root of each domain tree. As RED node knows other nearby RED nodes, they act as gateways to the neighboring domains. Finally, in Phase III, all the RED nodes coordinate among themselves, and the results collected by different RED nodes are merged, in such a way, that, eventually the highest weight RED node receives the

complete weight information of all the nodes in the network. It then chooses the top K highest weight nodes as directories and informs every other node. Each directory then executes a quorum generation algorithm described below and constructs its own quorum.

5.2.2 Construction of Directory Quorum

After the directory nodes have been elected, they execute a quorum generation algorithm to form directory quorums. In our protocol, the quorum is constructed following the method proposed by Lin et al. [65]. We choose their method because of several advantages. Firstly, the quorums are symmetric, i.e. each node is included in the same number of quorums and all the quorums have the same number of nodes. This will ensure even load distribution among the directory nodes. Secondly, this method allows flexibility in quorum size with respect to the number of directories. Finally, the method allows generation of multiple quorums for each node (based on multiple generating sets) to enhance the availability of the quorum system. Figure 5-2 presents the pseudo code of the quorum generation algorithm given by [65].

/***/
//The code executed by each node, i
(1) *elect set of directories* $K \rightarrow \{0,1,.....,K\text{-}1\}$;
//This code executed by each directory node
(2) *go to Q_Gen(K) to generate directory quorums*;
/*************************$Q\_Gen\ Algorithm$************************/
(2A) Var: c, τ, m, w, i, K
(2B) $c \leftarrow \left\lceil \frac{K}{m} \right\rceil$, $\tau \leftarrow \left\lceil \frac{K+1}{2m} \right\rceil$, $1 \leq m \leq K$;
(2C) Quorum generation set
$Q_g = \{0, 1, ..., m\text{-}1, w_1, w_2, ..., w_{\tau-1}\}$ ..... (i)
where, $m - 1 \leq w_1 \leq 2m - 1$, $0 < w_{i+1} - w_i \leq m$ for all $1 \leq i \leq \tau\text{-}2$, and $w_{\tau-1} \geq (K\text{-}1)/2$

Figure 5-2: Pseudo-code for the Construction of Directory Quorums

From equation (i) in the *Q_Gen Algorithm* in Figure 5-2, there can be more than one quorum generation set for a given m. Lin et al. [65] has proved that the size of a quorum

can be smallest when m equals $\sqrt{\frac{K+1}{2}}$. Let us consider an example with 25 directory nodes $\{K_0, K_1, \ldots, K_{24}\}$, so that, $K = 25$, $m = 4$, $c = 7$ and $\tau = 4$. Some of the possible quorum generation sets formed with the given values are: {{0, 1, 2, 3, 4, 8, 12}, {0, 1, 2, 3, 5, 9, 12}, {0, 1, 2, 3, 6, 9, 12}, {0, 1, 2, 3, 6, 9, 13},…}. We construct quorums using the generation set {0, 1, 2, 3, 5, 9, 12} as shown in Figure 5-3.

K0: {0, 1, 2, 3, 6, 9, 12},	K12: {12, 13, 14, 15, 18, 21, 24},
K1: {1, 2, 3, 4, 7, 10, 13},	K13: {13, 14, 15, 16, 19, 22, 0},
K2: {2, 3, 4, 5, 8, 11, 14},	K14: {14, 15, 16, 17, 20, 23, 1},
K3: {3, 4, 5, 6, 9, 12, 15},	K15: {15, 16, 17, 18, 21, 24, 2},
K4: {4, 5, 6, 7, 10, 13, 16},	K16: {16, 17, 18, 19, 22, 0, 3},
K5: {5, 6, 7, 8, 11, 14, 17},	K17: {17, 18, 19, 20, 23, 1, 4},
K6: {6, 7, 8, 9, 12, 15, 18},	K18: {18, 19, 20, 21, 24, 2, 5},
K7: {7, 8, 9, 10, 13, 16, 19},	K19: {19, 20, 21, 22, 0, 3, 6},
K8: {8, 9, 10, 11, 14, 17, 20},	K20: {20, 21, 22, 23, 1, 4, 7},
K9: {9, 10, 11, 12, 15, 18, 21},	K21: {21, 22, 23, 24, 2, 5, 8},
K10: {10, 11, 12, 13, 16, 19, 22},	K22: {22, 23, 24, 0, 3, 6, 9},
K11: {11, 12, 13, 14, 17, 20, 23},	K23: {23, 24, 0, 1, 4, 7, 10},
K24: {24, 0, 1, 2, 5, 8, 11}.	

Figure 5-3: An Example Directory Quorum

One can see that any two quorums have a nonempty intersection. One can also use a union of multiple generating sets as a single quorum generation set. This approach increases the availability of the quorum system. If the number of directories $K = n^2$ for some n, the size of quorums generated by the above described quorum construction method is $\approx (3n)/2$, for $m = n$. This quorum size is comparatively smaller than many other available schemes and that is why we chose this method for quorum generation over the others. For every directory node we generate a quorum of the smallest size.

5.3 The Proposed Service Discovery Protocol

Our service discovery protocol works using the backbone of domain trees and ensures fault tolerance by replicating services in the directory quorums. Key to the efficient functioning of our protocol is the proper and timely maintenance of the domain trees and the directory quorums in the face of sheer dynamicity posed by the mobile ad

hoc networks. In this section we shall describe our protocol in detail. The first part concerns our policies for the maintenance of service discovery infrastructure. It includes, techniques to maintain directory quorum and domain tree structures and mechanisms to handle frequent network partitions and partition merging. Following that we shall elaborate the methods for service registration and service information replication and finally we describe the mechanisms for service request and reply. A description of the data structures and message types used precedes the formal description of our protocol.

5.3.1 Data Structures and Message Types

While executing our protocol, each node i maintains necessary information about its state in the data structures listed in Table 5-1 and may exchange types of messages listed in Table 5-2.

Table 5-1: Data Structures for Reliable Service Discovery Protocol

Variable	Meaning
id_i	Identifier of node *i*
wt_i	Weight of node *i*
wt_sent_i	Binary variable indicating whether node *i* has informed its weight to the elector or not
root	Identifier of domain root node
dir	Binary variable indicating whether node *i* is a directory or not
$pred_i$	Predecessor of *i* in the diffusion tree
$succ_i$	Successors of *i* in the diffusion tree
elector	Identifier of the highest weight RED node in the network
new_wt	Set of new nodes joining the network and their weights maintained by the *root*
Dir_List	Set of domain directories known to the domain root
Node_List	Set of all nodes in the network and their weights maintained by the *elector*
LDR	Set of *K* directories and their weights maintained by the *elector*
Q_Mem	Set of quorum members of a directory
Pub_Host	Nearest directory of a service provider with which it registers its service
LookUp_Host	Nearest directory of a service requestor from which it requests a service

Table 5-2: Message Types for Reliable Service Discovery Protocol

	Message	**Purpose**
Infrastructure Maintenance Messages	ADV_{dd} *(root, pred, Dir_List, new_wt)*	Domain directory advertisement by domain root to its successors and the elector. Parameter *new_wt* is optional. Only when a new node joins the network, its weight is sent to elector.
	Dom_Dir_HB(wt_i)	Heart-beat message by a domain directory to the domain root upon receiving an ADV_{dd}. The weight value is sent to check whether the directory node is still capable of hosting services.
	ATTACH_REQ	A disconnected or newly joined node broadcasts a request to join a domain tree
	REP	A node replies upon receiving an *ATTACH_REQ*
	ATTACH(i)	Node *i* requests to attach to the node which first sends a *REP*
	ACK(i,wt_i*)*	A newly joined node sends its weight to the domain root, upon receiving a ADV_{dd}
	LEADER (LDR)	For the elector to announce the new set of K leaders to all other nodes
Service Discovery Messages	*REG (S)*	A service provider registers its service to its *Pub_Host*
	Q_REG(S)	*Pub_Host* of a service provider registers the service with each node in its *Q_Mem*
	Q_ACK(S)	Each node in the set *Q_Mem* of a *Pub_Host* acknowledges successful registration of a service to the *Pub_Host*
	REG_Done(S)	A *Pub_Host* acknowledges successful service registration to the service provider
	REQ(S)	A service requestor requests for a service to its *LookUp_Host*
	Q_REQ(S)	*LookUp_Host* forwards the service request to each node in its *Q_Mem*
	Service_Reply(S)	Reply containing a matching service

5.3.2 Maintenance of Service Discovery Infrastructure

After the preliminary operations have been completed (Section 5.2), every domain root periodically collects information about the directories present in its domain and sends a common directory advertisement message (ADV_{dd}) to other nodes which contains list of all the domain directories. To avoid broadcast, directory advertisements are distributed using the domain tree. A node receiving ADV_{dd} caches it and finds the nearest directory node.

Maintenance of service discovery infrastructure is crucial for our service discovery protocol. Below we discuss separately the procedures of maintaining domain tree structures and the directory quorums in presence of node mobility and arbitrary topological changes. We also discuss our mechanisms to cope with frequent network partitions and partition merging.

A. Domain Tree Maintenance

To maintain the domain tree structure we need to handle the following four cases that may happen mainly due to the node mobility:

- *Case I) A Node Leaves Current Domain*: When a node leaves the domain the domain tree gets disconnected and the successors of the departing node have no parent. To avoid this condition, it is important to detect such a scenario and handle it, in order to keep the domain tree connected. In our protocol, every node periodically checks its connectivity with its parent (pred) in the tree. If node *i* loses connection with parent *j*, node *i* broadcasts an *ATTACH_REQ* to all nodes within its wireless transmission range. Any node *n* receiving an *ATTACH_REQ* replies with a *REP* and starts a timer. When node *i* receives the first REP, it sends an *ATTACH* to the sender and ignores the other *REP*s. When node *n* receives an *ATTACH* from node *i*, it adds *i* in the successor set and forwards the next incoming ADV_{dd} to it along with other successors. If no *REP* is received by node *i* after sending an *ATTACH_REQ* then possibly a partition has occurred which can be handled as described in Section 5.3.2 C.

- *Case II) A Node Joins A New Domain*: A node can join a domain either as a node which freshly joins the network or as a node which migrates from another domain. In either case, the node assumes that it has no parent and it follows the

steps mentioned in *Case I* in order to join a domain tree.

- *Case III) A Directory Node Leaves Current Domain*: The domain roots periodically track the domain directory nodes to ensure their presence. When a directory node receives an ADV_{dd} message, it sends a *Dom_Dir_HB* to the domain root. When the domain root receives *Dom_Dir_HB* messages it updates the list of available domain directories in order to reflect joining of new directories to the domain and leaving of old directories from the domain. In case, the weight of a directory falls below some threshold, the domain root informs the elector to replace it considering failed. Also the domain tree is updated to keep it connected when a directory node leaves the current domain. When a directory node joins a different domain following *Case II* it will send the heart-beat message to the new domain root upon receiving the ADV_{dd}.

```
/*****************************************************/
// After domain directories have finished Quorum Generation
(3) send Dom_Dir_HB to domain root r;
(4) while node r receives Dom_Dir_HB from directory d;
      Dir_List ← d;
(5) when all domain directories have sent Dom_Dir_HB {
      send ADV_dd to each i ∈ succ_r and to elector;
      start the timer for next ADV_dd;
      }
(6) while node i receives ADV_dd from pred {
      if (dir = TRUE) {
            send Dom_Dir_HB to root r ;
            send ADV_dd to each j ∈ succ_i ;
      }
      else {
            if(wt_sent = = FALSE) { //newly joined node
                  send ACK(i, wt_i) to r;
                  wt_sent = TRUE;
            }
            Pub_Host = LookUp_Host ← nearest (Dir_List);
            send ADV_dd to each j ∈ succ_i ;
      }
  }
```

```
(7) while node r receives ACK(i, wt_i) from i
        new_wt ← (i, wt_i);
(8) send new_wt to the elector with the next ADV_dd ;
/*****************************************************/
//This part is executed by elector e (incremental directory election)
(9) while node e receives ADV_dd from domain root r {
          tmp ← Dir_List_r;  //Var: tmp => temporary directory list
          Node_List ← new_wt_r;
    }
(10) if all roots have sent ADV_dd {
        match tmp with LDR;
        if they match with original top-K
             do nothing;
        else {
              elect new nodes from Node_List for the failed ones;
              send LEADER to each l ∈ LDR ;
        }
    }
/*****************************************************/
//Domain Tree Maintenance
//This part is periodically executed by each node i
(11) check connectivity with pred;
(12) if connected
            do nothing;
       else {
              pred = NULL;
              broadcast ATTACH_Req;
       }
(13) if node j receives an ATTACH_Req
          send REP and start a timer T;
(14) if node i receives REP from j { //for the ATTACH_Req it sent
          if (pred = = NULL) {
               send ATTACH to j;
               pred = j;
          }
          else
              do nothing;
(15) while node j receives ATTACH from i {
           if (timer T not expired) {
                succ_j ← i;
                stop timer;
           }
    }
```

Figure 5-4: Pseudo-code for the Maintenance of Service Discovery Infrastructure

■ *Case IV) A Domain Root Leaves Current Domain*: This problem is tackled by

using a backup node for the domain root. Every root node selects a comparatively high resource node as the backup root. The backup root node will monitor the root node and in case the latter fails or migrates out of the current domain, the backup node can take over as the root node. If the domain root joins a different domain, it follows *Case II*.

B. Directory Quorum Maintenance

Maintaining the directory quorum structure is critical to satisfy the quorum intersection property which assumes that quorum nodes are stable and always available. Unless a failed quorum node is replaced readily, service lookup may fail. In order to ensure higher availability of quorum members, we propose an incremental directory election approach.

After a newly joined node attaches to some domain (refer to *Case II* of Section 5.3.2 A.), it receives a ADV_{dd} from the domain root, and sends an *ACK(id, wt)* message to the root informing its id and weight value. Domain roots periodically update the elector about the newly joined nodes, so that, the elector always maintains a list of all the nodes currently available in the network. Once a node has sent its weight value to the elector, it does not need to do such operation any more even if it migrates across domains. But, a node must update its weight to the domain root if its weight changes from the initial value. In case, the existing directories become unavailable, the elector chooses from other suitable nodes to replace for unavailable directories and sends *LEADER* messages to each directories informing about the changes. After a new directory takes charge for an old directory, it will contact its quorum members and register services maintained by them. It will also send heart beat to its current domain root, so that the domain root includes it as a domain directory in the next ADV_{dd} message.

C. Handling Network Partitions

Network partitions may occur at anytime during the service discovery operation. This can divide the network into two or more components with more or less than K directories in individual components. This will upset the quorum structure and hence the service discovery will be affected.

In order to cope with this frequent variation in number of directory nodes in a network component we propose to choose K as a ratio of the number of directories to the total number of nodes in the current network component. Thus, the value of K is adjusted when a network gets partitioned or when two or more network components merge together.

In case a partition occurs, it is important to detect the partition as quickly as possible. Our protocol can detect partition as described earlier in the *Case I* of Section 5.3.2 A. The detecting node then informs its nearest directory D, which tries to contact the elector. If the elector or its backup node exists in the same component it collects the node weights by traversing through the domain tree and adjusts the K directories by adding new directories or removing some directories depending on the number of nodes in the network component. The directory nodes then re-adjust the quorums accordingly to achieve maximum reliability. If no elector exists in the partition, the directory node D traverses the domain tree to collect different node weights. The highest weight directory then takes charge as the elector. Service discovery operation during the quorum reorganization process may be affected little bit but our experimentation shows that the effect is very little and the approach is scalable.

D. Handling Merging of Network Partitions

Node mobility may cause network partitions to merge at times. The number of K directories can be different for the two merging partitions depending on the total number of nodes present in them. When two partitions come closer, they form a link to inter-connect. The meeting nodes of two adjacent partitions exchange their elector information over the newly formed link. The two elector nodes can then communicate among themselves and the highest weight elector will be selected as the unique elector. The elector then chooses the top K weighted nodes, from the sets of directory nodes of the two merging components, as directories for the newly formed network component. The new directory nodes then require reforming the quorums to carry on with fault-tolerant service discovery. Experiments show that very few directory nodes are changed when partitions merge causing minimal quorum reformation overhead.

5.3.3 Service Registration

Service providers register only with functional information. Detailed service context (Figure 5-5) is forwarded to the user on request.

Service Identification: *Type (generic service class), Name, Description, Alias, Version number*
Service Functionality: *Specific functionalities provided by a service instance*
Service Attributes: *Cost, Time (time of service availability or validity), Security features, Provider's address (IP, URL, etc.), Make, Special attributes (color or B&W for a printer)*
Service Requirements: *Computational (CPU power), Storage (Memory size), Energy (remaining battery life), Display (screen size), Others (e.g., audio, input/output, software, etc.)*

Figure 5-5: Service Context Information

When a service provider ($\mathcal{P}$) wants to register a service ($\mathcal{S}$), he registers with the nearest directory, called the *publishing host* (*PH*). Similarly, for a user looking for a

service, the nearest directory with which he may register is called the *lookup host* (*LH*). The distance between two nodes can be obtained from the underlying routing protocol.

After receiving the registration request from $\mathcal{P}$, the *PH* registers the service locally and forwards the request to its quorum members. After the service has been registered at all the quorum members, *PH* receives an acknowledgement from each of them and then it sends a *REG_Done* message back to $\mathcal{P}$ informing $\mathcal{P}$ about the other directory nodes to which service $\mathcal{S}$ has been registered.

```
/************************************************/
// When a service provider P wants to register a service S
(16) P sends REG (S) to Pub_Host PH;
(17) while PH receives REG (S) from P {
        register (S);
        send Q_REG (S) to each q ∈ Q_Mem
     }
(18) while a Q_Mem of PH receives Q_REG (S) from PH {
        register (S);
        send Q_ACK (S) to PH;
     }
(19) while PH receives Q_ACK (S) from all q ∈ Q_Mem {
        send REG_Done(S) to P;
     }
/************************************************/
// When a service requestor R wants to receive a service S
(20) R sends REQ (S) to LookUp_Host LH;
(21) while LH receives REQ (S) from R {
        if (match (S) = = TRUE) //try to match S with registry
           send Service_Reply(S) to R;
        send Q_REQ (S) to each q ∈ Q_Mem
     }
(22) while a Q_Mem of LH receives Q_REQ (S) from LH{
        if (match (S) = = TRUE) //try to match S with registry
           send Service_Reply(S) to R;
     }
```

Figure 5-6: Pseudo Code for Service Registration and Service Discovery

The service registration is updated using a lease-based approach. Each registration is associated with a lease. The service provider $\mathcal{P}$ needs to renew its lease with the *PH* before timeout. Failing to do so, due to unavailability of either $\mathcal{P}$ or *PH*, will prompt the *PH* to delete the service information and to inform its quorum members to follow suit.

In case of any change made to $\mathcal{S}$, $\mathcal{P}$ informs the *PH* about the new version number. *PH* makes the change and updates its quorum members in order to maintain consistency.

If the *PH* is unavailable, $\mathcal{P}$ can try to contact other quorum members of *PH* - nearest to it - with whom it can renew the lease. This ensures the availability of $\mathcal{S}$ even if the *PH* fails. If $\mathcal{S}$ is updated in the mean time, to a higher version, its replicas will not be updated accordingly. So, if a user discovers $\mathcal{S}$, he will ask $\mathcal{P}$ for the copy with the highest version number.

When *PH* fails, $\mathcal{P}$ registers with a new *PH*, either from its cache, or sends out a directory request, to the neighboring domain, in case the cache is empty.

5.3.4 Service Request/Reply

When a user (U) is interested to discover a particular service $\mathcal{S}$, he sends a discovery request to its lookup host (*LH*). After LH receives a user request, it checks its own registration information for a matching service and if available, it replies the user with the identity of the service provider, $\mathcal{P}$. LH also forwards the discovery request to its quorum members which can directly reply to U if a matching service is found.

After U receives a reply, it checks the functional information and contacts the provider to receive complete service context information. Unused service replies are cached for possible future needs.

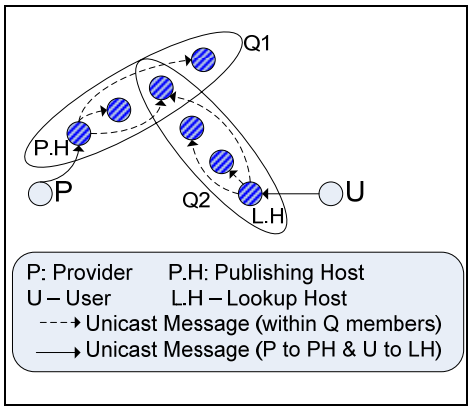


Figure 5-7: Quorum-based Service Matching Process

To illustrate the quorum based service discovery approach (Figure 5-7), we use the following example. Let us consider that, the service provider $\mathcal{P}$ has registered the service $\mathcal{S}$ with its publishing host, say $\mathcal{K}11$, (Figure 5-3) which has replicated the service to its quorum (Q_{11}), i.e., $\{\mathcal{K}_{12}, \mathcal{K}_{13}, \mathcal{K}_{14}, \mathcal{K}_{17}, \mathcal{K}_{20}, \mathcal{K}_{23}\}$. Now when U sends a service discovery request to its lookup host, say $\mathcal{K}_{23}$, it then forwards the request to its quorum Q_{23}, i.e. $\{\mathcal{K}_{23}, \mathcal{K}_{24}, \mathcal{K}_{0}, \mathcal{K}_{1}, \mathcal{K}_{4}, \mathcal{K}_{7}, \mathcal{K}_{10}\}$. Since, $Q_{11} \cap Q_{23} = \{\mathcal{K}_{23}\}$, $\mathcal{K}_{23}$ can match U's discovery request with $\mathcal{P}$'s advertisement.

Following the quorum intersection property, we can guarantee that, using our service discovery approach, if a service requested by a user is ever published and available, it must be matched through the user's lookup host or its quorum members.

5.4 Performance Evaluation

We have carried out extensive simulations to evaluate the performance of our proposed protocol. To prove the efficiency of our protocol, we have chosen to compare with the volunteer node based service discovery protocol presented in [57]. We generally follow the simulation settings from [57] for easy comparison. The time to live (TTL) values for the volunteer advertisement (TTL_{max_a}) and the service request (TTL_r) messages are fixed at 3 and 2, respectively to obtain the best performance of their

protocol. Moreover, for the optimal performance, every client belongs to a maximum of 2 volunteer regions and each volunteer advertises themselves once every simulation minute. Rest of the simulation parameters are same for both the protocols and are listed in Table 5-3.

Table 5-3: Simulation Parameters for Reliable Service Discovery Protocol

Parameters	**Values**			
Number of nodes, (N)	50	100	150	200
Territory scale (m^2)	700	1000	1200	1400
Territory scale (m^2) *[Simulation with variable node density]*	1500			
Number of service types (n_s)	20	40	60	80
K/n	25%			
Mean Link Delay (ms)	5			
Max Link Delay (ms)	100			
Transmission radius (m)	250			
Transmission radius (m) *[Simulation with variable node density]*	100			
Routing-protocol	Least hops			
Node failure rate (F_R) (in %)	10, 20, 30, 40, 50			
Mobility Model	Random Waypoint			
Max. node speed (V_{max}) (in m/s)	5, 10, 20			
Min. node speed (V_{min}) (in m/s)	5			
Pause time (ms)	10			
Parent Probing time (per minute)	1			
Directory advertisement (per minute)	1			
RED node to elector beacon (per minute)	1			
Service request (per minute)	2			

5.4.1 Simulation Setup and Metrics

The simulation system consists of two modules: the network backbone consisting of domain trees and directory quorums and the service discovery protocol. We consider 100 nodes moving at 10m/s with a 20% node failure rate as default. Unless otherwise specified, these default values will be used for our experiments. The network nodes are randomly scattered in a square territory. We have carried out simulations to test the effects of scalability and node density. The effect of scalability is studied by varying total number of nodes while scaling the territory size accordingly, so as to keep the node density constant. On the other hand, the effect of node density is studied by varying the total number of nodes while keeping the territory size constant. For message routing, we have implemented a simple protocol based on the "least hops" policy, which is adopted in many classical routing protocols in ad hoc networks. A routing table is proactively maintained at each node.

The directory election and quorum formation are carried out prior to the service discovery operations. Later, directory nodes are incrementally added to replace failed directories. The weight values of the nodes are assigned randomly. In case of node failures, the faulty nodes are randomly selected and a faulty node crashes in a randomly chosen time. The failure detection part is simulated based on the heartbeat-like approach.

We assume that, every node is a service provider that provides a service of a certain type. A service type is assigned to each node in a round robin fashion. If there are 10 service types in a network with 100 nodes, 10 nodes provide the same service. Therefore, with high n_s, service density ($d_s = N/n_s$ average number of service providers providing a same type of service) is low and vice versa.

In the simulations, we measure the performance of both the protocols using the following metrics:

NM (Number of Messages): The total number of messages exchanged for service discovery. This includes the messages required for directory election, quorum formation and backbone maintenance. Here, a "message" refers to an "end-to-end" message, i.e. a message from the source to the destination node. Such a message may be forwarded by several intermediate nodes in the network level.

NH (Number of Hops): The total number of hops of the messages exchanged to achieve the global decision. One "hop" means one network layer message, i.e. a point-to-point message. Compared with NM, NH can reflect the message cost of an algorithm more precisely.

HR (Hit Ratio): The ratio of the total number of successful discovery requests to the total number of discovery requests.

TD (Time Delay): This is the average delay between the time any successful request is sent from a client and the time corresponding reply is received by the same client. TD is measured in milliseconds.

5.4.2 Simulation Results and Analysis

Below we present our simulation results with analysis. We have simulated our protocol, labeled as "Q-SDP" and the volunteer-based SDP [57] labeled as "V-SDP". We run each simulation for 20 simulation minutes and each point is obtained by averaging over 10 different runs. As already mentioned before, we simulate the protocols twice, once by varying the nodes and territory size while keeping node density constant and again by varying the node density while keeping the territory size constant.

5.4.2.1 Simulation with Fixed Node Density

We first report the performance in general cases and then we shall discuss the effect of mobility and node failure rates on the performance, all in separate sub sections.

A. General Performance of Quorum-based Service Discovery Protocol

In this sub-section, we discuss the performance of "Q-SDP" and "V-SDP" under the default values of the simulation parameters. We plot the results of our experiments in Figure 5-8 (a)-(d).

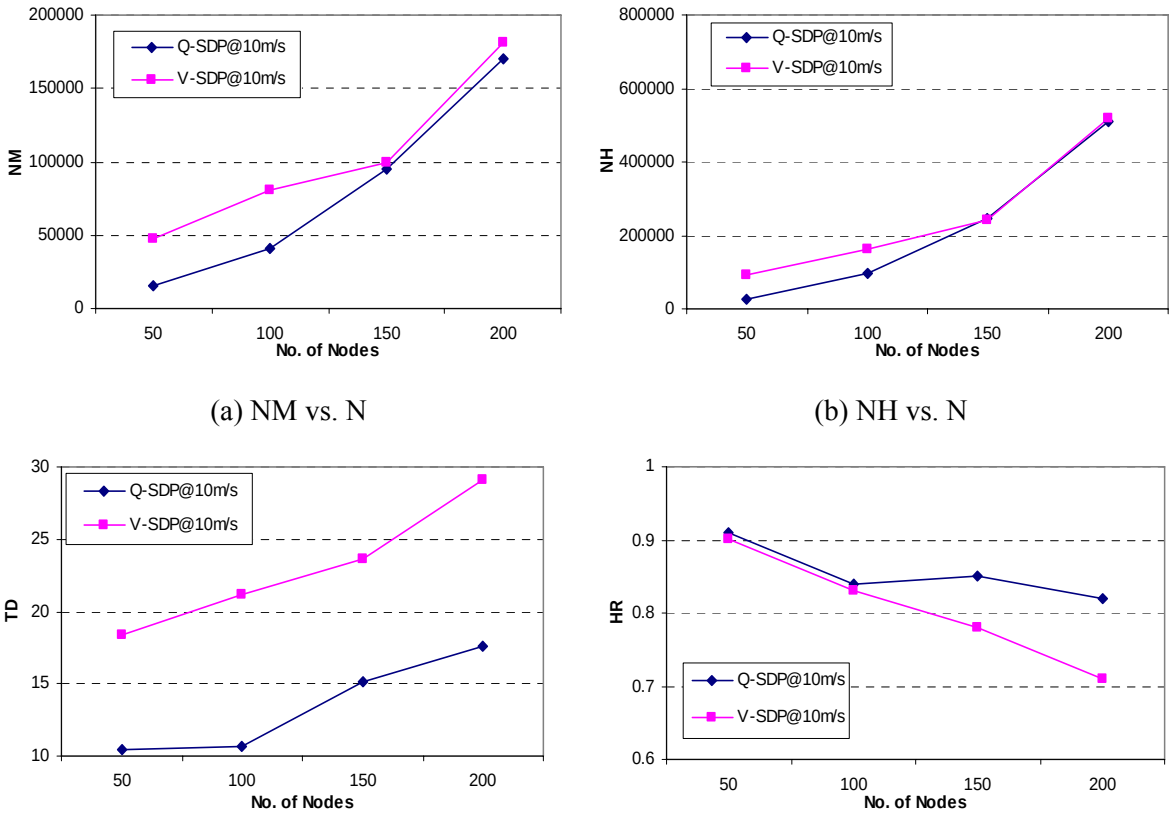
Performance of NM: From Figure 5-8 (a) we observe that NM increases with N and "V-SDP" consistently uses more messages than "Q-SDP". This is because, while "Q-SDP" disseminates messages using a tree, "V-SDP" broadcasts volunteer advertisements. Also, service discovery requests are sent through volunteer overlay which can have loops and cycles and hence increases NM. In presence of node failure, service providers and volunteer nodes may fail and the rate of broadcast significantly increases thereby increasing the NM.

Performance of NH: Figure 5-8 (b) shows that NH increases with N and "V-SDP" incurs higher NH than "Q-SDP". This observation follows directly from that of NM as increase in NM implies increase in NH. But, as "V-SDP" controls the message hops using pre-defined TTL values, hence the NH for "V-SDP" are not very significantly different than that of our protocol.

Performance of TD: From Figure 5-8 (c) we can see that increase of TD with respect to N is faster for "V-SDP" than "Q-SDP". This is due to the fact that with node mobility and node departure or failure, volunteer nodes are scarce and the delay in discovering service becomes higher as the volunteer nodes decrease in number. We, on

the other hand, significantly reduce TD by choosing nearest directory as *LH* or *PH* and forwarding discovery requests using the domain tree.

To further bolster the accuracy of our simulation results, we have calculated the confidence of the TD metric. Sample means of "Q-SDP" and "V-SDP" for N=100 are 10.715 and 21.131, respectively. Our measurement finds out that TD of both the simulated protocols have a 95% confidence level with confidence intervals of (10.1251 to 11.2681) and (20.5990 to 21.8421), respectively.



(a) NM vs. N

(b) NH vs. N

(c) TD vs. N

(d) HR vs. N

Figure 5-8: General Performance of the Service Discovery Protocol

Performance of HR: Figure 5-8 (d) shows that HR rapidly decreases with respect to N for "V-SDP" compared to "Q-SDP". This is because, with a failure rate (F_R) of 20%,

total number of nodes (N) decreases, and the service density (d_s) decreases rapidly, so the HR also decreases for "V-SDP" as matching services may not be found. "Q-SDP" can guarantee higher service availability as long as a matching service exists in the network. We also calculate the confidence on the simulated values of HR and we found that the sample mean with N=100 for "Q-SDP" and "V-SDP" are 0.84 and 0.83, respectively. HR of both the simulated protocols have a 95% confidence level with confidence intervals of (0.8337 to 0.8517) and (0.8179 to 0.8398), respectively.

In general, from the above results we can realize that, NM, NH, and TD increase with N, whereas, HR decreases with N. Comparing our protocol with "V-SDP", we can conclude that "Q-SDP" consistently performs better than "V-SDP".

B. Effect of Node Mobility on Our Protocol

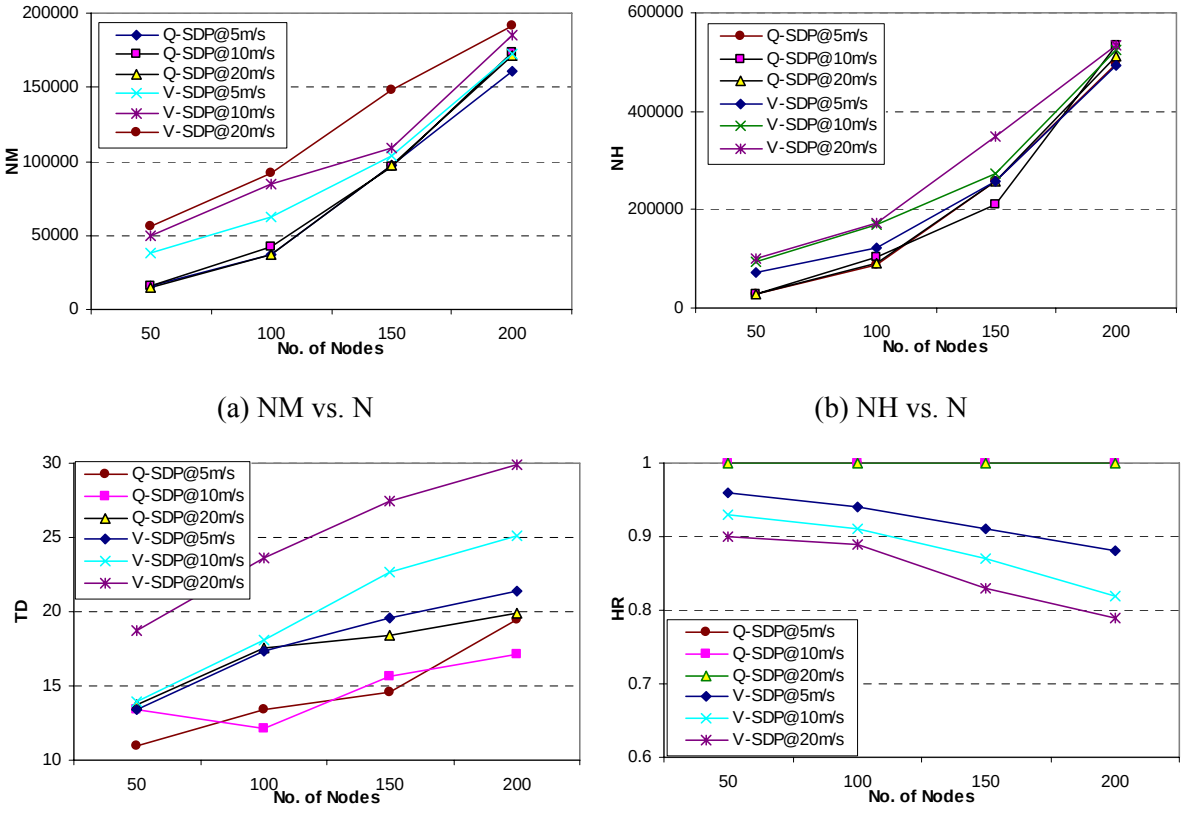
In this sub-section we study the effect of varied node speed on the performance of our protocol.

- *Effect of Node Mobility without Node Failure*

Initially we vary V_{max} from 5m/s to 20m/s without considering any node departure and plot the results in Figure 5-9 (a) to (d).

Effect of Mobility on NM: Figure 5-9 (a) shows the variation of NM with respect to N for increasing V_{max}, for both "V-SDP" and "Q-SDP". We can observe that NM quickly increases with N for both the protocols. However, "Q-SDP" incurs less message overhead than "V-SDP". One reason for this is that we disseminate messages using the domain tree structure and avoid all types of broadcasting or flooding. "V-SDP" on the other hand, broadcast volunteer advertisements which significantly increase the traffic. Also, NM increases with increase in V_{max}, though this effect is more pronounced for "V-SDP" than for "Q-SDP". Change of NM for "Q-SDP" is very

little. The only increase in NM is due to the backbone maintenance cost incurred by frequent node mobility. Since, no node failure has been considered here, the quorum maintenance cost is ignored. For "V-SDP", the NM increases rapidly with node speed. This is possibly because at high node speeds service providers move quickly through the system and users need to broadcast service requests if they do not find any service provider in the vicinity.



(a) NM vs. N

(b) NH vs. N

(c) TD vs. N

(d) HR vs. N

Figure 5-9: Effect of Node Mobility without Node Failure

Effect of Mobility on NH: Figure 5-9 (b) shows the variation of NH with respect to N for increasing V_{max}, for both "V-SDP" and "Q-SDP". We can see that NH increases with N and V_{max} for both the protocol and variation of NH for "V-SDP" is pretty close to the variation of NH for "Q-SDP". This is because, the "V-SDP"

restricts message hops using TTL values. Still, "Q-SDP" incurs slightly lower NH than "V-SDP" for different values of N and V_{max}.

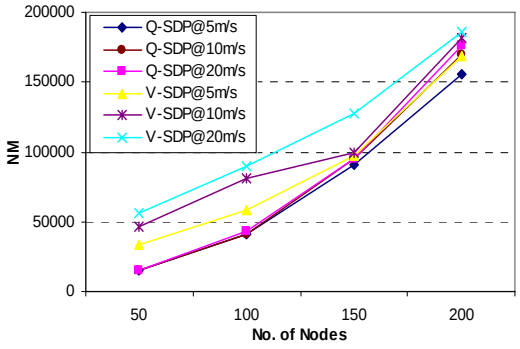
Effect of Mobility on TD: From Figure 5-9 (c) we can see the variation of TD with respect to N and V_{max} for "V-SDP" and "Q-SDP". TD for "Q-SDP" increases with. This is because, with increase in number of service requestors the average delay to reply individual query also increases. But, for "Q-SDP" with increase in N, K increases and the size of individual directory quorum also increases. This ensures higher replication of service information which holds back very fast increase of TD. "V-SDP", on the other hand, experiences fast increase in TD with increase in N as well as V_{max}. This is attributed to the fact that, with higher N, the numbers of service requestor increase, so, the TD increases. Also, at higher node speeds, the volunteer nodes move faster, rendering the service requestors deprived of volunteers. Then the service requestors need to find new volunteers before requesting a service and this goes to increase the overall TD.

Effect of Mobility on HR: Figure 5-9 (d) plots the change of HR with N and V_{max} for "V-SDP" and "Q-SDP". In absence of node failure, "Q-SDP" always maintains 100% HR for all values of N. This is achieved through the quorum intersection. The HR for V-SDP, however, decreases with increasing N and V_{max}. Thus, with many service requestors, there can be some unsuccessful service requests due to service provider or volunteer mobility. So, we can see that, if all nodes are available, "Q-SDP" can always find a service matching user request where "V-SDP" may not.

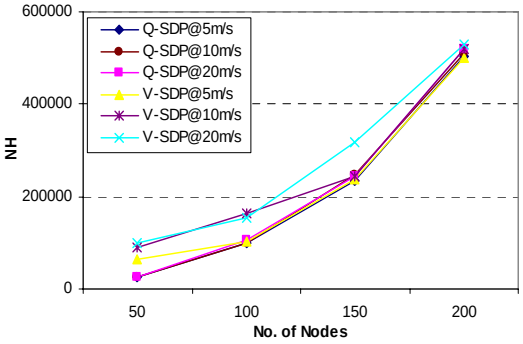
- *Effect of Node Mobility with Node Failure*

In Figure 5-10 (a) to (d), we present the performance results of "V-SDP" and "Q-SDP" by varying V_{max} from 5m/s to 20m/s and keeping the node departure rate at 20%.

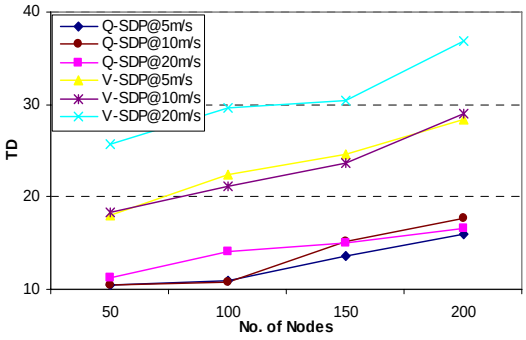
Effect of Mobility on NM: As shown in Figure 5-10 (a), NM increases for both "V-SDP" and "Q-SDP" with N and V_{max}. With 20% node failure rate, the NM increases with N as the directory nodes may fail which generates extra traffic for quorum maintenance. "V-SDP" also presents similar performance of NM. In their case, increase in NM can be explained by the fact that, at higher speeds, volunteer nodes may move away. So, service users and service providers may need to broadcast volunteer request message which increases the message cost. The change of NM is very similar to the trend observed in Figure 5-9 (a). However, NM slightly decreases in current case, due to the decrease in effective value of N caused by node failures.



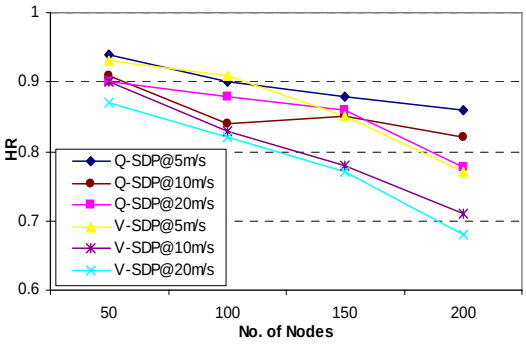
(a) NM vs. N



(b) NH vs. N



(c) TD vs. N



(d) HR vs. N

Figure 5-10: Effect of Node Mobility with Node Failure

Effect of Mobility on NH: Figure 5-10 (b) shows the variation of NH for "V-SDP" and "Q-SDP" with N and V_{max}. NH increases for both the protocols with increasing

values of N and node speeds. Though the changes are similar to those observed in Figure 5-9 (b), the current values of NH are little lower due to the lower N caused by 20% F_R.

Effect of Mobility on TD: We can observe from Figure 5-10 (c), that TD increases with N and V_{max} for both "V-SDP" and "Q-SDP". At higher node speeds "V-SDP" performs worse than "Q-SDP" because volunteer nodes can migrate away and the service requesters may not have any volunteer in range when they search for one. This increases the discovery delay (TD). Also, the TD here is higher than the one presented in Figure 5-9 (c) because, with failure of nodes, the number of service providers decrease which lowers the service density (d_s) in the network and increased TD.

Effect of Mobility on HR: Figure 5-10 (d) shows the decrease in HR for "V-SDP" and "Q-SDP" with N and V_{max}. With node departure, the service density (d_s) decreases and as a result the HR significantly decreases for "V-SDP". For "Q-SDP" the HR only decreases if there are no matching services in the network due to service provider failures.

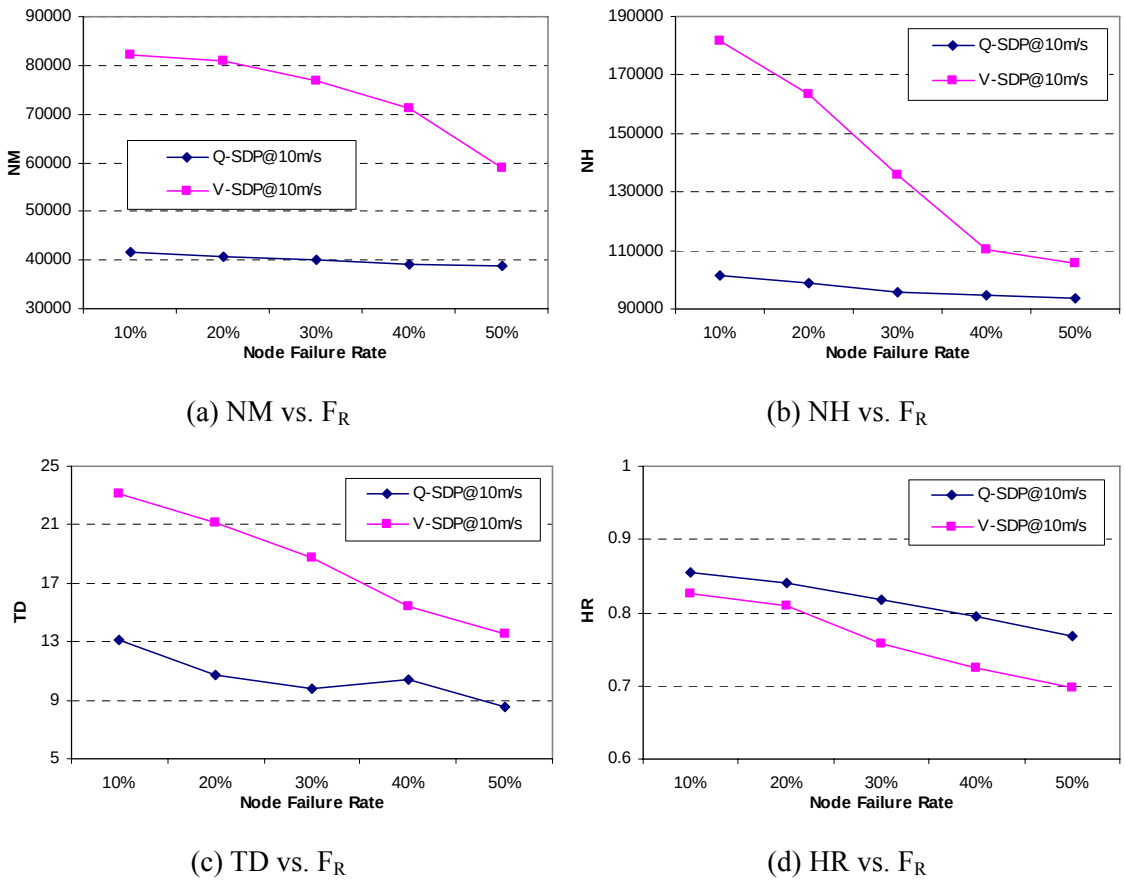
C. Effect of Node Failure on Our Protocol

In this sub-section, we report the results of varying node failure (F_R) rates on the performance of our protocol. We fail a percentage p of all the nodes (N), participating in the election process in such a way that throughout the execution time, at least as many as (p*N/100) nodes are always in the list of failed nodes. We release nodes in the head of this list if more than the expected number of nodes has been failed. The released nodes starts participating in service discovery protocol as freshly joined nodes as described in the *Case II* of Section 5.3.2 (A). Below we describe the results with varying node failure rate (F_R) from 10% to 50%. All the following experiments

are executed at default values, i.e., with 100 nodes moving at a speed of 20m/s. The results have been reported at Figure 5-11 (a)-(d).

Effect of F_R on NM and NH: We can observe from Figure 5-11 (a) and (b), that NM and NH decreases with increase in F_R. This is straightforward, because with increase in F_R, N decreases rapidly, so NM and NH also decrease for both "V-SDP" and "Q-SDP".

Effect of F_R on TD: Figure 5-11 (c) shows that TD decreases for "V-SDP" and "Q-SDP" with increasing F_R. This is because of the fact that, with high rates of node failure, number of service requesters decrease, so the average delay in service discovery (TD) decreases over the entire operational period.



(a) NM vs. F_R

(b) NH vs. F_R

(c) TD vs. F_R

(d) HR vs. F_R

Figure 5-11: Effect of Node Failure on the Service Discovery Protocol

Effect of F_R on HR: Figure 5-11 (d) shows that HR decreases for "V-SDP" and "Q-SDP" with increasing F_R. With high F_R, service density (d_s) in the system decreases rapidly, so the HR also decreases as matching services may not be found.

From the performance results and the analysis, we can see that all the performance metrics of our protocol ("Q-SDP") are significantly better than "V-SDP" under different node speeds and node failure rates. So, our protocol proves to be more robust under high node failure rates and higher node mobility.

5.4.2.2 Simulation with Variable Node Density

In this sub-section we show the effect of varying node density by keeping the network territory size constant and by increasing the total number of nodes. We initially report the general performance of "Q-SDP" and "V-SDP" at the default values of simulation parameters. Later we shall study the effect of node mobility on the performance of our simulated protocols while changing the node density continuously.

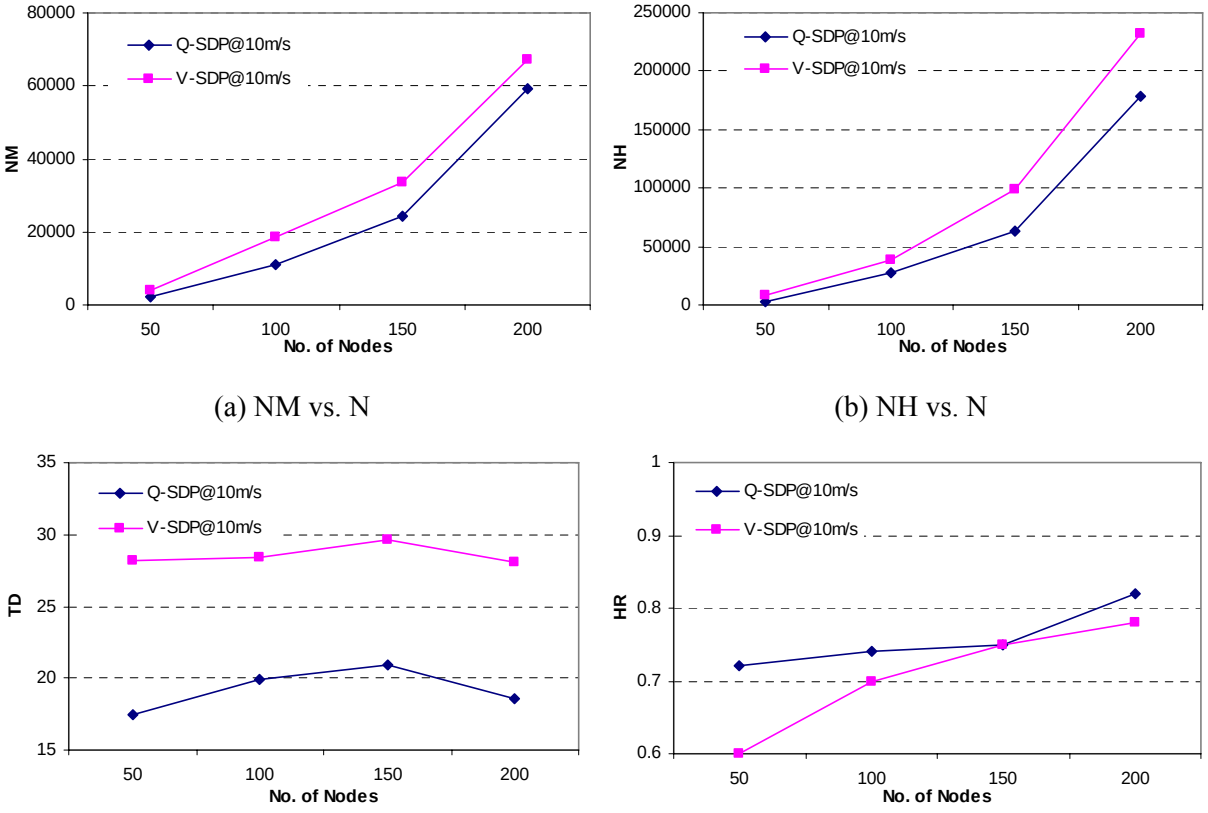
A. Performance in General Cases

Here we plot the results of our experiments in Figure 5-12 (a)-(d). The performance metrics used are same as before.

Performance of NM: From Figure 5-12 (a) we observe that NM increases with N for both the protocols and they use less messages than that depicted in the Figure 5-8 (a). This is because, in the current case, the network is heavily partitioned with low node density and hence the number of messages required for service discovery or to maintain the service discovery structures is pretty low. Also the transmission radius is just 100 m contrary to 250 m considered before. So, the numbers of neighbors for each node will also decrease. Moreover, we have considered a default node failure rate of 20% which

further reduces the total number of nodes. However, from Figure 5-12 (a) we can see that the "Q-SDP" gains over "V-SDP" in terms of NM as the network grows denser gradually.

Performance of NH: Figure 5-12 (b) shows that NH increases with N and "V-SDP" incurs higher NH than "Q-SDP". Also we observe that increase in NH with N for both the protocols are less than shown in Figure 5-8(b). All these observations and the related explanations can be obtained from that of NM as increase in NM implies increase in NH.



(a) NM vs. N

(b) NH vs. N

(c) TD vs. N

(d) HR vs. N

Figure 5-12: General Service Discovery Performance with Varied Node Density

Performance of TD: From Figure 5-12 (c) we can see TD for service discovery is quite high while compared with Figure 5-8 (c). The increase in TD can be attributed to the fact that, at low node density, service density (d_s) in each network component in a

partitioned network can be quite low. This hinders service discovery and hence, the TD increases compared to that shown in Figure 5-8 (c). Also, we can see that TD gradually increases and finally it drops little bit. This is because, with the increase in N, the service density increases and also, the network connectivity increase. Due to all these reasons, finding a service provider becomes more difficult and time consuming. We can observe from Figure 5-12 (c) that "Q-SDP" can find services quicker than "V-SDP". So, our protocol is still more efficient than "V-SDP" while operating at varying node densities.

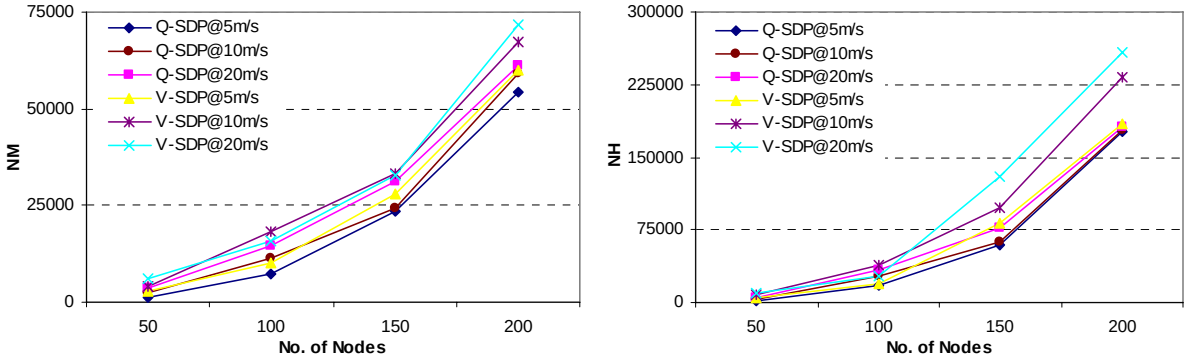
Performance of HR: Figure 5-12 (d) shows that HR is lower in sparse networks compared to that shown in Figure 5-8 (d) and it increases with N for both the simulated protocols. This is because, at very low node density, service density (d_s) in different network components is very low and hence the service discovery may not be successful. This lowers the average success rate of service discovery at lower N. The HR gradually increases with N as the network becomes gradually connected. Our protocol achieves a higher success rate for service discovery than "V-SDP" at similar network conditions. This proves the usability of a quorum-based service matching protocol.

In general, from the above results we can realize that, our protocol performs better than "V-SDP" at varied node densities and under dynamic network conditions where topologies change frequently. Next we shall discuss the effect of varying node speeds on our protocol.

B. Effect of Node Mobility

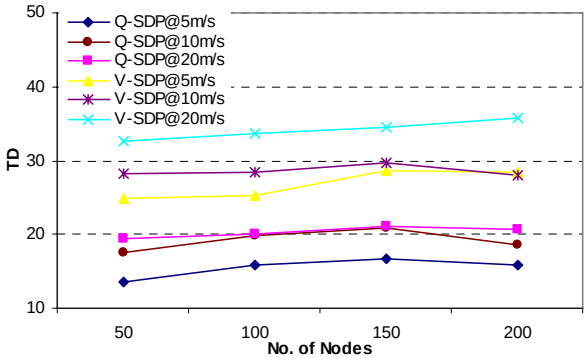
In order to test the effect of node mobility on the performance of "Q-SDP" and "V-SDP", we varied the V_{max} from 5 m/s to 20 m/s, keeping F_R at 20%. We can observe from Figures 13 (a)-(d) that, NM, NH, TD and HR values increase with V_{max} for both

the protocols. This is because, at higher node speeds, networks frequently get disconnected and it requires more messages for detecting network partitions and managing the service discovery structures on the face of partitions. This is why, NM and NH increases with increase in V_{max}. The increase in TD is also due to the highly dynamic network conditions that take place with increase in V_{max}. The frequent topological changes that occur by network partitioning and partition merges contributes in the delay of service discovery. The TD value slightly decreases at high values of N due to the increased service density and network connectivity. The same reasons are responsible for the decrease in HR with increased node speeds. The results also show that "Q-SDP" gains in performance over "V-SDP" even in the sparsely connected network environment.

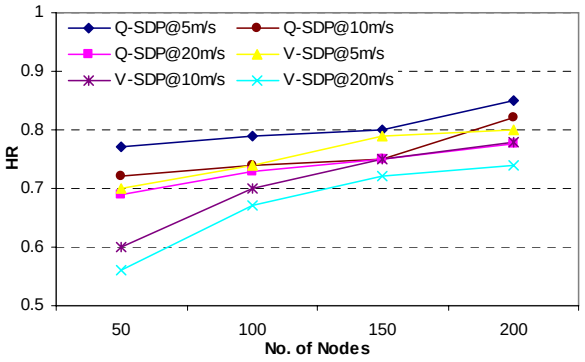


(a) NM vs. N

(b) NH vs. N



(c) TD vs. N



(d) HR vs. N

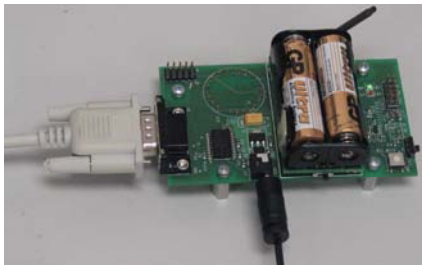
Figure 5-13: Effect of Node Mobility on Service Discovery with Varied Node Density

Thus, altogether, our simulation results prove that the proposed quorum-based service discovery protocol is message-efficient, fault-tolerant and can cope with node mobility and frequent topological changes caused by network partitions and partition merges.

5.5 Prototype Implementation

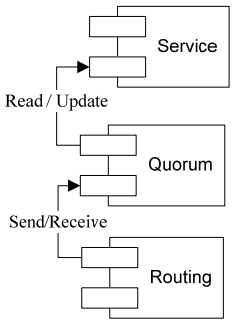
To further demonstrate the feasibility of our protocol in real applications, we implement it on a sensor network testbed. The testbed (Figure 5-14 (c)) contains 20 MicaZ [30] sensor nodes (Figure 5-14 (a)) running TinyOS [64] and a MIB600 gateway (Figure 5-14 (b)) and is distributed over a single floor of the Hong Kong PolyU Mong Man Wai building.

(a) MicaZ Node



(b) MIB600 Gateway

(c) Part of Our Testbed



(d) Implementation Overview

Figure 5-14: Prototype Implementation of Reliable Service Discovery Protocol

In order to make our system more flexible, we implement three major components as shown in Figure 5-14 (d). On the bottom, we have an end-to-end routing layer which is responsible for message delivery between nodes in the network. Above the routing layer, we have a quorum layer. It uses the send/receive interface provided by the routing layer to exchange messages between the quorum members. On the top layer, we have the service module. It uses the update and read interface provided by the quorum layer to allow the clients to register services. We assign random numbers to different sensor nodes as weights. Total number of directories is K where we choose K as $\lceil N/4 \rceil$, N being total number of nodes in the network. The nodes are considered static as it is difficult to make them mobile in a testbed. We have also implemented node failure. We arbitrarily switch off some sensor nodes during service discovery and consider them failed. We considered maximum node failure rate of 10%.

The sensors nodes are equipped with light, temperature and sound sensors and can provide different kinds of services. The client can request service from the nearest directory node. In addition to the testbed system itself, we also have a special node employed to monitor the performance of the system. Initially, when the execution starts, each node undergoes a bootstrapping phase to initialize the variables and chooses a random weight value. After that, the monitoring node broadcasts the topology and directory information to all the nodes in the network. When a node receives the topology, it populates its neighbor list and finds out the nearest directory. Then a node starts service discovery at randomly chosen time. When the execution finishes, the monitoring node sends the performance results to the computer.

We measure the performance of our protocol using two metrics – average service response time and average number of service discovery messages transmitted in the network. We test the system with 10 and 20 nodes. We generate random topologies. The

experiment is run for 10 times each with different topologies for a fixed number of nodes, and the average values are reported in Table 5-4.

Table 5-4: Implementation Results for Reliable Service Discovery Protocol

Performance Metrics	Values (N=10)	Values (N=20)
Average hit ratio	*100%*	*96.5%*
Average service response time (ms)	*12*	*37*
Average number of messages sent for each sensor node	*1*	*1*

The results show that the success rate of service discovery is very high and the discovery time is low. The results are very similar to that obtained by simulation. Since the average number of service discovery messages sent by each node is 1 at most, we can logically conclude that our protocol can minimize the energy spent by sensor nodes.

5.6 Summary

In this chapter, we address the problem of reliable service discovery in pervasive environments by tolerating directory failures. Directory failures during an ongoing service discovery and access operation in a MANET are difficult to handle as it implies that all services registered with a failed directory must re-register them with other available directories in order to publish themselves. The chances of directory failure are abundant in a highly dynamic network, like the pervasive computing system.

To tackle the directory failure problem we have proposed a fault tolerant service discovery protocol for mobile ad hoc networks using our directory community framework. Initially, nodes with top K highest resources are elected as directories and form a directory community. The nodes in the directory community are naturally more reliable than other resource constrained nodes. The elected directory nodes then form quorums and replicate services among the quorum members in order to enhance service

availability despite directory failures. Quorum-based approach has been adopted in order to achieve fault-tolerance with limited replication. Quorum intersection property is used for network-wide service availability with low message overhead. To further reduce message cost, the network is divided into several tree-structured domains. To tolerate the failure of directory nodes, an incremental directory election approach has been adopted. Our protocol can also cope with arbitrary topological changes caused by network partitions and partition merges. Besides extensive simulations we also have implemented our protocol on a testbed of wireless sensor nodes. The simulation and experiment results show that our protocol increases service availability and system robustness even with a high rate of node failure.

Chapter 6

Service Handoff Based Seamless Service Access

This chapter describes our proposed mechanism for seamless service access by mobile users in infrastructure-less pervasive environments using a service handoff approach. We make use of our directory community framework to develop the service handoff protocols. Section 6.1 presents a brief overview of our motivation and the issues that are required to be addressed for efficient and successful service handoff. Section 6.2 gives some background information about the previous endeavors to realize service handoff in pervasive computing environments. From Sections 6.3 to 6.5 we describe our service handoff protocols with detailed discussion of the handoff and load balancing techniques along with the performance results. Finally, Section 6.6 concludes the chapter by summarizing our contributions.

6.1 Overview

We have already introduced in Chapter 3, the general design principles and challenging issues of designing a seamless service access mechanism for mobile users in pervasive computing environments. We have also proposed our own service handoff

policy to achieve the same objective. Existing service discovery solutions in pervasive computing [50][42][95][82][21][103][76][104][32][60][94][85][57] rarely address the issues associated with continuous and seamless service access. One prominent work [80] in maintaining service execution continuity in ad hoc networks uses a migratory service [81] based model. Migratory services are built using Smart Messages [54], and can migrate like mobile agents from one device to another, if the device currently hosting the service becomes unsuitable to host it any longer. Though this approach is useful, there are plenty of service types which are not suitable for migration as a mobile agent.

Service handoff is an alternative approach to keep service access ongoing for a mobile user, when it gets disconnected with the service provider. Service handoff refers to the process by which new matching service providers are automatically selected for a user once the original service provider becomes unavailable. Basically an experience of seamless connection with a single service provider is simulated for the user. The objective of service handoff is to maintain service consistency by handing over the service execution states from one provider to another. Service handoff is, however, different from traditional handoff operations for mobility management in wireless networks. Handoff for mobility management focuses on the quality of network connection and initiates handoff based on signal strength or other related metrics. Their primary objective is to reduce the handoff delay.

Service handoff operation has two basic steps - initiating handoff and selecting handoff destination. While the former decides when to start handoff the latter chooses the provider to which the service should be handed off.

Initiating service handoff is challenging in a pervasive environment because pervasive devices are connected in an ad hoc manner and can move freely at will. In

such a dynamic environment, it is hard to detect the disconnection between a user and a service provider in a timely manner and to trigger the handoff process.

Selecting handoff destination also has some key challenges. Firstly, due to device resource constrains, heavily loaded service providers are often prone to failure by quick resource depletion. To avoid this scenario and to maximize the lifetime of a service discovery system, load balancing among service providers is absolutely necessary. Secondly, pervasive devices are fault-prone. So, failure of service providers will result in many other handoffs. Thus, carefully choosing a service provider will surely reduce handoff frequency and related costs. Finally, proper selection of handoff destination can control the handoff delay.

In this research, we propose a service discovery solution which especially focuses on providing seamless service access to mobile users using the service handoff approach. To the best of our knowledge, service handoff for seamless service access has not been investigated earlier. In order to realize the objective of ensuring reliable and continuous service access for mobile users, we make use of our directory community framework. We assume that the entire network is divided into multiple service provisioning domains with each domain containing many service providers whose scope is limited by the domain boundaries. Each domain also contains a set of top-K weighted directory nodes. We have proposed three different service handoff protocols depending on the action performed by the initiating node. The proposed protocols carefully selects handoff destination in order to achieve a load balance among the service providers. The directory community based service discovery and handoff policy appears to be fairly scalable, robust and fault-tolerant. Our performance evaluation shows that our service handoff protocols can achieve good load balance among the service providers. They are highly scalable and can cope with frequent node mobility and node failures.

6.2 Background

Service continuity has not been investigated widely in the perspective of mobile and pervasive computing. There are some works on service migration [48][91] in pervasive computing, which studies the issues and challenges of migrating an ongoing service from one device to another to suit the conditions of continuous service execution. In [48], a centralized model for context-aware service migration has been proposed, and [91] considers a transport layer overlay to assist users to seamlessly migrate through heterogeneous networking environments. Both these works treat 'service' similar as user applications which can run on any suitable device. On the contrary, service discovery applications consider dedicated service providers which only can provide particular services hosted by them.

Service continuity in ad hoc network has been studied in [43] and [80]. The idea of "follow-me" services has been proposed in [43]. Services can migrate from node to node, following a mobile user, in order to maintain seamless interaction with the client application. Migration is triggered by disconnections (between client applications and services) which are assumed to be predictable and so, cannot cope with unpredictable failures. To address these shortcomings a migratory service based reliability approach has been proposed in [80]. In that, services periodically take check points and store it in a backup node. In case the original service fails, the backup takes over. Services are modeled as mobile agents and can move from node to node to provide fault tolerance against predictable failures of hosting nodes. Their protocol however, does not support fault tolerance in case of unpredictable device failures or network disconnection.

As already mentioned in Chapter 2, existing SDPs for pervasive and ad hoc mobile environment use either a directory-less [103][76][104][32] or a directory based [60][94][85][57] architecture. Directory-less SDPs are costly because, they often resort

to expensive message broadcast. On the other hand, directory-based SDPs are robust, but they incur high maintenance overhead. The general approach followed by directory-based SDPs is to develop a virtual backbone and to disseminate service advertisements and discovery requests using the backbone. Protocols mentioned in [60][94][85] all uses the backbone based approach.

Kim et al. [57], however, proposed a volunteer node based SDP for mobile ad hoc network where volunteers are relatively stable and resource rich nodes and form an overlay structure. This approach suffers from certain limitations. Firstly, the overlay may develop loops and cycles, which may increase service discovery cost. Secondly, the volunteer advertisement broadcast can significantly increase the traffic. Despite its limitations, this approach shares some of our design considerations. The volunteers, in fact, function as our directory nodes and the resource-based volunteer selection is also similar to our resource-based K-directory election approach (described in Section 6.3.2). So, we choose to compare our protocols with the volunteer node based SDP. But this SDP does not have any service handoff feature which is our key contribution and we can specifically illustrate how service handoff benefits seamless service access in mobile environment.

6.3 System Model and Preliminaries

In this section we introduce our system model and the assumptions and describe a preliminary service discovery protocol used by service users.

6.3.1 System Model

We have already mentioned that our service handoff protocols work using the directory community. The generic system model of the directory community framework

has already been described in Section 3.1. Here we state some specific assumptions. We know that the entire network is divided into multiple service domains (Figure 5-1) during the directory community creation (refer to Section 5.2.1). We assume that service providers restrict their service access scopes by the domain boundaries. Nodes can move freely across domain boundaries. So, when a user moves from one domain to the next, they require a different service provider. Nodes belonging to neighboring domains can communicate only through a gateway node in order to exchange handoff related messages. Each node is a router and any two nodes within a domain are neighbors if they are within each others signal coverage range. Also any two nodes within a domain can be connected using multiple hops. We assume that there are multiple service types in the environment and each domain has one or more instances of each type of service, without which service handoff cannot be successful.

6.3.2 Basic Service Discovery Protocol

The service handoff approach ensures service access continuity for mobile users. So, users must discover required services and start using them before the need arises for a service handoff. We use a very simple directory based service discovery protocol that can be implemented over the directory community. After the directory election is complete, the elector sends identifier of the domain directories to each node in the domain. All the service provider nodes then find out the nearest directory in their domain and register their services with the directory.

A service provider ($\mathcal{P}$) registers its service ($\mathcal{S}$) with directory ($\mathcal{D}$) using only the functional information (service type, name, etc). Detailed service context (Figure 5-5) is sent to the user on request. Directory nodes enlist the registered services with respect to types and share the list with the elector. The service registration is updated using a lease-based approach. Each registration is associated with a lease. The service provider $\mathcal{P}$

needs to renew its lease with $\mathcal{D}$ before timeout, else $\mathcal{D}$ removes the service. During periodic lease renewal, $\mathcal{P}$ also sends the checkpoint of service execution states for all the users it is currently serving. The stored checkpoints are used to resume service with a new provider when a service handoff takes place. Thus our directory-based approach can ensure fault-tolerance against sudden failure of service providers. If $\mathcal{D}$ fails, $\mathcal{P}$ registers with a new $\mathcal{D}$, either from its cache, or sends a directory request to the elector, if the cache is empty.

When a user (U) looks for a service $\mathcal{S}$, he sends a discovery request to its nearest directory ($\mathcal{D}$). On reception of a service request, $\mathcal{D}$ replies the requestor with a matching service. If no matching service is available with $\mathcal{D}$, it forwards the request to other directories.

6.4 The Proposed Service Handoff Protocols

Service handoff is triggered if a disconnection takes place between the service provider and the user. Depending on the information used and action taken to initiate the handoff, we have proposed three different handoff protocols – service provider initiated, user terminal initiated and hybrid. Before describing the protocols in detail, we list the notations in Table 6-1 and types of exchanged messages in Table 6-2

Table 6-1: Data Structures for Service Handoff Protocols

Symbols	Meaning
ε	elector node in a domain
P	A service provider
$\mathcal{S}$	A service type
U	A service user
$\mathcal{D}$	Directory node with which P registers $\mathcal{S}$ and from which U looks up $\mathcal{S}$
ϕ	Service execution state of a user

Table 6-2: Message Types for Service Handoff Protocols

Message	Purpose
DISCONN(P,U)	P (or U) sends to $\mathcal{D}$ after detecting a disconnection with U (or P)
PAGE(U,ϕ)	ε sends to another ε to search for U
SEARCH(U)	ε searches for U in its domain
ACK(U)	ε acknowledges the requesting elector about successful paging of U
NACK(U)	ε informs the requesting elector about unsuccessful paging of U
ElectorREQ	U sends to neighbor nodes querying for ε
ElectorREP(ε)	Nodes receiving a *ElectorREQ* sends the identifier of ε to the requestor
JoinREQ(U,ε,$\mathcal{D}$)	U sends to the current ε requesting a handoff from its previous ε and $\mathcal{D}$
StateQRY(U,ε,$\mathcal{D}$)	ε requests the state(ϕ) of a newly migrated U from the U's previous ε
StateSEND(U,ϕ)	ε sends the state(ϕ) of a migrated U to its current ε

6.4.1 Service Provider Initiated Handoff Protocol

When a service provider P gets disconnected with a user it is serving, it requests the directory node to search the user. The directory node probes the user to check for its availability. If the probing remains unsuccessful, then the directory contacts the elector and requests to start a handoff operation. The elector sends a PAGE message containing the user's id and its service execution state to the electors of adjacent domains.

Any elector receiving a PAGE message searches for the migrated user in its domain. Paging incurs high message and time costs which evidently increases with increasing node density. Again, with low node density paging may not be successful at the first try due to possible network disconnections. So, electors repeat paging when faced with network disconnections until a mobile node is found by an elector which then notifies others to stop searching.

If any elector finds the user in its domain it will choose a matching service for the user and will reinstate the service execution states, so that in the new provider, the

service starts executing from the point where it stopped in the previous provider. If the service states are large, it is inadvisable to send the states during paging. In that case, the original elector may choose to send the states after a neighboring elector finds the user in its domain. After the completion of a successful handoff, the new elector sends an acknowledgement to the old elector which instructs the old directory and service provider to delete the information of the migrated user.

```
/*********************************************************/
// When a P detects disconnection with a U
(1) P sends DISCONN(P,U) to D;
(2) if D receives DISCONN(P,U) from P {
        send probe to U;
        if no-reply to probe
            send DISCONN(P,U) to ε;
    }
(3) if ε receives DISCONN(P,U) from D
        handoff detection( );
/*********************************************************/
handoff detection( )
{
    ε sends PAGE(U,ϕ) to each ε' of adjacent domains;
    while node ε' receives PAGE from ε {
    ε' sends SEARCH(U) to look for U in its domain;
    if U NOT found in domain of ε'
        ε' sends NACK(U) to ε;
    else {
        ε' selects suitable service for U and resumes the service execution
        after reloading ϕ;
        ε' sends ACK(U) to ε;
    }
    while ε receives NACK from ε' {
        ε waits to receive response from all ε' to which PAGE was sent;
        if all ε' sends NACK,
            ε decides that U has crashed;
    }
    while ε receives ACK from ε'
        ε informs D to delete the information of U;
}
```

Figure 6-1: Pseudo-code for Service Provider Initiated Handoff Protocol

New service provider selection for handoff is done by the elector in such a way that the newly arrived request goes to the least loaded service provider. We consider that each user request has unit load value. So, if a service provider is serving 30 users, then the load on the provider is 30 units. Service providers can pre-set some load threshold while registering with an elector. In this way, the elector will not assign any more users to a provider which is already operating at threshold.

If a service domain covers small area, users entering that domain may have already left before a service handoff from its previous domain has been completed. In this case, the elector of the current domain will start a new handoff process until a successful service handoff is achieved.

6.4.2 User Terminal Initiated Handoff Protocol

When a user terminal experiences a disconnection with the service provider, it tries to contact the lookup directory. If the directory can be reached, it selects an alternate service provider for the user in case the current one is unavailable. This process is called intra-domain handoff. Directory nodes can detect failure of a service provider by the lease renewal timeout. In this case, a directory node proactively finds new service provider for all the users associated with the failed provider and resumes service execution from the checkpoints stored by the previous provider.

```
/*************************************************************/
//When a user terminal MT detects disconnection with P
(1) MT sends DISCONN(P,U) to D;
(2) if no-reply from D {      //also disconnected from D
        send DISCONN(P,U) to ε;
        if no-reply {
            // check current domain
            if(CheckForDomain( ) = TRUE);
                InitiateHandoff ( );      // start service handoff
             else
```

```
                find new ε;       //the elector may have changed
        }
    else { //connection with D persists
        D selects alternative provider P' for U and resumes the service execution
        after reloading ϕ;
    }
/*************************************************************/
bool CheckForDomain( ) {
  broadcast ElectorREQ to 1-hop neighbors;
  if node j receives a ElectorREQ
        send ElectorREP(ε(j));
  when node i receives ElectorREP from j
        if(ε (j) ≠ ε (i))  // a domain change has occurred
              return TRUE;
}
/*************************************************************/
InitiateHandoff( )
{
    send JoinREQ(U, ε', D) to ε;
    //ε' is previous elector and D is previous directory
    while node ε receives JoinREQ from U
        ε sends StateQRY(U, ε', D) to ε';
    while node ε' receives StateQRY from ε {
         retrieve service execution state (ϕ) of U from D;
        ε' sends StateSEND(U,ϕ) to ε;
    }
    while node ε receives StateSEND from ε'{
        ε selects suitable service for U and resumes the service execution after
        reloading ϕ;
    }
}
```

Figure 6-2: Pseudo-code for User Terminal Initiated Handoff Protocol

But if the user terminal cannot access the lookup directory, it checks for its current location (domain). This can be done by asking neighboring nodes about their domain elector. If the user terminal finds out a change in the domain, it sends a join request to the new elector containing its id, previous domain's elector id and previous lookup directory id. The new elector then queries the old elector about the user which in turn sends back the user's ongoing service execution states and removes the information

from local directory. The new elector then chooses a matching service provider for the user following the afore-mentioned load balance policy and reinstates the service execution states.

6.4.3 Hybrid Handoff Protocol

The paging process in service provider initiated handoff mechanism creates a burden on the network by consuming plenty of precious bandwidth. This process also wastes much of node resources. The user terminal initiated handoff, though significantly less costly, demands higher complexity of the mobile terminal. To strike a balance between these two handoff mechanisms, we propose a hybrid handoff mechanism which combines the previous two approaches for a better use of resources.

```
/*********************************************************/
// when a MT detects a domain change
(1) MT send JoinREQ(U, ε', Ɗ) to ε;
//ε' is previous elector and Ɗ is previous directory
(2) while node ε receives JoinREQ from U
      MigratedUserList ← (U, ε', Ɗ);
/*********************************************************/
//when Ɗ informs ε' about the disconnection of U from P
(3) ε' send PAGE(U,ϕ) to each ε' of adjacent domains;
(4) while ε receives PAGE from ε' {
      if U belongs to MigratedUserList {
          ε selects suitable service for U and resumes the service execution
          after reloading ϕ;
          ε sends ACK(U) to ε';
      }
      else {
          ε waits till time T to check if it receives a JoinREQ from U;
          if no JoinREQ from U comes until time > T
              ε sends NACK(U) to ε';
      }
   }
```

Figure 6-3: Pseudo-code for Hybrid Handoff Protocol

Upon a disconnection between the service provider and user, the user terminal detects its current domain, while the service provider asks the elector to page the neighboring domains. When the user joins a new domain, it sends a join request to the elector of the new domain informing its id, previous domain's elector and directory ids. The new elector then waits to receive a paging message from the old elector of the user. When the paging message is available, the new elector finds a matching service for the user following the load balance policy, and hands over the service execution states to it, while sending an acknowledgement for successful handoff to the previous elector. If a domain elector receives the paging message, but the user has not moved in to that domain, then the elector waits for a certain time expecting the possible arrival of the user. After a time limit the elector sends a NACK message to the elector which was paging for the user.

6.5 Performance Evaluation

We have carried out extensive simulations to evaluate the performance of our proposed protocols. In absence of any other service handoff protocols, we first compare the load balance and service discovery performance of our protocols with the protocol proposed by Kim et al. [57]. After that, we analyze in-depth the performance of our proposed handoff protocols. To obtain the best performance of the protocol presented in [57], the TTL values for the volunteer advertisement (TTL_{max_a}) and the service request (TTL_r) messages are fixed at 3 and 2, respectively. Moreover, every client belongs to a maximum of 2 volunteer regions and each volunteer advertises themselves once every simulation minute. All simulation parameters have been listed in Table 6-3.

6.5.1 Simulation Setup and Metrics

The network nodes are randomly scattered in a square territory. The total number of nodes is varied to examine the effect of system scale on the performance. In order to test the load balancing performance, the total number of nodes has been increased while keeping the territory size constant (thereby increasing the node density). The territory is divided into 3x3 square grids which are considered as service domains. For message routing, we have implemented a simple protocol based on the "least hops" policy, which is adopted in many classical routing protocols in ad hoc networks. A routing table is proactively maintained at each node. We consider 200 nodes moving at 20m/s with a 20% node failure rate as default values of simulation parameters.

Table 6-3: Simulation Parameters for Service Handoff Protocols

Parameters	**Values**		
Number of nodes, (N)	100	200	300
Territory scale (m^2)	1500		
Number of service types (n_s)	5		
Number of service domains	9		
K/n	25%		
Mean link delay (ms)	5		
Max link delay (ms)	100		
Transmission radius (m)	100		
Routing-protocol	Least hops		
Node failure rate (F_R) (in %)	10, 20, 30, 40, 50		
Mobility model	Random Waypoint		
Node speed V (in m/s)	10, 20, 30		
Pause time (ms)	10		

The directory election is carried out prior to service discovery and handoff operations. Later, directory nodes are incrementally added to replace migrated directories. The weight values of the nodes are assigned randomly. The load threshold of a service

provider is chosen as one-third of its weight and each user request is assigned random load value.

We assume that, every node can provide certain type of service. A service type is assigned to each node in a round robin fashion. If there are 5 service types in a network with 100 nodes, 20 nodes provide the same service. Therefore, with high n_s, service density ($d_s = N/n_s$ average number of service providers providing same type of service) is low and vice versa.

We simulate node failure by stopping a percentage, p, of all the nodes (N), participating in the protocol in such a way that throughout the execution time, at least as many as (p*N/100) nodes are always in the list of failed nodes. We release nodes in the head of this list if more than the expected number of nodes has been failed. The released nodes starts participating in service discovery and access as freshly joined nodes.

In the simulations, we measure the service discovery and load balancing performance using the following metrics:

NM (Number of Messages): The total number of messages exchanged for service discovery. This includes the messages required for directory election, backbone maintenance and service handoff. Here, a "message" refers to an "end-to-end" message, i.e. a message from the source to the destination node. Such a message may be forwarded by several intermediate nodes in the network level.

NH (Number of Hops): The total number of hops of the messages exchanged to achieve the global decision. One "hop" means one network layer message, i.e. a point-to-point message. Compared with NM, NH can reflect the message cost of an algorithm more precisely.

AD (Access Delay): For new service requests, AD is the average delay between the time any successful request is sent from a user and the time corresponding reply is received by the same user. AD is measured in milliseconds.

For continuous service access operations across multiple domains, we assume that the service provider in the user's current domain periodically (the period is specified by the user in the service request) sends service results to the user. In this case, AD is measured by the message delay from the service provider to the user.

SL (Normalized System Load): This is the number of service requests of a particular type generated within a domain divided by the total number of service requests of the same type generated in the entire network.

The following metrics are used specially to evaluate the cost and performance of our proposed service handoff protocols:

NM/HO (Number of Messages per Handoff): This indicates the average number of "end-to-end" messages exchanged to realize a single handoff operation.

NH/HO (Number of Hops per Handoff): This indicates the average number of hops to realize a single handoff operation.

HD (Handoff Delay): Average delay between the time a handoff is triggered for a user and the time the handoff is successfully completed for the same user. HD is measured in milliseconds.

HSR (Handoff Success Rate): The ratio of the total number of successful handoff operations to the total number of handoff operations triggered.

In the next section we shall describe all our performance results in detail.

6.5.2 Simulation Results and Analysis

Below we present our simulation results with analysis. We have simulated our three service handoff protocols and the Volunteer node based SDP presented in [57]. We have labeled the user terminal initiated, service provider initiated and hybrid handoff protocols as "UserInit", "ProvInit", and "Hybrid", respectively. The volunteer node based SDP has been labeled as "V-SDP".

We run each simulation for 20 simulation minutes and each point is obtained by averaging over 10 different runs. We first report the load balancing performance in Section 6.5.2.1, followed by the service discovery performance in Section 6.5.2.2, and then in Section 6.5.2.3 we provide a detailed analysis of the service handoff performances of our proposed protocols.

6.5.2.1 Performance of Load Balancing

Since, our proposed service handoff protocols assign new service requests or service handoff requests to the service provider with highest load capacity, we can achieve load balance on the service providers. This is a very crucial property given the resource-constrained nature of the MANET nodes.

In the network, the electors in each domain count the number of users of a particular type of service in its domain for the entire simulation period. Nine numerical values corresponding to nine discovery domains are collected for each type of service. These values are normalized by the total number of users requesting that particular service type in the entire network over the entire simulation period. We plot the average value of normalized load for 5 service types. The same process to calculate the domain load is also adopted in "V-SDP".

We evaluate the load balance performance of all our handoff protocols which show very similar performance due to the singular load balance policy adopted. We plot the average value of the normalized load in all the service domains for all three handoff protocols. We compare the load balance in our protocols with that of the "V-SDP", for different node speeds, and plot the results in Figure 6-4.

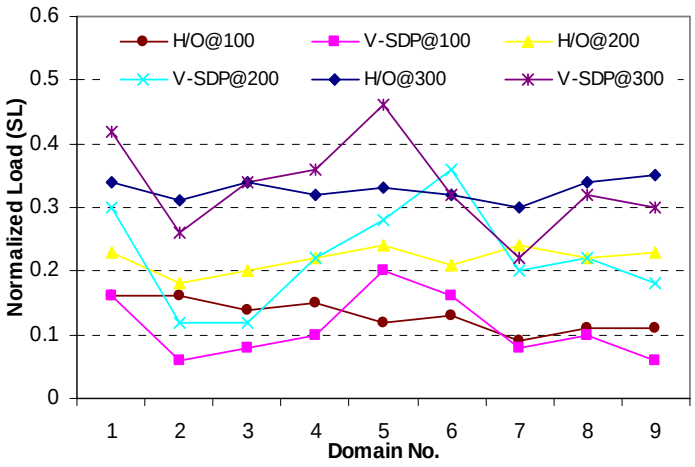


Figure 6-4: Performance of Load Balance on Service Providers

The results show that the load in different service domains is almost evenly distributed using our service handoff protocols. The "V-SDP", on the other hand, does not achieve proper load balance. One reason for this is, in our protocol, service requests are assigned to service providers by the directory nodes, so, the assignments are more or less uniform across different service providers. But, for "V-SDP", service providers themselves decide whether they can accept a new service request or not depending on their capacities. So, the load can widely vary from provider to provider. Since, in "V-SDP" terminally loaded service providers can refuse a service request which then must be reassigned to a different and less loaded service provider, the service discovery delay is higher in this case. Our protocol, on the other hand, can achieve lower discovery delay at the cost of maintaining the current load information of different service providers at the directory nodes.

6.5.2.2 Performance of Service Discovery and Access

In this sub-section, we compare the performances of "UserInit", "ProvInit", and "Hybrid" with respect to "V-SDP", first under default values of simulation parameters and then by varying node speeds and node failure rates.

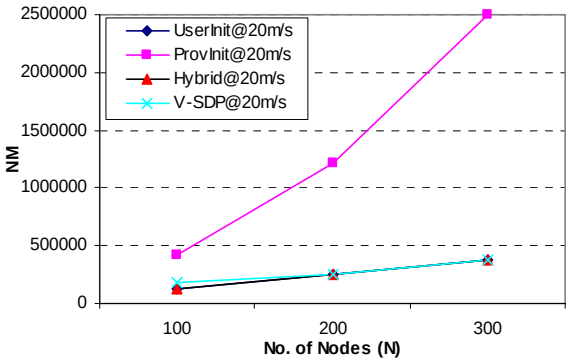
A. Performance in General Cases

We plot the results of service discovery experiments in Figure 6-5 which are carried out using default values of simulation parameters.

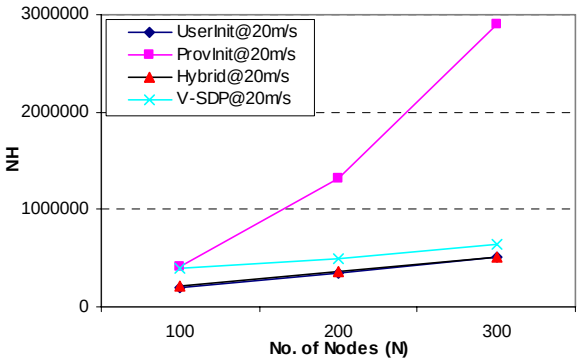
Performance of NM: From Figure 6-5 (a) we observe that NM increases with N for all the protocols and "ProvInit" incurs the highest message overhead. This is because, "ProvInit" uses message flooding to page a migrated user. However, "UserInit" and "Hybrid" use less number of messages than "V-SDP" as evident from Figure 6-5 (c). This is attributed to the fact that the handoff protocols, though incur extra overhead for handoff, can significantly reduce the recurrent service discovery cost. But, "V-SDP" broadcasts volunteer advertisements and requires recurrent service discovery when domain change occurs for a mobile user. In presence of node failure, service providers and volunteer nodes may fail and the rate of broadcast significantly increases to discover alternate service provider, thereby increasing the NM.

Performance of NH: Figure 6-5 (b) shows that NH increases with N for all the simulated protocols. This observation follows directly from Figure 6-5 (a) as increase in NM implies increase in NH. Similar observation can be obtained from Figure 6-5 (d) which shows that "V-SDP" incurs higher NH than "UserInit" and "Hybrid". The increase in NH for "V-SDP" is because of the absence of any handoff functionalities. Users move to new domains and try to connect to old service providers and directory

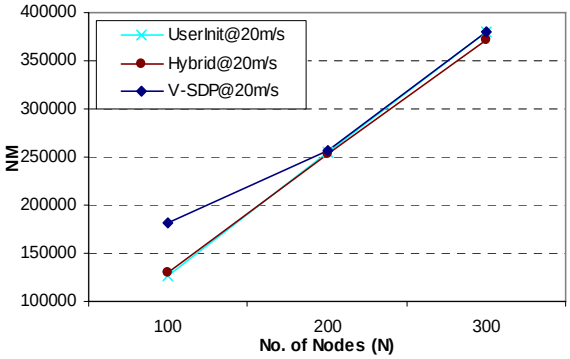
nodes. While unable to connect with previous service providers, they request for new services. This approach ultimately increases the message hops.



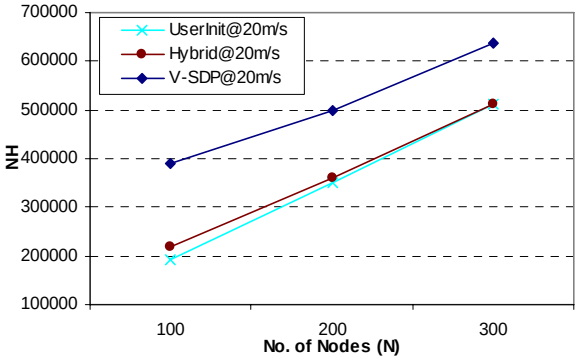
(a) NM vs. N (V = 20m/s)



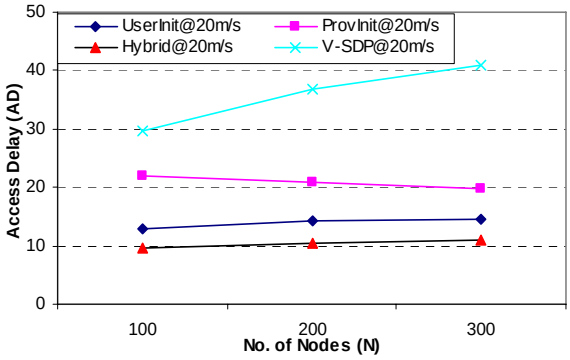
(b) NH vs. N (V = 20m/s)



(c) NM vs. N (“UserInit”, “Hybrid” “V-SDP”)



(d) NH vs. N (“UserInit”, “Hybrid”,“V-SDP”)



(e) AD vs. N (V = 20m/s)

Figure 6-5: General Performance of Service Discovery Operations

Performance of AD: Figure 6-5 (e) shows that AD also increases with N for all the simulated protocols. "V-SDP" shows highest AD as the users have to discover service every time they move to a new domain. Also, as already mentioned, the load balancing strategy of "V-SDP" has significant impact in increasing the AD. Among the handoff protocols, "ProvInit" has highest delay as the handoff delay is higher for this protocol (explained in Section 6.5.2.3).

From the results we can see that, with the exception of "ProvInit", which has the highest message overhead, our service handoff protocols, in general, performs better than "V-SDP" during service discovery and access operations. Thus we can claim that service handoff is an essential operation that can enable mobile users to have better service discovery and access in a dynamic environment.

B. Effect of Node Mobility

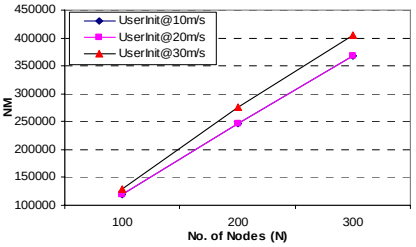
In this sub-section we study the effect of varied node speeds on the performance of our simulated protocols. Figure 6-6 shows the performance of NM, NH and AD with N by varying V from 10m/s to 30m/s and keeping F_R at 20%.

Effect of Mobility on NM: From Figure 6-6 (a1)-(a4) we can see the increase in NM with N is higher at increased node speed. This is because, with higher V, nodes frequently change domains requiring more handoff and discovery operations which results in higher NM. Also we can see that "V-SDP" has very slight difference in NM compared to "UserInit" and "Hybrid". So, considering NM, "V-SDP" is not really very bad than our protocols, but our protocols improve significantly over "V-SDP" with respect to NH, as described in the following point.

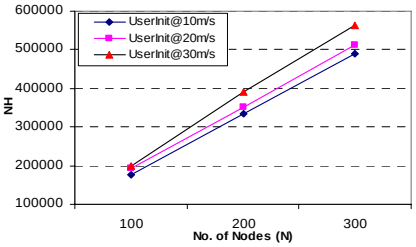
Effect of Mobility on NH: From Figure 6-6 (b1)-(b4) the same can be said about the faster increase of NH at higher V. Also, we can see that "V-SDP" incurs higher NH than

"UserInit" and "Hybrid" at higher V. The reason for higher NH for "V-SDP" has already been explained before. With increasing node speeds, service users move more frequently and which require more service requests to be generated.

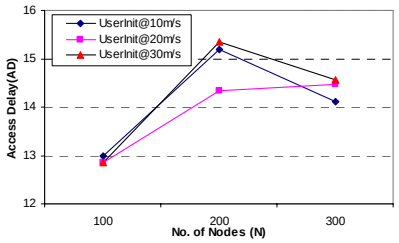
Effect of Mobility on AD: From Figure 6-6 (c1)-(c3) we can also observe that AD increases slightly with increase in V. This is because, with higher V, nodes frequently change domains requiring more handoff and discovery operations which results in higher AD. From Figure 6-6 (c1) and (c3) we can see that the AD slightly decreases at very high node density. This is because, at high node density, it is easy to find a service provider as there will be more service providers in a domain. From Figure 6-6 (c4) we notice that AD firstly increases with V for "V-SDP" but decreases at very high node speeds. This is because, at very high node speeds, the disconnected nodes often get re-connected before the check for disconnection is complete. We can also see that "V-SDP" incurs higher AD than our service handoff protocols.



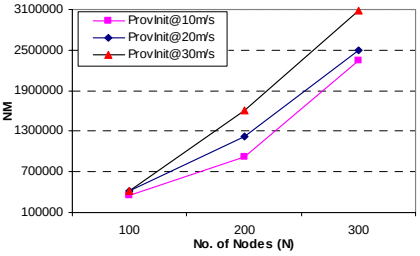
(a1) NM vs. N (V=10, 20,30m/s)



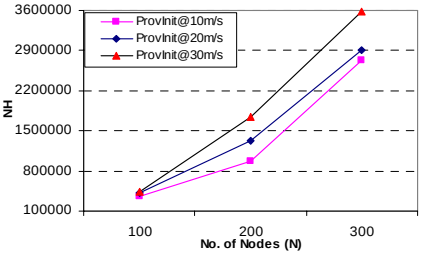
(b1) NH vs. N (V=10, 20, 30m/s)



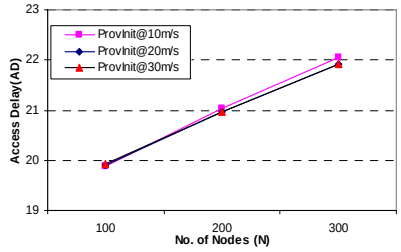
(c1) AD vs. N (V=10, 20, 30 m/s)



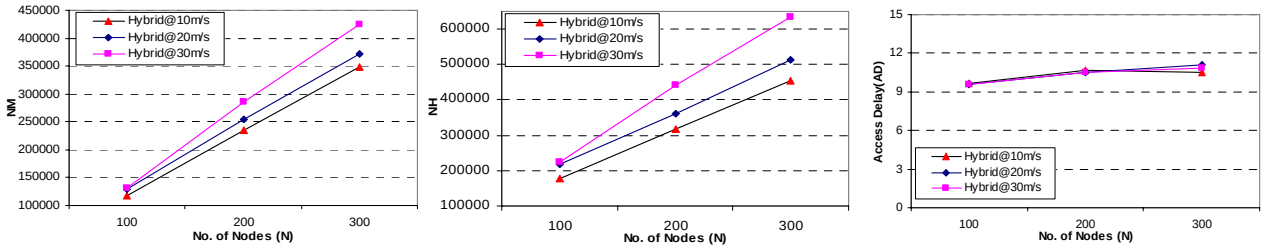
(a2) NM vs. N (V=10, 20,30m/s)



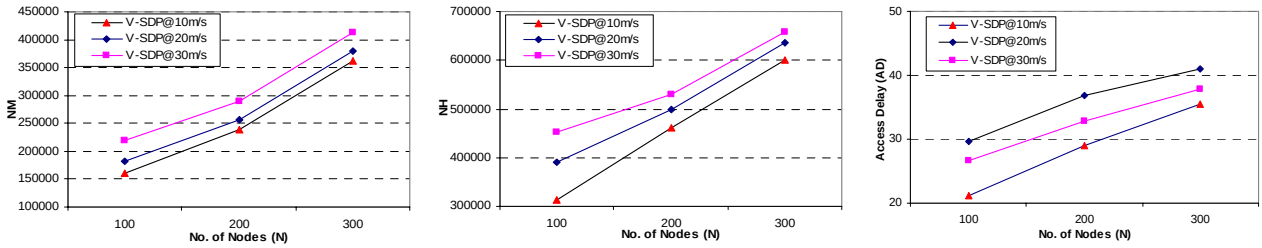
(b2) NH vs. N (V=10, 20, 30m/s)



(c2) AD vs. N (V=10, 20, 30 m/s)



(a3) NM vs. N (V=10, 20, 30m/s) (b3) NH vs. N (V=10, 20, 30m/s) (c3) AD vs. N (V=10, 20, 30 m/s)

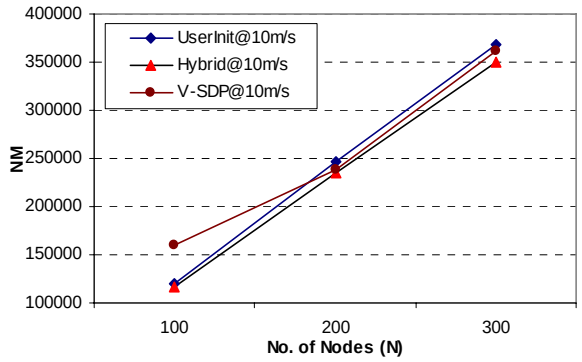


(a4) NM vs. N (V=10, 20, 30m/s) (b4) NH vs. N (V=10, 20, 30m/s) (c4) AD vs. N (V=10, 20, 30 m/s)

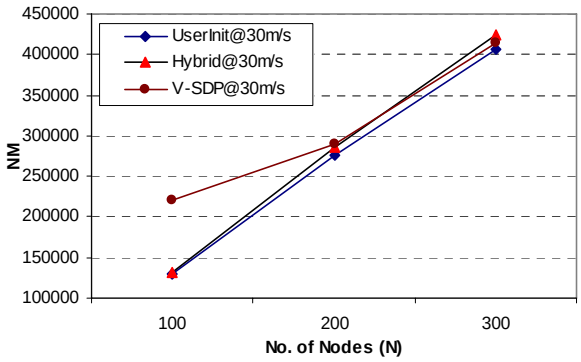
Figure 6-6: Effect of Node Mobility on the Service Discovery Operations (Study 1)

For clearer understanding regarding the variation of performance metrics for "UserInit", "Hybrid" and "V-SDP" we plot several graphs in Figure 6-7. The figures compare the performance of NM, NH and AD (also the AD values of "ProvInit" has been included) with N for the three protocols at node speeds of 10m/s and 30m/s and keeping F_R at 20%.

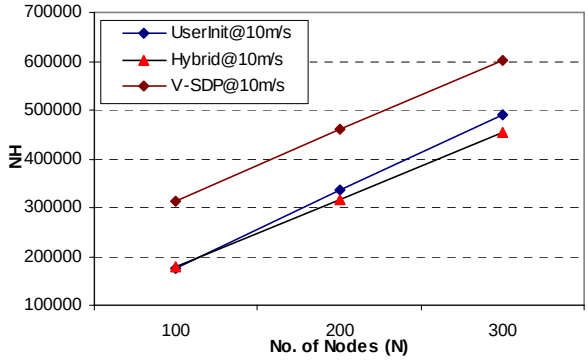
The results are similar to those observed in Figure 6-6. Figure 6-7 (a) and (b) shows that the three protocols use similar NM. But, Figure 6-7 (c) and (d) shows that the NH requirement for "V-SDP" is much higher than rest of the two protocols. The reason for higher NH in "V-SDP" has already been explained above. We can see from Figure 6-7 (e) and (f) that the AD is highest for "V-SDP" and least for "Hybrid" at both V= 10m/s and V= 30m/s.



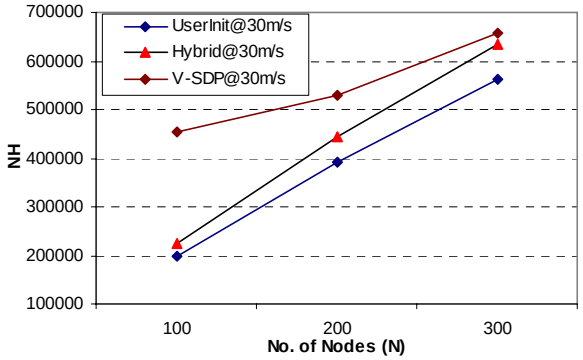
(a) NM vs. N (V=10m/s)



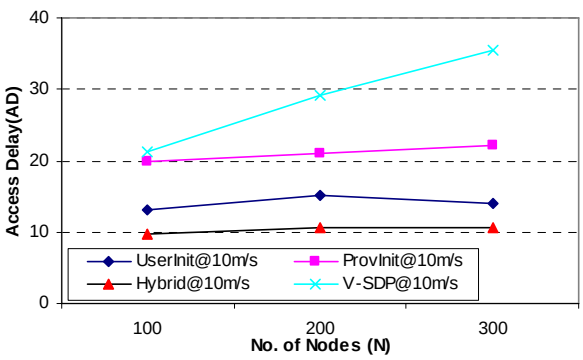
(b) NM vs. N (V=30m/s)



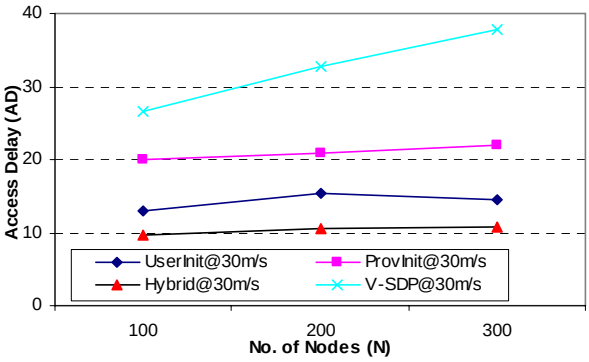
(c) NH vs. N (V=10m/s)



(d) NH vs. N (V=30m/s)



(e) AD vs. N (V=10m/s)

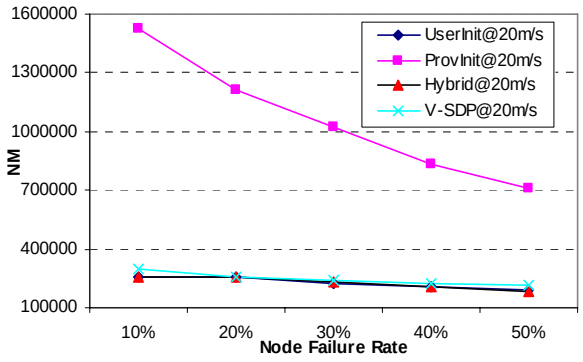


(f) AD vs. N (V=30m/s)

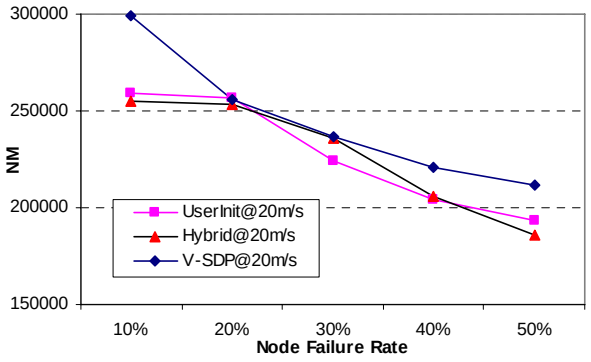
Figure 6-7: Effect of Node Mobility on the Service Discovery Operations (Study 2)

C. Effect of Node Failure

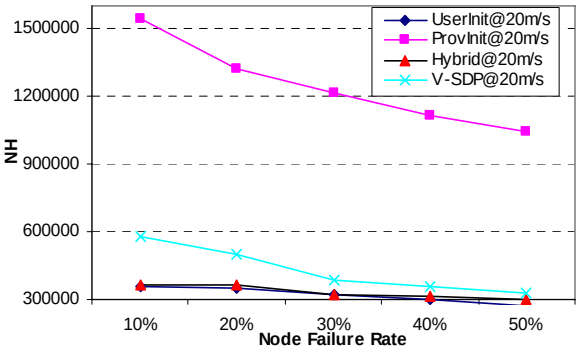
In this sub-section, we compare the service discovery performance of our proposed handoff protocols with "V-SDP" with regard to varying node failure rates (F_R). All experiments have been executed at default values of simulation parameters.



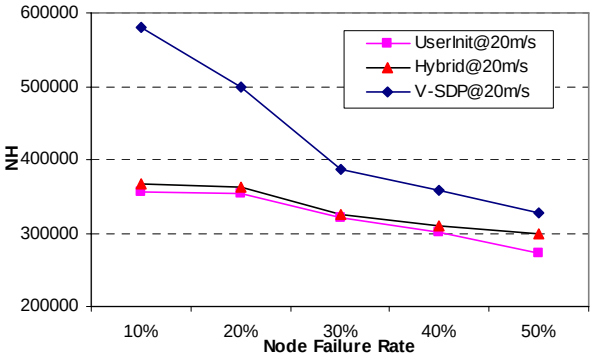
(a) NM vs. F_R



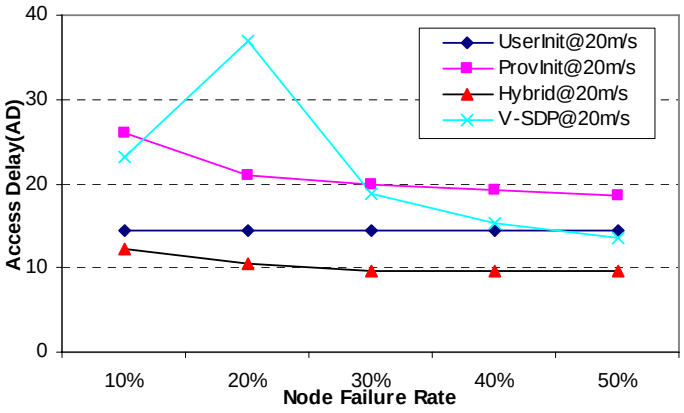
(b) NM vs. F_R (for "UserInit", "Hybrid", "V-SDP")



(c) NH vs. F_R



(d) NH vs. F_R (for "UserInit", "Hybrid", "V-SDP")



(e) AD vs. F_R

Figure 6-8: Effect of Node Failure on the Service Discovery Operations

Effect of F_R on NM: Figure 6-8 (a) shows that NM decreases for all the protocols with increasing node failure rates. The decrease is most prominent for "ProvInit" because with high F_R, the network gets less dense and hence the paging cost decreases rapidly. The other three protocols ("UserInit", "Hybrid" and "V-SDP") show the similar

trend of variation in NM. This is more clearly described in Figure 6-8 (b) where NM decreases rapidly for "UserInit", "Hybrid" and "V-SDP" with increase in F_R. This is because, at high F_R, number of service providers and users decreases.

Effect of F_R on NH: Figure 6-8 (c) shows the variation of NH decreases for all the protocols with increasing node failure rates. This trend is similar to that observed for NM. Figure 6-8 (d) shows the change of NH only for "UserInit", "Hybrid" and "V-SDP".

Effect of F_R on AD: Figure 6-8 (e) shows that AD decreases rapidly with increased F_R for "V-SDP". This is because, at high node failure rates, number of successful service discovery decreases, so the average AD also decreases. But for our handoff protocols, the access delay decreases very slowly. This is due to the fact that, at high F_R many service provider nodes fail which requires further service handoff and increases the AD.

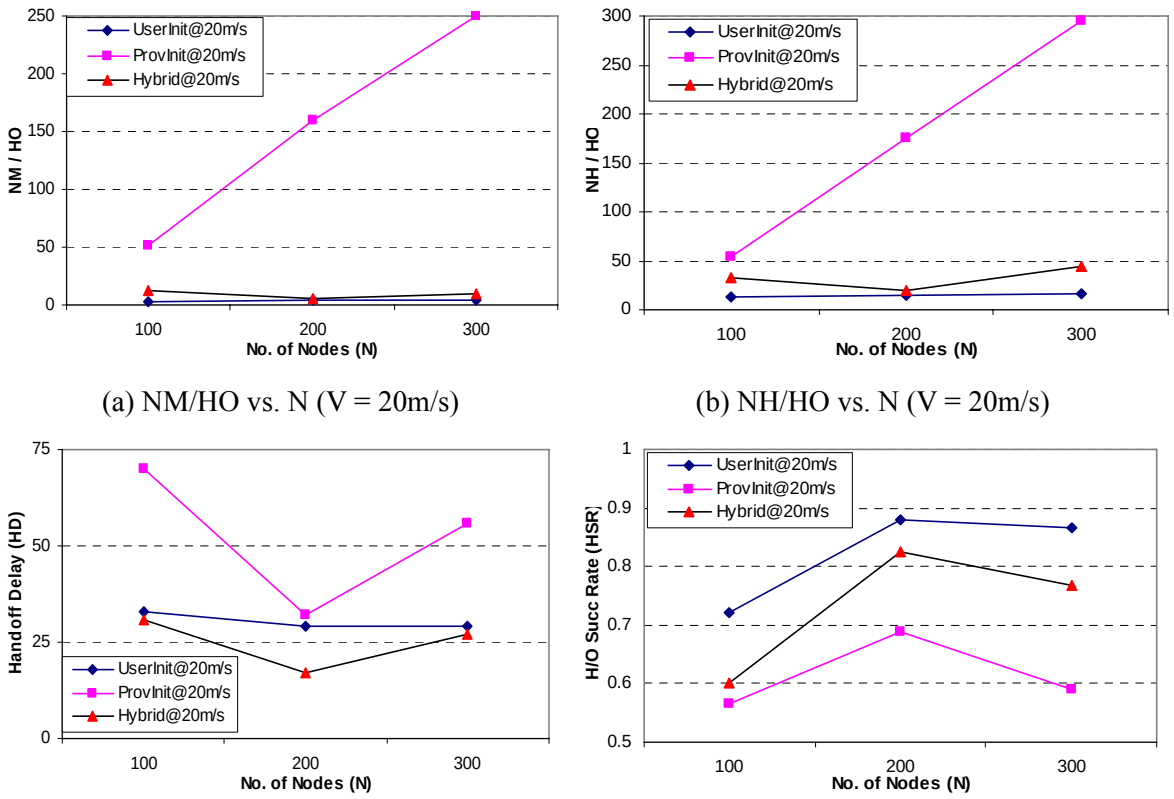
From the above discussions we can find out that our proposed handoff protocols are message efficient, incur low access delay and can cope with frequent node mobility and node failures. So, service handoff can be useful in service discovery operations in mobile and pervasive environments. In the next section, we shall analyze the performance of our proposed handoff protocols in greater detail.

6.5.2.3 Performance Comparison of Service Handoff Protocols

In this sub-section, we rigorously study the handoff performances of "UserInit", "ProvInit", and "Hybrid", first under default values of simulation parameters and then by varying node speeds and node failure rates.

A. Performance in General Cases

Here, we analyze "UserInit", "ProvInit", and "Hybrid" under the default values of the simulation parameters.



(a) NM/HO vs. N (V = 20m/s)

(b) NH/HO vs. N (V = 20m/s)

(c) HD vs. N (V = 10m/s)

(d) HSR vs. N (V = 20m/s)

Figure 6-9: General Performance of Service Handoff Protocols

Performance of NM/HO: Figure 6-9 (a) shows the results of NM/HO of our proposed handoff protocols. Both the metrics are highest for "ProvInit" and lowest for "UserInit". Paging, used in "ProvInit" increases the message cost per handoff. We can also find out that NM/HO for "Hybrid" slightly decreases with N initially and then increases with further increase in N. This is because of the fact that, at very low network density, the user terminal cannot reach the domain elector with single try due to frequent network disconnection. At very high node density, however, each message is forwarded through many intermediate hops, thereby increasing NM/HO. "UserInit" is the most message-efficient among all the handoff protocols as no paging is required here.

Performance of NH/HO: Figure 6-9 (b) shows the results of NH/HO of our proposed handoff protocols. Following the same trend as in NM/HO, NH/HO is highest

for "ProvInit" and lowest for "UserInit". Paging, used in "ProvInit" increases the message cost per handoff. We can also find out that NH/HO for "Hybrid" slightly decreases with N initially and then increases with further increase in N. This has already been explained above.

Performance of HD: Figure 6-9 (c) shows the change of HD with N. HD is the highest for "ProvInit" and lowest for "Hybrid". For "ProvInit" HD is highest when the network is sparse. This is because repeated paging may be required to reach all nodes in a sparse network. The HD decreases as the network gets connected. But, at very high node density, the HD increases again as the NM/HO and NH/HO increases. We can observe similar effects for "Hybrid" which can be explained with same logic except one thing. The high HD at lower node density is because of the fact that the user terminal may not reach the domain elector with single try due to the disconnected network status. The HD, however, decreases slowly with N for "UserInit" as the network grows denser. For "UserInit", the HD steadily decreases as the network grows denser. This is because, "UserInit" does not require paging and the handoff delay in this case is solely dependant on the combined message transmission delays (no paging delays). This also contributes to the higher rate of handoff success for "UserInit", as explained in the next paragraph.

To further bolster the accuracy of our simulation results, we have calculated the confidence of the HD metric. Sample means for "UserInit", "ProvInit" and "Hybrid" with N=100 are 29, 32 and 17.6, respectively. Our measurement finds out that HD of each of the simulated protocols has a 95% confidence level with confidence intervals of (28.3990 to 29.9421), (30.2513 to 32.9269) and (17.4932 to18.1067), respectively.

Performance of HSR: Figure 6-9 (d) plots the variation of HSR with N for the handoff protocols. For all the handoff protocols, HSR increases initially with N and then decreases with further increase in N. This is because, at lower node density the handoff

is often unsuccessful due to network disconnection. The success rate increases as the network grows denser. However, at very high network density the high handoff delay may render the handoff unsuccessful. For "ProvInit", at very high network density, the migrated user may move to a different domain even before it has been detected, thus the HSR becomes further low. The HSR values prove that "ProvInit" is the least efficient of the proposed service handoff protocols. But we can as well see that, "UserInit" and "Hybrid" achieve pretty high handoff efficiency.

We also calculate the confidence on the simulated values of HSR and we found that the sample mean with N=100 for "UserInit", "ProvInit" and "Hybrid" are 0.88, 0.6875 and 0.826, respectively. ET of each of the simulated protocols has a 95% confidence level with confidence intervals of (0.8757 to 0.8908), (0.6705 to 0.7379) and (0.7822 to 0.8923), respectively.

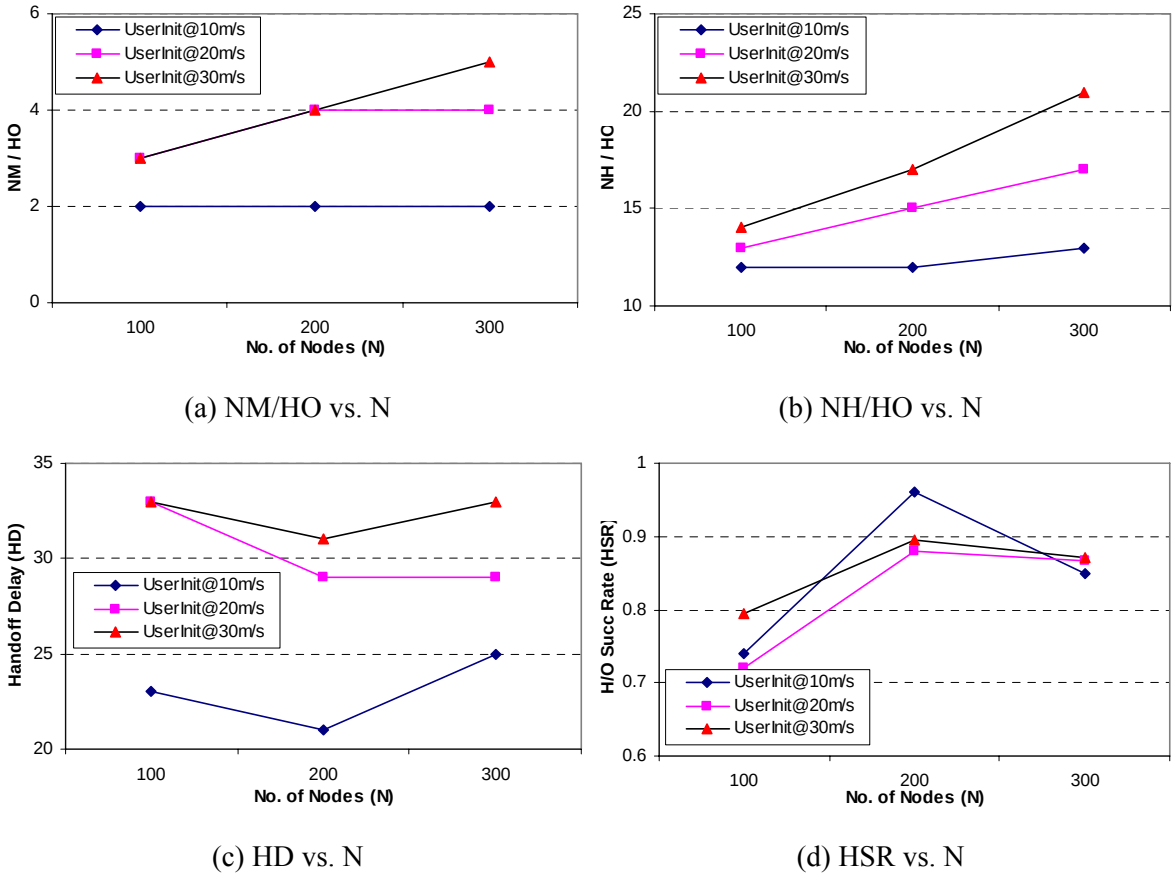
B. Effect of Node Mobility

In this sub-section we study the effect of varying V (from 10m/s to 30m/s, keeping F_R at 20%) on the performance of our handoff protocols, on an individual basis.

Effect of Mobility on "UserInit": Figure 6-10 (a) and (b) shows that NM/HO and NH/HO increases with increase in V. This is because at higher V number of handoff increases due to frequent node mobility.

As shown in Figure 6-10 (c), HD increases with V. This is normal as the users or service providers frequently change domain, thereby increasing the handoff completion time. Also HSR decreases with increase in V (Figure 6-10 (d)), because nodes frequently change domains before a handoff is successfully completed. But, at very high node speeds (V=30m/s) migrated nodes sometime rejoins their previous domain before

the handoff fails and thus makes the handoff successful. Same can be observed for "Hybrid" as described later.



(a) NM/HO vs. N

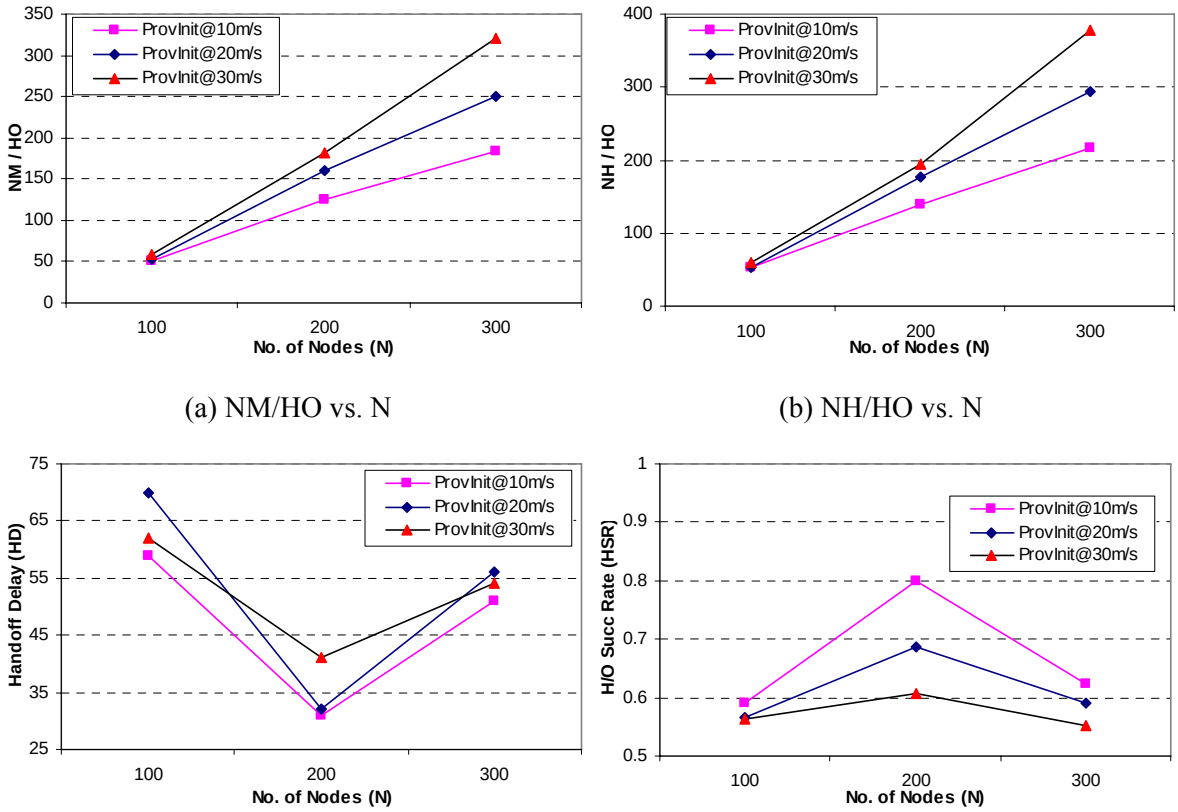
(b) NH/HO vs. N

(c) HD vs. N

(d) HSR vs. N

Figure 6-10: Effect of Node Mobility on "UserInit"

Effect of Mobility on "ProvInit": Figure 6-11 shows the "ProvInit" performances with varied node speeds. The trends of variation of different metrics are similar to those observed in Figure 6-10 and can be explained similarly. However, the NM/HO, NH/HO and HD in this case are considerably higher than those of "UserInit" because of the paging time involved. The paging delay increases the resulting HD (Figure 6-11 (c)) compared to other two handoff protocols. The HSR (Figure 6-11 (d)) is also lowest for "ProvInit" due to the high HD.



(a) NM/HO vs. N

(b) NH/HO vs. N

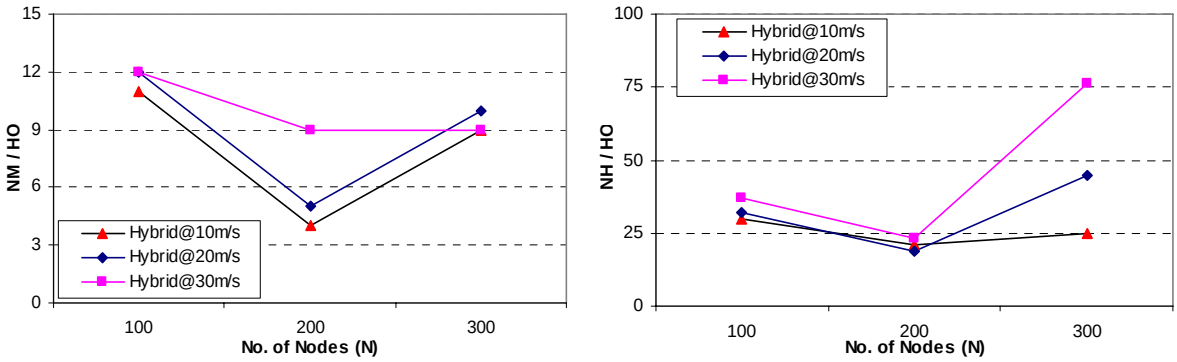
(c) HD vs. N

(d) HSR vs. N

Figure 6-11: Effect of Node Mobility on "ProvInit"

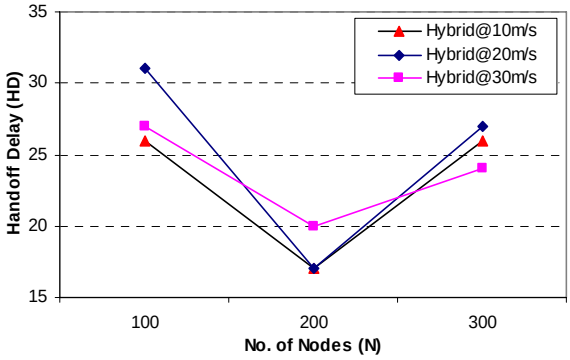
Effect of Mobility on "Hybrid": Figure 6-12 shows the performances of "Hybrid" with varied node speeds.

The trends of variation of different metrics are once again similar to those observed in Figure 6-10 and Figure 6-11, and can be explained similarly. The handoff message costs and time delay for "Hybrid" increases with increase in V. The handoff success rate decreases, in general, with increase in V, but at very high node speed (V = 30m/s), the HSR slightly increases. This has already been explained before.

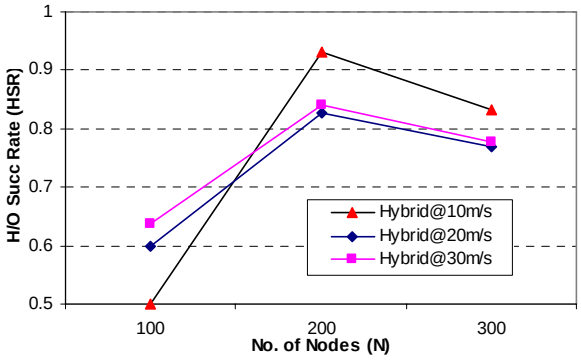


(a) NM/HO vs. N

(b) NH/HO vs. N



(c) HD vs. N

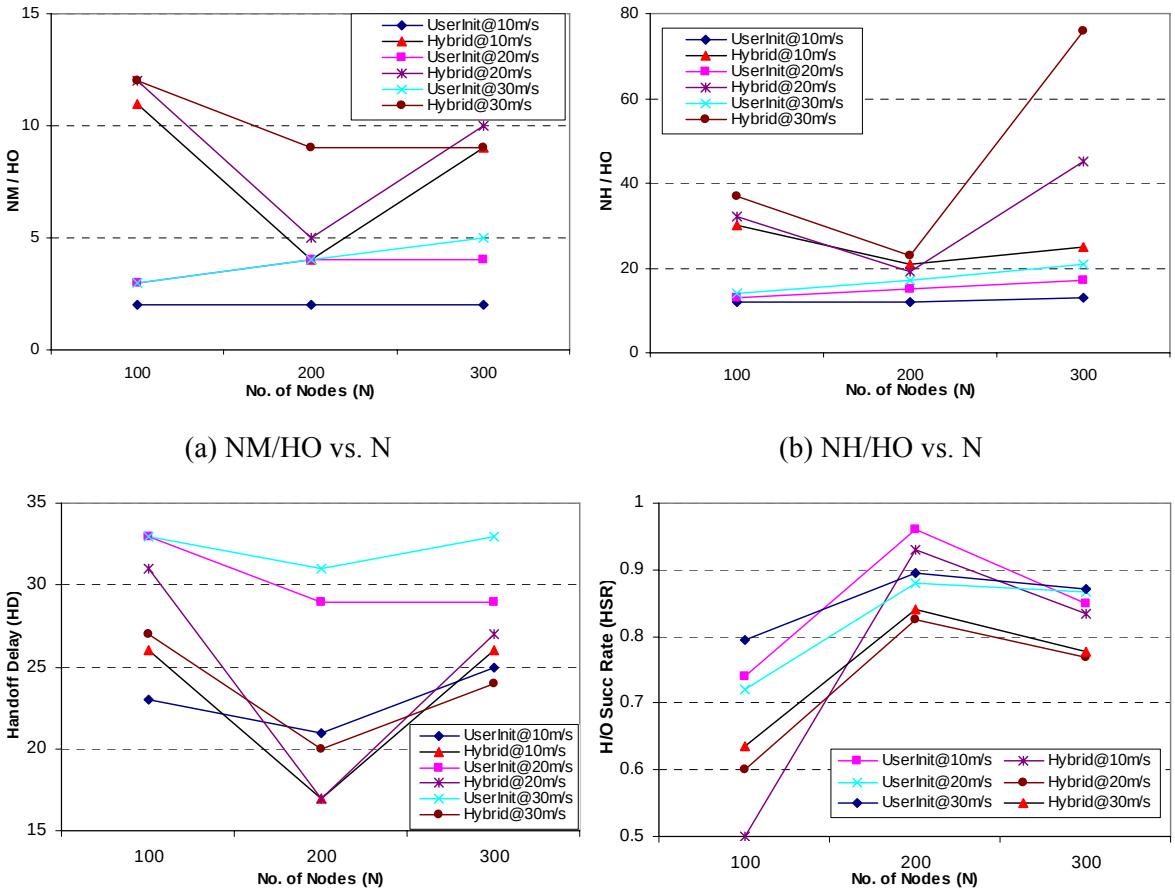


(d) HSR vs. N

Figure 6-12: Effect of Node Mobility on "Hybrid"

Performance comparison of "UserInit" and "Hybrid": We have plotted results (Figure 6-13) comparing "UserInit" and "Hybrid" at different node speeds. The values of the performance metrics of these two handoff protocols are closer to each other.

Figure 6-13 (a) and (b) shows the variations in NM/HO and NH/HO and we can find that for both the protocols, the message costs increase with node speeds. Also, "Hybrid" is always little more expensive than the "UserInit". At very high node density and node speeds, the NH/HO increases rapidly for the "Hybrid". This is because, "Hybrid" uses paging and at considerably higher node density, each message is forwarded through many intermediate hops, thereby increasing NH/HO.



(a) NM/HO vs. N

(b) NH/HO vs. N

(c) HD vs. N

(d) HSR vs. N

Figure 6-13: Comparison of Effect of Node Mobility on "UserInit" and "Hybrid"

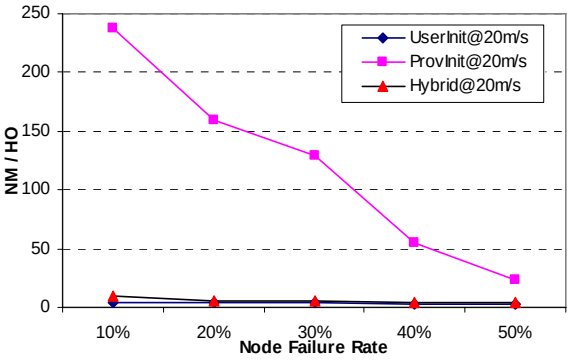
The change in handoff delay (HD) for "Userinit" and "Hybrid" has been depicted in Figure 6-13 (c). We can see that the HD increases with node speeds and "UserInit" incurs higher handoff delay than "Hybrid" in general. "Hybrid" can achieve a low HD compared to "UserInit" as the handoff process is started by both the parties concerned, i.e., the user and the service provider, which ultimately helps to reduce the total handoff delay. "UserInit", on the other hand, faces higher HD by waiting for response from the previous domain elector of the migrated user.

Figure 6-13 (d) shows the Handoff success rate (HSR) is higher for "UserInit" than "Hybrid" and decreases slightly at V=20m/s for both the protocols. This has already

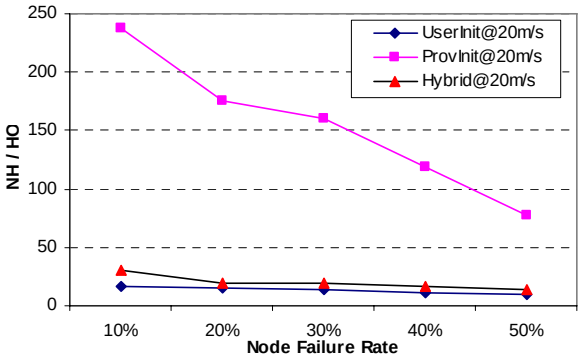
been explained earlier. "UserInit" has higher HSR compared to "Hybrid" as "UserInit" does not use paging for handoff. One interesting thing to notice from the Figure 6-13 is that the simulation parameters for service handoff mostly achieves best result with N=200. So, we can conclude that at this value of N, the network is ideally connected to realize the best handoff performance.

C. Effect of Node Failure

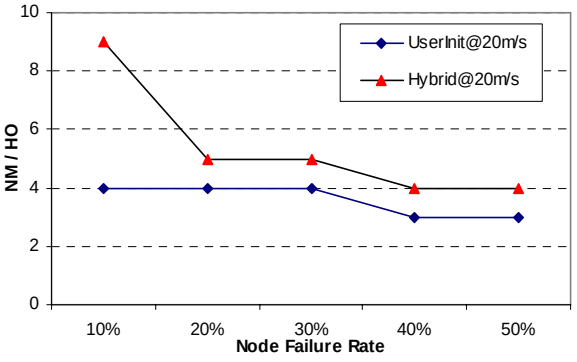
In this sub-section, we study the effects of varying F_R on the performance of our proposed handoff protocols. All experiments have been executed at default values of simulation parameters and the results are presented in Figure 6-14.



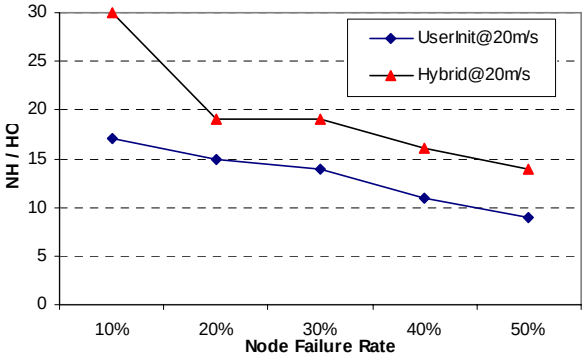
(a) NM/HO vs. F_R



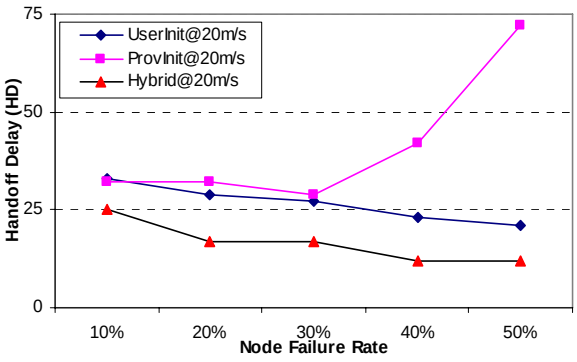
(b) NH/HO vs. F_R



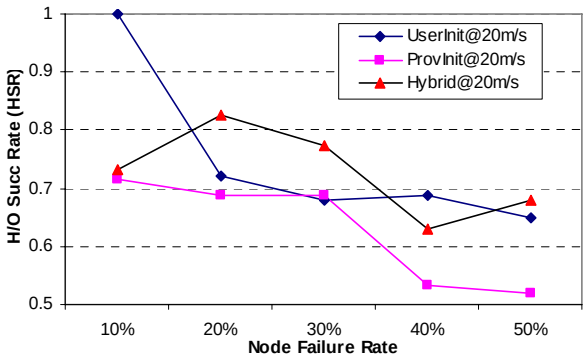
(c) NM/HO vs. F_R ("Userinit" & "Hybrid")



(d) NH/HO vs. F_R ("Userinit" & "Hybrid")



(e) HD vs. F_R



(f) HSR vs. F_R

Figure 6-14: Effect of Node Failure on Service Handoff

Effect of F_R on NM/HO: Figure 6-14 (a) shows that NM/HO decreases with increase in F_R. This is straightforward, because at higher F_R, N decreases, so NM/HO also decreases. But the decrease is not very fast for "UserInit" and "Hybrid", because, higher F_R means many service providers fail and their users need to be handed off to other providers. However, NM/HO decrease rapidly for "ProvInit" because increasing F_R signifies lower node density and less paging cost for a single handoff.

Effect of F_R on NH/HO: From Figure 6-14 (b) we can see that NH/HO decrease with increased F_R. This can be explained similar to NM/HO, because at higher F_R, N decreases, and thus NH/HO also decreases.

Effect of F_R on HD: The HD (Figure 6-14 (c)) also decreases very slowly with increased F_R except for "ProvInit". This is due to the fact that, at higher F_R, the network density decreases rapidly which results in network disconnection and hence the delay in finding migrated user by paging becomes higher for "ProvInit". For "UserInit" and "Hybrid", the decrease in HD is simply attributed to the low value of N.

Effect of F_R on HSR: The HSR also decreases (Figure 6-14 (d)) with increasing F_R due to frequent failure of nodes.

From the above results, we can see that our service handoff protocols can cope well with high node failure rates and still can guarantee HSR of more than 50%. So, our protocols are robust under high node failure rates and high node mobility.

6.6 Summary

In this chapter, we address the issues regarding continuous and seamless service access for mobile users in an ad hoc infrastructure-less pervasive computing environment. The continuous service access is often hampered due to service provider failure, network partitioning, or service scope outage by service provider or user mobility. Service handoff is needed to provide users with alternate matching services in case the original service becomes unavailable. However, existing service discovery solutions for pervasive computing do not address this problem.

We have studied the problem and have presented our novel mechanism for seamless service access for mobile users in pervasive computing environments using service handoff. Service handoff enables users to continue service access while on the move, by automatically transferring the service execution states to an alternative matching service provider when the original provider fails or the user moves out of its scope. We have developed three service handoff protocols - user initiated, service provider initiated and hybrid - depending on the initiator of the handoff. We have carried out extensive simulations to study the benefit of using service handoff for continuous service access and also to analyze the handoff efficiency of our proposed protocols. The results show that our protocols can support seamless service access for mobile users at low message cost and time delay while achieving high load balance among service providers.

Chapter 7

Conclusion and Future Directions

This chapter concludes this dissertation by summarizing our original contributions in Section 7.1 and by pointing towards the possible future directions of furthering our research in Section 7.2.

7.1 Conclusions

As a newly evolving paradigm of computing, pervasive computing has received significant research focus in the recent years. Pervasive applications have been developed over both wired and infrastructured wireless networks as well as ad hoc wireless networks in order to serve different requirements and objectives. Service discovery is a well-known application of mobile and pervasive computing. As the pervasive computing paradigm gradually changes the world into a service-based one more and more efficient service discovery solutions are being sought by users for various system and network structures. Though there are many existing service discovery protocols for wired and infrastructure-based wireless networks having support of fixed and stable network backbone, service discovery research over the ad hoc wireless network consisting of static or mobile entities is far from complete. One of the crucial issues of service discovery in pervasive computing is about how to ensure

reliability and fault tolerance during ongoing service discovery and access operations. Many pervasive applications, such as, health care, elderly care, or smart environments, are built to provide humans with unfaltering service support all the time. So, reliability or availability of services cannot be compromised in these critical applications. In this dissertation we mainly focus on addressing the reliability issues of service discovery and access in pervasive applications developed over extremely dynamic ad hoc and infrastructure-less network environments consisting of multiple resource-constrained static or mobile devices. We have modeled the underlying networks to be mobile ad hoc (MANET) in nature. We provide innovative and cost-efficient solutions to address the reliability concerns.

Our research in fault tolerant service discovery is based on a directory community framework which consists of a directory community structure and a suite of reliable service discovery and access protocols developed over the structure. A directory community is a collection of top-K weighted mobile nodes, where weight indicates various node resources, such as, the memory size, processing capacity, or remaining battery life, etc, and which perform as service discovery directories. We prefer directory based solution as it is more robust than a directory-less one and has the power to minimize both message and energy costs. Moreover, the weight-based election will ensure that the elected directories are more reliable and less fault-prone. We have designed an algorithm for top K directory election in ad hoc and mobile environment through localized interaction of participating nodes. The algorithm is phase-based and works using a diffusing computation approach. It first selects some coordinator nodes with highest weight among their 2-hop neighbors. The coordinator nodes then start diffusing computations to collaboratively collect the weight values of all the nodes in the environment to choose top K weighted nodes as K leaders. Besides proving the correctness properties of the proposed algorithm, we have also evaluated our algorithm

by extensive simulations. The results show that our algorithm is fault-tolerant and message-efficient and can cope with dynamic topological changes that frequently occur in mobile ad hoc environment. We have also implemented our algorithm on a wireless testbed to study its performance and applicability in real environments.

Based on the directory community framework, we have developed two different mechanisms – one for reliable service discovery in the midst of sudden failure of directory nodes and the other for reliable and continuous service access despite service provider failure, node mobility and network partitioning.

The mechanism for reliable service discovery consists of a discovery protocol which aims to address the different service unavailability issues caused by many possible failure situations. While simple service failure requires fresh service discovery attempts on the part of the users, failures of service directories are more complicated to handle. Directory failure necessitates all services registered with a failed directory to re-register with other available directories in order to publish them. To cope with this problem we propose to replicate services among multiple directory nodes. In order to limit replication and update costs, we choose a selective replication policy, by which, all the directory nodes form quorums among themselves and replicate services registered with them among its quorum members. This approach ensures network-wide service availability with minimal replication. Following the quorum intersection property, we can guarantee that if a service matching user request is available, the user can certainly find the service by forwarding a request only to its quorum members. This reduces service discovery cost. The message overhead is further reduced by dividing the network into one or more tree-structured domains, thereby eliminating loops, and also by restricting broadcast and flooding. We handle failure of directory nodes, by timely replacing a failed directory with a suitable node carefully picked up using an incremental directory election approach. We have carried out extensive simulations to

study the important properties supported by our protocol. The results show that our protocol is fault-tolerant, message-efficient, and incurs lower delay in service discovery. We have also implemented a prototype of our protocol on a wireless testbed system consisting of multiple sensor nodes. The implementation results corroborate with our simulation studies and prove the applicability of the protocol in practical scenarios.

The other mechanism for seamless service access is developed to support users in ad hoc pervasive environment with reliable and continuous service access. Given the dynamic nature of MANET, resource-constrain of the participating devices, and the unreliability of wireless connection, service access in pervasive environment is often unreliable and intermittent. Service providers and users frequently get disconnected due to their mutual mobility, node failures, service failures, and network partitions. To cope with these limitations, we propose a service handoff scheme which automatically selects new matching service providers for users, once the original service provider becomes unavailable. Service handoff is different from traditional handoff operations for mobility management in wireless networks which focuses on the quality of network connection and initiates handoff based on signal strength or other related metrics. Service handoff operation consists mainly of two steps – service handoff initation and handoff destination selection. Handoff initiation requires detecting the requirements of triggering a handoff, which is non-trivial given the extreme dynamics of the MANET environment. Handoff destination selection is also challenging as it requires us to achieve a load balance among the service providers; otherwise, resource depletion of service providers may result in increased handoff frequency. In this research, we present three novel service handoff protocols, depending on the action performed by the initiating node, which achieves load balancing among the service provider and increases reliability of service access. We have studied the performance of our protocol by simulations. The results indicate that our proposed service handoff protocols can support seamless service

access for mobile users at low message cost and time delay while achieving high load balance among service providers.

In summary, our algorithms and protocols are experimentally proven to support reliability, scalability and fault tolerance for service discovery and access applications in dynamic and ad hoc mobile environments consisting multiple service providers and users. So, we can claim to have fulfilled our objectives proposed at the beginning of this dissertation.

7.2 Future Directions

We close this dissertation with our comments and suggestions on the ways in which the current research can be advanced.

Presently our distributed algorithm for K-directory election works through localized collaboration of the participating devices, where, finally, one single node must collect the global knowledge to elect the top K weighted directory nodes. This approach requires extra maintenance overhead for the single elector node. In future, we want to design a new algorithm which will act in a purely localized manner to elect directory nodes. Localized algorithms with high degree of localization incur low message cost and help to reduce the maintenance overhead. Since, our protocols for reliable service discovery and access are based on the directory community framework, a more efficient approach for directory election and directory community maintenance will surely improve the overall performance of our protocols. But, designing a truly localized algorithm to ensure globally unique K directory nodes is a non-trivial task.

We have realized that, it is not always possible to guarantee exactly K directory nodes in a MANET at all the times. Instead, we can opt for selecting directory nodes in such a way that each directory node serves equal or almost same number of users. This

approach will be effective in achieving load balance among the directories and hence the average lifetime of individual directory nodes will be increased. When combined with our service handoff mechanism, which tries to achieve load balance among the service providers, this improved directory election policy will guarantee significant increase of the average lifetime of the pervasive system. To obtain further energy optimization, nodes lying in network boundary should not be chosen as directories, as they will be less utilized and hence their purpose will not be satisfied.

So far, our directory election algorithm and the quorum based reliable service discovery protocol have been implemented using a wireless sensor network tested with all the nodes being MicaZ sensor motes. Being sure that our protocol performs well using the resource constrained sensor nodes, we want to carry out further experimentation using multitude of devices consisting of sensor nodes, smart phones, laptops and PDAs. Device heterogeneity will surely help us to better understand the weaknesses of our protocols and to take necessary improvement measures. We also want to undertake experiments to study the feasibility of our service handoff algorithms under different operating environments.

Finally, we would like to investigate the performance of our protocols and algorithms in heterogeneous network environments rather than the pure mobile ad hoc networks we have considered so far. More challenges will be introduced if a heterogeneous network, comprising ad hoc as well as infrastructured components with multiple networking protocols and interfaces, is considered. Our protocols must be adapted to the different situations that may occur due to the mixture of various networks. Service handoff will have to take care of the issues related to inter-network handoff and service discovery among cross platform nodes will also be much more difficult. If different directory nodes support different network interfaces, the inter-directory communication must be facilitated through special protocol adapters.

Bibliography

Bibliography

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "*The Design and Implementation of an Intentional Naming System*," In Proceedings of Seventeenth ACM Symposium on Operating Systems Principles (SOSP'99), ACM Press, pages 186–201, Charleston, SC, December, 1999.

[2] M. Aguilera, C. Gallet, H. Fauconnier, and S. Toueg, "*Stable leader election*," In Proceedings of the Fifteenth International Symposium on Distributed Computing (DISC'2001), LNCS 2180, 2001.

[3] B. Al-Takrouri, K. Detken, C. Martinez, M. K. Oja, S. Stein, L. Zhu, and A. Schrader, "*Mobile Holsten Tour: Contextualized Multimedia Museum Guide*," In Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia (MoMM), pp. 460-463, Linz, Austria, November, 2008.

[4] A. Amis, R. Prakash, T. Vuong, and D.T. Huynh, "*Max-Min D-Cluster Formation in Wireless Ad Hoc Networks*," In Proceedings of Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), Volume 1, pp. 32 – 41, Tel Aviv, Israel, March, 2000.

[5] K. Arabshian and H. Schulzrinne, "*GloServ: Global service discovery architecture*," In Proceedings of MobiQuitous, pp. 319–325, June, 2004.

[6] I. Aydin, C. Jaikaeo, and C. Chen, "*Quorum-based Match-Making for Wireless Mesh Network*," In Proceedings of First IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob'2005).

[7] B. R. Badrinath, A. Acharya, and T. Imielinski, "*Designing Distributed Algorithms for Mobile Computing Networks*," Journal of Computer Communications. Volume 19, No. 4, April, 1996.

[8] M. Balazinska, H. Balakrishnan, and D. Karger, "*INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery*," In Proceedings of

International Conference on Pervasive Computing 2002, August, 2002.

[9] S. Banerjee and S. Khuller, "*A Clustering Scheme for Hierarchical Control in Multi-Hop Wireless networks*," In Proceedings of IEEE Conference on Computer Communications (INFOCOM), Anchorage, Alaska, April, 2001.

[10] M. Bang, A. Larsson, and H. Eriksson, "*NOSTOS: A Paper-Based Ubiquitous Computing Healthcare Environment to Support Data Capture and Collaboration*," In Proceedings of the 2003 AMIA Annual Symposium, pp. 46-50, Washington DC, Nov 8-12, 2003.

[11] S. Basagni, "*Distributed clustering for ad hoc networks*," In Proceeding of International Symposium on Parallel Architectures, Algorithms and Networks, pp. 310–315, June, 1999.

[12] S. Basagni, "*Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks*," In Proceedings of Vehicular Technology Conference, VTC, Vol. 2, pp. 889–893, 1999.

[13] P. Basu, N. Khan, and T. Little, "*A Mobility based metric for clustering in mobile ad hoc networks*," In Proceedings of International Workshop on Wireless Networks and Mobile Computing, April. 2001.

[14] Bluetooth SIG. Specification. http://bluetooth.com/.

[15] J. Bohn, "*The Smart Jigsaw Puzzle Assistant: Using RFID Technology for Building Augmented Real-World Games*," In Proceedings of the Pervasive Games Workshop, 2004.

[16] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer, "*Easyliving: Technologies for Intelligent Environments*," In Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing (HUC), Lecture Notes in Computer Science (LNCS), Volume 1927, pp. 12-29, Bristol, UK, September, 2000.

[17] J. Brunekreef, J. Katoen, R. Koymans, and S. Mauw, "*Design and Analysis of*

Leader Election Protocols in Broadcast Networks," Distributed Computing, vol. 9 no. 4, pages 157-171, 1996.

[18] M. Caesar, M. Castro, et al., "*Virtual ring routing: network routing inspired by DHTs*," In Proceedings of ACM SIGCOMM, pp. 351-362, 2006.

[19] M. Castro, P. Druschel, A.-M. Kermarrec. and A. Rowstron, "*One ring to rule them all: Service discovery and binding in structured peer-to-peer overlay networks*," In Proceedings of the SIGOPS European Workshop, Saint-Emilion, France, September. 2002.

[20] D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha, "*GSD: A novel group-based service discovery protocol for MANETs*," In Proceedings of Fourth IEEE Conference on Mobile and Wireless Communications Networks (MWCN), September, 2002.

[21] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin, "*Toward distributed service discovery in pervasive computing environments*," IEEE Transactions on Mobile Computing, February, 2006.

[22] M. Chatterjee, S. K. Das. and D. Turgut, "*WCA: A Weighted Clustering Algorithm for Mobile Ad hoc Networks*," Journal of Cluster Computing, Special issue on Mobile Ad hoc Networking, Vol. 5, Issue 2, pp. 193-204, April 2002.

[23] A. Chen, R.R. Muntz, S. Yuen, I. Locher, S. Park, and M.B. Srivastava, "*A Support Infrastructure for the Smart Kindergarten*," IEEE Pervasive Computing, Volume 1, No. 2, pp. 49-57, April-June, 2002.

[24] H. Chen, F. Perich, D. Chakraborty, T. Finin, and A. Joshi, "*Intelligent Agents Meet Semantic Web in a Smart Meeting Room*," Proceedings of the Third International Joint Conference on Autonomous Agents & Multi Agent Systems (AAMAS'04), July, 2004.

[25] L. Cheng and I. Marsic, "*Service discovery and invocation for mobile ad hoc networked appliances*," In Proceedings of Second International Workshop

Networked Appliances (IWNA 00), December 2000.

[26] S. Cheshire and M. Krochmal. "*DNS-Based Service Discovery*," IETF Internet draft, September 2008, http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt.

[27] C. Cho and D. Lee, "*Survey of Service Discovery Architectures for Mobile Ad Hoc Networks*," Unpublished term paper, Mobile Computing, CEN 5531, Computer and information sciences and Engineering Department, University of Florida Gainesville, USA, 2005.

[28] T. Clausen and P. Jacquet, "*Optimized link state routing protocol (OLSR)*," RFC 3626, October, 2003.

[29] C. Coore, R. Nagpal, and R. Weiss, "*Paradigms for Structure in an Amorphous Computer*," Technical Report 1614, Massachusetts Institute of Technology Artificial Intelligence Laboratory, October, 1997.

[30] Crossbow Technology: http://www.xbow.com/.

[31] B. Das and V. Bharghavan, "*Routing in ad-hoc networks using minimum connected dominating sets*," In Proceedings of IEEE International Conference on Communications (ICC), pages 376–380, 1997.

[32] F. Delmastro, "*From Pastry to CrossROAD: Cross-layer ring overlay for ad hoc networks*," In Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communication Workshops, pp.60–64, March, 2005.

[33] E. W. Dijkstra and C.S. Scholten, "*Termination Detection for Diffusing Computations*," Information Processing Letters, vol. 11, no. 1, 1980.

[34] E. W. Dijkstra, "*Self-stabilizing systems in spite of distributed control*," Communications of the ACM, 17:634-644, 1974.

[35] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "*Next Century Challenges: Scalable Coordination in Sensor Networks*," In Proceedings of ACM MobiCom, August, 1999.

[36] J. Favela, M. Rodriguez, A. Martinez, and V. Gonzalez, "*Ambient Computing*

Research for Healthcare: Challenges, Opportunities and Experiences," Computación y Sistemas Vol. 12 No. 1, 2008, pp 109-127, ISSN 1405-5546,

[37] P. Fergus, K. Kifayat, S. Cooper, M. Merabti, and A. El Rhalibi, "*A Framework for Physical Health Improvement using Wireless Sensor Networks and Gaming*," In Proceedings of ICST/IEEE International Workshop on Technologies to Counter Cognitive Decline (TCCD), in conjunction with the Third International Conference on Pervasive Computing Technologies for Healthcare (Pervasive Health), City University, London, UK, 31st March, 2009.

[38] G. Forman and J. Zahorjan, "*The Challenges of Mobile Computing*," IEEE Computer, vol. 27, no. 4, April, 1994.

[39] R. Gallager, P. Humblet, and P. Spira, "*A Distributed Algorithm for Minimum Weight Spanning Trees*," ACM Transactions on Programming Languages and Systems, vol.4, no.1, pages 66-77, January, 1983.

[40] H. Garcia-Molina, "*Elections in a Distributed Computing System*," IEEE Transactions on Computers, vol. C-31, no. 1, 1982.

[41] Gnutella Protocol Development, http://www.the-gdf.org, 2005.

[42] E. Guttman and C. Perkins, "*Service location protocol*," version 2, June 1999.

[43] R. Handorean, R. Sen, G. Hackmann, and G.-C. Roman, "*Context Aware Session Management for Services in Ad Hoc Networks*," In Proceedings of 2005 IEEE International Conference on Services Computing (SCC'05), pp.113-120, July, 2005.

[44] K. P. Hatzis, G. P. Pentaris, P.G. Spirakis, V.T. Tampakas, and R.B. Tan, "*Fundamental Control Algorithms in Mobile Networks*," In Proceedings of Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures, 1999.

[45] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "*Energy-Efficient Communication Protocol for Wireless Microsensor Networks*," In Proceedings of Hawaiian International Conference on Systems Science, January, 2000.

[46] S. Helal, W. Mann, H. El-Zabadani et al., "*The gator tech smart house: A*

programmable pervasive space," IEEE Computer, vol. 38, no. 3, pp. 50-60, 2005.

[47] S. Helal, N. Desai, V. Verma, and C. Lee, "*Konark- a service discovery and delivery protocol for ad hoc networks*," In Proceedings of the Third IEEE Conference on Wireless Communication Networks WCNC, March, 2003.

[48] H. Hemmati, A. Ranjbar, M. Niamanesh, and R Jalili, "*A Model to Support Context-Aware Service Migration in Pervasive Computing Environments*," In Proceedings of the Ninth World Multi-Conf. on Systemics, Cybernetics and Informatics, Orlando, Florida, USA, July 10-13, 2005.

[49] T. D. Hodes, Steven E. Czerwinski, Ben Y. Zhao, Anthony D. Joseph, and Randy H. Katz, "*An architecture for secure wide-area service discovery*," ACM Wireless Networks Journal, vol. 8, nos. 2/3, pp. 213–230, 2002.

[50] Jini Technology Core Platform Specification, v. 2.0, Sun Microsystems, June 2003; www.sun.com/software/jini/specs/core2_0.pdf.

[51] D. Johnson, D. Maltz, and J. Broch, "*DSR: The dynamic source routing protocol for multihop wireless ad hoc networks*," Chapter 5, pp.139–172, Addison-Wesley, 2001.

[52] S. Johnson. "*Emergence: the connected lives of ants, brains, cities, and software*," Penguin. 2001.

[53] E. Kang, M. J. Kim, E. Lee, and U. Kim, "*DHT-Based Mobile Service Discovery Protocol for Mobile Ad Hoc Networks*," In Proceedings of the Fourth International Conference on Intelligent Computing: Advanced Intelligent Computing Theories and Applications - with Aspects of Theoretical and Methodological Issues (ICIC '08), September, 2008.

[54] P. Kang, C. Borcea, G. Xu, A. Saxena, U. Kremer, and L. Iftode, "*Smart Messages: A Distributed Computing Platform for Networks of Embedded Systems*," Computer Journal, special issue on mobile and pervasive computing, pp. 475-494, 2004.

[55] Kazaa, http://www.kazaa.com.

[56] C. D. Kidd, R. Orr, G. D. Abowd, C. G. Atkeson, I. A. Essa, B. MacIntyre, E. D. Mynatt, T. Starner, and W. Newstetter, "*The Aware Home: A Living Laboratory for Ubiquitous Computing Research*", In Proceedings of Cooperative Buildings (CoBuild'99), pp.191-198, 1999.

[57] M. J. Kim, M. Kumara, and B.A. Shirazi, "*Service Discovery Using Volunteer Nodes in Heterogeneous Pervasive Computing Environments*," Journal of Pervasive and Mobile Computing, vol. 2, pp. 313-343, 2006.

[58] M. Klein, B. Konig-Ries, and P. Obreiter, "*Service rings – a semantic overlay for service discovery in ad hoc networks*," In DEXA Workshops, pages 180-185, 2003.

[59] M. Klein, B. Konig-Ries, and P. Obreiter, "*Lanes – a light weight overlay for service discovery in mobile ad hoc networks*," Technical Report 2003-6, University of Karlsruhe, May 2003.

[60] U. C. Kozat and L. Tassiulas, "*Service discovery in mobile ad hoc networks: an overall perspective on architectural choices and network layer support issues*," Ad Hoc Networks 2(1): 23-44 (2004).

[61] M. Kumar and S. K. Das, "*Pervasive computing: Enabling technologies and challenges*," In A. Zomaya, editor, Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies. Springer, 2006.

[62] C. Lee and S. Helal, "*A Multi-Tier Ubiquitous Service Discovery Protocol for Mobile Clients*," In Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'03), Montréal, Canada, 2003.

[63] W. Lee, W. Woo, and J. Lee, "*TARBoard: Tangible Augmented Reality System for Table-top Game Environment*," In Proceedings of the Pervasive Games Workshop 2005.

[64] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D.

Culler, "*The Emergence of Networking Abstractions and Techniques in TinyOS*," In Proceedings of NSDI, March, 2004.

[65] C. M. Lin, G. M. Chiu, and C. H. Cho, "*A New Quorum-Based Scheme for Managing Replicated Data in Distributed Systems*", IEEE Trans. Computers 51(12): 1442-1447 (2002).

[66] D. Lin and M. Gerla, "*Adaptive Clustering for Mobile Wireless Networks*," IEEE Journal on Selected Areas in Communications, 15(7):1265-75, September, 1997.

[67] N. Malpani, J. L. Welch, and N. vaidya, "*Leader Election Algorithms for Mobile Ad Hoc Networks*," In Proceedings of Fourth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, Boston, MA, August, 2000.

[68] R. S. Marin-Perianu, P. Hartel, and H. Scholten, "*A Classification of Service Discovery Protocols*," Technical Report TR-CTIT-05-25, Centre for Telematics and Information Technology, University of Twente, 2005.

[69] A. N. Mian, R. Baldoni, and R. Beraldi, "*A Survey of Service Discovery Protocols in Multihop Mobile Ad Hoc Networks*," IEEE Pervasive Computing, 8(1):66--74, 2009.

[70] M. Nidd, "*Service Discovery in DEAPspace*," IEEE Personal Communications, (2001) 39-45

[71] S.H. Park, "*An Election Protocol in a Mobile Environment*," In Proceedings of PDPTA2000, pp. 200-210, June, 2000.

[72] V. D. Park and M. S. Corson, "*A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks*," In Proceedings of IEEE INFOCOM, April, 1997.

[73] D. J. Patterson, O. Etzioni, D. Fox, and H. Kautz, "*Intelligent Ubiquitous Computing to Support Alzheimer's Patients: Enabling the Cognitively Disabled*", In Proceedings of the First International Workshop on Ubiquitous Computing for Cognitive Aids (UniCog), 2002.

[74] D. Peleg, "*Time Optimal Leader Election in General Networks*," Journal of Parallel and Distributed Computing, vol.8, no.1, pages 96-99, January, 1990.

[75] C. E. Perkins and E. M. Royer, "*Ad-hoc on-demand distance vector routing*," In Proceedings of Second IEEE Workshop on Mobile Computer Systems and Applications, pp.90–100, IEEE Computer Society, February, 1999.

[76] H. Pucha, S. Das, and Y. Hu, "*Ekta: An efficient DHT substrate for distributed applications in mobile ad hoc networks*," In Proceedings of Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), 2004.

[77] O. V. Ratsimor, D. Chakraborty, A. Joshi, T. Finin, "*Allia: Alliance-based service discovery for ad hoc environments*," In Proceedings of ACM Workshop on Mobile Commerce (WMC`02), September, 2002.

[78] V. Raychoudhury, J. Cao, and W. Wu, "*Top K-leader Election in Wireless Ad Hoc Networks*," In Proceedings of Seventeenth International Conference on Computer Communications and Networks (ICCCN'08), St. Thomas, U.S. Virgin Islands, August 3 - 7, 2008.

[79] V. Raychoudhury, J. Cao, W. Wu, and S. Lai, "*K-Directory Community: Reliable Service Discovery in MANET*," In Proceedings of Eleventh International Conference on Distributed Computing and Networking (ICDCN2010), January 3-6, 2010, Kolkata, India.

[80] O. Riva, J. Nzouonta, and C. Borcea, "*Context-aware Fault Tolerance in Migratory Services*," In Proceedings of Fifth Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous'08), July, 2008.

[81] O. Riva, T. Nadeem, C. Borcea, and L. Iftode, "*Context-aware Migratory Services in Ad Hoc Networks*," IEEE Transactions on Mobile Computing, 6(12):1313-1328, December, 2007.

[82] R. Robinson and J. Indulska, "*Superstring: A Scalable Service Discovery Protocol*

for the Wide Area Pervasive Environment," In Proceedings of the Eleventh IEEE International Conference on Networks, Sydney, September, 2003.

[83] M. Rodriguez, V. Gonzalez, P. Santana, and J. Favela, "*A Home-based Communication System for Older Adults and their Remote Family*," Computers in Human Behavior Journal. Vol. 25. Pp. 609-618. Elsevier Press. ISSN: 0747-5632

[84] A. Rowstron and P. Druschel, "*Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*," In Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Lecture Notes in Computer Science (LNCS), Volume 2218, pp. 329–350, Heidelberg, Germany, November, 2001.

[85] F. Sailhan and V. Issarny, "*Scalable Service Discovery for MANET*," In Proceedings of IEEE PerCom'05, pp. 235-246, 2005.

[86] The Salutation Consortium. Salutation architecture specification version 2.0c, June, 1999, available online at http://www.salutation.org/.

[87] C. Santoro, F. Paternò, G. Ricci, and B. Leporini, "*A Multimodal Mobile Museum Guide for All*," In Proceedings of Mobile Interaction with the Real World, Workshop at MobileHCI 2007, Singapore, Sep 11--14, 2007.

[88] Y. Shi, W. Xie, G. Xu, R. Shi, E. Chen, Y. Mao, and F. Liu, "*The Smart Classroom: Merging Technologies for Seamless Tele-Education*," IEEE Pervasive Computing, vol. 2, no. 2, pp. 47-55, April-June 2003.

[89] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan, "*Chord: A scalable peer-to-peer lookup service for internet applications*," In Proceedings of the 2001 ACM SIGCOMM Conference, pages 149–160, August, 2001.

[90] V. Sundramoorthy, J. Scholten, P. G. Jansen, and P. H. Hartel, "*Service discovery at home*," In Proceedings of Fourth International Conference on Information, Communications & Signal Processing and Fourth IEEE Pacific-Rim Conference on Multimedia (ICICS/PCM). IEEE Computer Society Press, Singapore. pp. 1929-

1933, December, 2003.

[91] K. Takasugi, M. Nakamura, S. Tanaka, and M. Kubota, "*Seamless Service Platform for Following a User's Movement in a Dynamic Network Environment*," In Proceedings of PerCom 2003, pp. 71-78.

[92] G. Taubenfeld, "*Leader Election in presence of n-1 initial failures*," *Information Processing Letters*, vol.33, no.1, pages 25-28, October, 1989.

[93] B. Traversat, M. Abdelaziz, and E. Pouyoul, "*Project JXTA: A Loosely-Consistent DHT Rendezvous Walker*," Sun Microsystems Inc. http://www.jxta.org/project/www/docs/jxta-dht.pdf, May, 2003.

[94] J. Tyan and Q.H Mahmoud, "*A Comprehensive Service Discovery Solution for Mobile Ad Hoc Networks*", In ACM/Kluwer Journal of Mobile Networks and Applications (MONET), Vol. 10 (8), pp. 423-434, August, 2005.

[95] UPnP Device Architecture 1.0, UPnP Forum, December, 2003, www.upnp.org/resources/documents/CleanUPnPDA10120031202s.pdf.

[96] A. Varshavsky, B. Reid, and E. de Lara, "*A Cross Layer Approach to Service Discovery and Selection in Manets*," In Proceedings of Second International Conference on Mobile Ad-Hoc and Sensor Systems (MASS'05), IEEE Press, Washington DC, USA, November, 2005.

[97] S. Vasudevan, J. Kurose, and D. Towsley, "*Design and Analysis of a Leader Election Algorithms for Mobile Ad Hoc Networks*," In Proceedings of ICNP, pp. 350–360, 2004.

[98] S. Vasudevan, B. DeCleene, N. Immerman, J. Kurose, and D. Towsley, "*Leader Election Algorithms for Wireless Ad Hoc Networks*," In Proceedings of IEEE DISCEX III, April 22-24, 2003.

[99] P. Wan, K. Alzoubi, and O. Frieder, "*Distributed construction of connected dominating set in wireless ad hoc networks*," Mobile Networks and Applications, 9(2), 2004.

[100] J. Wu, F. Dai, M. Gao, and I. Stojmenovic, "*On calculating power-aware connected dominating sets for efficient routing in ad hoc wireless networks*," Journal of Communications and Networks, 4(1), March 2002.

[101] J. Wu and M. Zitterbart, "*Service awareness in mobile ad hoc networks*," Paper Digest of the 11th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN), Boulder, Colorado, USA, March, 2001.

[102] S. S. Yau, S. K. S. Gupta, F. Karim, S. I. Ahamed, Y. Wang, and B. Wang, "*Smart Classroom: Enhancing Collaborative Learning Using Pervasive Computing Technology*," In Proceedings of American Society of Engineering Education 2003 Annual Conference, June, 2003.

[103] H. J. Yoon, E. J. Lee, H. Jeong, and J. S. Kim, "*Proximity-Based Overlay Routing for Service Discovery in Mobile Ad Hoc Networks*," In Proceedings of Nineteenth International Symposium on Computer and Information Sciences (ISCIS), 2004.

[104] T. Zahn and J. Schiller, "*MADPastry: A DHT substrate for practicably sized MANETs*," In Proceedings of Fifth Workshop on Applications and Services in Wireless Networks (ASWN), June, 2005.

[105] F. Zhu, M. Mutka, and L. Ni, "*Splendor: A secure, private and location-aware service discovery protocol supporting mobile services*," In Proceedings of the First International Conference on Pervasive Computing and Communication PerCom'03, Pages 235-242, 2003.

[106] F. Zhu, M. W. Mutka, and L.M. Ni, "*Service Discovery in Pervasive Computing Environments*," IEEE Pervasive Computing, vol. 4, no. 4, pp. 81–90, 2005.