A SERVICE-ORIENTED APPROACH TO TOPOLOGY FORMATION AND RESOURCE DISCOVERY IN WIRELESS AD-HOC NETWORKS

by

SERGIO GONZALEZ VALENZUELA

B.Eng., Instituto Tecnológico de Sonora, 1995
M.A.Sc., The University of British Columbia, 2002

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

April 2008

Abstract

The past few years have witnessed a significant evolution in mobile computing and communications, in which new trends and applications have changed the traditional role of computer networks into that of distributed service providers. User mobility has become an increasingly important design consideration, given the abundance of small electronic devices equipped with wireless communications capabilities. In addition, there is an increasing interest in advancing the ad-hoc networking capabilities of portable devices that do not depend on fixed data communications infrastructure.

This thesis approaches the above concerns from two perspectives. First, our research explores an alternative way to form wireless ad-hoc networks whose topologies can be customized as required by the users' software applications. In particular, we investigate the applicability of our solution to networks created by devices equipped with Bluetooth technology. Our method incorporates user-defined policies within the structure of mobile codes to enable service-oriented topologies. Computer simulation results suggest that our proposed approach can achieve this task effectively, while matching the level of efficiency seen in other salient proposals in this area.

This thesis also addresses the issue of service discovery in mobile ad-hoc networks. We propose the use of a directory whose network location varies in an attempt to reduce traffic overhead driven by users' hosts looking for service information. We refer to this scheme as the Service Directory Placement Algorithm, or SDPA. We formulate the directory relocation problem as a Markov Decision Process that is solved by using reinforcement learning, specifically, Q-learning. Performance evaluations through computer simulations reveal the effectiveness of the mobile directory approach with typical bandwidth overhead reductions that

range between 40% and 48% when compared with a basic broadcast flooding approach for networks comprising hosts moving at pedestrian speeds.

In the third part of our thesis, we elaborate on the mobile directory scheme and introduce a multi-directory service discovery system called the Service Directory Placement Protocol, or SDPP. This enhanced approach yields improved scalability when compared to SDPA. We incorporate the aspect of physical-memory availability as part of our investigation in an attempt to better understand the implications of this constraint in mobile computing environments. Our findings reveal bandwidth overhead reductions typically ranging from 15% to 75% in networks comprising slow-moving hosts with restricted memory availability.

In the fourth and final part of this work, we present the design foundations and architecture of a middleware system that called WISEMAN – WIreless Sensors Employing Mobile Agents. We employ WISEMAN for dispatching and processing mobile programs in Wireless Sensor Networks (WSNs). Our proposed system enables the dynamic creation of semantic relationships between network nodes that cooperate to provide an aggregate service. We present discussions on the advantages of our proposed approach, and in particular, how WISEMAN facilitates the realization of service-oriented tasks in WSNs.

Table of Contents

List of Tables

List of Figures

List of Acronyms

ACL	Asynchronous Connectionless Link
AODV	Ad-hoc On Demand Vector Routing
API	Application Programming Interface
BD	Bluetooth Device
CLK	Bluetooth Hardware Clock
DA	Directory Agent
DHCP	Dynamic Host Configuration Protocol
DM	Data Medium rate
DP	Dynamic Programming
DSR	Dynamic Source Routing
FD	Fixed Directory
FHSS	Frequency Hopping Spread Spectrum
HCI	Hardware Controller Interface
HTML	Hyper-Text Markup Language
IETF	Internet Engineering Task Force
IP	Internet Protocol
JXTA	Juxtapose
L2CAP	Logical Link Control Adaptation Protocol
LA	Learning Agent
LAN	Local Area Network
MAS	Mobile Agent system
MAC	Medium Access Control

MANET	Mobile Ad-hoc Network
MC	Monte Carlo
MD	Mobile Directory
MDP	Markov Decision Process
NP	Non-deterministic Polynomial-time
PAN	Personal Area Network
QoS	Quality of Service
RL	Reinforcement Learning
SA	Service Agent
SDP	Service Discovery Protocol
SDPA	Service Directory Placement Algorithm
SDPP	Service Directory Placement Protocol
SERS	Service Entry Ranking System
SFP	Scatternet Formation Protocol
SLP	Service Location Protocol
SMDP	Semi-Markov Decision Process
SOAP	Simple Object Access Protocol
SSDP	Simple Service Discovery Protocol
TCP	Transmission Control Protocol
TD	Temporal-Difference learning
TTL	Time-To-Live
UA	User Agent
UDP	User Datagram Protocol
UPnP	Universal Plug and Play

VF	Value Function
WISEMAN	WIreless Sensors Employing Mobile AgeNts
WLAN	Wireless LAN
WSDL	Web-Service Definition Language
WSN	Wireless Sensor Network
XML	Extensible Markup Language

List of Symbols

$E[s/m]$	Expected slave-to-master ratio
C	Minimized number of piconets in a scatternet
$e(s)$	Error for state s in the Q^* Value Function
R_t	Return of the sum of discounted rewards at time t
γ	Reward discount factor
π^*	Optimal decision policy
V^*	Optimal value function
$V(s)$	State value function
$V^{\pi}(s)$	Value of state s under policy π
$V^*(s)$	Optimal state value function.
$A(s)$	Set of actions in state s
(s,a)	State-action pair
$Q^*(s,a)$	Optimal reward value for state-action pair (s,a)
α	Learning rate of the system
β	Rate of exponential decay
ε_i	Decision epoch i
τ_i	Elapsed time between two decision epochs
$r(s,s',a)$	Reward obtained after a transition from state s to state s' when action a is taken.
$P_{ss'}^a$	Probability of going from state s to state s' when an action $a \in A(s)$ is taken .

Acknowledgments

I would like to express my deepest gratitude to my supervisors: Dr. Victor Leung and Dr. Son Vuong, for their invaluable guidance and support during my stay at UBC. It is not everyday that one has the privilege of being advised by two world-renowned experts in the area of data communications. This achievement could not have been possible without them.

I also want to express my sincere appreciation to the Natural Sciences and Engineering Research Council of Canada, the Canadian Institute for Telecommunications Research (CITR) Network of Centres of Excellence, the Advanced Systems Institute (ASI) of British Columbia, and UBC's Faculty of Graduate Studies for supporting my doctoral research.

I also want to thank the members of my thesis committee for my departmental and university exams: Dr. Clarence DeSilva, Dr. Satish Gopalakrishnan, Dr. Alan Hu, Dr. David Kirkpatrick, Dr. Thomas Kunz, Dr. Lutz Lampe and Dr. Vincent Wong for their insightful comments and suggestions. Finally, I am thankful to everyone, from conference/journal paper reviewers to my lab colleagues, who in one form or another, have helped me to achieve this goal. I would also like to thank Craig Wilson for helping me edit this manuscript.

Sergio González-Valenzuela

To my wife Brenda, for her love and support;

To my son Ruben, for his contagious smile and enthusiasm;

To my parents, for their lifelong teachings;

To my sister, for her words of encouragement when the going gets tough.

Chapter 1. Introduction

1.1 Service Discovery in Wireless Ad-hoc Networks

In this thesis we explore solutions to networking problems in wireless ad-hoc networks from a service-oriented perspective. We investigate how the physical topology of this kind of network can be explicitly manipulated to facilitate the interoperation of users' applications. We also explore alternative means of supporting the efficient dissemination of service-related information among the network's users.

The emergence of new technologies for wireless communications has enabled the seamless interconnection of portable electronic devices to the existing wire-line communications infrastructure, both of which have experienced significant growth during the past few years [1], [2], [3]. The Wireless Local Area Network (WLAN) technology has already played a cornerstone role in this respect by enabling people to effortlessly access localized data and the Internet over moderately large office spaces, commercial establishments, university campuses, and at home. This technology, however, incurs relatively high power consumption and cost, which has motivated the search for alternative solutions that overcome these disadvantages [4].

The proliferation of multimedia-capable devices equipped with distinct types of data communications interfaces has leveraged an ongoing and popular phenomenon known as social networking, whereby users rely on their personal devices to share or exchange digital assets with their peers [5], [6]. To a more limited extent, devices equipped with these communications interfaces may also interconnect on the fly to achieve this same goal regardless of location or the existence of a supporting data communications infrastructure [7]. To further exploit this capability, commonly referred to as service discovery, software applications first need to become aware of its availability in a manner that is transparent to users.

In computer networks, the term *service discovery protocol* (SDP) usually refers to the process of discovering the location of one or more devices providing a certain service. This term has traditionally been employed in the literature when referring to the act of advertising, locating and/or invoking either hardware or software entities that can be shared by users in a computer network. Service provisioning, access and composition tasks have been dealt with separately. In this thesis, we refer to SDP as a process involving both the act of locating services in a network and their advertising methodology, in the spatiotemporal context. We do not concern ourselves with the semantics of encoding information that describes features relevant to these services, such as their availability and access procedures.

From this viewpoint, there are a number of considerations that are of particular interest to our work. The first deals with the location proximity aspect of a service, particularly when the topology of an ad-hoc network is being defined. Traditionally, topology formation protocols for wireless ad-hoc networks do not take into consideration aspects relevant to the applications in the upper layers of the protocol stack. Instead, they focus on techniques that enable quick topology formation, or on methods that provide some Quality of Service (QoS) guarantees for a certain kind of traffic, as discussed later in this chapter. In our proposed approach, we attempt to manipulate wireless connections at will, in order to selectively link service providers and consumers on-demand to facilitate service discovery and provisioning as an individual instance of the current network topology. In particular, we consider the case of Bluetooth scatternets.

The second aspect deals with locating information about existing services in a mobile ad-hoc network, or MANET. Existing SDPs promote the use of one or more directories that act as information repositories of the services present in wired networks. However, researchers in the area often disregard the use of directories in MANETs, since this approach purportedly leads to significant traffic overhead incurred by devices that constantly have to locate services in the face

of a of changing network topology. As explained below, we investigate an alternative approach that relies on a single directory whose location in the network is deliberately altered. This approach enables leveraging service discovery processes in small ad-hoc networks where a limited number of services are available, rather than employing a de-facto broadcast flooding approach to locate services.

We then extend the mobile directory scheme and redefine a collaborative multi-directory system that facilitates the discovery of a larger number of services in conservatively larger ad-hoc networks of slow-moving hosts. Collaborative service discovery systems function much like two-tiered peer-to-peer systems, where a subset of devices is in charge of disseminating and updating service information regularly solicited by the rest of the network hosts. These two-tiered, content-driven, peer-to-peer, decentralized systems in wired networks have been shown to overcome performance limitations that occur when employing centralized approaches [8]. In addition, our work helps to better understand the advantages and disadvantages of the multi-directory approach over MANETs when the majority of devices in the network are constrained in physical memory availability.

Finally, we look again into the applicability of mobile codes technology as a viable candidate for the realization of diverse tasks in Wireless Sensor Networks (WSNs), in particular, the creation of overlay networks. Unlike existing network-management paradigms that promote the individual implementation and execution of the corresponding protocols, we advance the use of a middleware allowing networking tasks to be implemented as instructed by programmable software agents. This concept, known as active networking, has been widely studied in the wired networks realm for several years now, whereas the number of contributions in the wireless field has been more conservative. Our work in this area leads to a better understanding of the potential benefits obtained by deploying programmable codes in WSNs.

The rest of this chapter is organized as follows. In Section 1.2, we present an overview of existing systems for service-oriented networking in wireless and mobile ad-hoc networks. In Section 1.3, we explain our research motivations based on the information given. We outline the main contributions of our research work in Section 1.4, and the overall organization of the thesis is presented in Section 1.5.

1.2 Overview of Related Work

1.2.1 Service-Oriented Bluetooth Scatternets

Bluetooth is one of the most pervasive technologies available that was devised with service-oriented networking provisions in mind. Bluetooth has been the subject of intense research and commercial development in recent years [9], [10], [11]. As an enabler of Personal Area Networks, or PANs, Bluetooth facilitates the interconnection of portable devices featuring this technology, enabling data communication rates of up to 3 Mb/s by forming a *piconet* – a star-shaped network that can accommodate up to 7 devices in active communications. Two or more piconets may further interconnect to form a *scatternet* – a Bluetooth-enabled PAN [12].

The lack of a standardized procedure for the creation of scatternets in the Bluetooth specification has resulted in a number of Scatternet Formation Protocols (SFPs) being proposed throughout the literature. Some of the existing proposals generate scatternets with tree-shaped topologies [13] - [16], while others create mesh topologies [17] - [20]. A SFP that forms ring-shaped networks has also been proposed [21]. However, few, if any, considerations were made in these approaches to ensure that the resulting topology efficiently supports service provision, focusing instead on rapid scatternet formation. This is an important issue since the Bluetooth specification was created with strict service-oriented considerations, particularly those defined by Bluetooth's Service Profiles. To further complicate matters, the existing Bluetooth specification supports service discovery only at the piconet level (i.e., by means of point-to-point

connections). This limitation complicates matters for existing SFPs, since they create topologies that ignore this constraint. As a result, service providers and consumers might end up in different piconets altogether, hindering their subsequent interaction. A number of alternative SDPs for Bluetooth scatternets have been proposed to address this problem [22] - [25].

Outside of Bluetooth, the IEEE 802.15.3 Task Group has been active in the creation of a standard that incorporates application-aware issues for topology formation, though at the link-layer level, to ensure that the QoS requirements of multimedia applications are effectively provided [26] (i.e., focusing on service provision, not discovery).

1.2.2 Single-Directory Service Discovery in Mobile Ad-hoc Networks

The success of MANETs employed as open networks for commercial applications depends on several factors, among which the effective operation of SDPs is crucial in helping to locate available services. The use of a service directory for service discovery in MANETs appears at first counterintuitive, given the potentially high number of network packets generated by hosts when locating the directory in a network of variable topology. For this reason, both pure-flooding and multicasting mechanisms for service discovery have been often favoured, as pointed out in previous research [27], [28]. However, this approach places a heavy bandwidth and energy consumption burden on the network's hosts, especially on those commonly referred to as "thin" devices [28], [30].

Other possible solutions to this problem assume a relatively static network topology relative to the time granularity otherwise considered for MANETs. For instance, the directory-placement problem is directly related to both the facility-location problem and the minimum k-median problem [31], [32] of graph theory. The former seeks to find an optimal location for a service facility in order to minimize the incurred cost of requests generated by consumers in a graph *G,* whereas the latter disregards the costs associated with the creation of a service facility.

Both problems have been amply studied and are well known to be NP-hard [33]. Sub-optimal solutions to both of these problems have already been used for determining the placement of web-server replicas on the Internet [34], whereas a dynamic programming approach has been employed for a similar purpose under the same stable network assumption [35].

1.2.3 Multi-directory Service Discovery in Mobile Ad-hoc Networks

The context-dependent feature of SDPs has led to the involvement in this area of vendor-backed initiatives that promote the use of proprietary mechanisms in an attempt to favour their products. In fact, the most prevalent SDPs are those backed by Microsoft and Sun Microsystems – UPnP and Jini, respectively. On the one hand, the UPnP architecture relies on an orchestrated interaction of existing network protocols, such as XML, HTML, DHCP and SSDP in order to achieve a peer-to-peer system of distributed services [36]. However, UPnP's signalling mechanism would incur an inevitably heavy signalling burden that is detrimental to the power-conservation objective of thin devices in a MANET. On the other hand, devices enabled with Sun Microsystems' Jini system facilitate the dissemination of services information by forming a *federation* [37]. Services are advertised using a leasing scheme, allowing devices to become aware of new services being offered as well as the termination of existing ones. Jini's main shortcoming is the relatively heavy consumption of hardware resources in the devices that use it, along with a non-negligible signalling load placed on the MANET's communications links. IETF's Service Location Protocol (SLP) is another alternative to commercially-backed SDPs, and relies on the use of agent entities that enable clients to discover and access services on-demand [38].

There is also ample academic research on this subject in the literature [39] - [50]. In particular, three studies are relevant to our investigation. In the first, a high-level ontology-based approach is employed in conjunction with a peer-to-peer system that caches service

advertisements in order to selectively forward service discovery queries in MANETs to reduce overhead [48]. However, the effectiveness of this approach depends heavily on whether all MANET hosts support this higher level of information abstraction. In the second study, an unnamed field-theoretic approach is devised [49] for query routing in which the behaviour of the service discovery system mimics that of particles with electrostatic charges in order to direct queries to service providers. The efficiency of the proposed system seems to rely heavily on the accuracy with which the Capacity of Service parameter of service providers is assessed among them. Another approach, based on directories and service advertisements (i.e., a "push" model) for MANETs, has been presented in [28]. This scheme relies on a virtual backbone and surrogate hosts for the dissemination of service-related information and processing of queries, in accordance with a set of predefined heuristics with a more neutral approach that could be conceivably adapted to work with existing SDPs. Another approach, based on social models of host cooperation and trust that incorporates security issues, has been proposed [50].

1.2.4 Mobile Agents for Wireless Sensor Networks

Advances in signal processing, computing and radio technologies have enabled the realization of WSNs for sensing, measuring and distributing information for a wide range of applications, including: monitoring of environment phenomena and industrial environment settings, first-responder support, and defence [51] – [53]. In these applications, the orchestrated cooperation of nodes in a WSN effectively turns the system into a distributed service provider. A number of issues pertaining to the design and implementation of WSNs have already been identified in the literature, among which energy-efficient design remains by far the hardest to solve [54]. A great deal of research has therefore focused on new communications schemes that attempt to reduce the amount of energy consumed by a WSN node per bit of transmitted data.

The use of mobile code has gained significant attention as a plausible alternative to addressing energy-conservation issues in WSNs, in which compact programs (interpreted or binary) are distributed to the WSN nodes. The value of using code mobility is twofold. First, mobile codes allow the network to be conveniently re-tasked according to the users' needs. Second, a programmable approach enables the data computation elements of an application to be relocated to sites where relatively large amounts of data are collected, enabling potentially high energy savings in [55], [56].

A number of Mobile Agent Systems (MAS) for WSN have been proposed to date. For instance, MAS like Maté [57], Impala [58] and Deluge [59] enable network programmability by relying on code mobility in the form of binary images distributed throughout the WSN. We refer to this approach as hard re-tasking, whereby the WSN nodes' physical memories are flushed and provided with newer code, possibly aimed at supporting a different or improved service. On the other hand, systems like SensorWare [60], SmartMessages [61] and Agilla [62] have the ability to re-task the network by injecting code scripts. In these systems, the WSN nodes' program memories are not flushed; only the interpreted code that defines the current application is replaced. We refer to this approach as soft re-tasking.

Hard re-tasking approaches employing native code are usually expected to execute faster than their soft re-tasking counterparts, since the latter rely on interpreted code. On the other hand, the injection of mobile code implies a simpler way to re-task the network, while enabling potential energy savings through the re-location of the data-processing element. This convenience comes at a price, as code interpreters need to be efficiently designed in order to occupy the least possible amount of memory space, while maintaining robustness. As for code compactness, Impala, SensorWare and SmartMessages are designed to run over devices with

richer hardware resources, while Agilla, Deluge and Maté are specifically designed with stringent hardware considerations in mind.

1.3 Research Motivations

As explained in the previous section, several schemes have been proposed at different levels and in different contexts to tackle the issue of service discovery in wireless ad-hoc networks, ranging from link layer optimizations, to the efficient localization of service-related information. One of our motivations for this work is driven by the need to learn from the application-awareness/service-oriented aspects of topology formation. In the case of Bluetooth scatternets, existing approaches focus almost entirely on the manipulation of the device-pairing procedure geared towards rapid connection setup. Yet, it is unknown whether these SFPs produce topologies that effectively support the applications that users might want to run. In any case, vendors of Bluetooth-enabled devices might want to be able to set up scatternet topologies that favour their proprietary devices when used collectively for any given purpose. Another motivation is that Bluetooth's own SDP was designed to work with point-to-point connections, and not in the scatternet mode. Therefore, we would like to be able to set up scatternets by directly linking service providers with consumers whenever the interested parties are in close proximity. Moreover, we note that a number of unrealistic assumptions have been commonly adopted in order to simplify the inherent complexity of the proposed SFPs. We would like to formulate a scatternet formation scheme that takes into account more realistic settings within a service-oriented context.

We also investigate the issue of service discovery in MANETs. While there are indications of improved packet overhead efficiency in schemes promoting ontology and semantic information processing, their estimated computational overhead is largely unknown. In this regard, the high-level programming language (i.e., Java) used therein is well known to be

computationally more intensive than those that run natively at lower levels. This issue raises the question as to whether the amount of energy these schemes save at the network layer actually offsets whatever amount of extra energy consumption they incur by processing information in this fashion. This factor is important to consider since these schemes are assumed to run on small portable devices. The same arguments apply to schemes that employ the JXTA framework [63], [64], which is also commonly known to be computationally more intensive. For our work, we investigate enhancements that can be viably incorporated into existing popular frameworks in order to leverage their performance, as opposed to creating a new one from the ground up. In particular, we would like to employ functionalities readily available in these protocols to reduce their signalling overhead. We develop the idea of reducing traffic overhead due to broadcast flooding by bringing service information closer to its consumers, as reported before [65], [66].

In addition to the above considerations, our work is also motivated by the need to consider the impact that hardware limitations in portable devices have on the proposed system's performance when employing two-tiered architectures for service discovery in MANET. Specifically, we are interested in investigating the impact of having limited amounts of memory available to the SDP system in small user devices, unlike the usually abundant amount of memory found in desktop computers. If overlay networks are employed to distribute service-related information over these types of devices, then we would like to know the best policies that the system should employ in order to distribute information in a resource-conscious manner. Our survey of the existing literature reveals that the implications of incorporating memory constraints into decentralized service discovery systems have been completely overlooked, which motivates this part of our thesis. We are also interested in understanding the impact that user mobility and the dynamics of service discovery processes have on traffic overhead.

Finally, our work on MAS for WSNs is motivated by the need to create a system that fulfills different but complementary needs in these kinds of systems. First, we are interested in creating an agent-based system that supports soft re-tasking, and favours flexibility in the coordination of distributed tasks over fine-grained control of functionalities in the WSN's nodes. In fact, the case for code mobility in the form of agents for WSN deployment hardly holds if the task of these agents is rather deterministic, or requires a minimal degree of flexibility. The use of mobile agents in WSN is therefore justified by the need for flexibility in the evolution of a distributed process. In addition, we are interested in creating a system with a simple architecture, whose language constructs enable programmability of diverse network tasks.

1.4 Technical Contributions

We summarize the technical contributions of our research as follows:

- **Programmable Agents for Efficient Topology Formation of Bluetooth Scatternets:** We reconsider some of the assumptions made by the proposed SFPs in the literature, and aim at designing an efficient SFP that addresses the scatternet formation problem from a completely different perspective. In particular, we make use of compact programs that are able to relocate themselves in a highly controlled fashion and employ available resources at the local host in order to configure scatternets on-demand. We present our proposed architecture for processing mobile codes in Bluetooth-enabled devices, and describe the design of a compact program that can be forwarded in a single packet at the link layer of the Bluetooth protocol. We then evaluate the effectiveness of the proposed approach, which is comparable to existing schemes amply cited in the literature. However, our approach has the added benefit of being programmable, thus enabling the scatternet architect to embed into the mobile program's structure as needed the adequate policy for configuring the scatternet's topology.

- **A Mobile-Directory Approach for Service Discovery in MANETs**: We present the Service Directory Placement Algorithm (SDPA), a directory-placement scheme that leverages the performance of existing service discovery protocols over wireless ad-hoc networks. SDPA promotes the deployment of a nomadic service directory whose location in the network varies according to the dynamics of service-discovery queries driven by the users' applications and by partial knowledge of the network's topology. SDPA uses a heuristic approach, whose performance is optimized by formulating the directory-placement problem as a Semi-Markov Decision Process solved by means of a reinforcement-learning technique known as Q-Learning. Performance evaluations obtained through computer simulations reveal average bandwidth savings of close to 50% over a broadcast approach for service discovery, once an efficient directory-placement policy is obtained.
- **A System of Mobile Directories System for Service Discovery in MANETs:** We propose the Service Discovery Placement Protocol (SDPP), a scalable service discovery scheme that relies on a multi-directory system for efficient service discovery in wireless ad-hoc networks. SDPP builds on SDPA's framework, and promotes the directory duplication of fixed service providers according to the current service discovery dynamics. However, we also take into consideration the issue of limited memory availability in user devices, in an attempt to mimic the overall system's inability to support full service information exchange when a relatively large number of service providers are available in the MANET. Hence, we incorporate into SDPP a simple Service Entry Ranking System (SERS) that helps to identify service information entries whose spatiotemporal popularity can be employed to prioritize directory duplication, in an attempt to achieve better system performance. Computer simulations reveal typical traffic reductions resulting from SERS that range from 15% to 75% in moderately large MANETs operating under distinct conditions.

- **A Mobile Code Platform for Service-Oriented Tasks in Wireless Sensor Networks:** In this final part of our thesis, we introduce the architecture and design foundations of WISEMAN for its deployment in WSNs. Key features of our system include an ultra-compact language construct that embraces high-level control of repetitive tasks inherent to this type of networks, as well as the ability to support semantic navigation of mobile codes. Our approach derives from an earlier system originally designed for wired networks unconstrained by power resources. We outline the changes made to the original system needed to support its implementation over WSN devices. We also describe its implementation as a computer simulation, with the objective of identifying potential problems and improvements to the system.

1.5 Organization of the Thesis

The rest of this thesis is organized as follows. In Chapter 2, we introduce a framework that employs mobile codes for the formation of Bluetooth scatternets, and present the methodology employed to reconfigure scatternet topologies. We then discuss performance results and present comparisons against existing work. In Chapter 3, we describe an enhancement to SLP that employs SDPA in order to leverage the performance of the service discovery system. We describe the heuristics behind the use of a mobile directory, and the use of a reinforcement-learning technique to fine-tune the performance of the proposed system. We then discuss results obtained through computer simulations. In Chapter 4, we describe our multi-directory approach to service discovery in MANETs. The methodology behind our SERS is presented, along with its incorporation into SDPP. Extensive computer simulation results are presented and discussed. In Chapter 5, we present WISEMAN's architecture and language constructs, and discuss qualitative aspects of our approach and implementaton. Chapter 6 presents our conclusions, accompanied with some suggestions for future work.

Chapter 2. Programmable Agents for Efficient Topology Formation of Bluetooth Scatternets*

This chapter describes the advantages obtained by applying mobile code technology to the formation of Bluetooth scatternets [67]. In particular, we describe a methodology that decouples the Bluetooth device discovery procedure from the scatternet formation process, facilitating the association of service providers and consumers. This is an important consideration that existing scatternet formation protocols have failed to address, given that the Bluetooth's service discovery operation requires physical proximity of the corresponding devices. In fact, all of the surveyed SFPs for Bluetooth rather aim at improving topology formation time by manipulating the device discovery process. In addition, the existing discovery mechanism for Bluetooth does not make provision for service context awareness. We address this issue from a high-level perspective through the use of mobile code technology. Performance evaluations show that our proposed scheme achieves the same efficiency seen in other salient schemes, while fulfilling the need to provide a flexible solution that enables device association oriented to services.

We begin, in Section 2.1, with a concise overview of the Bluetooth technology for a better understanding of how devices establish a wireless connection. In Section 2.2, we offer a short discussion on the shortcomings observed in existing SFP proposals and the assumptions they are based on. We then discuss the applicability of the mobile code paradigm to the scatternet formation problem in Section 2.3. In Section 2.4, we describe in detail the implementation of our proposed solution, followed by a protocol overhead analysis in Section 2.5. We then present and discuss the results of our simulations in Section 2.6, which also include

* This chapter is based on a paper to appear in the International Journal of Wireless and Mobile Computing, Inderscience Publishers [67].

a comparison of our work to existing ones. We conclude this chapter with remarks and suggestions for future work in Section 2.7.

2.1 Networking With Bluetooth-Enabled Devices

At the radio level, Bluetooth Devices (BDs) gain access to the wireless medium by means of the Frequency-Hopping Spread-Spectrum (FH-SS) technique, which is widely regarded for its superior bandwidth efficiency and noise immunity [68]. Allocated bandwidth is divided into smaller partitions that each BD accesses cyclically in a pseudo-random fashion, in which BDs arbitrarily assume either the *INQUIRY* or *INQUIRY SCAN* state with the purpose of becoming aware of their mutual existence. During this initial phase, the device assuming the INQUIRY state hops rapidly in the allocated frequency spectrum in comparison to the device that assumed the INQUIRY SCAN state. As time elapses, BDs eventually find themselves in the same instantaneous operating frequency and in complementary states. This event leads to the exchange of preliminary control data just before they switch into the *PAGE* and *PAGE SCAN* states, respectively, in which both BDs become time-synchronized prior to their entering the CONNECTED state. The device that initially assumed the PAGE state then takes on the role of *master*, and the other assumes the role of *slave*.

Master and slave roles are merely logical, and depend exclusively on the states that the respective BDs assumed at the beginning of the discovery phase explained before. The connection set-up delay may take up to 10 seconds, depending on the number of devices simultaneously accessing the wireless medium [13]. In addition, devices equipped with class-B transmitters will only be able to connect to other devices located at distance no greater than 10 metres [12]. Once in the CONNECTED state, BDs may momentarily enter the INQUIRY/PAGE states in order to discover additional devices in the proximity. Furthermore, two or more piconets

may interconnect to form a scatternet through bridge (slave) BDs, whose presence at each piconet is shared in a time-division basis (as exemplified in Figure 2.1).

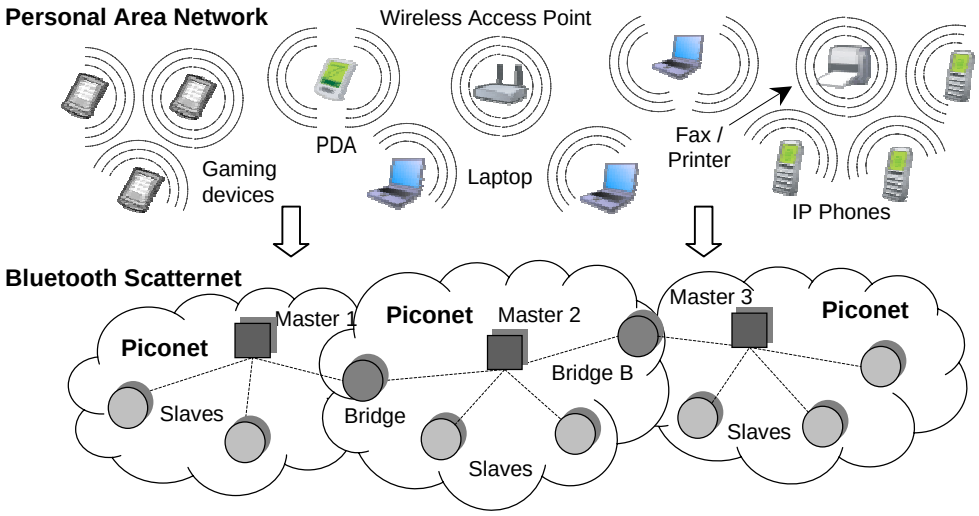

Figure 2.1 Mapping of a personal area network to a Bluetooth scatternet.

2.2 Review of Assumptions Made by Existing SFPs

The lack of a standardized procedure for the creation of scatternets in the Bluetooth specification has resulted in a number of SFPs being proposed throughout the literature. As mentioned before, some of the existing proposals generate scatternets with tree-shaped topologies, while others create meshed topologies. Despite the lack of predefined rules or procedures for such purpose, a number of assumptions are commonly adopted in order to simplify the inherent complexity of the proposed SFPs. Three of common assumptions made by existing SFPs are:

- BDs that participate in the protocol are all located within radio range.
- A BD cannot join a scatternet once it has been already formed.
- BDs are able to synchronize the time at which their participation in the SFP begins.

If we consider the distinct scenarios in which Bluetooth-enabled PANs are most likely to occur, then it becomes clear that the previous assumptions are inconsistent with the very purpose

that such portable electronic devices were originally created for. SFPs adhering to the first of these assumptions become unable to support the creation of PANs at venues spanning areas larger than the 10-metre radio range constraint. The second assumption clearly contradicts the natural flow of events of user-oriented PANs by disallowing people to spontaneously join or leave an existing network. Finally, unless BDs employ external means to synchronize the time at which they begin to participate in the formation of a scatternet exist, the third assumption would hardly hold in a realistic PAN scenario. Indeed, it is an instance of the communications paradox that BDs would be able to realize the desired synchronization, given that the purpose of creating a scatternet is to establish communications links among them in the first place. These and other limitations observed in existing SFPs motivated our considering the use of mobile software agents as a suitable solution to the scatternet formation problem as explained in the next section.

2.3 Mobile Agent Applicability for Wireless Network Management

Existing SFPs employ traditional communications models such as message passing and client-server to achieve their goal. In essence, message passing is employed during the initial BD discovery phase when connections are being established. They then choose leader nodes that take on the role of servers that coordinate the distributed execution of the protocol, with the rest of the BDs behaving as clients (e.g., [13], [18], [69]). However, we argue that these communications models limit the flexibility of the proposed SFPs and have a clear influence over the assumptions behind them in order to keep the problem tractable.

Contrary to the communications models described above, we employ encapsulated software entities known as mobile software agents to perform a concise task on behalf of the user. These agents may act individually, or may be instances of a larger process when they cooperate with other agents to accomplish a distributed task. A relatively large number of systems and applications that support the use of mobile agents exist, most of which are based on

the Java programming platform. In our case, we promote the *Wave* system as an efficient tool for implementing and deploying mobile processes given its enhanced mechanisms for distributed control and spatiotemporal coordination of software agents [70]. Wave's architecture helps reduce much of the management overhead observed in other agent systems, while its similarity to scripting languages offers code compactness not seen in other mobile processing platforms, making programs easier to modify [71]. This latter feature is of foremost importance since it helps to reduce the amount of bandwidth overhead that wireless ad-hoc network management protocols incur. We have previously reported efficient Wave-based solutions to wire-line network management issues [72].

2.4 On-Demand Scatternet Formation Through Mobile Processing

In this section, we present *BlueScouts* – our proposed approach for scatternet formation based on the use of mobile programs that enable service-oriented topologies [73]. First, we define our assumptions for the formulation of our protocol:

- BDs are powered up sequentially according to an arbitrary (non-batched) arrival process.
- BDs possess a basic data forwarding mechanism to their one-hop neighbours.
- BDs need not be within radio range of every other BD to form a scatternet.
- Existing BDs in a piconet devote a certain amount of time discovering new BDs once they have entered the CONNECTED state.

From the last assumption, we note that master BDs become potentially busier than their slave counterparts as piconets and scatternets grow in size. Therefore, it is reasonable to expect master BDs having limited spare time to discover new BDs attempting to join an existing scatternet. As a consequence, new BDs are most likely to be discovered by existing slaves, which force them into assuming master roles. Clearly, new BDs assuming the role of master translates into the arbitrary creation of new piconets, leading to larger scatternet diameter, increased average path

length, and longer data forwarding delays. For simplicity, we follow the tacit consensus of existing SFPs and aim at minimizing the number of piconets in a scatternet. However, as explained later, this need not be a fixed scatternet policy. Let *s/m* denote the slave-to-master ratio of any given piconet in a scatternet *S*. Our objective is to maximize this ratio in every piconet P_i in order to minimize the number of piconets. The maximum value of *this* ratio is obtained by allocating as many slave devices as can be handled by the respective master device, which is limited to a maximum of 7 slave devices, as defined in the Bluetooth specifications.

2.4.1 Reconfiguration Attempt at the Local Piconet

To describe the details of our protocol, we employ a sample situation in which users with portable electronic devices arrive at a certain venue according to our previous assumptions. Having assumed the master role as per our previous assumptions, this BD attempts to reconfigure its role to become a slave. To achieve this, a *wave agent* is dispatched in an initial intra-piconet reconfiguration attempt, as seen in Figure 2.2(a). This wave agent, carrying both the current values of the hardware clock (CLK) and Bluetooth device address (BD_ADDR), is forwarded by the same slave node that discovered the new BD, herein referred to as anchor node, to its own piconet master. The agent hops from M_2 to M_1 via S_2. At M_1, the agent verifies whether an additional slave BD can be accommodated in the current master node's schedule. If so, the agent employs the already-existing Application Programming Interface (API) of the local host in order to access its hardware through the Host Controller Interface (HCI). By doing so, the agent instructs the local host (M_1) to temporarily enter the PAGE state in order to probe the new BD for radio proximity.

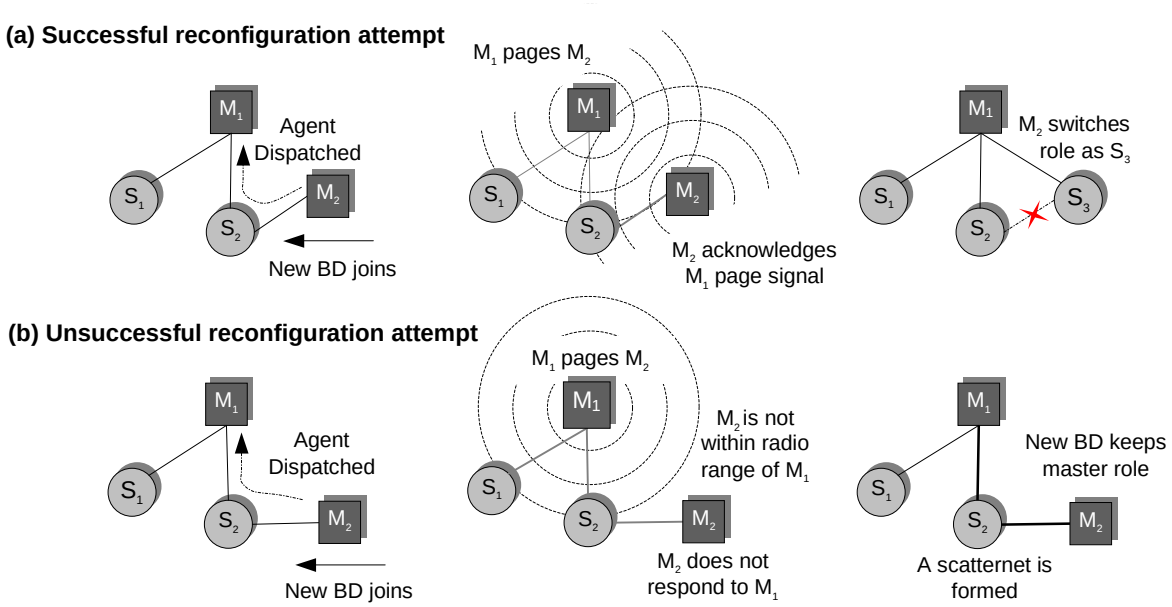

Figure 2.2 Reconfiguration attempt at the local piconet.

Knowledge of the new BD's internal clock and the BD_ADDR carried by the agent allows the calculation at M_1 of the current operating frequency (and subsequent ones) of M_2 [12]. However, for this to be possible, the agent must instruct the new BD (M_2) to enter the PAGE SCAN state just before hopping to the slave node (S_2). A message received in response to the paging signal indicates immediate radio proximity between the new BD and the existing piconet, in which case the new BD switches to a slave role and the BDs establish a connection. Finally, the agent echoes a termination message back to the new BD via the slave BD before being discarded. This event instructs the new BD to disconnect from the slave node to avoid creating a loop.

Sharing adjacent physical space might become difficult as the number of people in a meeting venue increases, making it impossible for all BDs to be within immediate radio proximity of each other. Therefore, a failed intra-piconet reconfiguration procedure causes the new BD to keep its initial master role, leading to the creation of a new piconet. In this case, a secondary process is launched from the anchor BD to create an overlay network. This network

functions as a logical corridor that subsequent agents will use to reach other master nodes that may be able to accommodate new BDs that are found. We refer to this overlay network as the scatternet backbone, which is formed by piconet masters linked through bridge (slave) nodes. Conversely, master-slave links are regarded as leaf links. This logical backbone also prevents unnecessary waste of time and bandwidth, as it guides agents only through paths leading to master BDs. In other words, there is no use in having agents migrate to leaf (slave) nodes to probe radio proximity. An unsuccessful reconfiguration attempt is also depicted in Figure 2.2(b), where backbone links are shown bold and leaf links are shown thin.

2.4.2 Reconfiguration Attempt at Remote Piconets

As mentioned before, the reconfiguration procedure can be extended by performing a scatternet-wide search at remote piconets. However, if agents are arbitrarily dispatched, then they might cause PAGE signals to collide if two or more agents concurrently instruct the BDs they reach to enter this state. To overcome this problem, agents are sequentially dispatched from the current node over the logical backbone using a depth-first technique as shown in Figure 2.3. This process is automatically performed by Wave interpreters as instructed by the language constructs of the corresponding agent. The agent injected by the new BD initiates a depth-first distributed search process in which agents are cloned and activated in a tokenized fashion to traverse the Bluetooth scatternet and probe for radio proximity as recently explained. The desired coordination is achieved by having Wave interpreters issue the necessary signals that notify the preceding interpreter in the corresponding BD of either a successful or failed outcome of the current search stage. Failure signals lead to the continuation of the search process, whereas a successful radio probe distributes the necessary process completion signals.

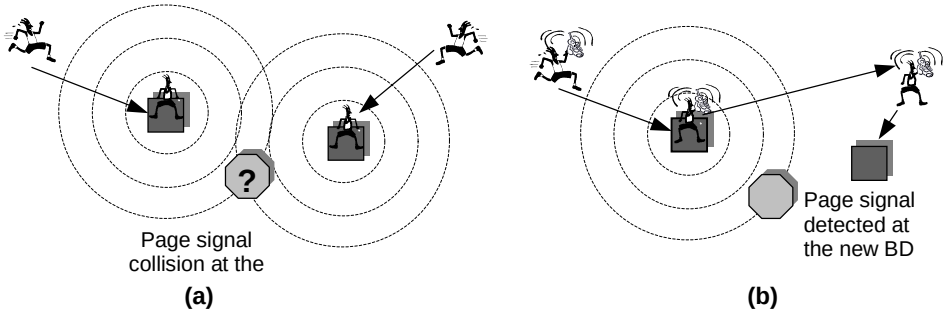

Figure 2.3. Collision avoidance of paging signals through spatial agent coordination.

At the current stage, the next hop is chosen according to the BD's lookup table of one-hop neighbours, which is updated every time new BDs are discovered. If the propagation from the current master BD is no longer possible (e.g., when only leaf links remain), then the mobile process backtracks to the preceding BD and continues over the next unexplored path. The search process stops either once a suitable candidate has been found, or after the totality of the masters has been probed. In the latter case, the new BD keeps its initial master role, and the scatternet backbone is extended as previously described. The corresponding algorithm shown in Figure 2.4 has $O(V + E)$ time-complexity, where V equals the number of BDs visited, and E is the number of links that connect them.

```
Hop to anchor node
For all backbone links in the current node {
   DepthFirstSearch {
      Execute the following in sequence {
          Hop to next neighbour node
          Probe new BD for radio proximity
          If success, terminate search }
      While next entry in table not empty
          DepthFirstSearch   }
}
Hop back to anchor node
If search was successful {
   Hop back to new BD
   Tear down initial link
   Finalize setting up new link }
Else, label link to new BD as backbone
```

Figure 2.4 Pseudo-code for the Bluescouts.

Our agent-based protocol creates tree-shaped scatternets that grow in an organic-like fashion as new BDs join the existing network topology. Figure 2.5 shows an example of a

reconfiguration attempt as a new BD joins a scatternet, wherein an existing slave S_3 discovers a new BD that takes on the role of master (M_4). A failed reconfiguration attempt at the local piconet's master (M_1) leads to the scatternet-wide search, in which M_3 is finally able to accommodate the new BD, triggering a master-to-slave role change. Finally, any BD that unknowingly becomes disconnected as a result of another BD leaving the scatternet must trigger the reconfiguration process again. Interrupted data transfer sessions would have to be dealt with by higher layer protocols dedicated to routing and reliable data transport.

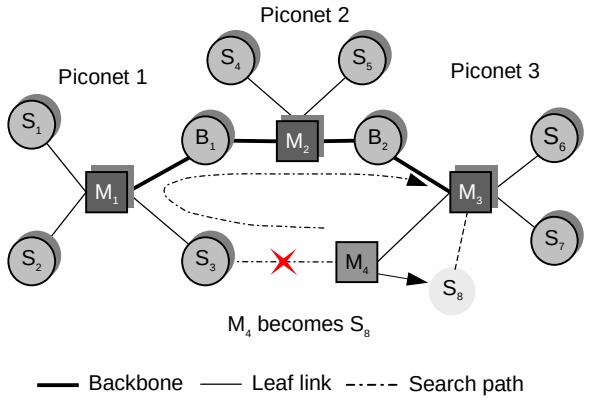

Figure 2.5 Reconfiguration attempt at remote piconets.

2.5 Protocol Overhead Analysis

Mobile code interpreters, or Wave Interpreters (WIs), can be placed on top of the L2CAP layer of the Bluetooth protocol stack as an enabling middleware for wave agents to be directly carried by L2CAP packets. The size of the wave packet being transmitted may vary slightly, depending on whether an echo signal or agent codes are being forwarded. In either case, this packet amounts to no more than 28 bytes [74]. Taking into consideration an agent's codes, its program variables, the BD clock (CLK) and hardware address (BD_ADDR) values carried, the total payload of the L2CAP packet amounts to 202 bytes. Including the 2-byte header, the L2CAP packet would amount to 204 bytes at maximum, fitting within the limits of a single DM5

ACL baseband packet [12]. The agent can therefore be transmitted in a one-shot fashion, effectively reducing the forwarding delay experienced at piconet masters and inter-piconet bridge nodes. However, according to the architecture of the Wave system, agents are forwarded in two steps. The first segment of the wave agent, known as the head, is forwarded first to ensure that the purported next hop is actually reachable. A successful hop of the agent's head then triggers an echo signal from the node reached, prompting the agent's tail forwarding from the previous node in order to resume execution at the new node. The agent's tail size also requires a DM5 packet-type to be forwarded, whereas the head can be transmitted in a single DM3 packet-type. Based on this data, we now examine the factors accounting for the total time delay that the mobile process incurs, also illustrated in Figure 2.6:

- Inter-piconet switching time of bridges (D_{SW}) and the new BD (2 slots = 2 × 625 μs).
- Transmission time (D_H) of a DM3 packet carrying the agent's head (3 slots = 3 × 625 μs).
- Transmission time (D_T) of a DM5 packet carrying the agent's tail (5 slots = 5 × 625 μs).
- Transmission time (D_E) of a DM3 packet carrying a Wave echo (3 slots = 3 × 625 μs).
- Paging delay (D_P) at visited masters (4 slots = 4 × 625 μs).

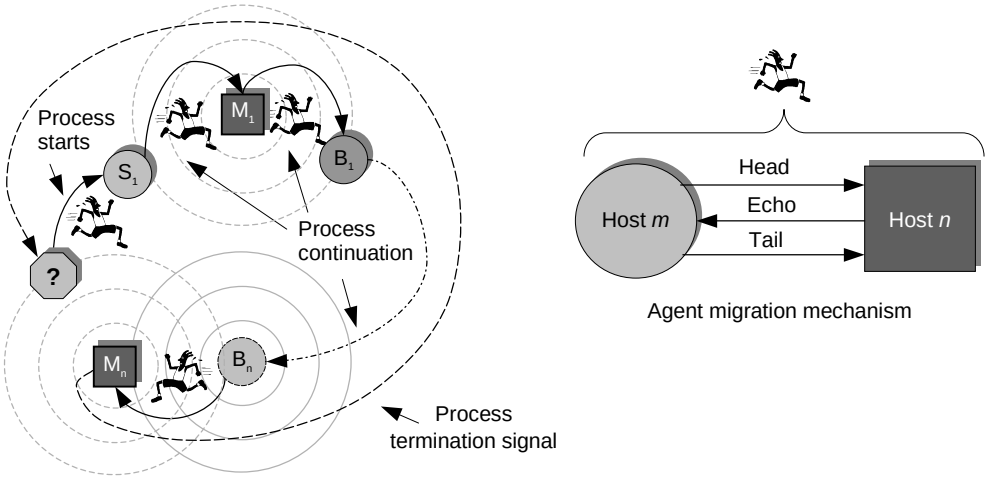

Figure 2.6 Recursive propagation of agents and process evolution-state echoes.

Thus, we see that the total reconfiguration delay D_R of the process is equivalent to the combined sum of the piconet membership switching delay, the data transmission delays for every agent's head, tail and echoes, and the delay incurred by masters during the new BD paging attempt. This can be formally quantified as:

$$D_R = nD_{SW} + 2nD_H + 2nD_T + 4nD_E + D_P \tag{2.1}$$

where n represents the number of piconets searched. The processing time incurred by the wave agent is considered negligible. We also assume that the scheduler would give the maximum transmission priority to the wave agents and immediately pre-empt any other type of packet.

2.6 Simulation Results

We now discuss the outcome of our computer simulations, which are based on an experimental version of the Wave system for Linux. BDs are individually powered on at a random coordinate following a uniform distribution for areas of 10, 20, and 40 square metres. This helps to obtain a more insightful view into the mechanics of the scatternet formation process, providing a more realistic scenario. This is in contrast to existing systems that assume BDs powering up at the same time. Results depict averages of 50 simulation runs per test area for up to 200 BD arrivals, which we consider to be a reasonably large number of devices for a Bluetooth scatternet. Direct comparisons of our scheme to existing SFPs that also create tree topologies are discussed where possible.

In accordance to the protocol's run-time complexity, we observe in Figure 2.7 a nearly linear trend on the average number of piconets probed during our agent-based, depth-first search process. The average number of piconets obtained is also similar to the one seen in [13]. We estimate a lower bound with respect to the total average reconfiguration delay experienced by a new BD switching from master to slave role, as seen in Figure 2.8. We also observe in Figure 2.9 how deploying a scatternet on different test areas affects the average slave-to-master ratio (*s/m*),

where the largest fluctuations occur on scatternets that are initially formed in smaller areas. The larger the area is, the smaller the number of slaves that a master hosts, which converges asymptotically to a given value. We also note that, since scatternets deployed in smaller areas yield a higher *s/m*, it is harder to find a piconet with available space to accommodate a new BD. Therefore, it takes less time to reconfigure a BD scatternet formed in a larger area than one deployed in a smaller area. The proposed schemes in [13]-[16] do not report *s/m* ratio performance results with test areas of distinct dimensions.

Figure 2.10 depicts a logarithmic growth of the scatternets' diameter, adhering to the characteristics of tree-shaped graphs. It can also be seen that, on average, the size of the physical area where a scatternet is deployed has little effect on its diameter. An intuitive explanation for this is that only a few BDs having assumed master roles are initially required to cover a small venue in its entirety, diminishing the need for new BDs taking on a master role in order to accommodate new slave BDs, which in turn yields a higher *s/m* ratio. In addition, we note a linear growth on the number of piconets as new BDs arrive, while a consistent difference of around 5 piconets between the 10 m^2 and 40 m^2 scenarios can be seen once scatternets have reached a size of around 60 BDs as seen in Figure 2.11. This difference can be considered as being relatively small since a 40 m^2 area is actually 16 times larger than one measuring 10 m^2. Similar performance results regarding diameter and number of piconets was obtained in [13], but the effect of running their SFP in areas of different dimensions is unknown.

Figure 2.12 illustrates the estimated average consumed bandwidth during the reconfiguration process of a new BD, which also depicts a linear growth. However, this value decreases as scatternets form in larger areas. Given that larger areas yield a smaller *s/m* ratio, the chances of finding a piconet able to accommodate a new BD are higher. This in turn reduces the number of piconets that need to be searched, thus reducing consumed bandwidth.

Unfortunately, bandwidth overhead has not been accounted for in any of the existing approaches. Results on the number of packets sent by the proposed protocol were reported in [13], although the actual packet size is unknown. In any case, the amount of bandwidth incurred by our approach can be considered reasonable small, which is an advantage of employing mobile agent technology efficiently.

2.7 Summary

We have presented a novel scatternet formation protocol based on mobile processing technology. This protocol allows a gradual organic-like growth of the scatternet as users arrive, and scales well to moderately-sized scatternets. Our solution uses a programmable system that facilitates the deployment of mobile codes that achieve the scatternet reconfiguration task. Our protocol executes in a completely distributed and asynchronous fashion, enabling failure survivability by eliminating the need of concurrent protocol execution. We have revisited some of the assumptions commonly made by other protocols and have replaced them with more realistic ones. The importance of studying the performance impact that an area on which a scatternet is deployed has on a scatternet formation protocol has also been stressed. We provide simulation results of that are comparable to previous protocols despite our employing tighter constraints.

Bluescouts enables service-oriented configuration of scatternet topologies, since agents can be explicitly programmed to establish direct connections between service providers and consumers as needed, effectively facilitating service provision, as portrayed in Figure 2.5. For instance, a BD near a printer can launch a reconfiguration process that: (1) disconnects from one particular device that initially discovered it, and (2) sets up a new connection to the service provider's piconet. In this case, it is assumed that service providers would initially take on master BD roles to subsequently accommodate up to 7 slaves wishing to use its service

concurrently. In this regard, agents can be programmed as needed in order to favour effective service discovery and provision.

Finally, the main advantage of employing mobile codes over simple message passing is that the former enables open programmability of distributed process coordination, whereas the latter approach entails limited flexibility. In other words, in the message passing approach, the corresponding interpreter/processor parses and interprets the information contained therein in order to execute a well-defined task. In the agent approach, the local interpreter obtains the necessary instructions from the language constructs that define the mobile codes, making the system much more flexible.

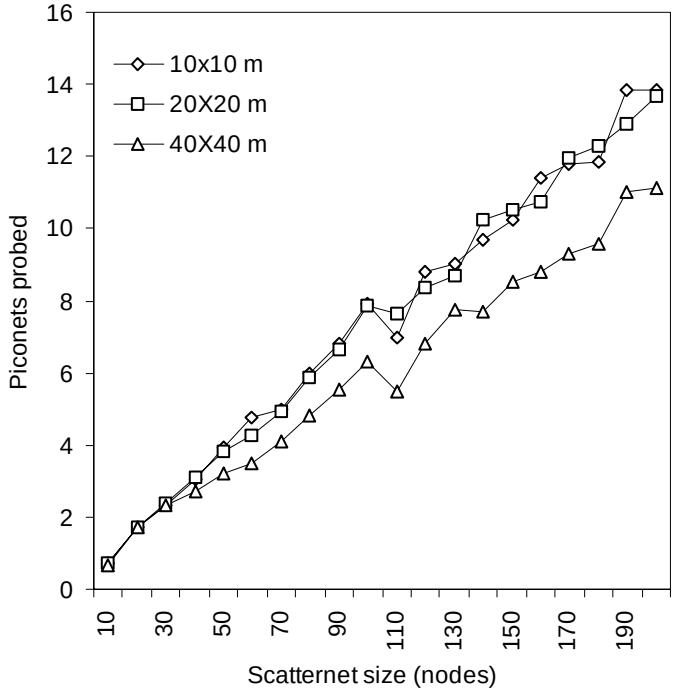

Figure 2.7 Number of piconets probed

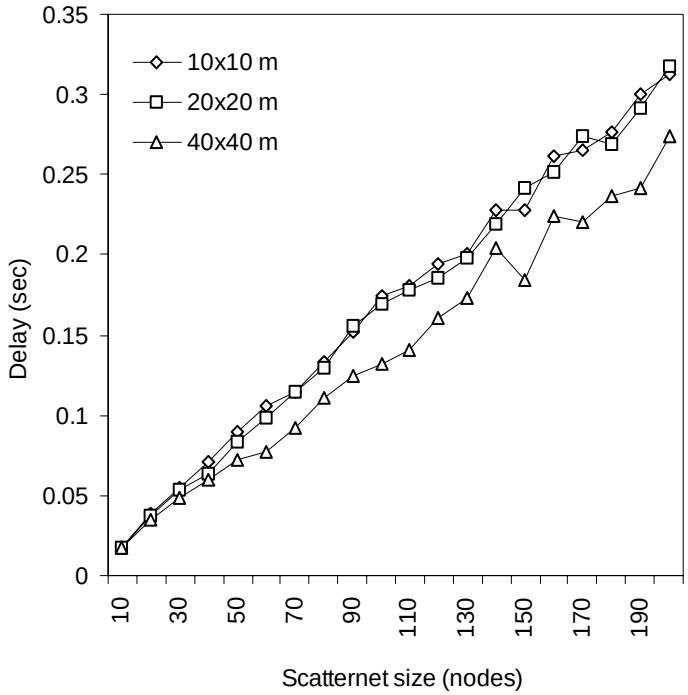

Figure 2.8 BD-role reconfiguration delay

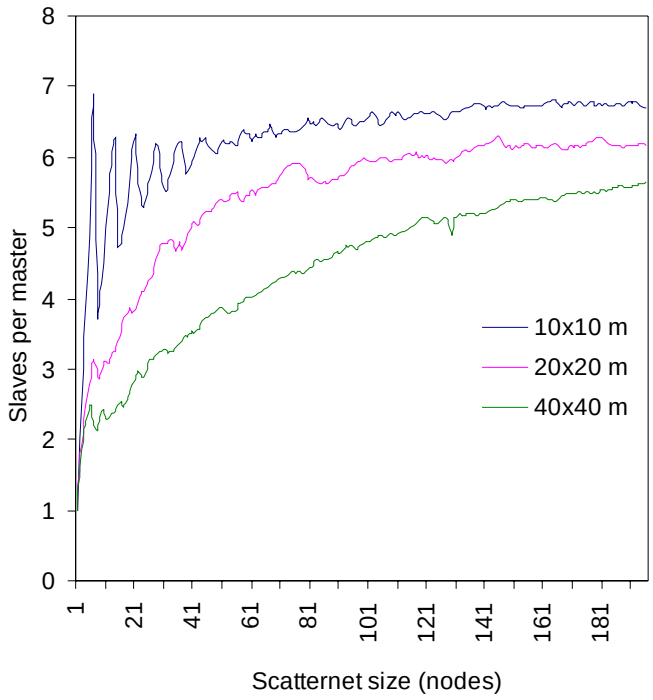

Figure 2.9 Slave-to-master ratio

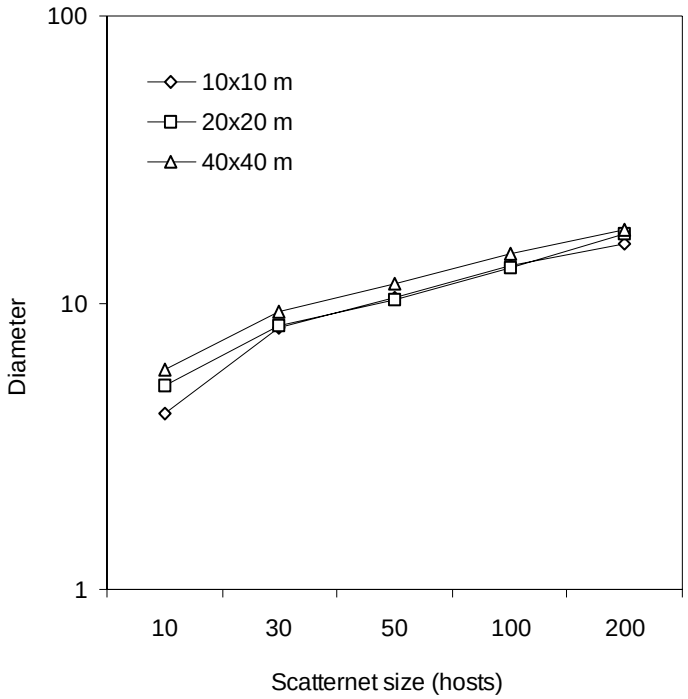

Figure 2.10 Scatternet diameter

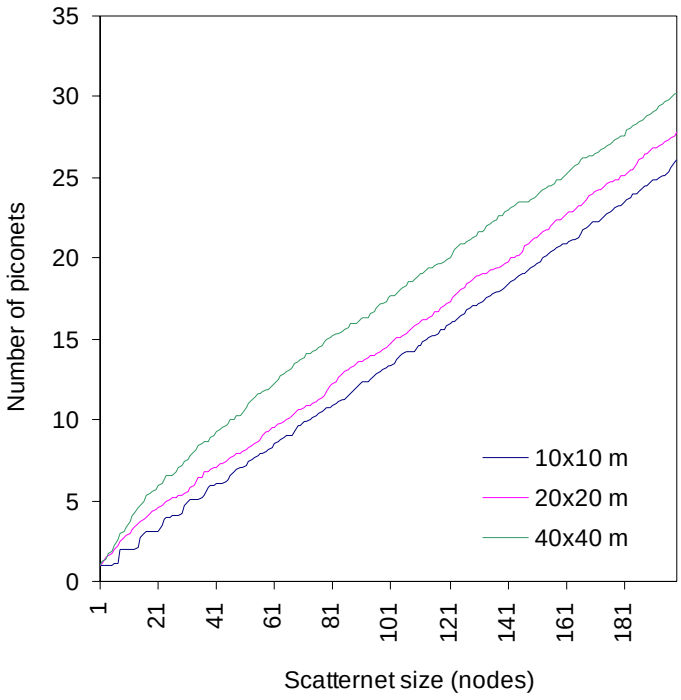

Figure 2.11 Total number of piconets

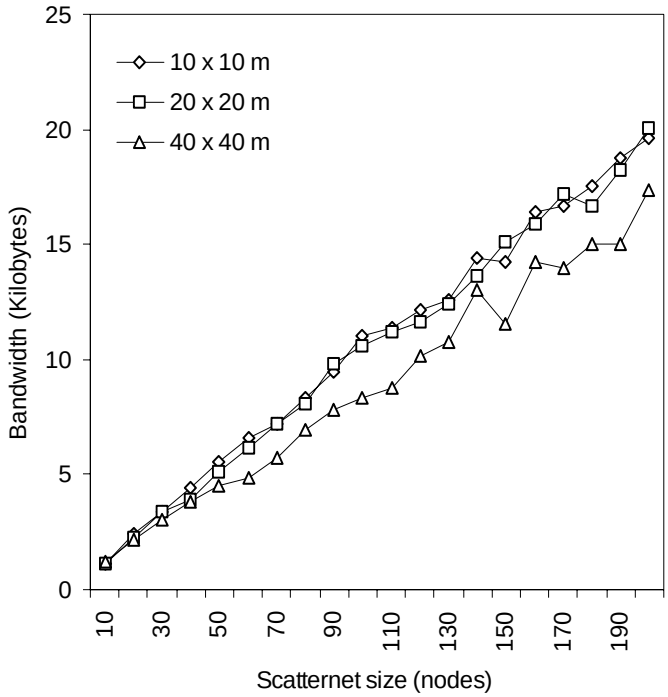

Figure 2.12 Consumed bandwidth

Chapter 3. A Mobile-Directory Approach for Service Discovery in MANETs*

3.1 Introduction

In the previous chapter, we introduced an alternative approach that can be employed to facilitate service discovery and provisioning by manipulating the topology of the wireless ad-hoc network (Bluetooth scatternet) in an attempt to link devices in close proximity to the desired service provider. If this were not possible, then these devices would have to make use of the multi-hop network to first discover the service and then access it. In addition, the topology-based approach favours wireless ad-hoc networks with static hosts and infrequent disconnections, which might not always be the case. In this chapter, we present the Service Directory Placement Algorithm (SDPA) as an alternative solution that brings service-related information closer to hosts that need it. SDPA promotes the deployment of a nomadic service directory whose current location in the network varies according to the dynamics of service-discovery queries driven by the users' applications, and partial knowledge of the network's topology. Whereas the skeleton of SDPA is based on a purely heuristic approach, we use a formal methodology to obtain the thresholds upon which the decisions to move the directory from one host to another are based. We formulate the problem as a Markov Decision Process (MDP) in order to find a cost-efficient strategy that incorporates some energy-conservation features as well. MDPs have already been successfully used to solve a variety of engineering problems, including mobility-related issues in the cellular networks arena (e.g., [75], [76]), and more recently in wireless sensor networks [77].

The rest of this chapter is organized as follows. In Section 3.2, we describe the proposed SDPA, and introduce MDPs as an effective approach for the directory-placement problem, as

* This chapter is based on a paper to appear in IEEE Transactions on Mobile Computing [66].

well as the Q-Learning method for solving MDPs without having full knowledge of the system states and transition probabilities. In Section 3.3, we describe the implementation and experimental set-up of SDPA. In Section 3.4 we present performance results through computer simulations that measure bandwidth conservation, discovery success rate improvements, potential battery preservation features and scalability aspects. We discuss issues pertaining to the practicality of our approach, its limitations, and future work in Section 3.5, and conclude with a summary of our scheme's highlights in Section 3.6.

3.2 The Service Directory Placement Algorithm

3.2.1 SDPA as a Heuristic Approach

In this chapter, we consider the case of MANETs configured as multi-hop networks, and assume the existence of a fixed network Access Point (AP) through which initial service directory interactions can be made, as shown in Figure 3.1.

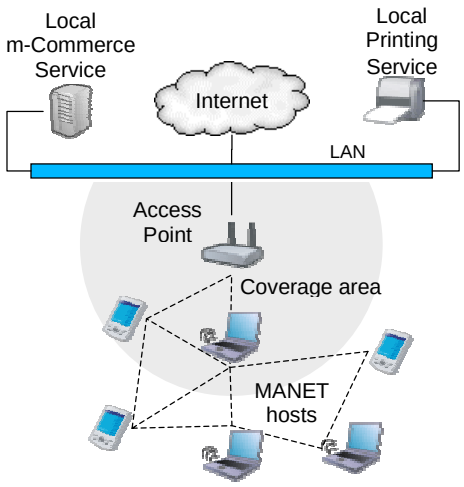

Figure 3.1 MANET as a multi-hop network

Hosts initially locate the service directory by issuing queries wrapped in UDP packets that are disseminated in an incremental breadth-first fashion. To this effect, time-to-live (TTL) values at the network layer are initially set to 2, and are subsequently doubled if additional

location attempts are needed. When found, the directory unicasts a response to the querying node, which in turn advertises the network location of the directory to its 1-hop neighbours. As a result, these neighbours avoid incurring a bandwidth-expensive directory location process if the network topology's variation rate allows the current path to remain unchanged by the time they need to issue a query of their own to the directory. Unicast queries will eventually fail to arrive at/from the directory at a later time when either: (1) there has been an error in the wireless link, or (2) the topology of the MANET has changed, in which case a service discovery process is repeated as portrayed in Figure 3.2.

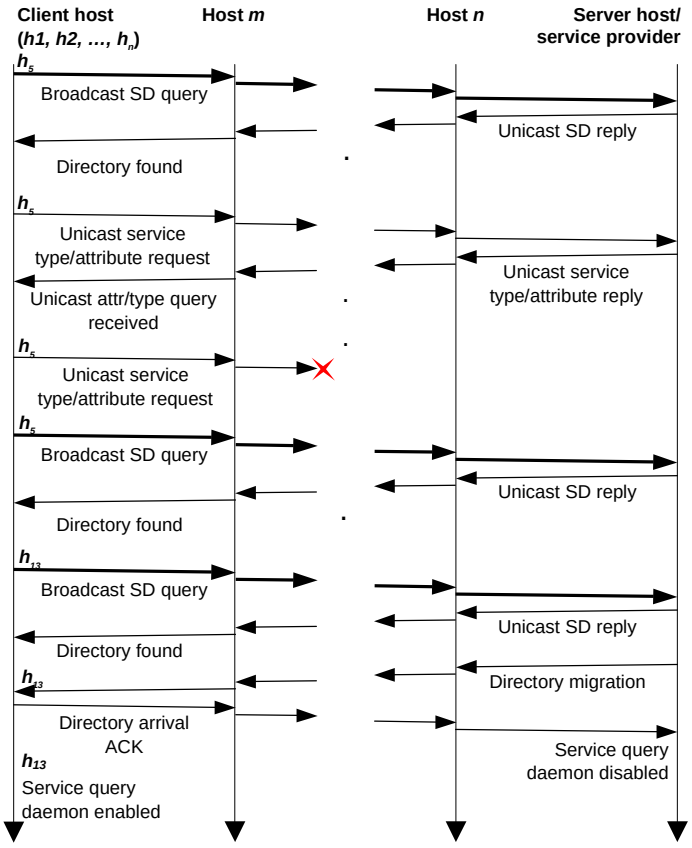

Figure 3.2 Signalling sequence for service discovery/query in SDPA

Up until now, we have described a fairly generic scheme for any SDP that supports the use of a directory. However, under SDPA's enhanced scheme, the directory eventually detaches from the AP and begins to recursively relocate from one host in the MANET to another. This

relocation decision process requires both the query pattern and localized topology information obtained separately by the directory and by the host issuing the query, at their current location. We define SDPA as a shim layer positioned between the transport layer and the actual SDP being used, as shown in Figure 3.3.

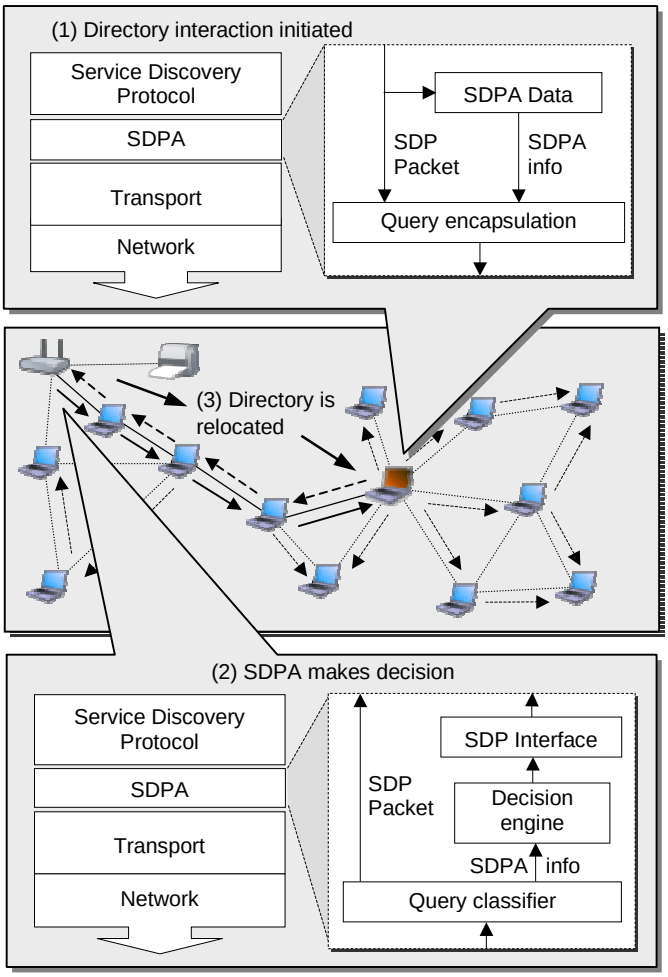

Figure 3.3 Directory relocation scheme of SDPA

The process begins at any host, where a regular SDP request is wrapped along with the aforementioned information and sent to the directory. Upon arrival, the packet is unwrapped to separate the data needed by SDPA and the actual service query that is sent up to the SDP for due processing. Once the information received has been processed, SDPA's task is limited to instructing the SDP whether relocating the directory to the host that issued the query will likely result more cost effective in terms of bandwidth consumption in the MANET as a whole.

However, as mentioned previously, SDPA requires an efficient decision making policy to rely upon. The process employed for obtaining this policy is described next.

3.2.2 The Directory Placement Problem as a Markov Decision Process

We begin by describing the environment's conditions surrounding a mobile service directory hosted by a user device in a MANET. First, we consider that users' mobility patterns are independent from both one another and their own previous movement history, which is a common practice in MANET research that employs random mobility models [78]. Evidently, this memory-less property has a time-varying effect over certain topology-related conditions, such as the number of neighbouring hosts and the hop distance of the current query, as observed at a mobile directory's current location. A second assumption is that the number of queries received at the current directory's location also varies in accordance with the users' independent preferences and needs. And even though users' mobility patterns and querying processes might not be completely independent in MANETs that are formed on-the-fly for group collaborations [79], [80], we still allow a certain degree of individuality in users' subjective behaviours.

Based on the previous arguments, we find no constraints for any particular network topology or query pattern condition from recurring sometime into the future, provided that both the service directory can effectively be relocated to any other host in the MANET, and that the long-term spatial distribution of hosts remains statistically homogeneous [81]. Moreover, if the circumstances surrounding a MANET environment warrant a limited number of observable conditions (hereafter referred to as states), then the time until a mobile service directory revisits any given state is finite. We can thus regard our system as *ergodic* [82], [83], [84]. These assumptions are readily applicable to MANETs wherein hosts move at low speeds in open networks of relatively small size [79], and can be reasonably justified in situations where users' devices provide a packet forwarding service for the common good.

In addition to the above, the memory-less nature of the model also indicates that the limiting probabilities of finding the system in any given state will converge to a certain value once it has been operating for a long time. Therefore, our bandwidth-saving objective can be maximized on an averaged basis over a long-term time frame if a suitable policy is employed to dynamically relocate the service directory in accordance to the users' combined mobility and service querying patterns. We can effectively approach this sequential decision-making process by modeling our proposed system as an MDP, as described next.

MDPs consist of several elements: an agent, a decision epoch, states, actions, a model and a reward [85]. In essence, the *agent* is a decision-maker exposed to the system that we wish to control, which can be described as being in one of a potentially many of states. The agent takes actions in an attempt to steer the system into a more desirable state; each of which is assigned a subjective value of its worthiness for a certain task. This value is recursively re-evaluated during the agent's learning period. The more successful the agent is in steering the system into a more desirable state, the greater the *reward* it obtains. Because the behaviour of the system is considered stochastic, there is a certain probability of the system entering a given state after a certain action is taken. The objective of the agent is thus to maximize the accrued sum of rewards, also known as the return R, received over the long term, at which point the steady-state transition probabilities are obtained. The time elapsed between the agent's decisions instants (τ_i-τ_{i-1}) is known as the *decision epoch*. Systems possessing real-valued and variable transition times between decision epochs are categorized as *semi*-Markov Decision Processes (SMDPs), and are of particular interest for our research.

MDPs can be solved by means of a technique known as Reinforcement Learning (RL). There are three main RL methods [86]: Dynamic Programming (DP), Monte Carlo (MC), and Temporal Difference learning (TD). The objective of these methods is the same – learn the true

mapping of every state to its value, known as the Value Function (VF). Whereas for MC methods the main drawback is the time it takes to compute an optimal solution, the main problem for DP is the formulation of an accurate model of the system. For the case of service discovery in MANETs, producing an accurate model of the system is a particularly cumbersome task given all of the factors that would have to be accounted for, such as changes in the radio signal propagation and the network topology, as well as hosts' mobility pattern, and service discovery/attribute query models. Previous research on computer networks has also relied on the use of reinforcement learning to solve a variety of problems (e.g., [87] - [90]). In our research, we employ reinforcement learning, and in particular, TD learning, which possess characteristics that are highly appealing for our purposes, as described next.

Initially, the VF can have arbitrary (deviated) values that are gradually adjusted to their true values by the RL method being employed. The state value function, defined as $V(s)$, describes a measure of worthiness or merit that the agent obtains by being in a given state s, or by proceeding with a certain action in this state. The particular values assigned to $V(s)$ are obtained by allowing the agent to learn which actions yield the highest rewards over a long period of time. There rewards are sequentially accrued into a value known as the return R. However, a problem arises if we let R accumulate an unbounded sum of rewards during the execution of the RL algorithm, especially if no absorbing final state exists that the system might eventually reach. In such *infinite-horizon* processes, a mechanism known as *discounting* is applied to control the rate at which rewards are accrued. A discounting factor γ with an initial value within the range (0,1) is thus introduced to reduce the weight of future rewards r:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\,, \tag{3.1}$$

where the return R_t is the accrued sum of rewards at time t for an infinite-horizon RL process.

The value of state s is equivalent to the expected return $E\{R\}$ for a process that starts at state s and chooses an action a from the set $A(s)$ according to policy π at each step thereafter. This value is denoted as $V^{\pi}(s)$ [86]:

$$V^{\pi}(s_t) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\} \tag{3.2}$$

where E denotes the expectation. The Q-learning algorithm does this by mapping state-action pairs to their corresponding maximum reward value, known as the optimal policy π^*. Hence, the optimal VF described by $V^*(s) = \max_{\pi} V^{\pi}(s)$ can be defined in terms of Q-values representing these state-action pairs as $Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \forall\, s \in S, a \in A(s)$. The Bellman Optimality Equations define the conditions under which the agent can maximize the rewards over the long run [91]:

$$Q^*(s,a) = \sum_{s'} P(s,s',a)\left[r(s,s',a) + \gamma \max_{a' \in A(s')} Q^*(s',a')\right], \tag{3.3}$$

where $P(s,s',a)$ denotes the probability of transitioning from state s to state s' when an action $a \in A(s)$ is taken, and $r(s,s',a)$ is the reward obtained during the same transition. For the case of SMDPs solved by means of Q-learning, the discount factor γ must be re-defined, given that the states' sojourn time is a continuous random variable. In this case, integrals are employed to compute the expected return, as opposed to summations: $\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \cong \int_0^{\infty} e^{-\beta\tau} r_{\tau}\, d\tau$, as defined in [92]. This leads to the reformulation of the Bellman Optimality Equations described above as:

$$Q^*(s,a) = \sum_{s' \in S} p(s' \mid s,a) \cdot \int_0^{\infty} \int_0^{t} e^{-\beta\sigma} \rho(s,a)\, d\sigma\, dF_{ss'}(t \mid a) + \sum_{s' \in S} p(s' \mid s,a) \cdot \int_0^{\infty} e^{-\beta t} \max_{a' \in A(S)} Q^*(s',a')\, dF_{ss'}(t \mid a). \tag{3.4}$$

We can see that, for a SMDP, $F_{ss'}(t|a)$ describes the probability that the next decision epoch occurs within t time units after the agent takes action a when transitioning from s into s'. (For a

discrete-time MDP, these transition times are identically 1). This yields the following result to account for the real-valued sojourn time between state transitions:

$$Q_{t+1}(s,a)=Q_t(s,a)+\alpha_t\left[\frac{1-e^{-\beta t}}{\beta}r(s,s',a)+e^{-\beta t}\max_{a'\in A(s)}Q_t(s',a')-Q_t(s,a)\right] \quad (3.5)$$

The reward $r(s,s',a)$ incorporates the value owing to the event that leads into state s'. Therefore, the Q-value for state-action pair (s,a) at $t+1$ depends a combination of the Q-value in the previous time step, a discounted reward value that penalizes/rewards the agent according to its previous action, and the maximum Q-value for (s',a'). A learning rate α is introduced to allow for the algorithm to converge to one of several possibly existing policies, and β is a factor that controls the rate of exponential decay. Both values are initially chosen from the (0,1) range and are gradually decremented to 0 as the algorithm proceeds. An important advantage of working with Q^* values is that it allows for the selection of actions without the need to know all of the possible successor states. In other words, knowledge about the underlying environment's dynamics as described by its state transition probabilities is eliminated, and a system model becomes unnecessary. However, sub-optimal results can be obtained if the actual system's behaviour lacks strict adherence to both the memory-less (Markovian) and ergodic properties as explained before, which is often the case in real-life situations [86]. The model of Q-Learning is shown in Figure 3.4.

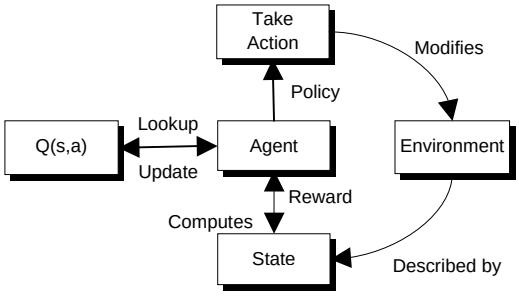

Figure 3.4 The Q-Learning model

3.2.3 Defining SDPA as an SMDP

As mentioned before, we formulate our directory-placement algorithm as an SMDP wherein the delivery of a service discovery's query at the directory's location marks the end of a decision epoch and the beginning of a new one. A directory-relocating policy can be learned off-line by the agent and later incorporated into our scheme as threshold decision parameters employed to determine the most cost-efficient action. Figure 3.5 portrays the sequence in which decision epochs occur. The decision epoch when the learning agent decides to take an action according to a service-related query is represented by ε_i, and τ_i is the elapsed time between two decision epochs. The elements of the RL process for SDPA are described below:

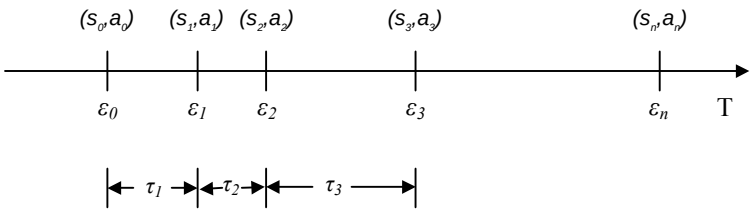

Figure 3.5 Time-line of events for the formulation of SDPA as an SMDP

States

The state of the network at time epoch ε_i is an abstract representation that takes into account the following factors as measured for the duration of each time epoch τ_i:

- The difference between the number n of neighbouring hosts at both the location of the querying node U and at the current location of the directory V at time t.
- The hop distance d between the querying host U and the directory V at time t, which can be less than or equal to the network's diameter D, also at time t.
- The average hop distance $\bar{d}$ traversed by queries q generated at hosts U_i received by the directory V in its current host location at time t.
- The difference between the number of queries q generated in the neighbourhood of the querying host U and those processed by the directory V in its current location at time t.

State space S for our system will therefore grow exponentially with the number of factors taken into account (and their magnitude). Figure 3.6 shows an example of how the previously described factors are taken into account to formulate the system state for the current time epoch.

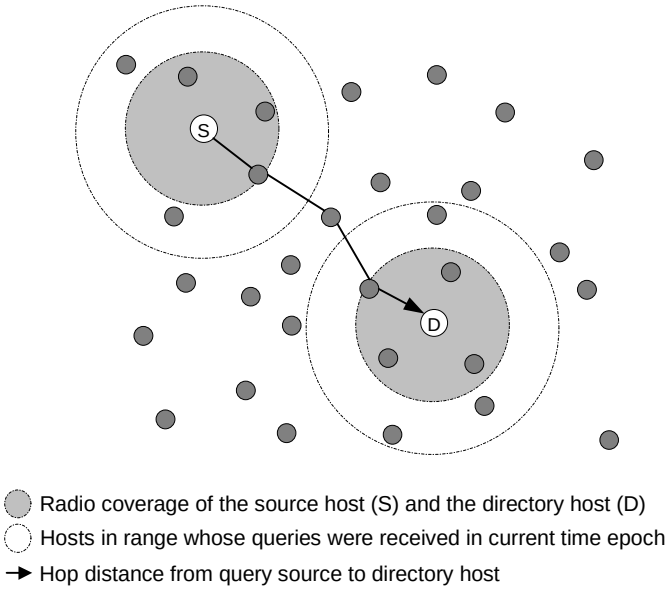

Figure 3.6 Depiction of factors taken into account in the computation of the current system state

Actions

For our system, the agent may choose to either let the directory remain at the current host, or to move the directory to the host that generated the last service discovery query. In other words, the action space can be described as $A(s)=\{relocate, do\ not\ relocate\}$. The well known ϵ-*greedy* scheme [86] is employed to define whether to choose an action a in $A(s)$ that has the largest Q-value for the current state with probability $1-\epsilon$, or take the other action in $A(s)$ as an exploratory move. In any case, the starting point for the agent is to choose one of the two actions at random since all the Q-values are initialized to zero. During the course of the learning process, ϵ is slowly decremented to 0 to ensure convergence to the optimal policy π^*.

Reward

The reward value is obtained by dividing the number of packets η incurred by service queries over the actual number of queries χ processed for the current time epoch t. This value

includes the SRV_RQST query that leads the system into entering s'. the This expression yields a negative value that results in lower magnitude values depicting a larger reward, and higher magnitude values representing a smaller reward:

$$r(s,s',a) = \frac{-\int_{t_\varepsilon}^{t_{\varepsilon+1}} \eta_t(s,s',a)\,dt}{\int_{t_\varepsilon}^{t_{\varepsilon+1}} \chi_t\,dt}, \tag{3.6}$$

3.3 Implementation and Simulation Setup

We implemented and evaluated SDPP using the Mobility Framework of the OMNeT++ discrete-event simulator [93] and employed the random-waypoint mobility model [94]. All hosts simulate the ability to support a directory-capable SDP (e.g., Jini, SLP), whose operation is complemented by use of SDPA. We simulated a generic wireless ad-hoc network, in which low-mobility hosts are randomly dispersed in a confined area of 500 m x 600 m forming a mesh topology. Additional parameters employed for this simulation are shown in Table 3.1.

We tested SDPA with a slightly modified version of IETF's SLPv2 framework. The architecture of SLPv2 consists of User Agents (UAs) in individual hosts, Service Agents (SAs) associated with service providers, and Directory Agents (DAs) that maintain service directories. The SAs register their service information with the local DA, which in turn periodically broadcasts DA_ADVERT packets to advertise services. A DA acts as a proxy for UAs attempting to discover services. To discover a service, a UA issues a service request query (SRV_RQST) to a DA, which replies with the service information in a (SRV_RPLY) packet. In addition, attribute reply packets (ATTR_RPLY) follow attribute request (ATTR_RQST) queries. However, for our proposed system, the host where the SRV_RQST query originated is in charge of issuing the Directory Agent (DA) advertisement (DA_ADVERT) to its 1-hop neighbours to inform them of the DA's location just discovered, as proposed earlier.

Table 3.1 Simulation Parameters for SDPA

Parameter	Value
Number of hosts	15, 25, 35, 45
Simulation area	500 x 600 metres.
Simulated time	10000 hours
Hosts' speeds	0.5, 1.0, 1.5, 2.0 m/s
Hosts' pause time	0 seconds
α	0.1
β	0.1
ε	0.1
Query retry attempts	3
$\max\{n_U - n_V\}$	100
$\max\{d_{UV}\}$	10
$\max\{\bar{d}_{q_{U,V}}\}$	10
$\max\{w_q\}$	100
Query interval distribution	Exponential
Mean query interval	2 minutes
Query timeout	5 seconds
Device transmission power	2 mW
Signal-to-Noise Ratio (SNR)	3 dB
Attenuation threshold	-110 dBm
Carrier frequency	2.4 Ghz
Path loss coefficient	3.2

For simplicity, the rest of the DA-query signals are modeled as generic signals since we are interested in measuring the number of packets generated at the network layer by the interactions between the DA at its current location and rest of the hosts. In other words, we do not model the details of the other types of queries since we assume that they generate the same number of network-layer packets. For example, the number of packets incurred by a service registration (SRV_REG) interaction is the same as one for a service de-registration (SRV_DEREG) interaction, and this applies to other types of interactions. Therefore, hosts individually issue a random number of DA generic-interaction signals upon discovering the current location of the DA (e.g., two SRV_TYPE_RQST interactions, and a single ATTR_RQST interaction). Hosts have to re-discover the new DA's location when the MANET topology changes before regular interactions resume. DA-interaction signals are issued with exponentially

distributed inter-arrival times that have a mean value of 2 minutes at every host. Finally, hosts that provide a service do not broadcast Service Advertisement (SA_ADVERT) signals.

At the network layer, our simulation incorporates a fairly generic routing scheme that is comparable to a significantly scaled-down version of the Ad-hoc On-Demand Distance Vector (AODV) routing protocol [95], wherein paths are discovered on-demand as triggered by interactions with the DA. Our down-sized AODV implementation omits the use of route-error packets (RERR), although sequence numbers are still used in order to avoid route loops. Routes discovered as a result of service-discovery queries issued by the individual hosts are stored in a <*destination,next_hop*> fashion. This nearly generic routing scheme allows us to isolate the effect that any particular routing algorithm (e.g., AODV, DSR [94], etc.) might have on our system, and therefore measure the effectiveness of our mobile directory approach to service discovery in a reasonably unbiased fashion.

```
1 Set DA location at the AP
2 Initialize α, β, ε to 0.1; Q(s,a) to 0, algorithmStep = 0
3 Initialize s, s_{t-1}, a, a_{t-1}
4 while(SimulatedTime < 10000 hours) do
5   if((algorithmStep > 0) && (signal==SRV_RQST || signal == GENERIC_QUERY))
6      DA-bearer host collects state data
7      s_{t-1} = s, a_{t-1} = a
8      Learner agent computes system state
9      Choose action according to ε-greedy scheme
10     Compute reward
11     Update Q(s_{t-1},a_{t-1})
12     Update α, β, ε
13     Learner agent informs action to DA-bearer host
14  DA executes learner agent's action
15  If(a==migrate)
16     Old DA-bearer host reschedules DA-interactions
17     DA information is moved to new DA-bearer
18     New DA-bearer host dequeues scheduled DA-interactions
19     DA-bearer host issues DA_ADVERT to 1-hop neighbours
```

Figure 3.7 Service Directory Placement Algorithm

SDPA is implemented as described in Section 3.2 (see Figure 3.7). The decision-making module is implemented as an omniscient agent (not to be confused with SLPv2's agents) that chooses among one of two available actions upon being referred to by the MANET hosts. Mobile

hosts are thus regarded as the network environment's state sensors for the agent, instead of each host having its own agent. In a practical implementation, real devices would possess a local copy of both SDPA and the decision-making policy (learned off-line) in order to make decisions independently as explained later.

We regard the service directory as a single data object containing compressed services' information, which is relocated at once employing a simulated TCP connection, following SDPA's decisions during the simulation's lifetime. Naturally, the agent is expected to err during the initial stage of the learning process given the uncertainty of the worthiness of the actions it takes for the current system state. However, as the simulation progresses, the agent becomes better at making decisions with Q-values progressively approaching their optimal point. *Q(s,a)* values are computed by employing expression (3.5), and are stored in a table at the end of every decision epoch. Parameters α, β and ϵ are all initialized to 0.1, and are linearly decremented until they reach 0 at the end of the simulation, according to the following expression:

$$\upsilon'(\tau) = \upsilon(\tau) - \left[\frac{\upsilon(\tau)\cdot\tau}{\mathrm{T}}\right], \tag{3.7}$$

where υ is the value being computed (α, β, or ϵ), τ is the current simulation time, and T represents the total simulation time, which is known in advance.

3.4 Simulation Results

Results are collected in experiments equivalent to 10000 simulated hours per run. To measure the performance of the learning system, packets are counted in a moving-window fashion by sampling the packet count every ten simulated hours, at which point the packet count is reset to 0. Our simulations contemplate scenarios wherein hosts move at constant walking speeds that are subjectively categorized as: slow (0.5 m/s), normal (1 m/s), speedy (1.5 m/s), and swift (2 m/s).

No pause times were incorporated in order to test our system under more stringent mobility situations (as perceived by real users), which also avoids a well-known pitfall pertaining to the random waypoint mobility model [96]. Network sizes of 15, 25, 35 and 45 hosts are employed to assess the efficiency of employing only one service directory, as per our assumption of small-sized MANETs discussed before. We also assume that collisions in the wireless medium are rare, given the fairly low density of hosts per square-metre, and the existence of an underlying mechanism that coordinates data transmission at the MAC layer. 500-hour averages of samples are taken for smoother plots. Tables 3.2 - 3.5 summarize the performance gains obtained by SDPA without any knowledge of the underlying system dynamics as seen at the beginning of the simulations (raw gain), the additional performance gain introduced by the RL system, and the combined performance gain seen at the end of each respective simulation.

Figures 3.8 - 3.15 show a breakdown of the number of packets measured in experiments that incorporate host speeds and network sizes as defined before. Figures 3.8, 3.10, 3.12 and 3. 14 depict the results obtained by employing a simple broadcast flooding scheme, which is used here as a reference for comparisons against the results obtained by employing SDPA as portrayed in Figures 3.9, 3.11, 3.13 and 3.15. This comparison reveals a clear superiority of SDPA against the regular broadcast scheme. From the beginning of the simulations there are noticeable packet savings even when no policy was yet available, proving the efficiency of SDPA as a purely heuristic approach.

From these results, it is evident that the use of SDPA and its RL-based scheme reduce the number of network layer packets attributable to DA-hosts interactions to nearly half of those that were produced by the pure (incremental) broadcast flooding scheme for the network sizes tested. We observed only a minor degradation of our system's total performance gain, as evidenced by a

slight decrease in efficiency ranging from only 3.6 % in the 15-host network to 4% in the 45-host network, given a 400% increase in host speed (from 0.5 to 2.0 m/s.) respectively.

Figures 3.11, 3.13 and 3.15 show that by the time the simulations have run for approximately ⅛ of their scheduled completion time, the policy obtained yields roughly ⅓ of the performance gains that can be attained overall, with the rest attained afterwards. This is not the case for the 15-host network shown in Figure 3.9, in which most of the performance gains are achieved in the first ⅓ of the simulation, followed by much lower gains thereafter. In other words, learning took place faster in networks of smaller size. We attribute this behaviour to networks of larger size evidently incurring a larger number of system states, and in turn a longer time that the agent spends visiting them before Q-values approach optimality and the system achieves top performance as seen in Table 3.6, and explained in Section 3.3.

The most striking results were obtained for the 35- and 45-host network cases when the lower speeds were tested. Here, the performance gains obtained by SDPA's pure heuristics make up for the performance degradation observed in the RL-based system as hosts' speed is increased from 0.5 m/s to 2.0 m/s. In other words, SDPA's raw performance gain alone (without the optimization policy) enables scalability in terms of host speed and network size as shown in Figure 3.10. On the other hand, the scalability of SDPA's packet overhead in terms of host speed degrades exponentially as revealed in Figure 3.11.

Accurate comparisons against other directory-based approaches for MANETs are not always straightforward given the variety in methodologies, evaluation platforms, simulation parameters and performance metrics employed in each of them. As a result, the existing literature on SDPs reports comparisons mainly against a default broadcast/multicast approach. For instance, Figure 13 (c)/(d) in the GSD approach [48] shows a network load just over 2000 messages processed per node on average in a MANET setting operating under somewhat similar

parameters after 75 simulated minutes. This amounts to roughly 100,000 messages for the MANET as a whole. In contrast, SDPA incurs an average of 375,000 packets during the same time period according to our results in Figure 3.15. This would indicate a clear advantage of using GSD over SDPA. However, these GSD messages are not network layer packets, but application layer messages. It is unclear how many TCP packets GSD yields, given the available experiments' information. In addition, their experiments incorporate pause times for the hosts' movements, making their topologies somewhat more stable. On the other hand, SDPA also shows better performance than VSD under somewhat similar circumstances [50]. Control overhead in [49] is reported in the form of traffic measurements, making direct comparisons somewhat impractical since no packet sizes were reported. In the case of [28], network load is in fact reported as control packets at the network layer, although based on the usage of popular MANET routing algorithms such as AODV and DSR. This same approach was followed as in [50], whereas [48] reports the use of a customized routing protocol.

Another factor to consider is the difference in query inter-arrival times employed in pull-based approaches, or the equivalent advertisement intervals in push-based approaches, overall ranging from 5 to 30 or more seconds. For the case of SDPA, the combined generation of queries follows an aggregate Poisson process with average inter-arrival times that may be as low as 1.33 seconds in the 45-host case, causing a larger number of overhead packets.

Figures 3.18 - 3.25 compare the directory-less approach against SDPA's improved discovery success rate. Clearly, a success rate of 100% is impossible to attain due to hosts in the MANET becoming momentarily isolated from the others as a result of their independent mobility pattern and limited radio coverage. Also, discovery success rates were higher in denser networks comprised of hosts moving at faster speeds. This was expected since hosts moving at a faster pace are less often in momentary isolation, which is the main cause of failed service-

discovery attempts. We can also observe that SDPA performs better than the unnamed approach in [28], despite the fact that our experiments involved no pause times (i.e., the rate of topology change in our case is grater). SDPA also performs better than VSD [50], although the hosts speeds employed are higher. As for [28] and [48], SDPA's performance is similar.

As explained early in this paper, an additional objective of SDPA is to reduce the amount of broadcast packets incurred by interactions between MANET hosts and the service directory in order to collectively conserve battery power. The results shown in Figures 3.26 – 3.29 underscore SDPA's battery-saving potential. These values show that the average number of broadcast packets incurred by using SDPA becomes a decreasingly smaller fraction of the average amount of packets that are generated when using the incremental broadcast flooding counterpart as the simulation progresses. This result confirms the ability of SDPA and its RL-based decision system to drastically reduce the amount of flooding packets, which are ultimately to blame for a good portion of the battery power consumption in MANET networking [28].

Finally, Table 3.6 shows the number of states observed by the RL system employed in our simulations. This number fluctuates in the range of tens of thousands, despite the fact that only four factors were employed to represent the system's state. The state space can actually be considered relatively conservative, given that the inter-arrival time of service queries was not taken into account as part of the system's state representation. Instead, this parameter is employed to compute $Q(s,a)$ values through equation (3.6), which contains the terms that adjust both the reward and the $max\ Q(s',a')$ values. The duration of each individual epoch is employed directly in the calculation of the reinforcement signal's magnitude as a delayed reward, whose value is measured in seconds by the learning system and incorporated accordingly.

3.5 Practical-Implementation Aspects

We now elaborate on a number of factors that would need to be considered should SDPA be put to work in real network. First, we note that existing SDPs such as Jini and SLP do provide the necessary structure for deploying a mobile directory-based service advertisement/discovery scheme, although the actual SDPA-SDP interface would depend on the architecture of each individual protocol. The learning system cannot be implemented "as-is" in a real network given the long time it takes for the Q-function to approach optimality and the resource constraints of the MANET devices to store and update a large Q-table. Therefore, the system must be put to learn "off-line" (i.e., in a simulation) prior to its incorporation into real devices. Different implementation approaches can be subsequently employed. The simplest one would be to apply hard-coded rules and a decision threshold table digested directly from the Q-function that SDPA would reference upon making directory-relocation decisions. In that case, the fewer the number of system states, the more compact the decision threshold table can be made. Not all of the state-action pairs need to be stored, only the subset of states with the smallest cardinality (e.g., either the *relocate*, or the *do not relocate* subset). Therefore, given the relatively small number of states observed by the learning system, a few tens of kilobytes would suffice to store the whole Q-table. However, this table might require further compressing to save memory if the cardinality of both subsets reaches parity and the available memory in the device is extremely limited. Thus, the information on system states that lead to the greatest performance improvements should be preserved, allowing SDPA to adapt more efficiently to changes in the MANET topology. Alternatively, function approximation in the form of a neural network could conceivably be employed, although some performance degradation might be experienced [86]. Also, reference values could be adjusted dynamically once the system is brought "on-line" in order to improve efficiency. As for the system's state abstraction, this information can be readily obtained by

interfacing with the MAC layer in an 802.11 network, the Baseband layer in a Bluetooth scatternet, or from a topology-configuration protocol being employed to estimate the number of neighbours observed at any given host location. Similarly, hop distance measurements can be obtained by interfacing with the routing layer.

Another issue to consider is that while some of the simulation parameters and system implementation details employed are intended to capture realistic conditions, the Markovian property upon which our system relies might not hold if certain conditions are violated. For instance, users' mobility patterns might not be independent, as mentioned before. Additionally, there is always a certain degree of subjectivity in choosing a mobility model when formulating a problem of this kind, which has been a matter of ongoing debate in the literature. Therefore, different mobility models could be pre-evaluated in an attempt to capture more realistic situations with as much accuracy as possible if deemed appropriate. Another factor to consider is that the frequency with which one or more users query for services does not necessarily have the same statistics or follow the same probability distribution, which most likely depends on the application/service being used. In addition, although the network size in our simulations was kept constant in order to measure the effectiveness of the proposed scheme in a systematic way, this condition is unlikely to hold in real MANETs. Therefore, the performance of the proposed system would vary depending on variations in the number of network hosts and their average speeds. For larger networks, a multi-directory approach would be needed to address scalability issues. We believe that our simulation set-up effectively captures the circumstances observed in practical situations where a moderate number of people in a relatively large area move at reasonable walking speeds as their devices participate in the MANET. As for system resilience, SDPA could sporadically send a copy of the current directory to the AP to ensure that the available service information is preserved in case of failure.

3.6 Summary

We have presented the Service Directory Placement Protocol for service discovery in wireless ad-hoc networks. SDPA is designed to work in conjunction with any SDP that enables directory-based service discovery. We observed that the formulation of SDPA's decision scheme as a Semi-Markov Decision Process proved useful in defining a policy that helps to accomplish our bandwidth-savings objective. The use of Reinforcement Learning, and in particular the Q-Learning technique, proved an invaluable tool to solve the SMDP. Results obtained through computer simulations helped to measure both the effectiveness and the limitations of the directory-based approach against the directory-less approach for different network sizes and host velocities.

Our experimental setup relied on a parameter space that is intended to help us understand both the feasibility of our proposed approach, and its variations in performance as these values are gradually changed. In practice, the characteristics of a MANET can be expected to constantly vary, implying that our system's performance would also fluctuate in accordance to the current network circumstances. Nonetheless, the data obtained through computer simulations indicate that a clear superiority can be attained by allowing the service directory to roam through the network's hosts when compared to employing a pure broadcast-flooding approach. SDPA decreased the amount of broadcast packets employed for service discovery, while maintaining or improving the overall discovery success rate. The fact that SDPA largely reduces broadcast and not unicast packets underlines its ability to enable a reduction in battery usage in the MANET as a whole. Limitations of our system, and practical implementation issues and possible solutions were also discussed.

Table 3.2 Comparison of Packets Generated for the 15-Host Network

Host speed (m/s)	Raw SDPA performance gain (%)	Performance gain by RL (%)	Total performance gain (%)
0.5	38.1	9.3	47.4
1.0	39.5	7.7	47.3
1.5	39.9	5.7	45.7
2.0	39.1	4.6	43.8

Table 3.3 Comparison of Packets Generated for the 25-Host Network

Host speed (m/s)	Raw SDPA performance gain (%)	Performance gain by RL (%)	Total performance gain (%)
0.5	18.0	31.9	49.9
1.0	27.3	20.8	48.1
1.5	29.9	16.6	46.5
2.0	31.2	14.5	45.7

Table 3.4 Comparison of Packets Generated for the 35-Host Network

Host speed (m/s)	Raw SDPA performance gain (%)	Performance gain by RL (%)	Total performance gain (%)
0.5	15.8	33.0	48.8
1.0	25.6	21.7	47.3
1.5	29.0	16.9	45.9
2.0	30.6	14.5	45.1

Table 3.5 Comparison of Packets Generated for the 45-Host Network

Host speed (m/s)	Raw SDPA performance gain (%)	Performance gain by RL (%)	Total performance gain (%)
0.5	14.9	33.9	48.8
1.0	26.0	21.1	47.1
1.5	29.6	16.5	46.1
2.0	30.7	14.1	44.8

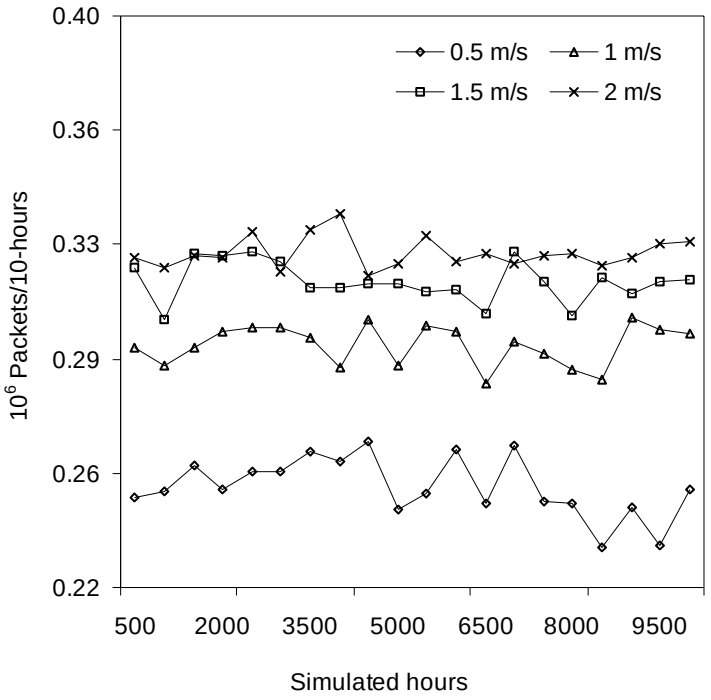

Figure 3.8 Total number packets per 10-hour period, averaged at 500-hour periods when broadcast flooding is employed for service discovery in a network of 15 hosts moving at the indicated speeds in meters/second.

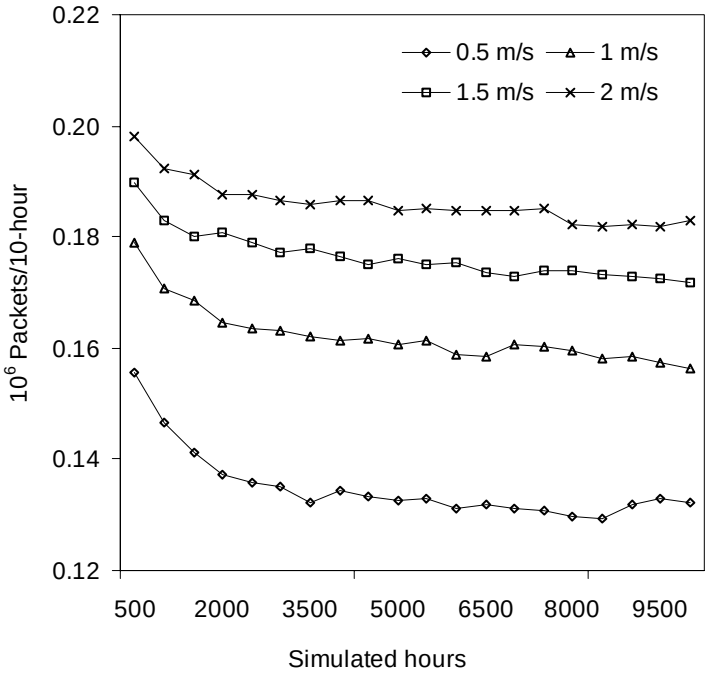

Figure 3.9 Total number packets per 10-hour period, averaged at 500-hour periods when SDPA is employed for service discovery in a network of 15 hosts moving at the indicated speeds in meters/second.

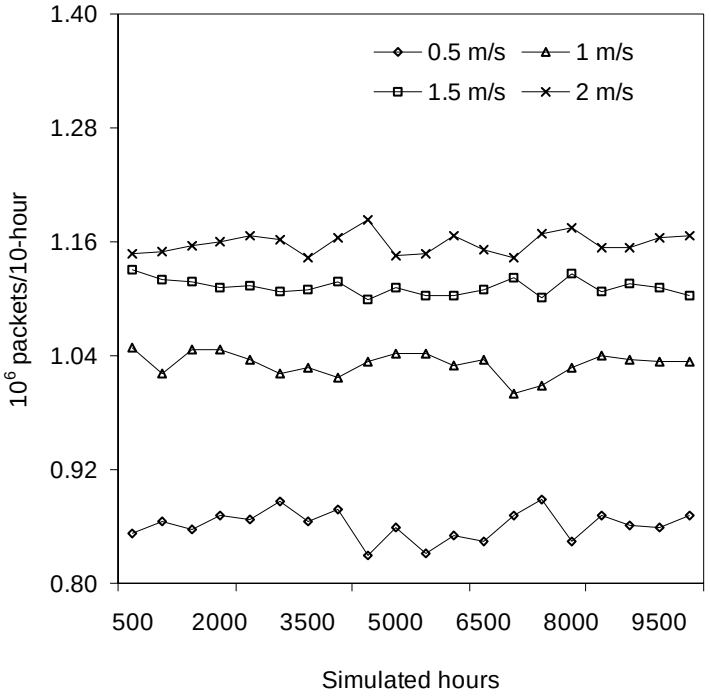

Figure 3.10 Total number packets per 10-hour period, averaged at 500-hour periods when broadcast flooding is employed for service discovery in a network of 25 hosts moving at the indicated speeds in meters/second.

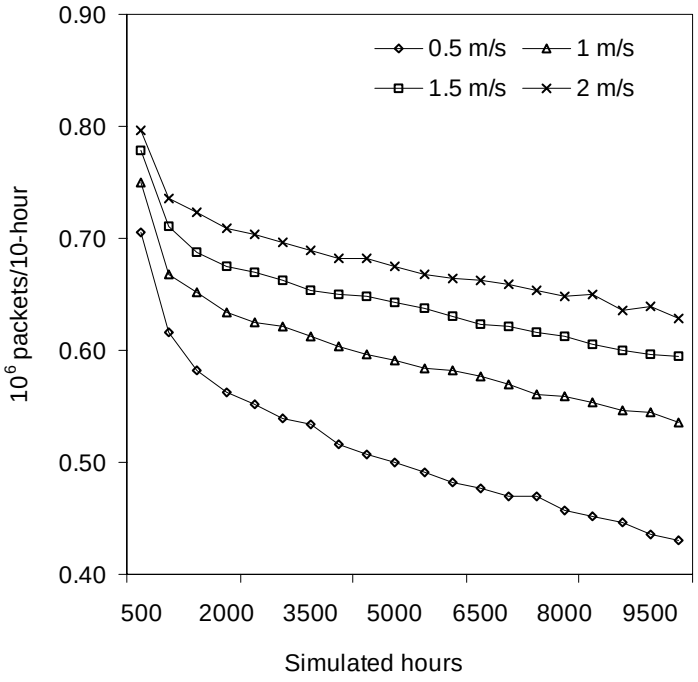

Figure 3.11 Total number packets per 10-hour period, averaged at 500-hour periods when SDPA is employed for service discovery in a network of 25 hosts moving at the indicated speeds in meters/second.

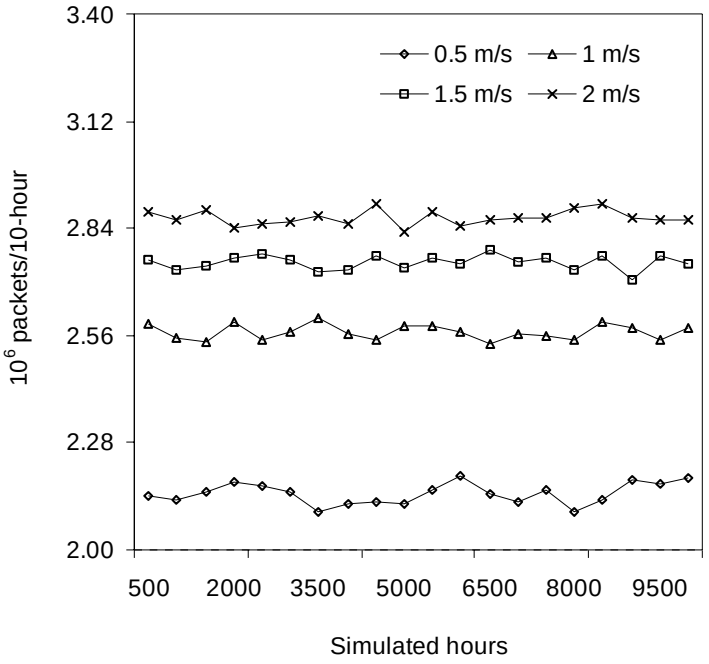

Figure 3.12 Total number packets per 10-hour period, averaged at 500-hour periods when broadcast flooding is employed for service discovery in a network of 35 hosts moving at the indicated speeds in meters/second.

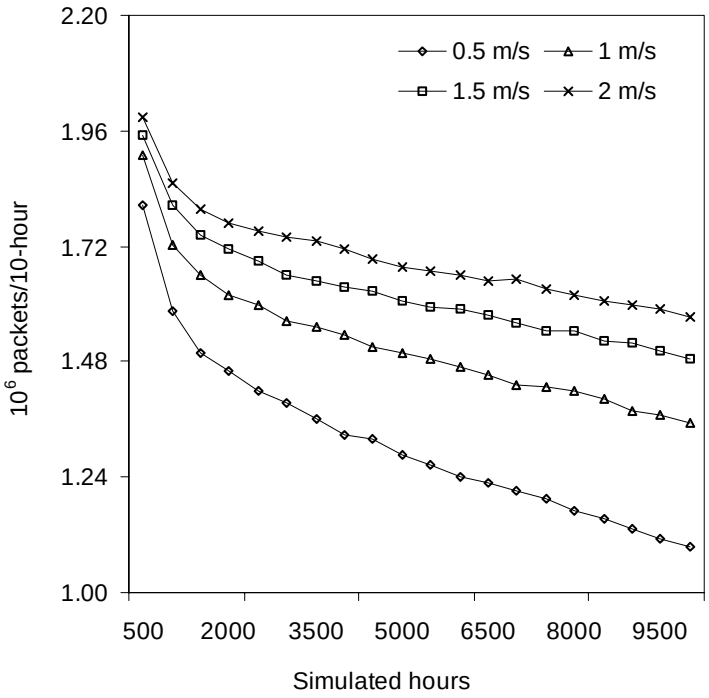

Figure 3.13 Total number packets per 10-hour period, averaged at 500-hour periods when SDPA is employed for service discovery in a network of 35 hosts moving at the indicated speeds in meters/second.

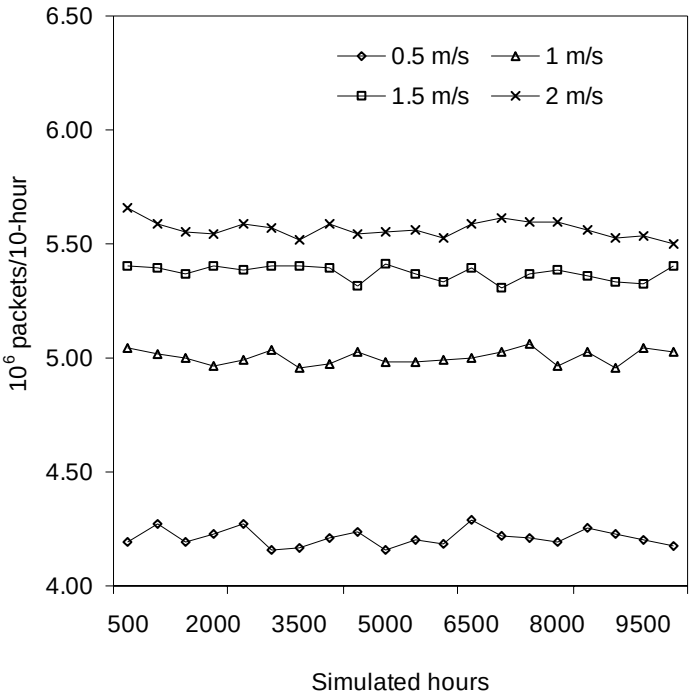

Figure 3.14 Total number packets per 10-hour period, averaged at 500-hour periods when broadcast flooding is employed for service discovery in a network of 45 hosts moving at the indicated speeds in meters/second.

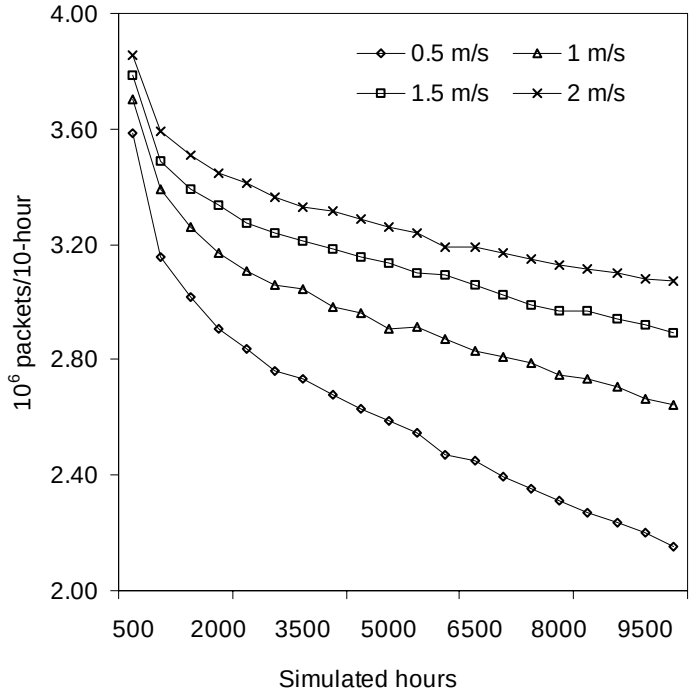

Figure 3.15 Total number packets per 10-hour period, averaged at 500-hour periods when SDPA is employed for service discovery in a network of 45 hosts moving at the indicated speeds in meters/second.

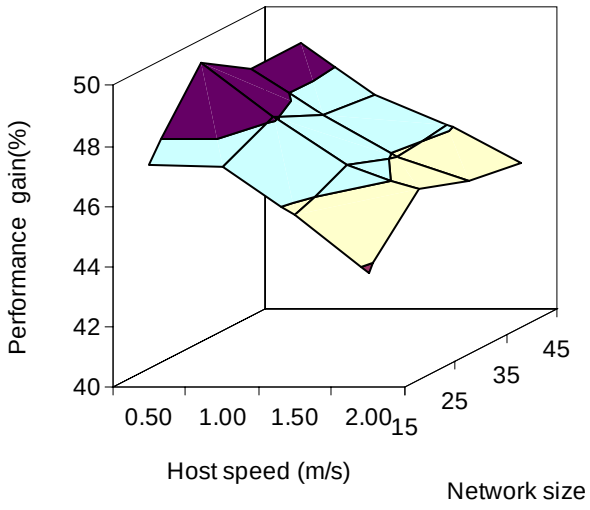

Figure 3.16 Scalability of SDPP's performance gain in terms of host speed and network size

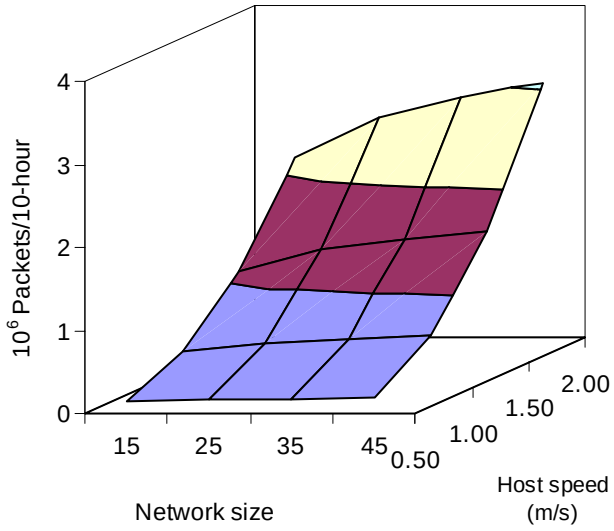

Figure 3.17 Scalability of SDPP's packet overhead in terms of network size and host speed

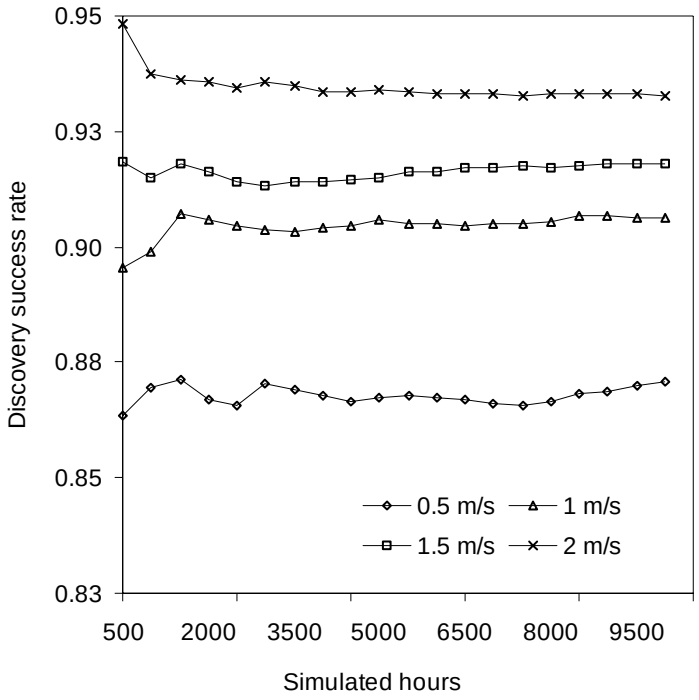

Figure 3.18 Discovery success rate measured in query responses over query attempts, averaged at 500-hour periods when incremental broadcast flooding is employed in a network of 15 hosts moving at the indicated speeds in metres/second.

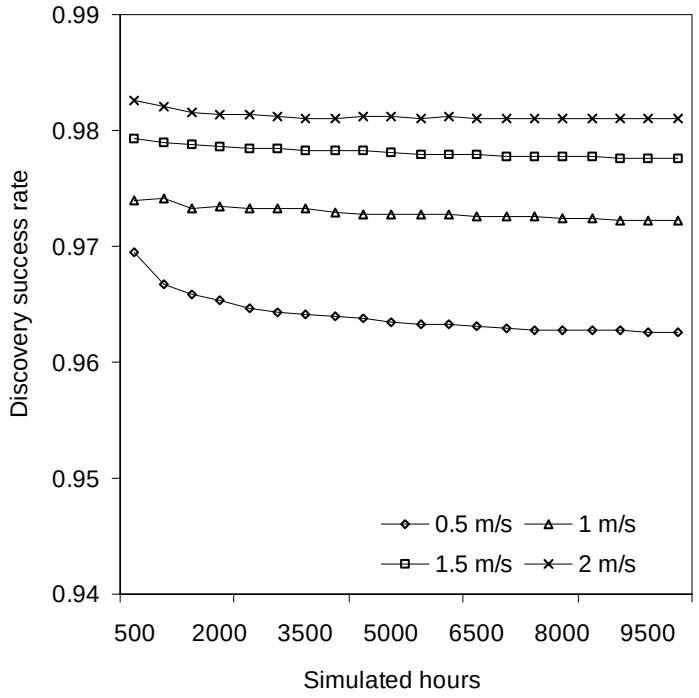

Figure 3.19 Discovery success rate measured in query responses over query attempts, averaged at 500-hour periods when SDPA is employed in a network of 15 hosts moving at the indicated speeds in metres/second.

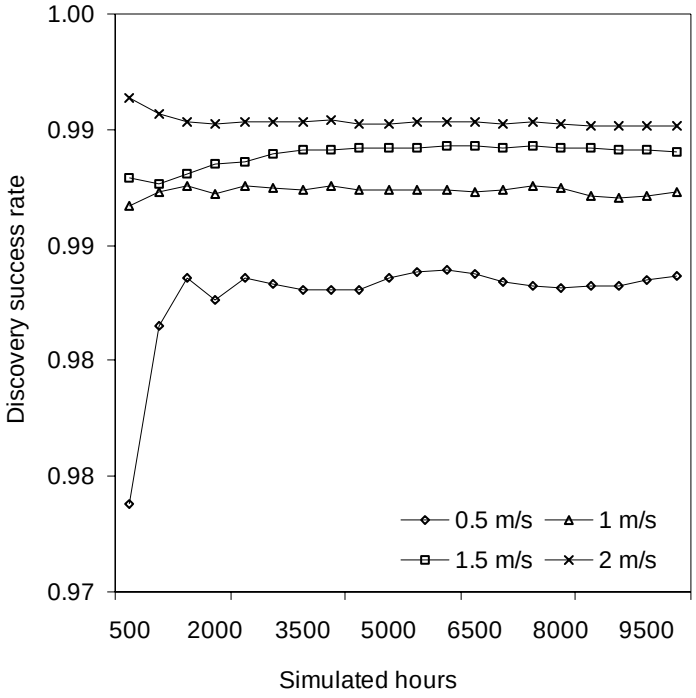

Figure 3.20 Discovery success rate measured in query responses over query attempts, averaged at 500-hour periods when incremental broadcast flooding is employed in a network of 25 hosts moving at the indicated speeds in metres/second.

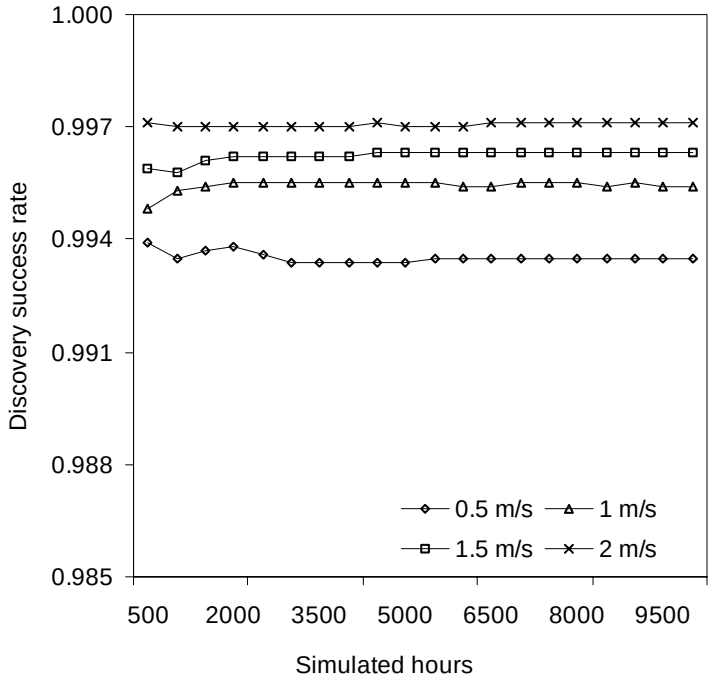

Figure 3.21 Discovery success rate measured in query responses over query attempts, averaged at 500-hour periods when SDPA is employed in a network of 25 hosts moving at the indicated speeds in metres/second.

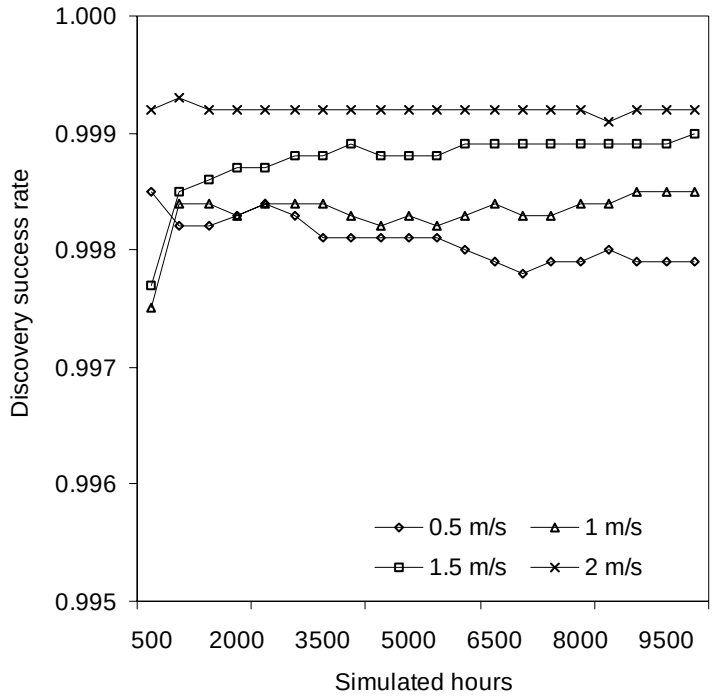

Figure 3.22 Discovery success rate measured in query responses over query attempts, averaged at 500-hour periods when incremental broadcast flooding is employed in a network of 35 hosts moving at the indicated speeds in metres/second.

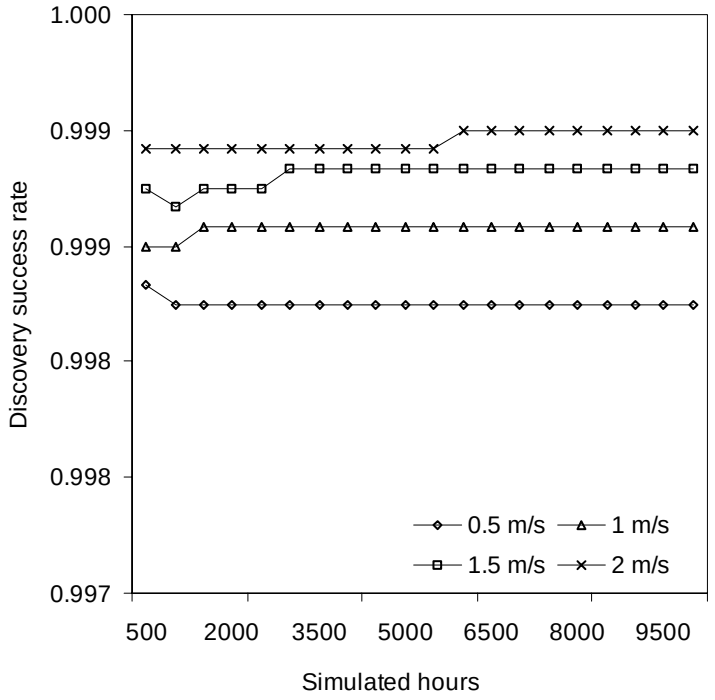

Figure 3.23 Discovery success rate measured in query responses over query attempts, averaged at 500-hour periods when SDPA is employed in a network of 35 hosts moving at the indicated speeds in metres/second.

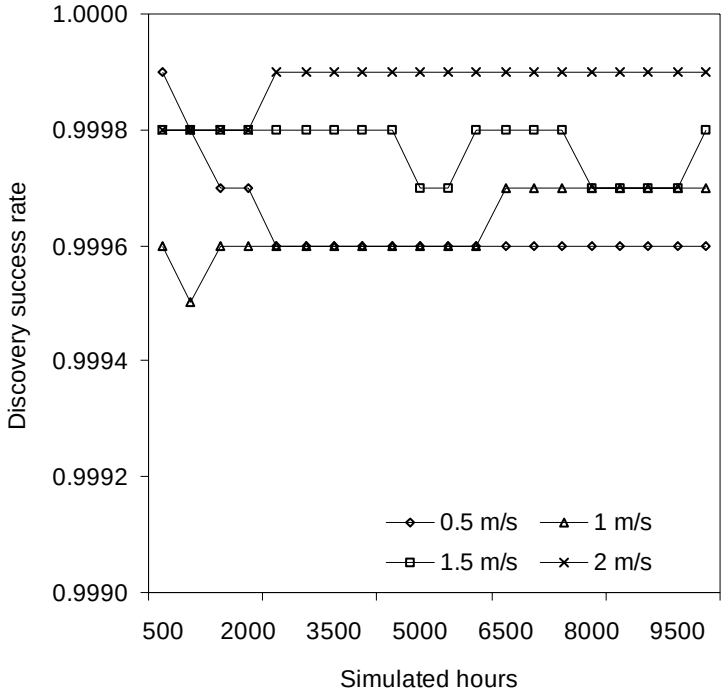

Figure 3.24 Discovery success rate measured in query responses over query attempts, averaged at 500-hour periods when incremental broadcast flooding is employed in a network of 45 hosts moving at the indicated speeds in metres/second.

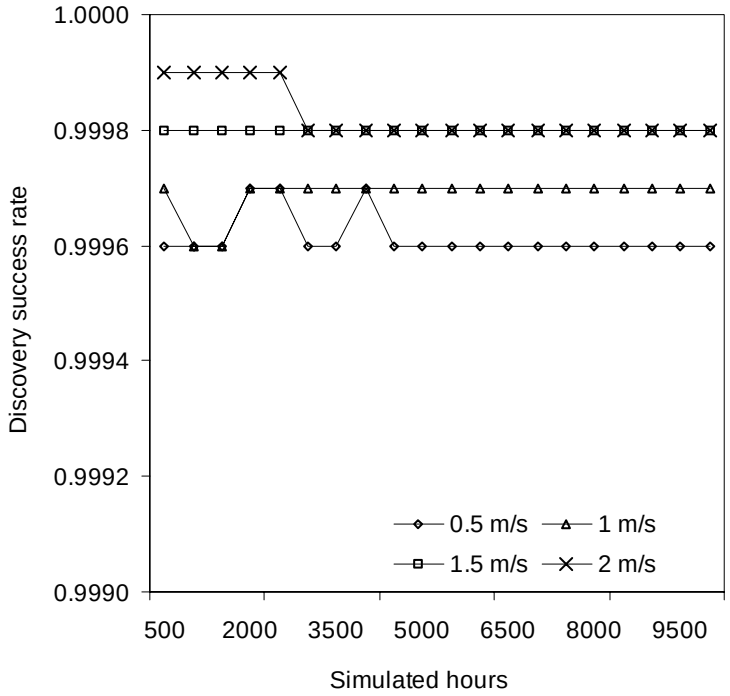

Figure 3.25 Discovery success rate measured in query responses over query attempts, averaged at 500-hour periods when SDPA is employed in a network of 45 hosts moving at the indicated speeds in metres/second.

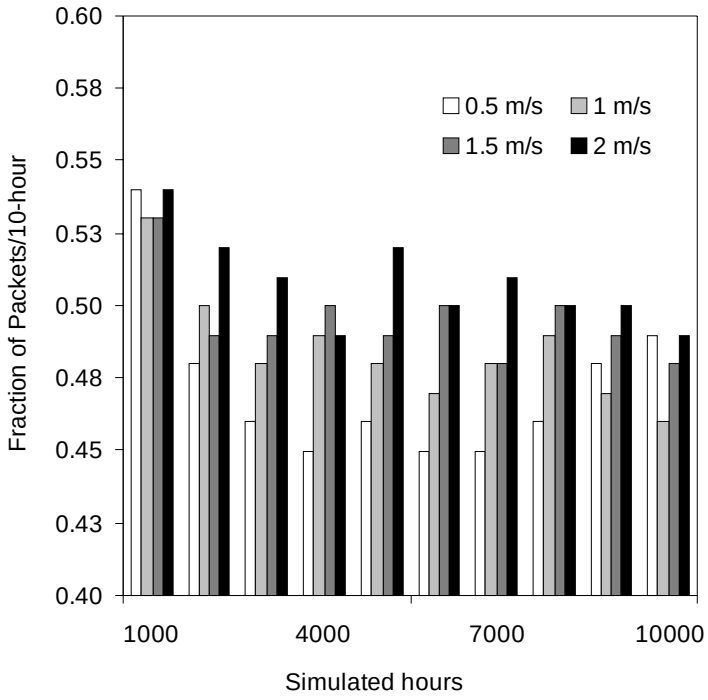

Figure 3.26 Break-down of flooding packets incurred by SDPA as a fraction of flooding packets from the total observed by employing regular broadcasting for service discovery, averaged at 500-hour periods for a network size of 15 hosts.

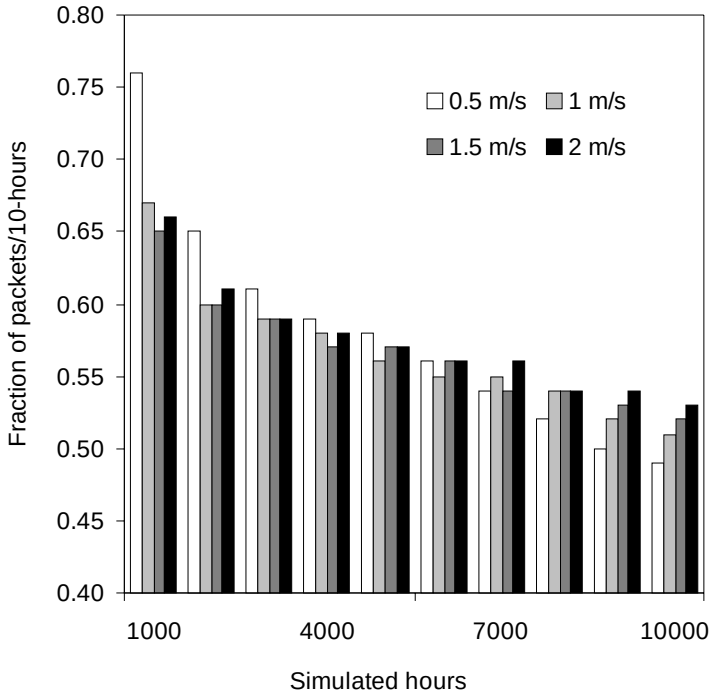

Figure 3.27 Break-down of flooding packets incurred by SDPA as a fraction of flooding packets from the total observed by employing regular broadcasting for service discovery, averaged at 500-hour periods for a network size of 25 hosts.

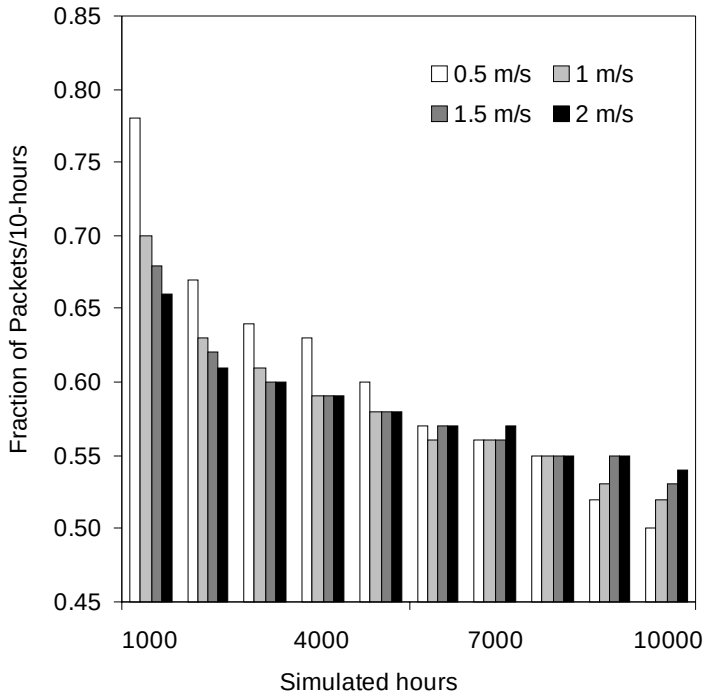

Figure 3.28 Break-down of flooding packets incurred by SDPA as a fraction of flooding packets from the total observed by employing regular broadcasting for service discovery, averaged at 500-hour periods for a network size of 35 hosts.

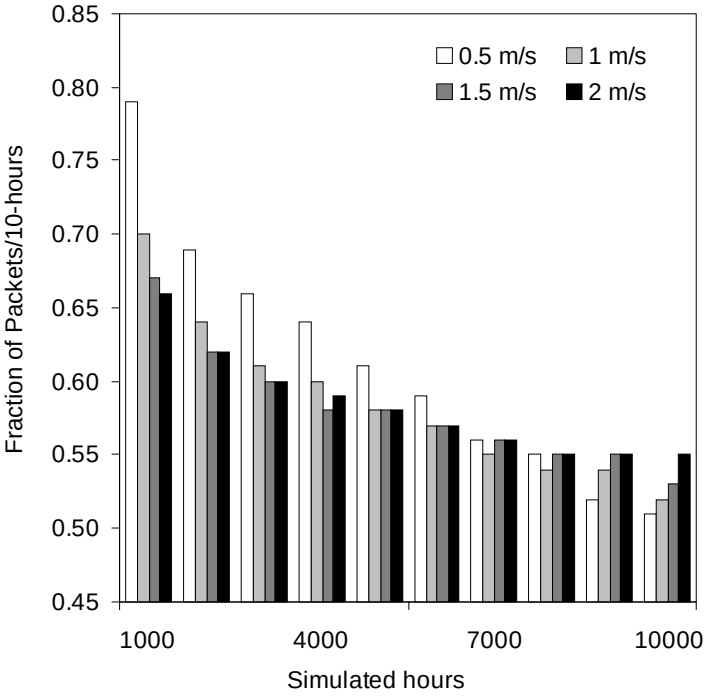

Figure 3.29 Break-down of flooding packets incurred by SDPA as a fraction of flooding packets from the total observed by employing regular broadcasting for service discovery, averaged at 500-hour periods for a network size of 45 hosts.

Table 3.6 Number of States Observed by the Learning System

Host speed (m/s)	15-host network	25-host network	35-host network	45-host network
0.5	13449	21785	25784	29722
1.0	13225	21604	25459	29173
1.5	13132	21519	25253	29139
2.0	13111	21299	25452	29284

Chapter 4. A System of Mobile Directories for Service Discovery in MANETs.

4.1 Introduction

In the previous chapter, we introduced and discussed the advantages of implementing a mobile directory approach for service discovery in wireless ad-hoc networks of relatively small size and limited host mobility. However, as observed in the corresponding results, a single mobile directory does not scale well with growing network sizes. In addition, SDPA's efficiency depends on the assumption of having short service descriptions that can be transferred in a one-shot transaction when the directory is moved from one host to another. This scheme may favour small networks where only a handful of services are present and the service descriptions are short (e.g., "printer", "Internet", "VoIP"). Nevertheless, SDPA's efficiency would evidently decay if more services were made available, or if the service's information footprint is larger, both of which require several transactions at the transport layer to migrate the directory.

In this chapter we present our Service Directory Placement Protocol, or SDPP, a multi-directory scheme for service discovery in MANETs. SDPP builds on the framework of SDPA, and enables multiple mobile directories co-existing with other fixed directories in a hybrid environment comprising by multiple types of devices, ranging from desktop computers to firmware-based service providers with limited memory. SDPP also builds on Version 2 of IETF's SLP protocol [38], with some modifications and enhancements that we propose to cope with the issues of host mobility. SDPA's original learning system is also modified to accommodate the multi-directory approach. In addition, we introduce a Service Entry Ranking System (SERS) that we employ as a simple measure to cope with network performance degradation that is indirectly introduced by memory limitations in users' devices. The rest of this chapter is organized as follows. Section 4.2 briefly discusses the state of existing service

discovery frameworks and their shortcomings. Section 4.3 introduces SDPP and describes its operation principles, followed by a description of the modified learning system in Section 4.4. Section 4.5 introduces SERS, which is employed to leverage the performance of SDPP. The proposed system's experimental setup is described in Section 4.6, and performance evaluations are presented and discussed in Section 4.7. A summary of this chapter's highlights is presented in Section 4.8.

4.2 Service Discovery in MANETs Using Multiple Directories

4.2.1 Limitations of Existing Frameworks

As explained in Chapter 1, UPnP and Jini have prevailed over a number of approaches that support scalable service discovery in wired LANs, whereas other schemes based on the JXTA framework have also recently surfaced in the literature. However, the efficiency in MANETs of these systems is largely unknown. We also note that these systems share some basic characteristics regardless of the specific methodologies followed to accomplish their objective: (1) they were designed to rely on minimal configuration in support of a user-friendly computing environment; (2) they support the use of moderately-complex service-advertisement architectures that allow for some degree of context-awareness; (3) they assume an underlying operating-system and hardware platform powerful enough to efficiently support their operation in conjunction with other system processes; (4) they make few provisions that facilitate the interoperability of devices employing their architecture with devices that employ a distinct SD framework.

The first aspect above raises concerns regarding the amount of overall protocol traffic needed to maintain the service discovery system operational in the presence of mobility. This is particularly a problem for UPnP if it were to be run over MANETs, as it relies on the combined use of HTTP, TCP, UDP, SSDP, XML, SOAP and DHCP in lieu of device drivers to simplify

device interoperability at the expense of increased network traffic. Jini is also prone to generating unnecessary traffic when directories issue the appropriate signals in order to avoid their service registration lease from expiring, which is a built-in feature of this and other systems. This procedure might trigger unwanted path discovery processes at the network layer if the MANET's topology is prone to constant changes.

The second aspect follows as per the recent trend in service-oriented computing that favours the use of information-rich service descriptions [36], [48], [97] - [101]. For instance, reseachers in [48] propose a high-level ontology-based approach in conjunction with a peer-to-peer system that caches service advertisements to selectively forward service discovery queries in MANETs and reduce overhead, as pointed out previously. However, the effectiveness of this approach depends heavily on whether all MANET hosts support this higher level of information abstraction, which might not be prevalent given the diversity of commercial platforms that tend favour the use of proprietary service discovery mechanisms. In general, a number of service discovery schemes based on the so called *web-services* framework have been proposed in the literature. In these web-services approaches, consumers and providers exchange XML messages in accordance to the SOAP and WSDL standards to discover service capabilities from a lexicographical perspective. In other words, network services are described and categorized in a highly-structured and information-rich fashion in an attempt to facilitate both service matching, and client-server interoperability. As a consequence of this, service descriptions have larger memory footprints, which might become a problem for MANETs with thin devices that have memory limitations and when a relatively large number of services are made available. An example of this would be a food/beverage-provision service advertising products in a MANET deployed at a convention/expo venue, in which attributes' and/or descriptions might include images or video advertising a promotion.

The third aspect is one of the most overlooked issues by proponents of service discovery systems based on Sun Microsystems' Java and Microsoft's .Net technologies. Both of these platforms require resource-rich computing systems that might be either scarce or unavailable in thin devices, which in turn aggravate energy consumption concerns. Once again, performance evaluations that help to estimate the extent of energy drain and the memory requirements of these and other proposed schemes are lacking in the literature. The fourth aspect is an issue that has been explored to a greater extent by researchers in the area, most of whom have proposed the use of sub-systems that facilitate interoperability in otherwise fundamentally differing approaches. Since UPnP has a more rigid architecture than Java-based systems (i.e., Jini, JXTA), several of the proposed schemes promote the use of bridges on the Java side that act as software drivers obtained on-demand, oftentimes made available from the service providers. Evidently, this solution adds to the memory availability problem as discussed before.

4.3 The Service Directory Placement Protocol

4.3.1 Operational Principles

SDPP is designed to work in conjunction with two-tiered service discovery architectures. In particular, we employ IETF's Service Location Protocol Version 2.0, in which a subset of the network devices creates an overlay network for the distribution of service-related information. SDPP is aimed at leveraging the efficiency of SLPv2 in a MANET by allowing an overlay directory network to adaptively extend its topology in a resource-conscious manner. SDPP builds on SDPA's architecture, and promotes the dynamic relocation of multiple service directories in an attempt to maximize the time-average performance of the network.

SLPv2's original architecture defines User Agents (UAs), Service Agents (SAs) and Directory Agents (DAs) as its core elements. SAs register their service information with a pre-designated DA, which acts as its proxy for UAs attempting to discover services in the MANET.

In addition, DAs periodically broadcast DA_ADVERT packets to advertise services to nearby UAs currently tuned into a predefined multicast channel. If a UAs is unaware of any DA, it initiates the corresponding interaction by issuing a SRV_RQST query to locate a DA, which in turn replies with a SRV_RPLY packet to inform the UA of the available services known to it. The UA then follows with an ATTR_RQST query of a given service, which is responded to with an ATTR_RPLY packet containing the corresponding information. This type of interaction continues until the UA is unable to reach the DA due to a change in the network topology, prompting the UA to execute the DA location process as before. Hence, the probability P that any given host will trigger the DA re-discovery process after n interactions is geometrically distributed, as in $P\{X = n\} = p^{n-1}(1-p)$, with mean $1/p$ for n. Here, p represents the probability of a successful query, and n assumes a value in the range $[1, 2 \ldots]$. The elements of SLP's/SDPP's architecture are shown in Figure 4.1.

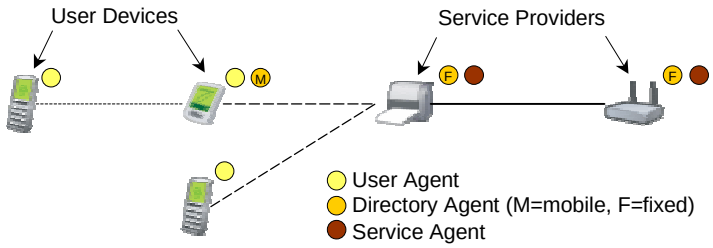

Figure 4.1 Agent elements in SLP's architecture

In addition to the above, DAs may exchange pertinent service information in a peer-to-peer fashion, enabling the systematic dissemination of service information as required by user hosts when the requested information is locally unavailable. SDPP employs SDPA's framework where DAs individually duplicate service-related information to one or more mobile hosts as needed, depending on whether the learning system's policies determine that this action yields a time-averaged performance improvement in the MANET. Therefore, although DAs are considered optional in the SLPv2 specification, they stand as the cornerstone of our SDPP

approach. For aspects relating to SDPP, we further categorize DAs as either Fixed Directories (FD), or Mobile Directories (MD), the former existing at the local (fixed) service provider, and the latter possibly existing in mobile hosts.

4.3.2 Formation and Maintenance of the Volatile Backbone Network

First, we note that the existing SLPv2 standard does not specify an interaction procedure between DAs. However, we assume that SDPP can coordinate this interaction by controlling the SLPv2 module through the use of APIs, much in the same way it is proposed to do so when cloning local directories into MDs. In other words, SDPP can instruct SLPv2 to issue the corresponding signals for service (de-)registration (SRV_REG/DEREG) as needed, in addition to those required to discover the presence of other DAs as explained shortly. In this respect, we recall that we propose SAs and DAs co-existing at the local service provider, and so remote DAs may process regular SLPv2 signals issued by the local SAs according to the specification, thus complying with the IETF specification. Hereafter, we refer to the cited SA-DA interaction simply as DAs' interaction.

Traditionally, DAs (FDs) are created and configured by a local area network administrator. However, the spontaneous and infrastructure-less nature of MANETs demands an automated means to deploy and maintain any directory-based approach in a fully unsupervised manner. To accomplish this, SLPv2 must first ensure that DAs are able to communicate with their nearest peers in the presence of mobility. Therefore, we allow FDs to become logically tethered by forming an overlay peer-to-peer network by means of spatiotemporal wireless connections that may relay on mobile hosts. FDs may use this backbone network to relay queries onto other FDs if the information requested by an ATTR_RQST query cannot be found locally. Thus, FDs form a "volatile" backbone, whose links may depend on non-fixed hosts. When any of these hosts move, the topology fractures and the backbone has to be fixed.

A significant amount of research has been devoted to the concept of MANET backbone formation in the past years (e.g., [102] - [106]). Once again, we decided to employ a simple generic approach that isolates the effect that any particular backbone formation algorithm might have on our system. This approach relies on a stateless peer location process that is triggered on-demand, ensuring that any given FD remains logically linked to at least one more neighbour FD insofar as the MANET's current topology allows it, as shown in Figure 4.2. We introduce a LOCATE_PEERS packet type, which is employed by FDs to locate their peers. The moment a service provider is enabled, a LOCATE_PEERS packet is broadcast in an incremental fashion, as done with SRV_RQST packets. Neighbour FDs reply with the corresponding LOCATE_PEERS_RPLY packet as seen in Figure 4.2, causing the FD peer discovery process to stop.

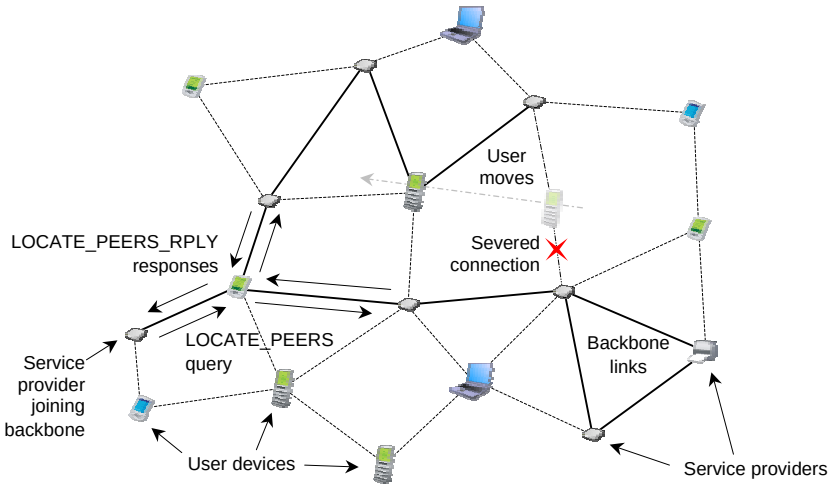

Figure 4.2 Multi-directory wireless ad-hoc network

If a service provider becomes temporarily isolated from its peers, and a predefined maximum number of attempts are reached, then the process continuation is deferred for a later time. However, if a previously isolated FD becomes reachable, and it receives a LOCATE_PEERS request from another FD during this wait period, then this FD replies accordingly, and its own deferred attempt to locate peers is discarded. The identities of unknown

FDs replying to the current query are individually added to the current list of logical neighbours upon receiving a LOCATE_PEERS packet. Decisions of whether or not to update service tables at the local directory are based on a timestamp value that the replying FDs include within the LOCATE_PEERS_RPLY packet issued. If the timestamp of an FD is newer than the one locally stored, then it is inferred that this particular FD had its contents changed, triggering a directory update process. This allows already-known FDs to avoid unnecessary directory updates if no changes have been made to the descriptions that they have registered. The inter-directory content update procedure is realized by employing the existing SLPv2's registration/deregistration process, which employs SRV_REG/DEREG signals to individually update the availability of entries at the DA. ATTR_RQST packets are duly processed by DAs (either an FD or an MD) depending on whether the service attribute sought for exists locally or not. If it does exist, an ATTR_RPLY packet is immediately issued as seen in Case A of Figure 4.3.

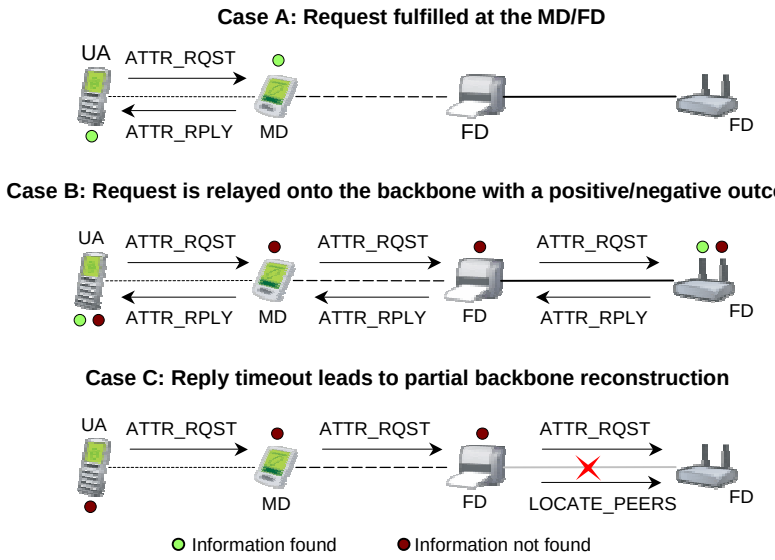

Figure 4.3 Propagation of ATTR_RQST queries onto the backbone network

If the service attribute does not exist, then the packet is propagated onto the backbone network in an attempt to find the desired attributes' info at another FD, as seen in Case B of Figure 4.3. An FD containing the information sought for immediately issues a corresponding

ATTR_RPLY packet, which implies that the query originator might receive more than one reply. However, if the query's search depth is reached before a match is found, then any involved FDs issue an ATTR_RPLY packet with a "not found" indication back to the sender. On the other hand, if the FD that initially forwarded the query onto the backbone network receives no response within a given time frame, then it is assumed that the existing backbone structure has failed. At this point, the local neighbour table's is flushed, and a LOCATE_PEERS packet is issued in order to find new FD neighbours as before (Case C of Figure 4.3).

4.3.3 The Mobile Directory Subsystem

The mobile directory subsystem is introduced to bring service information closer to users' hosts that become physically distant from the FDs. This approach attempts to diminish the system's performance degradation caused by queries traversing longer paths in the MANET. To accomplish this, the SDPP module at the sender appends special meta-data within a SRV_RQST query as done in SDPA. At the receiving end, the FD processes the SRV_RQST query according to SLPv2's specification, whereas the meta-data is passed onto the local SDPP module. The learning system there uses the available information to make a decision of whether sending a copy of the local directory to the query sender results in time-averaged performance improvements. However, it is possible for multiple FDs to receive a copy of the original SRV_RQST query broadcast. To avoid concurrent DA duplication attempts, we introduce a DA_RQST packet that FDs employ to notify the target host of their directory copy intent. The target host schedules a timeout event upon receiving the first DA_RQST packet, since more requests might be subsequently received. When this time window expires, the target host randomly chooses one of the FDs and sends to it a DA_RPLY response accepting the directory copy intent. Then, the FD employs SLPv2's existing service registration (SRV_REG) messages to gradually copy entries from the FD sender to the target device, as seen in Figure 4.4.

The new MD advertises its presence to all one-hop neighbours by broadcasting a DA_ADVERT packet. Neighbouring hosts in turn refer to the new MD when forwarding subsequent ATTR_RQST queries, as is done in SDPA. If an MD is unable to service an ATTR_RQST due to lack of information about a particular service, then it relays the query onto the FD backbone network as explained previously. For simplicity, the MD forwards the ATTR_RQST packet through the FD that initially spawned it (i.e. it's 'anchor' FD).

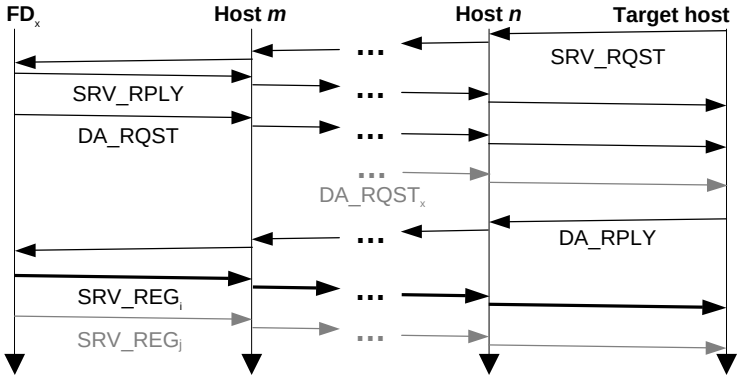

Figure 4.4 Signalling sequence for SDPP

Intuitively, the deployment of the MD results favourable in the case when the MD carrying most of the services' entries that are currently being queried by neighbouring devices, enabling the possibility of substantial bandwidth savings. However, if the MD's memory is limited, then the number of directory entries that it can hold will also be limited, leading to a greater number of unwanted backbone-relayed queries, which are considered overhead. Furthermore, if user mobility causes an MD-to-FD path rupture, then the MD will first have to broadcast a LOCATE_PEERS query in order to find a FD gateway to the backbone network, similar to Case C in Figure 4.3, which is also accounted for as overhead. Therefore, the FD must learn the conditions under which directory duplication improves network performance, and when it does not. The details of this learning process are explained in the following section.

4.4 Multi-Directory Learning System

For our extended multi-directory system, we assume that all of the considerations previously pondered in Chapter 3 for defining the directory placement problem as an SMDP still hold. However, as mentioned before, the learning system was refit to accommodate the needs of our multi-directory approach. First, we note that Fixed Directories (FDs) and Learning Agents (LAs) coexist in symbiotic pairs in each of the MANET's service providers, in contrast to having a sole "omniscient" LA responsible for finding the optimal directory duplication policy. This allows reward values being assigned to individual actions in response to a decision made by an LA. In addition, we continue to employ the Q-learning technique to maintain consistency with the framework advanced in the previous chapter. However, unlike the SDPA approach, MDs do not perform any learning function. They only exist as simple directories that process user queries. We also note that LAs perform their work in a non-cooperative fashion, meaning that the policies learned are not shared with one another. This implies that the combined actions learned by each LA will eventually help reduce service discovery traffic in the MANET as a whole. The specific aspects of the learning system are now described:

States:

A new system state is added to the ones previously considered for SDPA, which is represented by the total directory memory g available at host U where duplication is being pondered. Knowledge of this value helps the LA to decide when it is more efficient to duplicate service data than otherwise.

Actions:

Only two possible actions are considered at the current directory in the modified system, and so the action space can be described as $A(s)=\{copy,\ do\ not\ copy\}$, which we refer to with the keywords STAY, and DUPLICATE. An LA may duplicate its content to the extent allowed by

the available memory in target host, but the FD is always preserved at the corresponding service provider.

Reward:

The reward function was changed to enable individual LAs to assess whether their actions lead to better system performance within the corresponding decision epoch [$t_\varepsilon - t_{\varepsilon-1}$]. After repeated experimentation with different schemes yielding unsuccessful outcomes, we dismissed the use of broadcast flooding traffic as being an inaccurate feedback mechanism for overall efficiency assessment. This is due to the LAs' inability to determine whether fewer SRV_RQST packets in the MANET could be directly attributed to their individual actions, or the actions of other LAs working independently in overlapping time frames. Moreover, having omniscient access to the number of SRV_RQST packets in the MANET would be impractical in a real setting. Therefore, the reward function was modified to reflect the following random variables with values that depend on the duration of the current time epoch:

- J ATTR_RQST queries QH. This value is obtained by summing over the corresponding queries issued by each of I possible hosts in the current time epoch.
- L ATTR_RQST queries QD. This value is obtained by summing over the corresponding queries relayed onto the backbone network by each of K possible mobile directories in the current time epoch.
- 1 SRV_RQST query received at the corresponding FD from any given MD that marks the end of the current decision epoch.
- M ATTR_RQST packets PH. This value is obtained by summing over the corresponding packets issued by each of I possible hosts in the current time epoch. This process yields an equal number of ATTR_RPLY packets, and so the double summation is multiplied by 2.

- *N* ATTR_RQST packets PD. This value is obtained by summing over the corresponding packets relayed onto the backbone network by each of *K* possible mobile directories. This process yields an equal number of ATTR_RPLY packets, and so the double summation is multiplied by 2.
- *H* network-layer packets comprising the entire SRV_REG interactions when a FD copies its content into an MD in the current decision epoch.

The reward signal is obtained by dividing the total number of service discovery queries for the current time epoch over the number of packets generated by these same queries, resulting for the current epochs, the later of which ends when the corresponding SRV-RQST query is received at the local FD. This indicates that the LA's reward increases when it sees a larger number of queries with respect to the number of packets. This can be achieved when an FD is copied to a host that ends up servicing a larger number of queries from neighbouring hosts. However, if the directory-recipient host (MD) possesses limited memory to accommodate a sizable number of service descriptions along with their corresponding attributes, then the host that duplicated the directory will inevitably observe rely ATTR_RQST queries originating from the new MD that yields unwanted overhead traffic, and in turn a negative number for the reward $r(s,s',a)$. Therefore, the LA will eventually infer the threshold memory value at which the amount of saved bandwidth surpasses the overhead attributed to ATTR_RQST packets relied from MDs onto the backbone network. At this point, the optimal policy is found in accordance to the rest of the factors that account for the system state and the current system parameters as seen later.

Finally, MDs must have a way of knowing when their existence actually contributes to bringing improved performance to the overall system. A lack of feedback information on this matter might otherwise lead to increased packet overhead when their existence is unwarranted

given the existing circumstances. To avoid this, MDs simply keep count of the number of service queries successfully processed against those that had to be relayed to the FD network. The MDs' continuation is guaranteed upon a positive difference in the number of successfully processed queries, whereas a negative outcome leads to its termination without any FD being notified.

4.5 The Service Entry Ranking System

The SERS operates on the assumption that it results counter-intuitive to copy service description/attributes that are seldom queried in the present time frame. Doing so would result in the MD's eventual relaying of ATTR_RQST queries to the anchor FD, incurring unwanted traffic. Also, not all of the information contained in a directory might be copied due to memory limitations in small, portable devices. To address this issue, we introduce a straightforward solution that helps to determine which directory entries should be copied. Our scheme is based on a method that ranks each individual directory entry according to its popularity. The system is based on the frequency with which directory entries are queried, as opposed to employing the number of query hits that each entry receives. This makes the system fairer to those services that have been available for a shorter portion of the time that the MANET has been operational. Otherwise, services that have been available for longer periods would see their ranking unfairly favoured regardless of their current popularity status.

Initially, every entry's SER is set to 0. Then, their SER is subsequently updated according to a simple method that mimics the way in which a capacitor in an electronic circuit might charge/discharge. A query hit for any given entry symbolizes a narrow pulse that charges a capacitor to a level commensurate with its previous value or "charge" (the service's rank). Thus, the more often an entry is queried, the higher its SER value grows. Therefore, the query inter-arrival time for each respective entry is recorded at the local directory (either FD, or MD), for the recurrent calculations of each SER value when the corresponding service query is processed.

Similarly, a lack of queries received for an entry leads to its SER value decay according to expression (4.1), as though the capacitor were discharging.

SER values are all normalized to 1, and their actual magnitude depends in the current number of entries $N(t)$. In addition, SER values converge asymptotically to a given value if no queries for a service are received after a relatively long time. This equalization policy makes the system fairer when the time that a given service remains more popular than others becomes statistically less significant over the long run, as exemplified in Figure 4.5. Note that SER values in one MD might differ from those at another MD, which reflects the spatiotemporal popularity of a given MANET service. Therefore, the need of a mechanism to maintain the consistency of the SER values across the network is eliminated, since SER values are not shared among MDs.

$$SER_i(t) = \frac{1 - e^{-1/\lambda_i}}{\sum_{j=0}^{N(t)} \left(1 - e^{-1/\lambda_j}\right)} \tag{4.1}$$

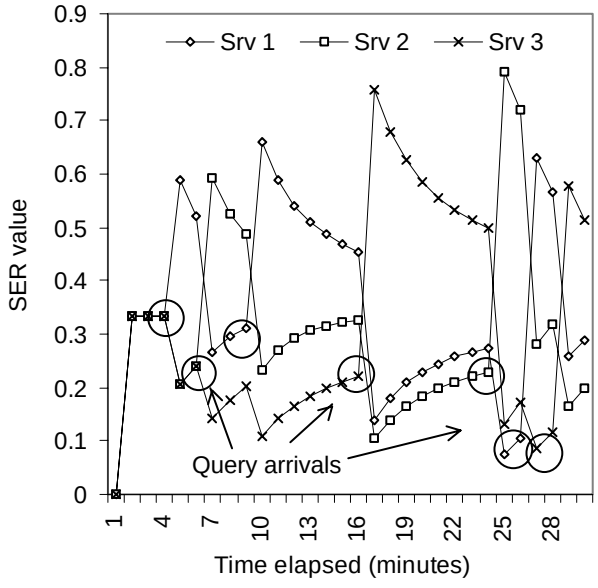

Figure 4.5 Sample behaviour of SER values.

During the copy process, priority is given to entries that possess the highest SER values. However, if the attributes' memory footprint of the current entry is too big to fit into the target

memory space, then only the service description is copied, and the process attempts to copy the entry with the second highest SER. This process is repeated until either all of the local entries have been copied should the new MD have enough memory, or the memory available at the target host is depleted. Both the inter-arrival query time and the SER for each directory entry are also copied to the new MD, since these become part of the attributes for every single service. Entries with lower SER values are routinely replaced at the local FD/MD by entries fetched from other FDs upon receiving an ATTR_RPLY packet in response to a previously relayed ATTR_RQST query onto the backbone network as explained previously.

4.6 Implementation and Simulation Setup

In this section we describe the features of our experimental evaluation for SDPP. Our computer simulations were implemented using the OMNeT++ Discrete Network Simulator, as with SDPA. We probed the behaviour of the proposed system with networks of 50, 100 and 200 hosts, plus a fixed number of stationary service providers whose physical positions in the network are assumed uniformly-distributed. By default, each service provider possesses a FD that remains enabled during the network's lifetime. Mobile hosts are uniformly dispersed in areas of 500, 700 and 1000 m^2 correspondingly, forming a multi-hop network. They travel at speeds of 1 or 2 m/s, and have pause times of 900 seconds according once again to the random waypoint mobility model. All hosts remain within the area's boundaries for the duration of the simulation. The transmission range of all hosts is 100 m. For the sake of simplicity, mobile hosts do not provide any service. Instead, services are provided by either 10 or 20 fixed devices for various experiments, in addition to the mobile hosts in the MANET. The packet length at the network layer is arbitrarily set at 1500 bytes, and the query inter-arrival time is exponentially distributed with a mean of 2 minutes. For simplicity, we assume that the RAM available in the network's hosts exists in banks of 2^n kilobytes. For the 50- and 100-host simulations, n may assume a

minimum value of 4 (16 KB), and a maximum within the range [7…11] (i.e., from 128KB to 2MB). For the 200-host simulations, the maximum value for n is increased to the range [*9…13*] (i.e., from 512 KB to 8 MB) for reasons that will be explained later in the next section. Each host has a uniformly distributed probability of being assigned 2^n bytes of memory within the specified ranges, which remains fixed during the course of the current simulation.

Our experiments run a total of 3000 hours of simulated time, which includes a 600-second warm-up period wherein service providers are enabled in a random sequence to allow for a gradual discovery of their service provider peers. FDs discover their peers and exchange information of their advertised services through a simulated TCP connection. The learning system remains disabled during this "warm-up" period, after which user hosts begin issuing service discovery queries and both the learning and SER systems are enabled. Each service entry is comprised of a single description and an arbitrary number of attributes, each of which occupies a geometrically distributed number of bytes. Therefore, the total byte-count per service entry results in an Erlang-distributed value, which is a special case of the Gamma distribution that we use to represent integer-type memory footprints for service entries. The probability of each host's choosing any given service to query is also uniformly distributed.

FDs and learning agents coexist in symbiotic pairs at each service provider. This allows reward values to be individually attributed to each of the agent's actions. Here, we also employ Equation (3.6) seen in Chapter 3, where values for α and β in the hosts' respective learning system is set to 0.1. We also note that learning agents perform their work in a non-cooperative fashion. Therefore, the policies they learn individually are not shared with one another, as explained before. Each MD computes the number of service queries successfully processed as a feedback mechanism of its performance. If an MD observes more packets relayed onto the backbone network than those that are locally processed (resulting in a negative-valued reward),

then the MD surrenders its role. When this happens, all local directory-related information is flushed, and the host assumes a regular status. We tested SDPA with a slightly modified version of SLPv2 to accommodate our needs. Service/attribute requests/replies are routinely processed according to the specification described before. As with SDPA, hosts broadcast a DA_ADVERT to 1-hop neighbours to inform them of the DA's location just discovered upon receiving a SRV_RPLY packet. In other words, FDs do not issue any DA_ADVERT signal in our scheme, but MDs do so as soon as they assume the role. At the network layer, we employ the same routing scheme described in Chapter 3.

4.7 Simulation Results

In this section we evaluate the network packet overhead incurred by the compound SDPP/SERS, and the corresponding success rate for directory location/discovery. In our experiments, the fewer packets attributed to service discovery queries while maintaining a high success rate for directory localization, the better. Once again, we gauge these performance metrics against a baseline broadcast flooding system and against existing multi-directory approaches where possible. Packets are counted in a moving-window fashion by sampling their number every ten simulated hours, at which point the count is reset to 0.

Figures 4.6 and 4.7 illustrate the typical behaviour of two sample sets of measured data in our system once SDPP is put into action under the simulation parameters specified. We employ a simplified nomenclature to represent simulation parameters. For instance, <100 H, 64 KB, 10 P, 1 m/s> refers to a simulation run where 100 hosts are employed, the average memory footprint for each service is 64 KB (service description plus attributes), there are 10 service providers in the MANET, and hosts move at 1 m/s. Each dot in the plots corresponds to the number of packets counted in the corresponding 10-hour period, as explained.

Our first observation is that the learning system is incapable of finding an efficient copy decision policy that yields better system performance when the maximum memory available for directory copies is relatively small. In this case, the constant swapping in and out of the low-SER entries causes oscillation in the learning system when only a maximum of 128 KB are available, as seen in Figure 4.6. For the most part, it can be seen that the majority of the gains attributed to the learning system are attained approximately one-third into the total simulation period. On the other hand, Figure 4.7 illustrates a peculiar behaviour of our system, wherein the limited number of service entries can be effectively handled by a surplus of memory at the hosts. In particular, the random memory allocation up to a maximum of 2 MB leads to the unrestricted replication of service directories to all mobile hosts until they all become MDs and the use of SRV_RQST packets is no longer needed. In other words, each mobile host has a directory copy that it can reference to enquire about any of the available MANET services. To facilitate comparisons with the baseline broadcast flooding approach, we summarize the performance of SDPP in the form of average number of packets measured in 10-hour periods during the last third of the total simulation period according to the parameters employed. For example, Figure 4.16 depicts how the system's packet overhead evolved as the maximum directory memory was gradually increased from one simulation (128 KB) to another (2 MB), as initially shown in Figure 4.6.

Figures 4.8 through 4.15 depict the summarized outcomes for simulations that incorporate 50 hosts. Although results are highly variable, we observe that the performance of the broadcast flooding approach rivals or surpasses more often than not the one yielded by SDPP. In fact, there are only 16 out of 40 cases where SDPP performs equally or better than broadcast flooding. We ascribe this outcome to a combination of possible issues. First, we note that the number of factors that comprise the states of the learning system we devised might be insufficient. In other words, the lack of more information about the natural process (the actual

system state) yields a "subjective" policy that does not match the true optimal policy. Therefore, the learning system obtains an optimal policy relative to the (limited) amount of information at hand about the underlying environment. For instance, the current system state representation conveys no information about the actual physical distribution of service providers in the MANET. In fact, we can see that this particular set of simulations possesses the largest proportion of service providers per host in the MANET. Given that they are uniformly distributed, as little as 10 service providers may suffice to cover the whole deployment area, rendering the use of MDs as unnecessary. On the other hand, service providers in a separate simulation might end up more or less clustered within a particular deployment area. In such cases, the MANET may benefit from a limited use of MDs, and SDPP may conservatively outperform the baseline approach. In either case, SDPP's learning system should be able to learn the best policy. Our results suggest that this is indeed the case when the average performances of both approaches match, which occurs 40% of the time. However, SDPP was unable to find the "use no MDs" optimal policy in the remaining 60% of our 50-host simulations. Other factors may also be partially responsible for this outcome, such as the policy employed to choose an action (ϵ-greedy), or the values employed for α and β, all of which have been shown to exert strong influence on the final results of a Q-learning system [86].

Figures 4.16 through 4.23 depict the summarized outcomes for simulations that incorporate 100 hosts. Unlike the 50-host simulation set, the number and/or physical position of service providers might not always be enough to cover the whole deployment area, which hints at to the need to use MDs more often. In fact, this time SDPP performed better than the baseline approach in roughly 75% of the time. Simulation results reveal reductions of service discovery traffic typically ranging from 15 to 75% when SDPP is employed. These results also allow us to make more detailed assessments of SDPP's behaviour. For instance, we can observe a dramatic

decrease in packet overhead as the maximum allotted directory space is increased from 128 KB to 256 KB (from one simulation outcome to the next) when the memory footprint of service entries averages 64 KB, hosts move at 1 m/s, and there are 10 service providers in the network (see Figure 4.16). Similar arguments can be made for the case of service entries consuming on average 128 KB, as shown in the corresponding plot of Figure 4.17. Here it can be seen that the byte-size of the services' memory footprint has little effect on the system's performance.

Variations in the simulations' parameters have a significant impact on the performance of SDPP. The largest of these are observed when there are 20 service providers in the network. These variations in overall performance are attributed to the wide range of values allocated for directory memory in each host at the beginning of the simulations. We regard this as a reasonably realistic expectation in an actual MANET setting. Consequently, the overall service discovery system underperforms when the subset of hosts that picks smaller memory amounts outnumbers those that allocate larger amounts. In these cases, MDs with less memory will see a larger number of service entries swapped in and out of the local directory, leading to increased bandwidth consumption until the learning system decides not to use these hosts as MDs. Again, in the extreme case, no MDs are employed, and the pure-FD approach is favoured. To the contrary, a MANET sees improved performance as more mobile hosts allocate directory memory on the higher end of the predefined range. This can be confirmed in those cases where hosts have enough memory to store all of the available services' information, so that each of them becomes an MD, as explained before. Therefore, these performance variations are not necessarily attributable to SDPP, but to the MANET hosts' intrinsic operating parameters. Our proposed system learns to work in a best-effort fashion, instantiating as many MDs in hosts that are able to allocate larger amounts of memory for directory use in order to avoid larger performance degradations.

Figures 4.24 through 4.31 depict the summarized outcomes for simulations that incorporate 200 hosts. This simulation set yielded the best outcome of our proposed system against the broadcast-flooding approach, with 90% of the results favouring the use of SDPP. As in the 100-host case, performance improvements are highly variable too. These improvements may range from one being fairly conservative to one spanning nearly a whole order of magnitude, as depicted in the 512 KB case of Figure 4.24. We recall that, compared to the preceding simulation sets, the maximum amount of memory allocated to service directories (as a simulation parameter) here was increased. The reason for this is that a larger number of MANET hosts translates into service entries being queried more often, causing a more frequent swapping in/out of service entries in the directories at a higher rate. This hinders the policy learning process, as observed in the 128 KB case of Figure 4.6 explained before. By increasing the range of the maximum amount of memory allocated to service directories, we allow for more entries that can be kept for longer periods of time until they are eventually swapped out, thus reducing traffic, and facilitating the job of the LA. As in previous experiments, there were instances in which the system seems to reach a stabilization point when 1 MB of memory was allocated to the directory, though this is not always the case. Our results also reveal that SDPP introduces improved scalability, as evidenced by a small increase in service discovery traffic from the 50-host case to the 200-host case. On the other hand, the traffic performance observed for the broadcast-flooding approach increases significantly, as expected. We also note that although the number of service providers remains the same as in the previous simulations, the deployment area is conversely increased. As a result, larger portions of the deployment area that are not covered by the DAs can be effectively covered by MDs, thus leveraging their overall usefulness.

Tables 4.1 through 4.6 depict the discovery success rates of our simulations. In accordance with the results obtained for packet traffic, the discovery success rate of the baseline

approach often meets or surpasses the one obtained by SDPP in the 50-host case. However, results become more favourable for SDPP as the size of the network increases to values nearing the full 100% mark. Our simulations also reveal that the use of SDPP indeed meets or exceeds the performance reported by the approaches referenced for comparison in Chapter 3. In particular, we revisit the case of GSD, which is often cited in the literature, and is the stronger contender for SDPP. We recall the case of Fig. 13(c)/(d) in [48] also mentioned in Chapter 3, where a network load that amounts to roughly 100,000 messages in a 50-host MANET after 75 simulated minutes is reported. Contrary to SDPA's performance, SDPP's multi-directory approach yields roughly the same amount of packets as GSD in a MANET comprised by 100 hosts. In other words, SDPP incurs the same amount of network traffic in a MANET that is twice as large. However, if the application-layer packets that they report translate into two or more network-layer packets, then SDPP's superiority would be further confirmed. In fact, SDPP incurs roughly this same traffic load in several of the 200-host cases measured when enough directory memory is available. Unfortunately, more detailed comparisons become impractical since the results reported in [48] do not present a breakdown of GSD's performance for every MANET size. In addition, their proposed system does not take into consideration memory availability. Therefore, GSD's performance with this constraint is unknown. As for the discovery success rate, SDPP's performance still meets or exceeds the one reported for GSD's and the other approaches previously referenced.

4.8 Summary

We have shown the effectiveness and limitations of employing both fixed and mobile directories for service discovery in mobile computing environments. Once again the formulation of SDPP's as an SMDP and its solution through the Q-Learning technique proved helpful in defining a policy that saves bandwidth versus using a purely heuristic approach. The learning

system was adapted to meet the needs of our multi-directory approach, and the drawback of working with limited system state information was discussed. We showed that allowing directories to become mobile can leverage the overall system's performance, which is also highly dependent on the MANET's size and the directory memory availability. SDPP was shown to perform in a best-effort fashion, depending on the operating parameters of the MANET. The more memory that hosts allocate to the service directory, the better that SDPP's performance becomes. Limited amount of literature data suggests that the mixed multi-directory scheme of SDPP is also able to either meet or exceed the performance seen in other approaches. As for practical consideration issues, we observe that the same arguments apply for SDPP as in SDPA. Given the relatively small number of states observed by the learning system, a few tens of kilobytes would suffice to store the whole Q-table.

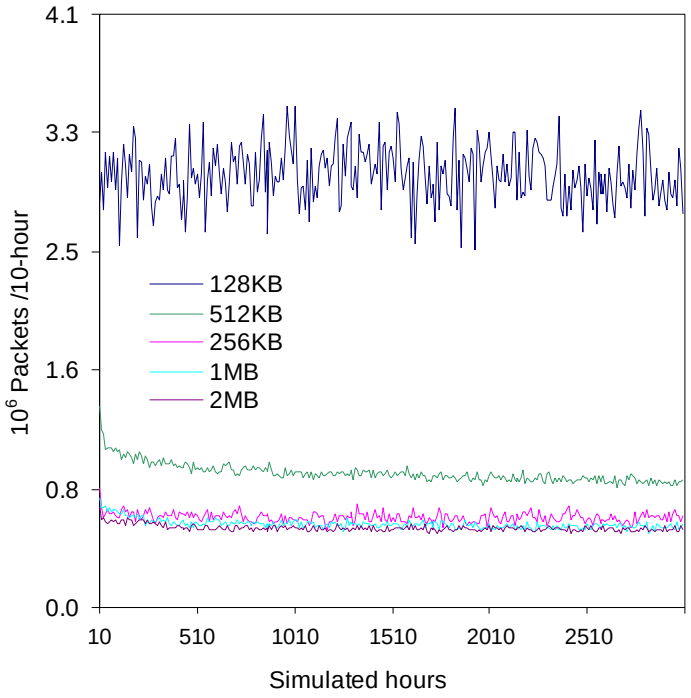

Figure 4.6 Compound System behaviour with parameters 100 H, 64 KB, 10 P, 1 m/s.

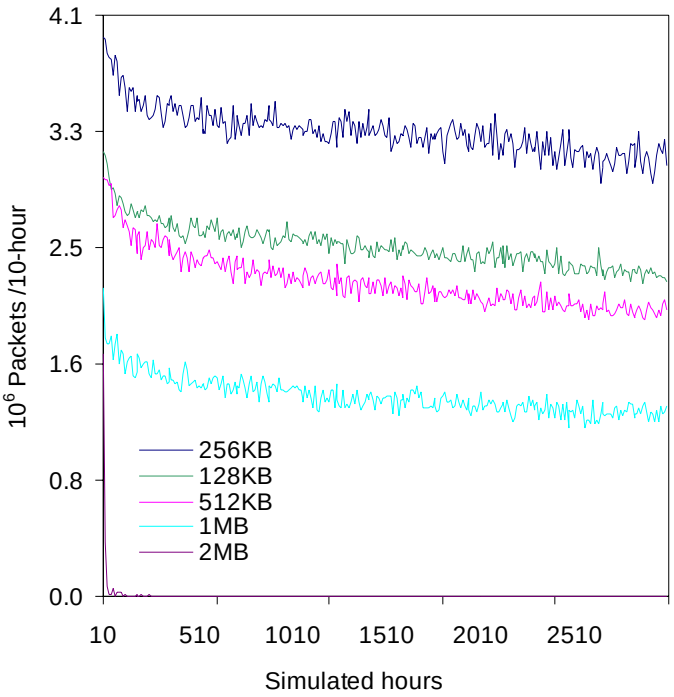

Figure 4.7 Compound System behaviour with parameters 100H, 128 KB, 20 P, 2 m/s

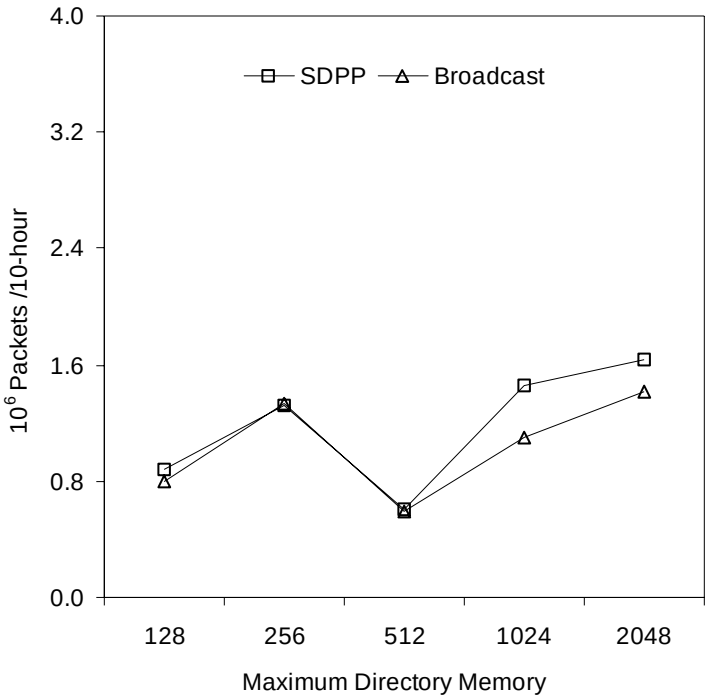

Figure 4.8 Summarized System behaviour with parameters 50 H, 64 KB, 10 P, 1 m/s.

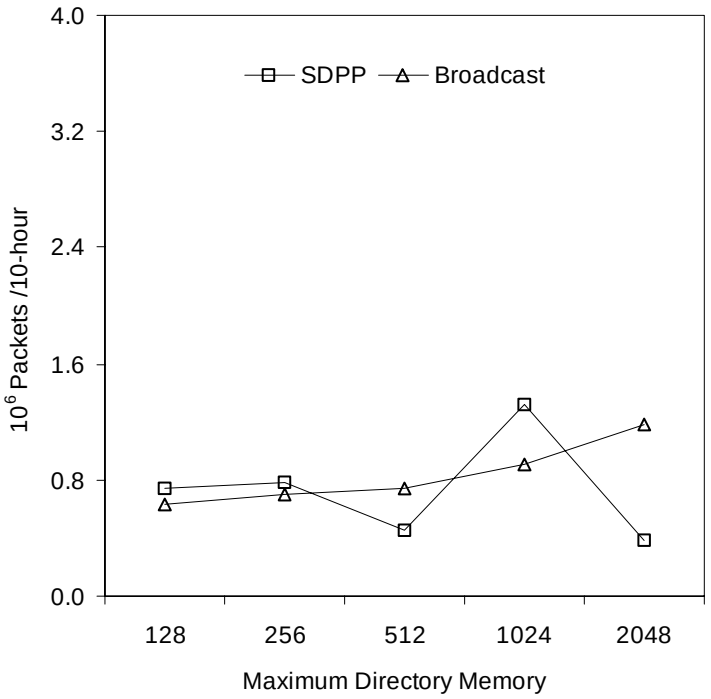

Figure 4.9 Summarized System behaviour with parameters 50 H, 128 KB, 10 P, 1 m/s.

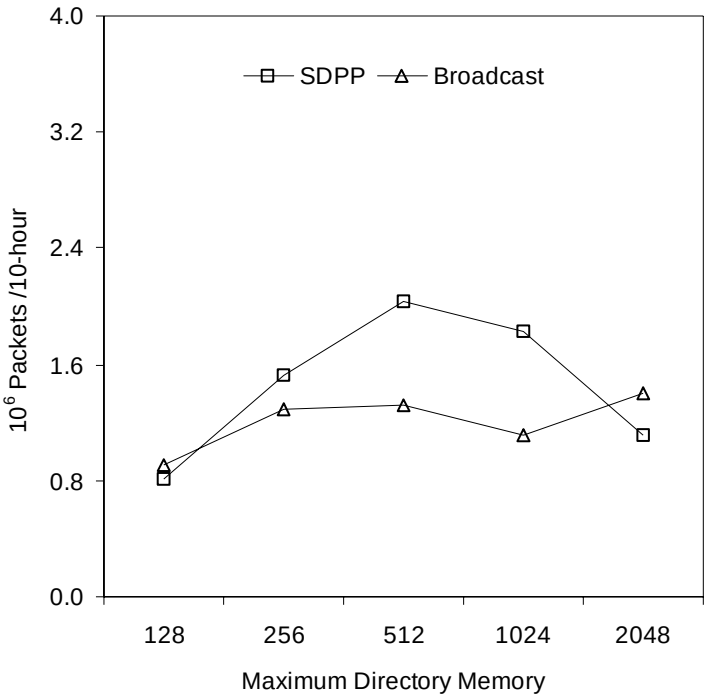

Figure 4.10 Summarized System behaviour with parameters 50 H, 64 KB, 20 P, 1 m/s.

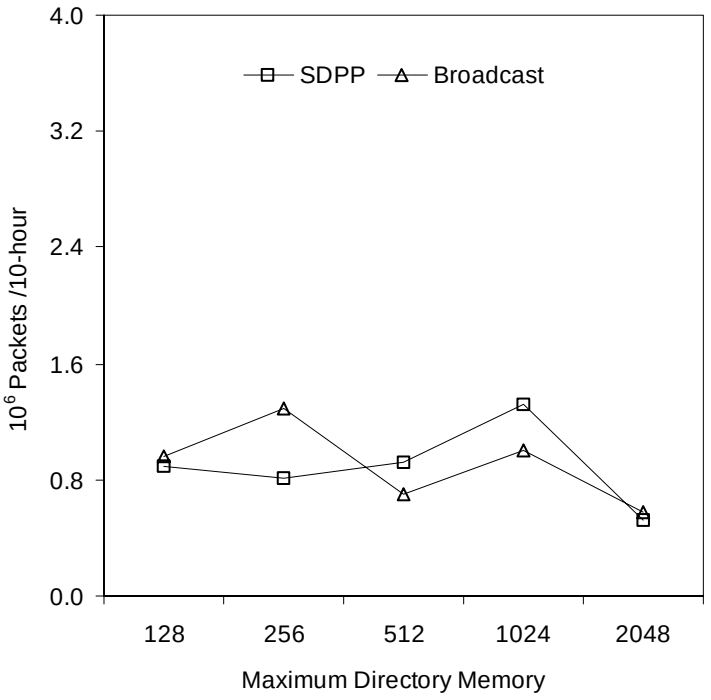

Figure 4.11 Summarized System behaviour with parameters 50 H, 128 KB, 20 P, 1 m/s.

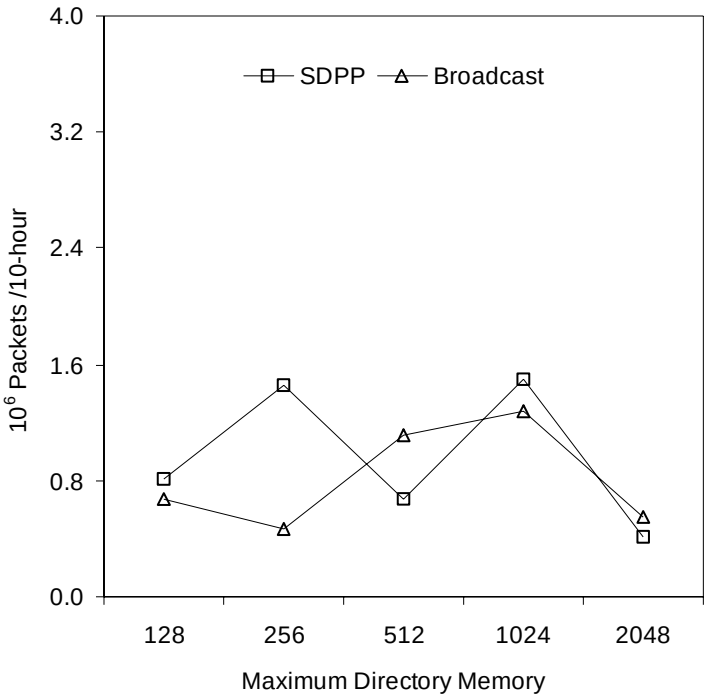

Figure 4.12 Summarized System behaviour with parameters 50 H, 64 KB, 10 P, 2 m/s.

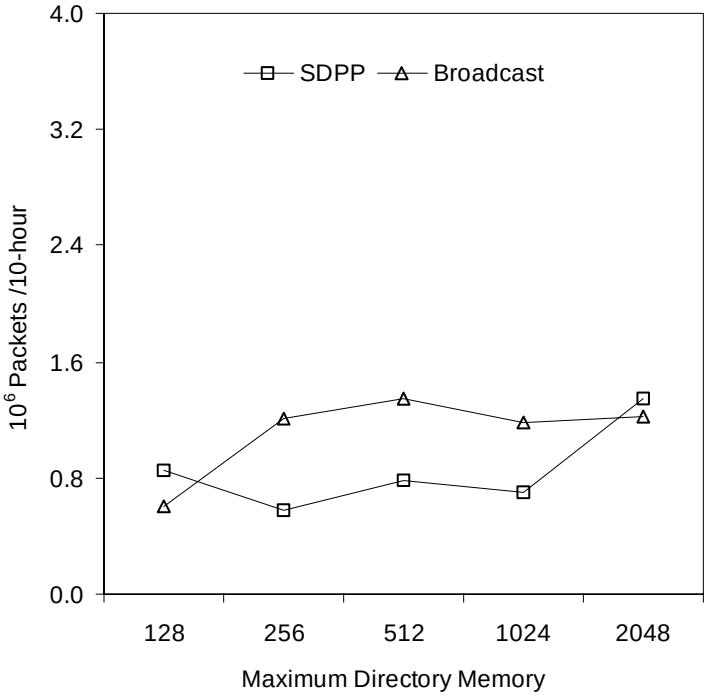

Figure 4.13 Summarized System behaviour with parameters 50H, 128 KB, 10 P, 2 m/s.

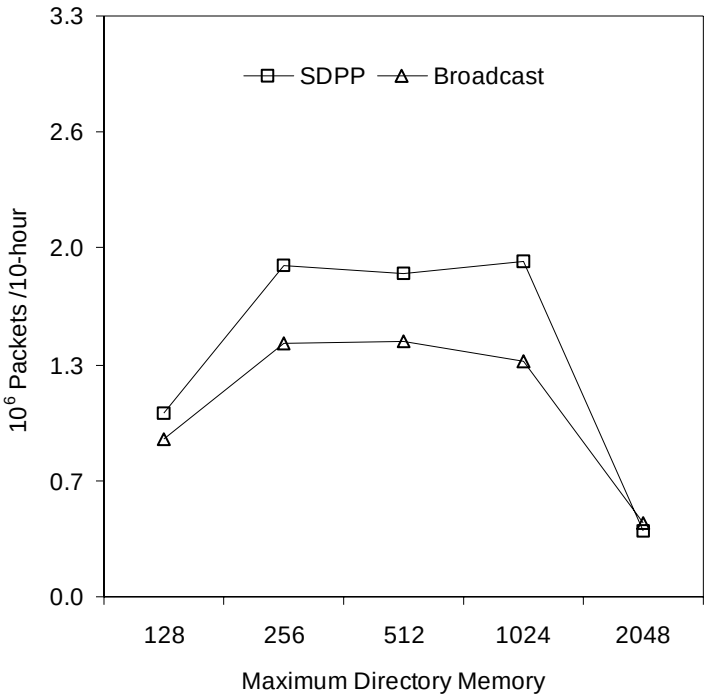

Figure 4.14 Summarized System behaviour with parameters 50H, 64 KB, 20 P, 2 m/s.

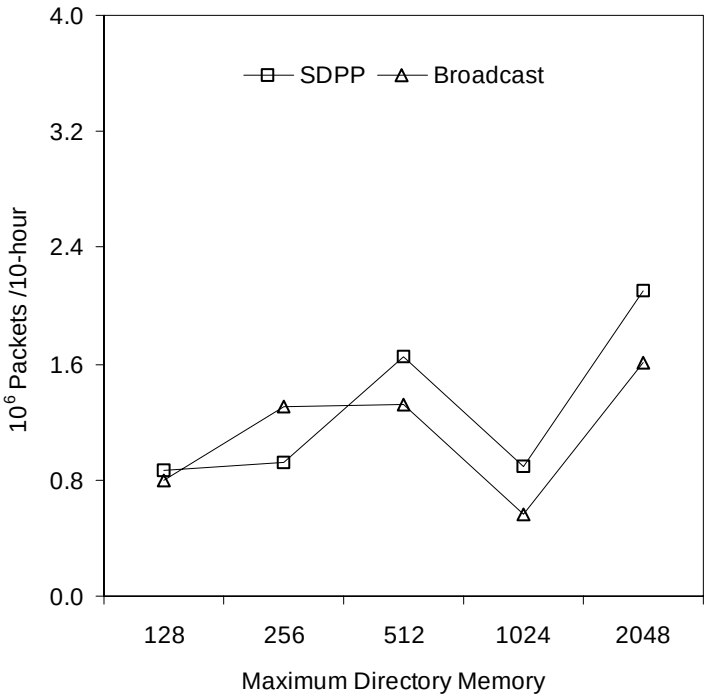

Figure 4.15 Summarized System behaviour with parameters 50H, 128 KB, 20 P, 2 m/s.

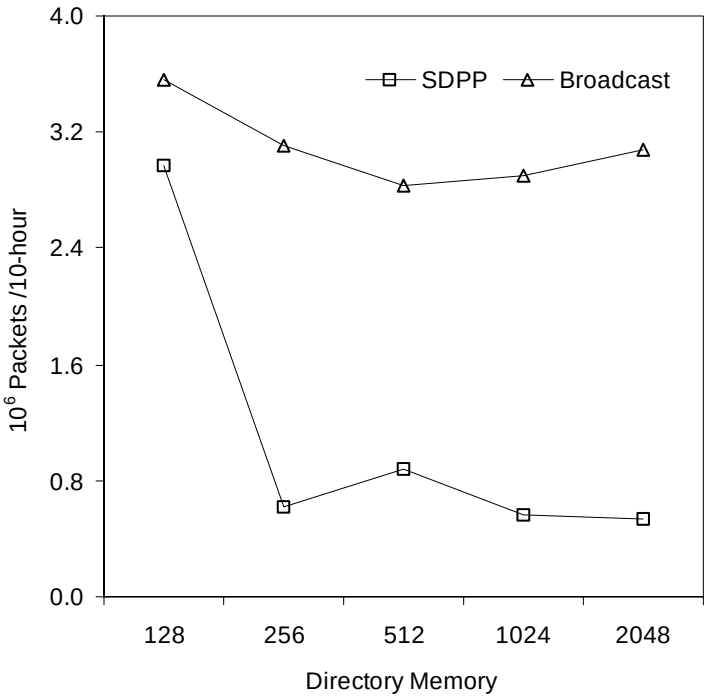

Figure 4.16 Summarized System behaviour with parameters 100 H, 64 KB, 10 P, 1 m/s.

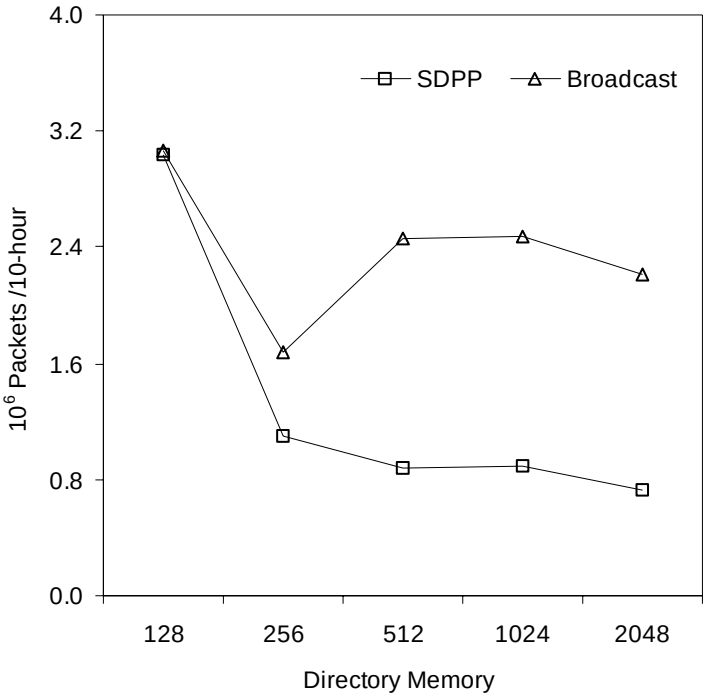

Figure 4.17 Summarized System behaviour with parameters 100H, 128KB, 10P, 1 m/s.

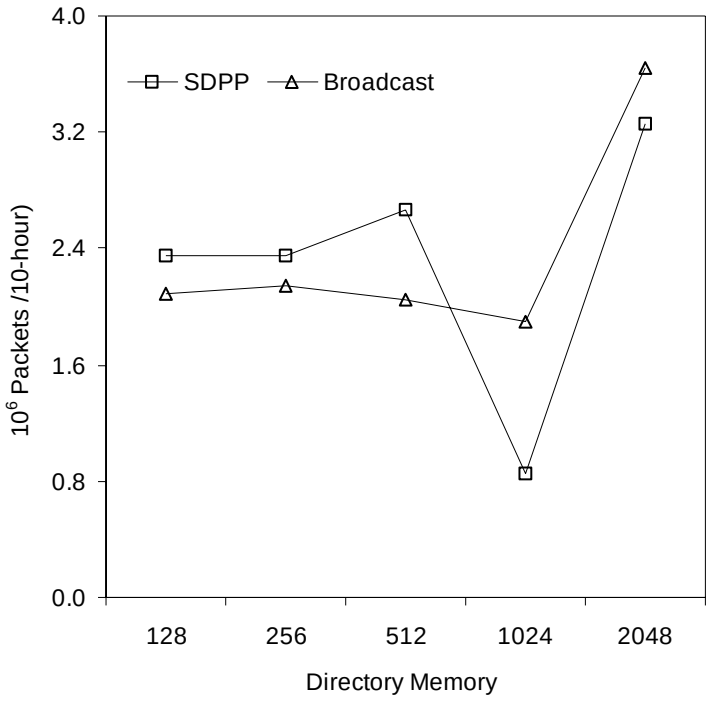

Figure 4.18 Summarized System behaviour with parameters 100H, 64 KB, 20 P, 1 m/s.

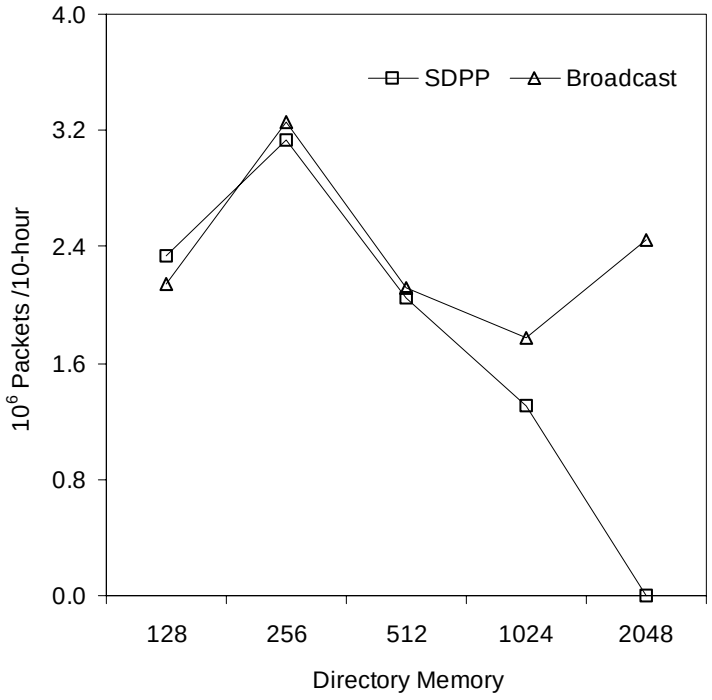

Figure 4.19 Summarized System behaviour with parameters 100 H, 128 KB, 20 P, 1 m/s.

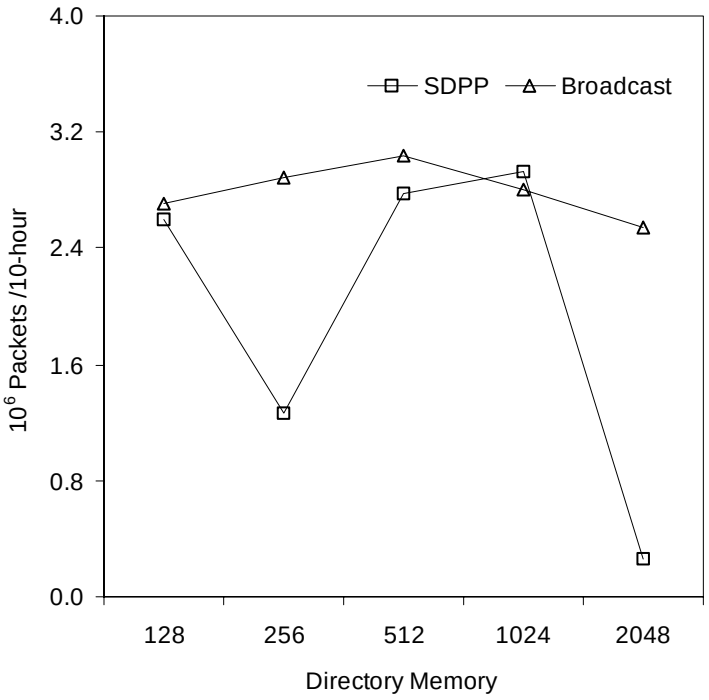

Figure 4.20 Summarized System behaviour with parameters 100 H, 64 KB, 10 P, 2 m/s.

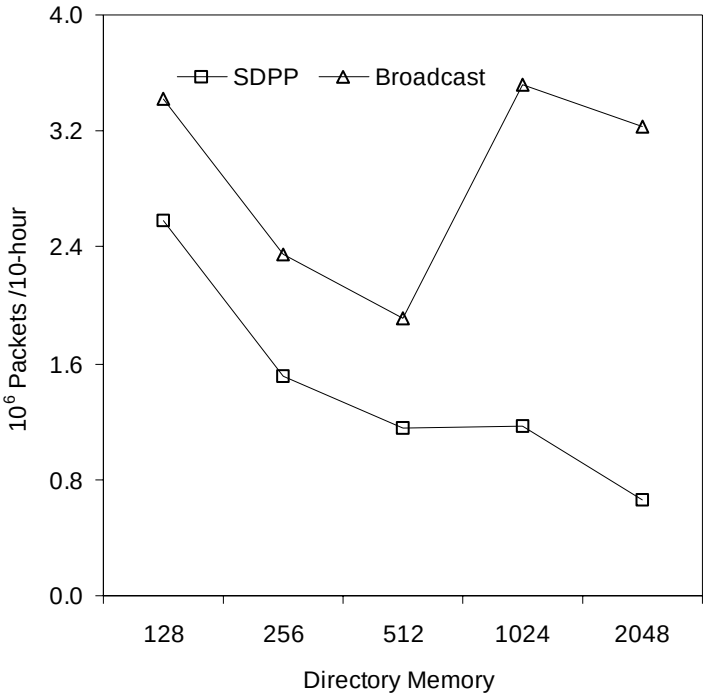

Figure 4.21 Summarized System behaviour with parameters 100H, 128 KB, 10 P, 2 m/s.

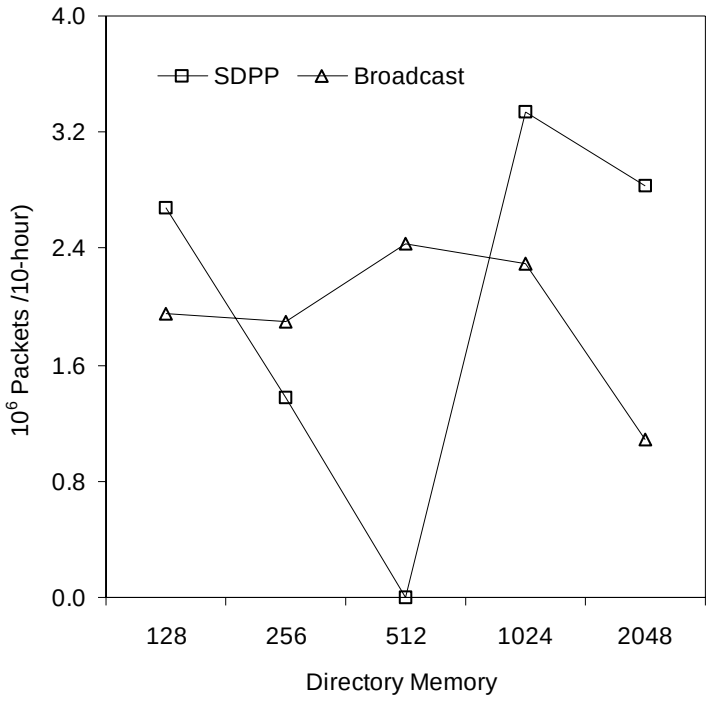

Figure 4.22 Summarized System behaviour with parameters 100 H, 64 KB, 20 P, 2 m/s.

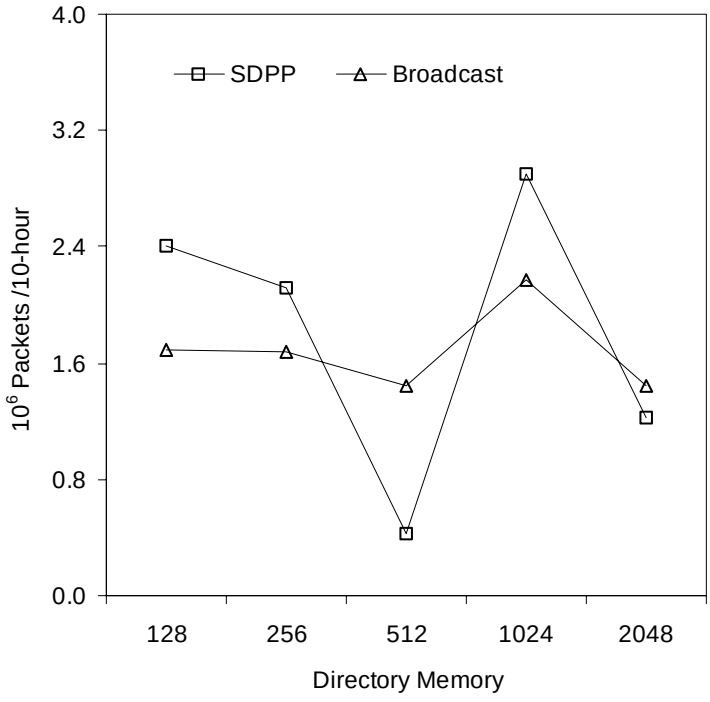

Figure 4.23 Summarized System behaviour with parameters 100 H, 128 KB, 20 P, 2 m/s.

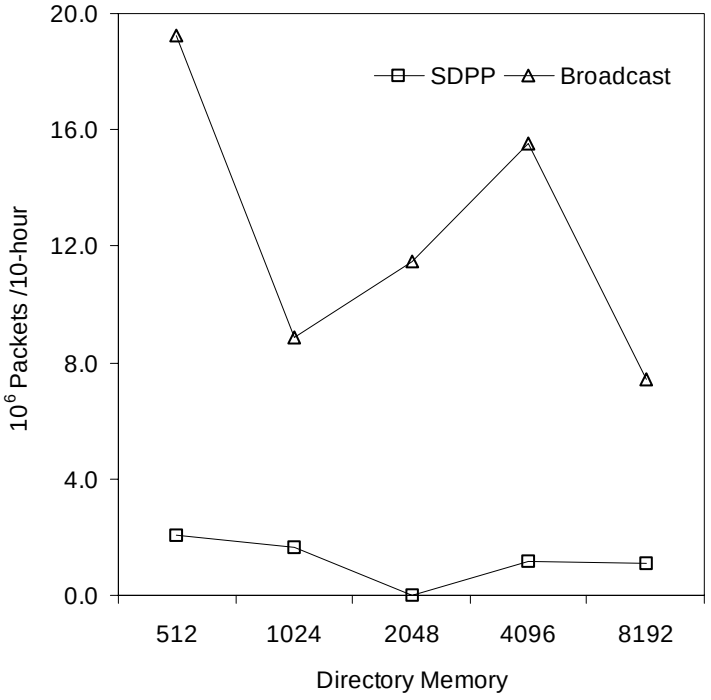

Figure 4.24 Summarized System behaviour with parameters 200H, 64 KB, 10 P, 1 m/s.

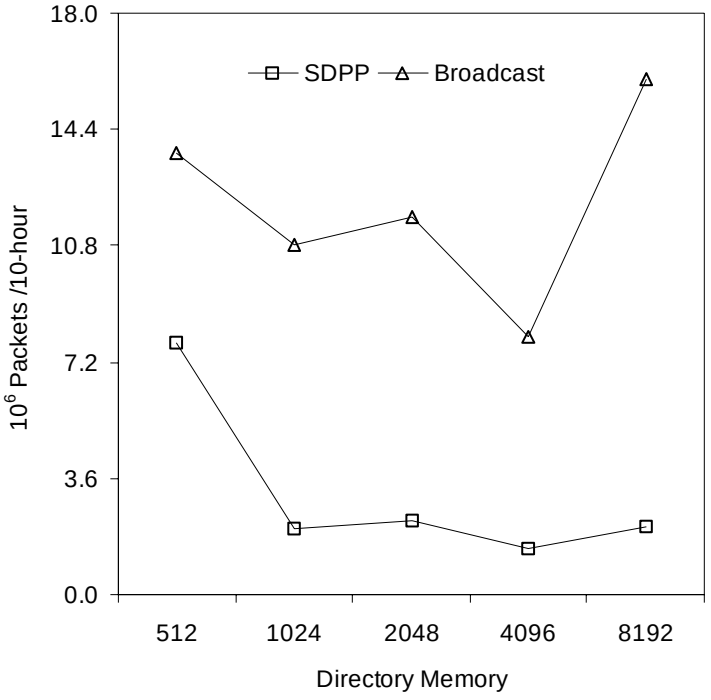

Figure 4.25 Summarized System behaviour with parameters 200H, 128 KB, 10 P, 1 m/s.

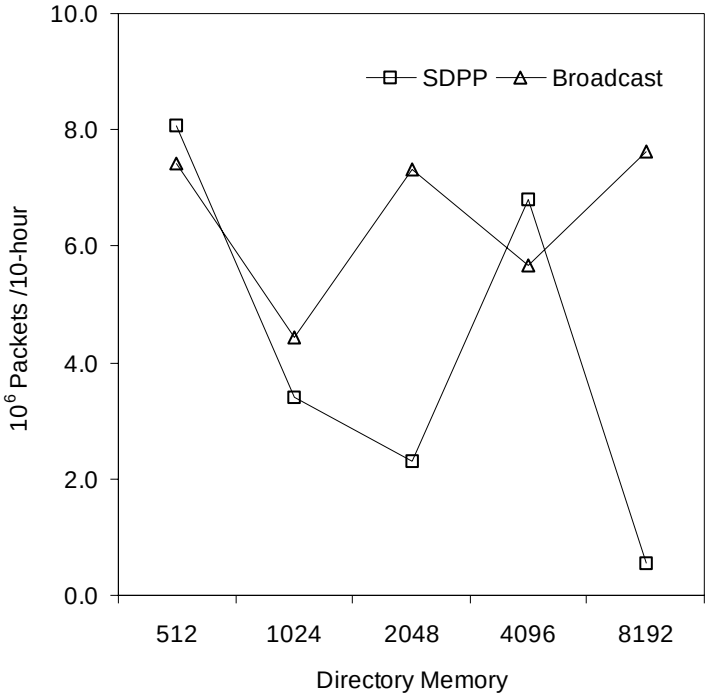

Figure 4.26 Summarized System behaviour with parameters 200H, 64 KB, 20 P, 1 m/s.

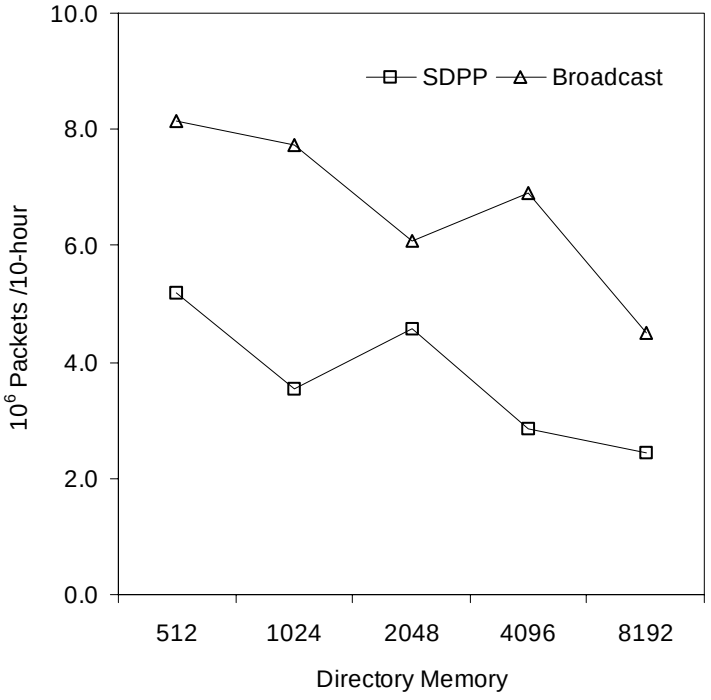

Figure 4.27 Summarized System behaviour with parameters 200H, 128 KB, 20 P, 1 m/s.

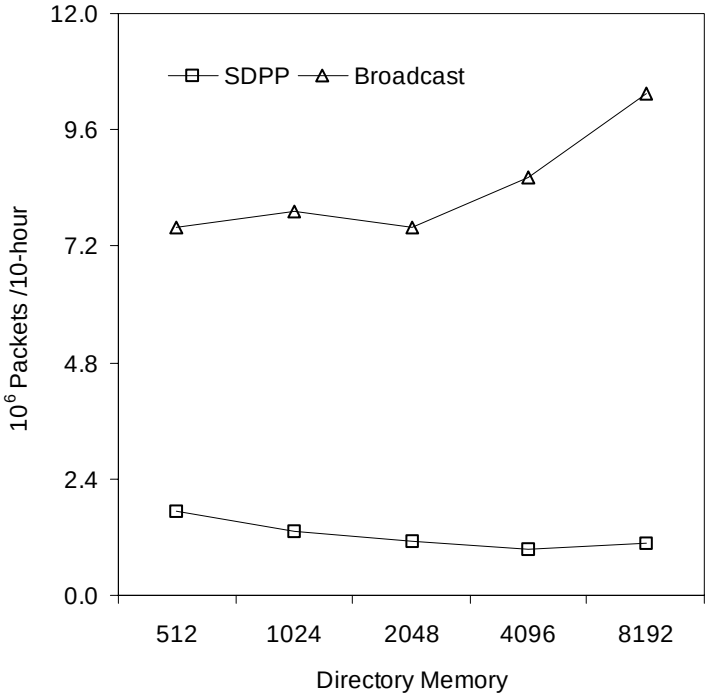

Figure 4.28 Summarized System behaviour with parameters 200H, 64 KB, 10 P, 2 m/s.

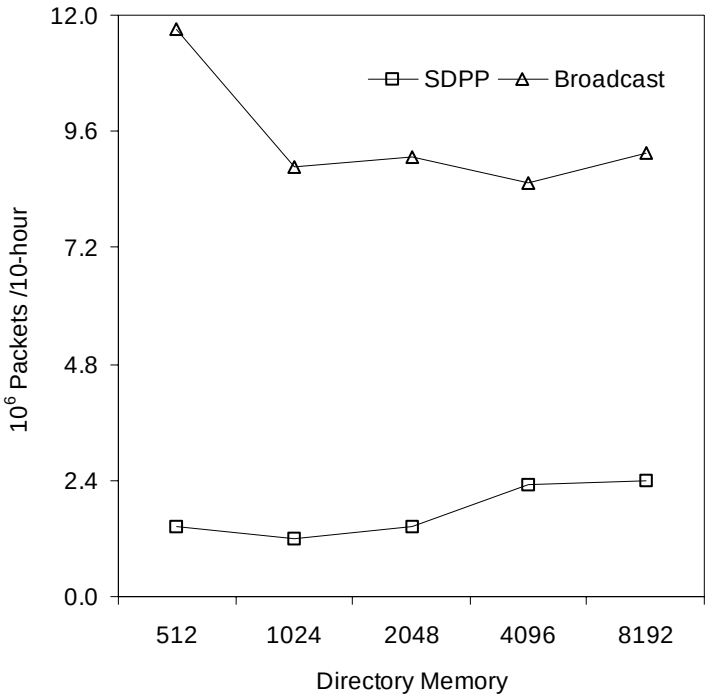

Figure 4.29 Summarized System behaviour with parameters 200H, 128 KB, 10 P, 2 m/s.

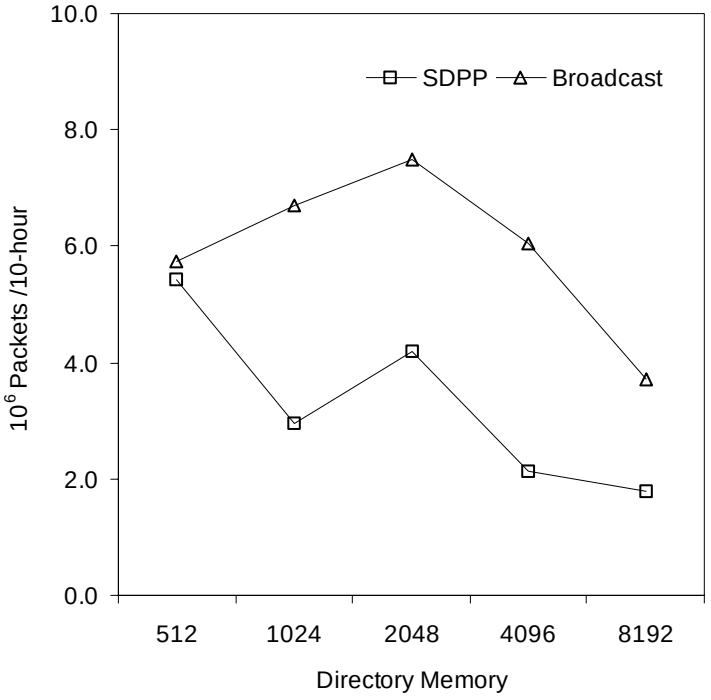

Figure 4.30 Summarized System behaviour with parameters 200H, 64 KB, 20 P, 2 m/s.

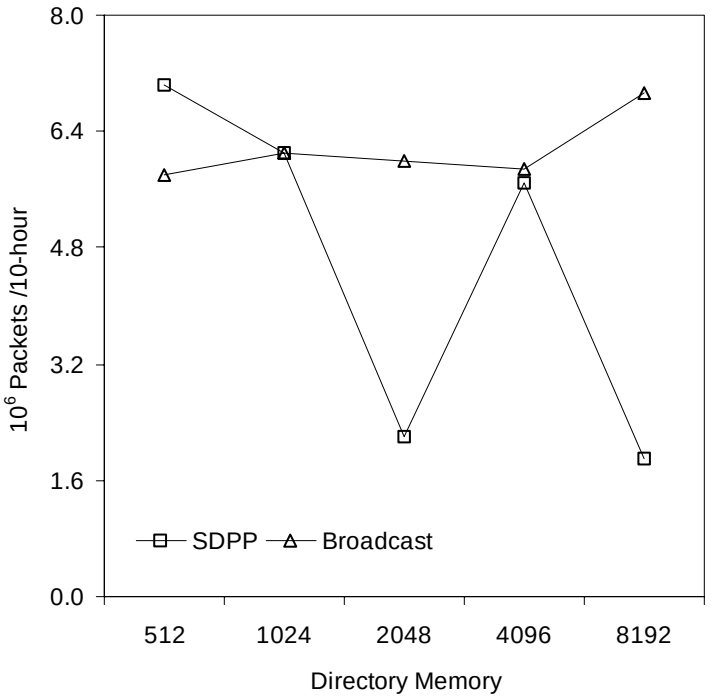

Figure 4.31 Summarized System behaviour with parameters 200H, 128 KB, 20 P, 2 m/s.

Table 4.1. Directory Localization Success Rate for a network of 50 hosts when SDPP is used.

Success rate for hosts moving @ 1 m/s			
Average entry footprint 64 KB		*Average entry footprint 128 KB*	
Directory size	Number of service providers 10/20	Directory size	Number of service providers 10/20
128KB	0.94 / 0.99	128KB	0.95 / 0.98
256KB	0.97 / 0.99	256KB	0.96 / 0.99
512KB	0.96 / 0.99	512KB	0.94 / 0.96
1024KB	0.98 / 0.99	1024KB	0.93 / 0.99
2048KB	0.97 / 0.99	2048KB	0.96 / 0.99
Success rate for hosts moving @ 2 m/s			
Average entry footprint 64 KB		*Average entry footprint 128 KB*	
Directory size	Number of service providers 10/20	Directory size	Number of service providers 10/20
128KB	0.93 / 0.99	128KB	0.96 / 0.99
256KB	0.98 / 0.99	256KB	0.97 / 0.99
512KB	0.98 / 0.99	512KB	0.94 / 0.99
1024KB	0.95 / 0.99	1024KB	0.94 / 0.99
2048KB	0.96 / 0.99	2048KB	0.97 / 0.98

Table 4.2. Directory Localization Success Rate for a network of 50 hosts when broadcast-flooding is used.

Success rate for hosts moving @ 1 m/s			
Average entry footprint 64 KB		*Average entry footprint 128 KB*	
Directory size	Number of service providers 10/20	Directory size	Number of service providers 10/20
128KB	0.91 / 0.98	128KB	0.95 / 0.98
256KB	0.95 / 0.98	256KB	0.98 / 0.99
512KB	0.96 / 0.99	512KB	0.96 / 0.96
1024KB	0.98 / 0.99	1024KB	0.94 / 0.99
2048KB	0.94 / 0.99	2048KB	0.96 / 0.99
Success rate for hosts moving @ 2 m/s			
Average entry footprint 64 KB		*Average entry footprint 128 KB*	
Directory size	Number of service providers 10/20	Directory size	Number of service providers 10/20
128KB	0.93 / 0.99	128KB	0.93 / 0.99
256KB	0.97 / 0.98	256KB	0.97 / 0.99
512KB	0.98 / 0.98	512KB	0.92 / 0.99
1024KB	0.93 / 0.98	1024KB	0.94 / 0.99
2048KB	0.93 / 0.98	2048KB	0.91 / 0.97

Table 4.3. Directory Localization Success Rate for a network of 100 hosts when SDPP is used.

Success rate for hosts moving @ 1 m/s			
Average entry footprint 64 KB		*Average entry footprint 128 KB*	
Directory size	Number of service providers 10/20	Directory size	Number of service providers 10/20
128KB	0.97 / 0.96	128KB	0.96 / 0.97
256KB	0.98 / 0.99	256KB	0.95 / 0.99
512KB	0.98 / 0.98	512KB	0.95 / 0.99
1024KB	0.96 / 0.98	1024KB	0.97 / 0.98
2048KB	0.98 / 0.98	2048KB	0.96 / 0.99
Success rate for hosts moving @ 2 m/s			
Average entry footprint 64 KB		*Average entry footprint 128 KB*	
Directory size	Number of service providers 10/20	Directory size	Number of service providers 10/20
128KB	0.95 / 0.96	128KB	0.96 / 0.97
256KB	0.97 / 0.99	256KB	0.99 / 0.99
512KB	0.96 / 0.99	512KB	0.98 / 0.99
1024KB	0.94 / 0.96	1024KB	0.95 / 0.97
2048KB	0.99 / 0.98	2048KB	0.99 / 0.99

Table 4.4. Directory Localization Success Rate for a network of 100 hosts when broadcast-flooding is used.

Success rate for hosts moving @ 1 m/s			
Average entry footprint 64 KB		*Average entry footprint 128 KB*	
Directory size	Number of service providers 10/20	Directory size	Number of service providers 10/20
128KB	0.95 / 0.96	128KB	0.94 / 0.95
256KB	0.96 / 0.98	256KB	0.91 / 0.97
512KB	0.95 / 0.97	512KB	0.90 / 0.95
1024KB	0.94 / 0.97	1024KB	0.94 / 0.99
2048KB	0.96 / 0.97	2048KB	0.99 / 0.96
Success rate for hosts moving @ 2 m/s			
Average entry footprint 64 KB		*Average entry footprint 128 KB*	
Directory size	Number of service providers 10/20	Directory size	Number of service providers 10/20
128KB	0.93 / 0.97	128KB	0.94 / 0.96
256KB	0.95 / 0.98	256KB	0.93 / 0.98
512KB	0.91 / 0.99	512KB	0.92 / 0.99
1024KB	0.92 / 0.95	1024KB	0.94 / 0.97
2048KB	0.97 / 0.96	2048KB	0.95 / 0.99

Table 4.5. Directory Localization Success Rate for a network of 200 hosts when SDPP is used.

Success rate for hosts moving @ 1 m/s			
Average entry footprint 64 KB		*Average entry footprint 128 KB*	
Directory size	Number of service providers 10/20	Directory size	Number of service providers 10/20
512KB	0.96 / 0.96	512KB	0.96 / 0.95
1024KB	0.96 / 0.96	1024KB	0.95 / 0.97
2048KB	0.99 / 0.98	2048KB	0.98 / 0.99
4096KB	0.97 / 0.97	4096KB	0.96 / 0.98
8192KB	0.98 / 0.99	8192KB	0.96 / 0.96
Success rate for hosts moving @ 2 m/s			
Average entry footprint 64 KB		*Average entry footprint 128 KB*	
Directory size	Number of service providers 10/20	Directory size	Number of service providers 10/20
512KB	0.95 / 0.94	512KB	0.95 / 0.97
1024KB	0.96 / 0.97	1024KB	0.96 / 0.97
2048KB	0.97 / 0.98	2048KB	0.96 / 0.98
4096KB	0.97 / 0.99	4096KB	0.96 / 0.97
8192KB	0.96 / 0.97	8192KB	0.96 / 0.98

Table 4.6. Directory Localization Success Rate for a network of 200 hosts when broadcast-flooding is used.

Success rate for hosts moving @ 1 m/s			
Average entry footprint 64 KB		*Average entry footprint 128 KB*	
Directory size	Number of service providers 10/20	Directory size	Number of service providers 10/20
512KB	0.88 / 0.95	512KB	0.92 / 0.93
1024KB	0.90 / 0.95	1024KB	0.85 / 0.95
2048KB	0.96 / 0.96	2048KB	0.92 / 0.96
4096KB	0.88 / 0.96	4096KB	0.90 / 0.95
8192KB	0.94 / 0.96	8192KB	0.84 / 0.96
Success rate for hosts moving @ 2 m/s			
Average entry footprint 64 KB		*Average entry footprint 128 KB*	
Directory size	Number of service providers 10/20	Directory size	Number of service providers 10/20
512KB	0.90 / 0.91	512KB	0.84 / 0.95
1024KB	0.92 / 0.92	1024KB	0.89 / 0.95
2048KB	0.92 / 0.94	2048KB	0.88 / 0.96
4096KB	0.94 / 0.96	4096KB	0.89 / 0.94
8192KB	0.79 / 0.93	8192KB	0.91 / 0.93

Chapter 5. A Mobile Code Platform for Service-Oriented Tasks in Wireless Sensor Networks*

5.1 Introduction

In the past chapters, we studied alternative methods for topology formation and service discovery, where the networks under consideration were treated as the means to provide one or more types of services made available by individual devices. In this chapter, we consider a special kind of wireless ad-hoc network that can be regarded as a provider rather than an enabler of services. As discussed before, considerable research efforts are currently being targeted at WSNs employed for a variety of applications. The intrinsic uncertainties in the behaviour of the environments where WSNs are deployed have led researchers to seek efficient re-tasking solutions to address unforeseen circumstances. In other words, it is highly desirable that the application or service that WSNs provide be programmable. As explained previously, this WSN programmability can be achieved through the use of Mobile Agent Systems, or MAS. The individual architectures of existing MAS for WSN determine the methodology employed to achieve these objectives and the programming style devised [109].

In this chapter, we explore the benefits of employing a service-oriented re-tasking approach in WSNs. We first note that similar approaches attempt to achieve a balance between the functionality incorporated into the actual code interpreter, and the one provided to the agents. On the one hand, a coarse-grained agent system incorporating a high degree of functionality in the interpreter entails simple constructs on the agent side to accomplish a certain task. On the other hand, a fine-grained code interpreter requires the design of a language construct comprehensive enough to accomplish all of the desired functionalities targeted at a wide range of

* This chapter is based on a paper published in IEEE Wireless Communications [113].

WSN applications. As with other software systems, a coarse-grained control provided by the MAS leads to the incorporation of a high-level of code abstraction, whereas fine-grained control can only be achieved through low-level code abstraction. We promote the use of a programmable system based on mobile codes that enables the efficient control of spatial tasks in WSNs by means of the WISEMAN framework. The rest of this chapter is organized as follows. Section 5.2 explains the proposed deployment setting for WISEMAN. Section 5.3 explains the interpreter architecture of WISEMAN, and Section 5.4 describes its language constructs. Section 5.5 presents a discussion on WISEMAN, accompanied by sample application scenarios. Section 5.6 discusses the implementation details of WISEMAN. Concluding remarks are presented in Section 5.7.

5.2 Deployment Setting

Since WSN design is specifically tailored to meet the needs of a particular application, it also makes sense for MAS to be customized, so that agents can efficiently accomplish their tasks in the corresponding deployment setting. We consider multi-hop WSNs comprised of stationary sensor nodes, possibly deployed in large and open geographical areas, and assume that nodes may be heterogeneous with respect to their hardware and software platforms. We also allow for the possibility of external transitory devices becoming part of the WSN. In addition, although a network gateway is typically assumed to be the entry point of mobile codes, our system allows the codes' injection at any node. The reason for this is that relying on a single entry/exit point for mobile codes or raw data injection may render the whole WSN useless should the gateway become unavailable. The same applies if one or more nodes serving as data relays between the sensor nodes and the gateway fail. In addition, the WSN may support device heterogeneity as an inherent characteristic of its application during network re-tasking [110] – [111]. For example, consider a WSN deployed in a sub-alpine setting used primarily for avalanche prediction

purposes as shown in Figure 5.1. Here, the WSN may switch from a monitoring service mode to a rescue operation support mode. In the latter case, on-site access to the WSN by devices possibly different from the typical sensor nodes and the ability to bypass the WSN gateway becomes critical.

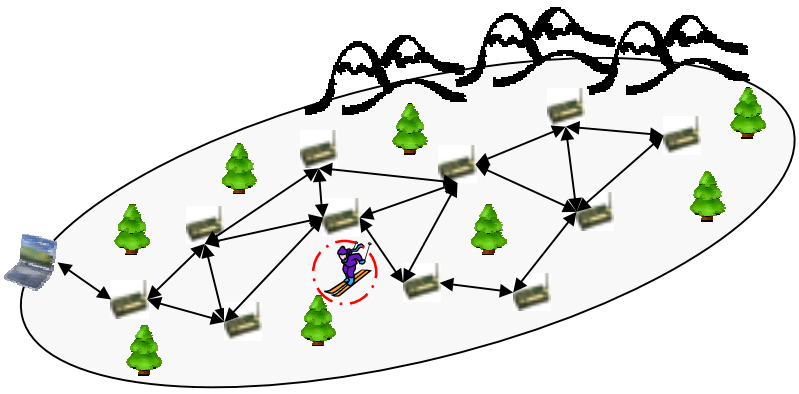

Figure 5.1 Sample deployment setting for a WSN

5.3 The WISEMAN Framework for WSNs

In view of the circumstances surrounding the applicability of MAS to WSNs, we propose an alternative approach that incorporates coarse-grained programmability to support distributed task control. We rely on our previous experience with mobile codes in data networks, and propose a simplified version of the original Wave system (initially referred to in Chapter 2) for its incorporation into WSNs, which we refer to as WISEMAN [112], [113].

The Wave system can be considered as one of the precursors of code mobility in data networks [70], [114], and is founded on the idea of efficient task coordination in distributed environments. Wave's high-level language construct allows the creation of highly-compact programs that encompass appropriate degrees of distributed coordination. Additionally, the Wave system promotes the use of native code programs (external to the interpreter) to execute complex algorithms that are constantly invoked by the agents. Consequently, the need to carry

additional code is avoided, promoting system efficiency in terms of execution speed and code compactness that enable potential energy savings. As a result:

- The process evolution part of the distributed application is decoupled from the data processing element.
- Agent compactness can be achieved by defining language constructs that are sufficient to reduce its forwarding time as well as to describe the desired coordination methodology of the mobile process.
- A condensed language construct translates into a simplified interpreter's architecture, which in turn helps to reduce its memory footprint.

The original Wave system is structured into four layers: code, track, knowledge and computer. Here, mobile programs ("waves") are injected into a computer network whose hosts are assumed to possess a Wave code interpreter. Initially, Wave programs are used to setup semantic relationships between hosts according to predefined conditions specified into the agent codes, forming a subset of hosts known as a knowledge network (i.e., a virtual overlay network). One or more agents can then be employed to navigate these virtual links in order to perform a secondary task. When migrating between hosts, agents leave "tracks" that are subsequently used by the interpreters to back-propagate signals, also called *echoes*, between them. Therefore, depending on the outcome of the local computations or situation assessments they make, Wave agents instruct the local interpreter to issue echo signals that may reach the interpreters in one or more hosts to coordinate the evolution of the mobile process in a tokenized style.

The proposed architecture of our WISEMAN MAS is shown in Figure 5.2. While several of the language constructs of the original Wave interpreter are kept, others are simplified in order to reduce the complexity of the interpreter. We decided not to incorporate the Track Layer to reduce the memory footprint of the interpreter, given the significant amount of complexity that it

introduces [74]. Also, the number of control fields comprising the agent packets has been simplified. This contributes to reduced energy expenditure incurred at the time of forwarding, as explained shortly.

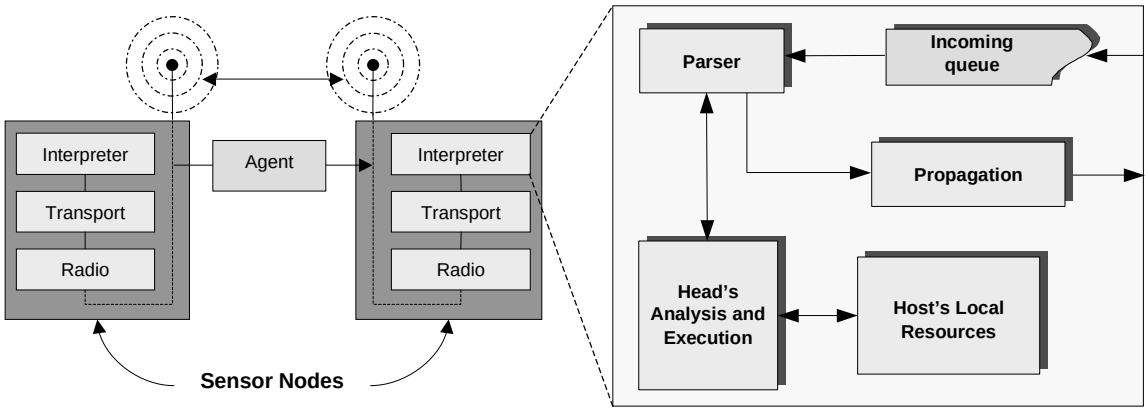

Figure 5.2 System architecture of WISEMAN for WSNs.

The interpreter is comprised of an incoming agent queue, a code parser, an execution block, and an agent propagation block. The incoming agent queue works in a simplistic first-in-first-out fashion, and accepts agents arriving either from other nodes, or those injected at the local node. The parser is in charge of processing WISEMAN's language constructs that comprise the agent. Initially, the parser removes a single agent from the incoming queue and fragments it into its code and data fields, which are temporarily stored in predefined memory spaces. Agents are further separated into two segments, hereto referred as HEAD and TAIL. The HEAD is the first fragment of codes that appears before an operation or precedence delimiter as defined by the language constructs explained in the next section. The rest of the codes that follow are referred to as the TAIL. The HEAD is recursively unwrapped from within any precedence delimiters or top-syntactic operations until a single indivisible operation is found and subsequently processed by the execution block. The parser continues to send the next indivisible operation as indicated by the execution block if the previous one is successfully processed.

The execution block's task is to perform the operation indicated in the HEAD, whose outcome is defined either as success or failure, and is always returned to the parser. In addition, the execution block handles any external function calls in response to an agent's need for data processing at the current node. Therefore, the agent must know in advance the function name or identifier that the execution block shall use. The agent can employ the value(s) returned by the function, if any, to make decisions that affect the subsequent execution of the distributed process, as noted before.

The local agent execution process is halted when any of the following conditions occur: an operation yielded an unsuccessful outcome, a hop operation is encountered, or an explicit process termination is indicated. In the first case, the agent's TAIL is discarded and the agent simply terminates executing, unless the HEAD is contained within a language construct that instructs the parser to proceed otherwise. If the operation is successful, then the parser sends the next indivisible operation to the execution block and the process continues as defined by the agent's code construct. In the second case, the agent may instruct the execution block to forward the agent to another node, and so the TAIL of the agent is sent to the propagation block for dispatching onto other node or set of nodes. In the third case, the agent may simply instruct the interpreter to explicitly halt the current process execution as needed.

Figure 5.3 shows the agent processing cycle in the WISEMAN interpreter. When an agent is forwarded, the propagation block first ensures that the agent's transfer is actually possible by looking up the destination node in a local table that contains a list of the neighbouring nodes. The agent will be dispatched out of the interpreter only if the destination node is found in the table. Otherwise, the agent is discarded. However, the agent may be sent in the broadcast mode if no particular destination is specified. The propagation block signals the

parser once an agent has been forwarded, which in turn removes the next agent from the incoming queue if one exists.

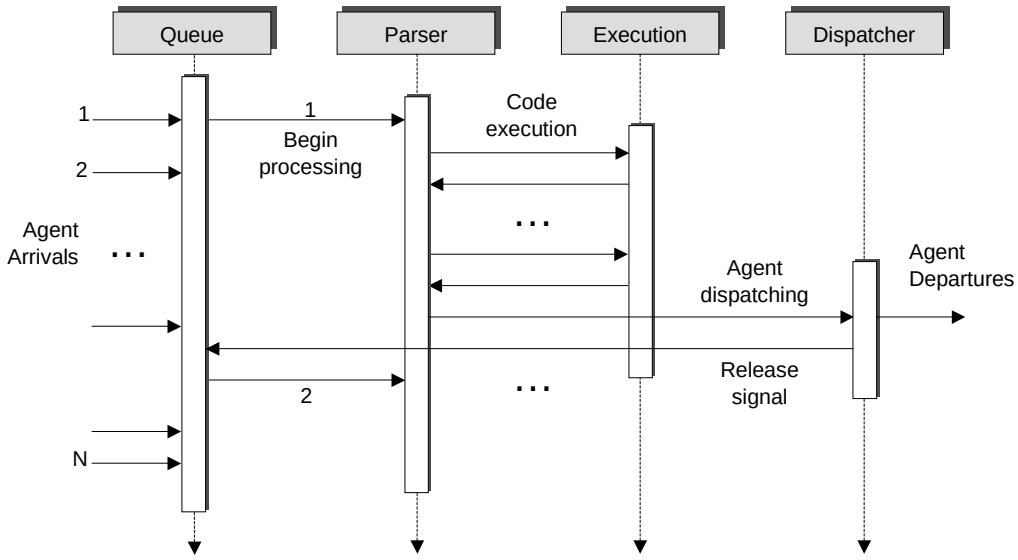

Figure 5.3 Timing sequence for agent processing in the WISEMAN interpreter

Agents perform maintenance on the table of neighbouring nodes, as well as on any semantic relationships that may have been established. The purpose of this functionality is to enable the creation of virtual links that the agents can employ in their WSN navigation process. For simplicity, semantic relationships are mapped to simple ASCII characters as defined by the programmer. For example, '*t*' may be mapped to the following meaning: "*a neighbouring node has reported a temperature beyond the maximum threshold*". These semantic relations can be reset or modified as needed by the agents that use them.

Another powerful feature of WISEMAN is defined by its metamorphic capability. Since WISEMAN agents are transmitted and processed in their raw text-string from, the architecture effectively allows the interpreter to substitute portions of the agent's code as instructed by the very agent code. This characteristic introduces a degree of flexibility unmatched by existing agent approaches in WSNs. For example, a target-tracking agent may initially carry and execute

the corresponding code that executes an energy-efficient algorithm that works efficiently for slow moving objects. However, if the object's rate of mobility increases, then the agent instructs a WSN node to replace its target tracking algorithm (its code structure) section by one that is more energy-demanding, but otherwise necessary to keep up with fast-moving objects. This process can be carried out in two ways. In the first, WSN nodes can be preloaded with a library of algorithms to be used when swapping portions of the agents' codes as instructed. In the second, agents carry, say, one primary algorithm, and a secondary one that might be dropped as the agents navigate through the WSN. Later, the primary algorithm might be replaced by the secondary one as instructed.

5.4 Language Constructs

5.4.1 Variable Types

WISEMAN's simplified constructs are defined by operators, variables, rules, and delimiters, as shown in Table 5.1. The original Wave interpreter defines typeless "Nodal" and "Frontal" variables that a programmer uses to store any type of variable. However, the overhead associated with the operations necessary to achieve this same task is unsuitably large in the case of middleware in WSNs.

The WISEMAN interpreter supports three kinds of variables of fixed-type, all of which employ pre-assigned memory segments in the local node. The first type is coined as *Numeric* variable, represented by the letter '*N*' (e.g., "*N3=1"*). These variables are semantically similar to public variables defined in object-oriented programming, meaning that all agents that arrive to the interpreter have access to them. The second kind is the *Character* variable, which simplifies the manipulation of character strings as needed by agents through the letter "*C*" (e.g., "*C=d*"). Finally, agents may individually employ and carry one or more *Mobile* variables represented by the letter "*M*" (e.g., "*M2=2"*), which resemble the role of private variables in object-oriented

programming. In addition, the interpreter employs the *Clipboard* variable to temporarily store data that agents can retrieve by using the letter "*B*". An agent may always create, modify or delete a mobile variable as needed. The manipulation of *Numeric* and *Character* variables at the local node has no effect on the variables of remote nodes. Likewise, manipulations of an agent's own *Mobile* variables have no effect on other agents' *Mobile* variables, as shown in Figure 5.4.

Table 5.1 WISEMAN language constructs

(a) Variables

Identifier	Description
N	Numeric
M	Mobile
C	Character
B	Clipboard
D	Address
P	Predecessor
L	Link

(b) Operators

Identifier	Description
~	Belongs to
/~	Does not belong to
<	Less than
<=	Equal or less than
==	Equal to
+	Add
-	Subtract
*	Multiply
/	Divide
&	Append
:	Retrieve by index
=	Assign
#	Hop
!	Halt

(c) Execution rules

Identifier	Description
R	Repeat
O	Or
A	And
S	Sequential

(d) Delimiters

Identifier	Description
;	Segment delimiter (sequential)
,	Rule delimiter (parallel)
()	Atomic execution
[]	Embraces And/Or rule
{}	Embraces Repeat rule

All variables are expected to maintain their semantic meaning across the network according to how they are individually manipulated by programmers through the agents. In addition, *Numeric* variables are predefined as being of floating point type, whereas *Mobile* variables are implemented as character arrays, so that multiple values of any type can be stored (e.g. Boolean, integers, fractions and characters). The interpreter converts the contents of one or

more mobile variables to their corresponding value types and temporarily stores them in predefined memory locations when an agent visits a node. Other agents visiting the node may overwrite these values later.

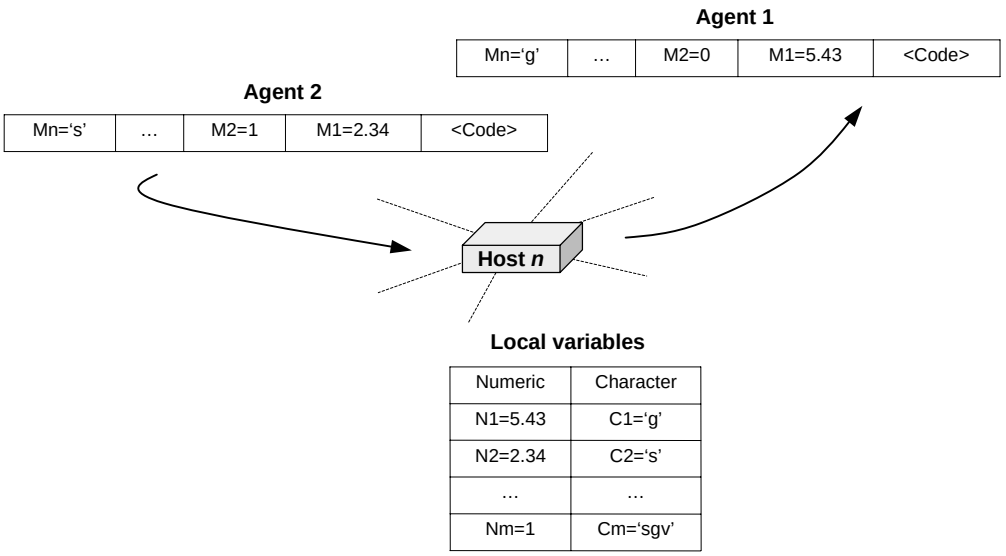

Figure 5.4 Mobile and local variable access.

The execution environment also defines three extra *Environmental* variables. *Address* is a read-only variable, whose content is defined by the local node's address. The *Predecessor* variable contains the address value of the node where the agent currently being processed came from. Similarly, the *Link* variable holds the label of the virtual link through which the agent arrived from. The *Propagation* block of the preceding node is responsible for appending to the agent's control field both these variables.

5.4.2 Operator Types

Table 5.1(b) shows the operators defined for WISEMAN. The first two operators allow an agent to test whether the value of a variable is contained within another variable or not. For example, "*M1~C1*" tests whether the value of *M1* brought by the current agent is contained within *C1*, if used as an array. These operators are useful for keeping lists, or when agents keep

track of the sensor nodes that observed an event. The next three operators are standard comparison operators that also return Boolean values that agents can evaluate (e.g. "*M3<45*").

The next four operators can be employed to perform simple arithmetic operations. For example, "*N3*4*" multiplies the existing value of *N3* by 4, and the result is re-assigned to *N3*. The same logic applies to the subsequent arithmetic operators. On the other hand, the "*&*" operator appends a value to the indicated variable when employed as an array (e.g. "*N2&`b'*"), which can be later retrieved by the ":" operator. For example the code segment "*M2:3;N1=B*" copies the fourth element of *M2* to the clipboard variable *B*, which is then assigned to *N1*.

The *Hop* operator is employed to indicate that the agent needs to be forwarded to the node specified in the right-hand side of the "#" character through the link specified in its left-hand side. For example, the code segment "*a#M1*" indicates a hop through the virtual link "*a*" to the node whose identifier is stored in the mobile variable *M1*, whereas "*s#*" indicates a hop to all neighbouring nodes that are virtually linked from the local node through the label "s". Here, the programmer defines the semantic connotation of the letter "s". This construct has the advantage of automatically cloning the agent with as many copies as outgoing virtual links exist in accordance to the left-hand operand. That is, if there are 3 such virtual links labelled as "*s*", then 3 identical agent copies will be forwarded by the dispatcher. Finally, the halt operator "*!*" indicates explicit termination of the current agent with success if the right-side operand is 2 (i.e., "*!2'*'), or failure if the operand is 1 (i.e., "*!1*").

5.4.3 Rule Constructs

Table 5.1(c) illustrates the rule constructs' identifiers defined in WISEMAN. The *Repeat* "*R*" rule indicates that the code embraced by curly brackets will be repeatedly executed (e.g. "*R{...}*"). The *Or* and *And* rules are employed as a way to manipulate execution of the agent's thread by testing whether the code embraced within square brackets yields a *true* or *false* value

for every code segment it includes. Therefore, an "*O*[*...;...;...*]" string indicates that the code segments delimited by "*;*" are to be sequentially executed, stopping as soon as one of these segments results in a *true* value. Otherwise, the *Or* rule returns a *false* value and the agent's process stops. A similar logic applies for the *And* rule "*A*[…;…;…]", except that all of the segments must return *true* in order for the rule to return a *true* value as well.

5.4.4 Code Delimiters

As hinted in code examples above, the main delimiter employed to separate code segments is the semicolon ";". The comma delimiter "," is also defined within the rule constructs *And/Or*. The semicolon delimiter instructs the interpreter to execute the embraced code segments in sequence, whereas the comma indicates possible parallel execution if supported by the underlying operating system. In addition, round, square and curly brackets have been designed to delimit code segments whose execution depends on a rule construct, as explained before. Distinct types of brackets are employed since they facilitate the parser's task when tokenizing nested code segments prior to being processed (e.g., "*R*{...*O*[...(...)...]...}"). The use of a single type of bracket (e.g., "(...)") would have implied significant parsing functionality requirements, and therefore, added overhead.

5.5 Programming in WISEMAN

5.5.1 A Sample Event Monitoring Application

To better understand the usefulness of our proposed system, consider a sample WISEMAN program deployed in a WSN whose main objective is to provide an avalanche risk assessment service at a sub-alpine terrain setting as mentioned before.

```
 1. Assign a risk threshold level to mobile variable M1
 2. Set N2=1 as a flag
 3. Repeat
 4.     Hop to all neighbouring hosts through all virtual links
 5.     if M1 is smaller than the latest assessment N1 then
 6.         Assign a "risk" ('r') label to the traversed link L
 7.     else halt the current process
 8.     end if
 9. end Repeat
10. Repeat
11.     Hop to neighbouring hosts through "risk" virtual links
12.     if N2 indicates no assessment performed yet then
13.         run "eval" to assess the localized avalanche risk level
14.         if risk level in N3 is greater than F1 then
15.             run "notify" to send an alert signal to base station
16.             Set N2 to flag current host as evaluated
17.         else halt the current process
18.         end if
19.     else halt the current process
20. end Repeat
```

Figure 5.5 An avalanche risk assessment algorithm in WISEMAN

```
M1=6;N2=1;R{#;O[(M1<N1;L="r");!2]};
R{r#;N2==1;"eval"; O[(M1<B;"notify";N2=0);!2]}
```

Figure 5.6 An avalanche risk assessment program in WISEMAN

Lines 1-9 in Figure 5.5 explain the first line of code in Figure 5.6, which performs an initial assessment probing, possibly triggered by a node once one of its sensors reached a fictitious magnitude reading of 6. After setting a flag in *N2* to indicate that the current node has been visited, agents are cloned and dispatched through all of the outgoing links for preliminary situation assessment according to the *hop* operation first encountered in the *repeat* rule. The next code segment is embraced by an '*Or'* rule, which evaluates the outcome of three code segments embraced by round brackets. Here, if the value in the local variable *N1* exceeds the threshold specified in *M1* (indicating the possibility of an avalanche), then the agent begins the creation of a virtual network starting at the current node, whose nodes are linked by an "*r*" (i.e., "avalanche *r*isk") semantic label. A false outcome triggers the evaluation of the halt operator, resulting in a

forced success that enables process continuation of the *Repeat* rule. This process executes recursively until the intended virtual network is created.

A secondary assessment process is immediately run afterwards, as shown in the second line of Figure 5.6, and described in lines 10-20 of Figure 5.5. The agents use the "*risk*"-labelled links to reach only those nodes that initially reported a potential avalanche risk. An agent first checks *N2* to verify whether other agents executing the *second* part of the process have reached that node. If so, the local program "*eval*" is invoked to further analyse sensor readings and assess the current situation. The return value stored is in the Clipboard, and is used to decide whether an alert should be sent to the base station as indicated within the '*Or'* rule. As explained before, the agent carries no data processing algorithm. Instead, the "*eval*" program is made available beforehand on every node, given the likelihood of repeatedly being invoked in the future. The agents' task is limited to controlling the way in which the risk assessment is made, and can always be changed. A fine-grained agent approach would have implied incorporating into the interpreter the necessary functionality and language constructs, so that this task could be performed by the agent. However, this would make little sense if this algorithm is run in a continuous basis.

Figure 5.7 depicts the previous example, where the dark-shaded nodes are assumed to issue an alert notification to the base station by employing the locally available "*notify*" program. Note that no program counter or process state information transfer is needed in the WISEMAN framework to enable a simple form of strong mobility. Here, a non-recursive program gradually executes until its segments are depleted and the agent terminates. However, for recursively-executing programs, the interpreter extracts the code embraced by the Repeat rule and inserts it before the Repeat rule itself (i.e., "*code*;*R*{*code*}"). Thus, the successful execution of all code segments leading to the *Repeat* rule delimiter ensures the extraction and re-insertion of its

embraced code once again. If a hop act is encountered during execution, the remaining segments (i.e., the agent's TAIL) are forwarded to the new node, where execution resumes with the next code segment. Hence, a simple form of strong mobility is accomplished as an intrinsic feature of how the agent is handled by the interpreter without the need to additionally forward a program counter and execution stack as traditionally done in other MAS.

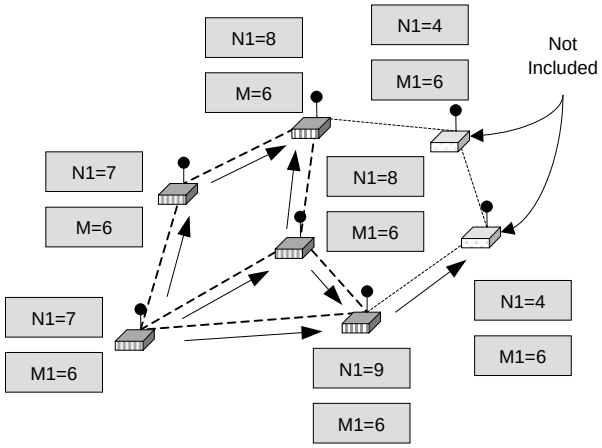

Figure 5.7 WISEMAN agent propagation example

5.5.2 Service Discovery with WISEMAN

An important feature introduced by our approach is that it enables the creation of several virtual overlay networks within the same WSN. Since various WSN's nodes may cooperate to provide an aggregate service, then different subsets of nodes can be logically tethered too, depending on the particular service that they provide. Therefore, a WSN may belong to one or more of these logical planes, and their service-oriented association may vary in time according to the current needs. WISEMAN agents may thus be programmed to navigate through one or more virtual networks and carry out different tasks, which may well be the case in WSNs comprised of heterogeneous nodes, also known as wireless grids [111].

Since WSNs are usually considered static, service discovery tasks can be implemented in the form of selective query forwarding. Figure 5.8 shows an example of a WISEMAN program

that can be launched at any of these service provider nodes to set up a labelled tree to direct queries for service "s". The agent is first cloned as many times as outgoing links exist from the current node, and the mobile variable *M1* is incremented by 1 ("*#;M1+1*"). If the value brought by *M1* is smaller than the one stored in the local numeric variable *N1*, then *N1* receives the value of *M1* and the identifier value of the predecessor node is stored in *N2* ("*M1<=N1;N1=M1;N2=P*"). Once this initial process is finished, a second code segment is propagated to the nodes whose identifiers were stored in the variable *N2*, and a virtual link denoting the service "*s*" is assigned between the corresponding nodes. *N1* is used as a flag to restrain agents from traversing paths that have been already labelled.

```
M1=0;R{#;M1+1;O[(N1==0;N1=99);];M1<=N1;N1=M1;N2=P};
R{#N2;N1>0;L=s;N1=0}
```

Figure 5.8 Setting up query forwarding labels for service discovery in WSNs using WISEMAN

Once this process is finished, service queries originating at any WSN node can traverse the corresponding virtual network to reach the desired destination. Additional nodes that also provide this service can be later incorporated into the existing virtual network by checking first whether these logical links already exists.

5.5.3 Topology Configuration with WISEMAN

Topology configuration can be accomplished by allowing agents to access the WSN nodes' APIs in order to configure the physical links. For instance, a node might want to find an alternative connection to the WSN when another node through which communications were originally relied upon is currently running low in battery. In this case, the WISEMAN program shown in Figure 5.9 can be conceivably employed for this purpose.

```
M1=A;listen;
R{#;N1==0;B=M1;page;
  O[(B==1;B=M1;connect;#M1;#6;shutdown);
    (N1=1;M2+1;M2<4)]};
```

Figure 5.9 Localized WSN topology reconfiguration program

This program first stores the identifier of the local node into *M1* and instructs the local node to enter a listening mode before being dispatched to its neighbours ("*M1=A;listen*"). Upon arrival, each agent checks for a clear "node visited" flag and instructs the local node to send a page signal to the process initiator node whose identifier is temporarily written to the clipboard ("*#;N1==0;B=M1;page*"). If the page signal is acknowledged, then the node is further instructed to establish the connection sought for, followed by the agent's hopping to the originator node. An additional hop to the node whose battery is almost depleted (e.g., '6') is then performed to turn it off ("*B==1;B=M1;connect;#M1;#6;shutdown*") and successfully terminate the process. If on the other hand, the paging process was unsuccessful, then the agent sets the "node visited" flag and increments the hop-count variable *M2*. In this example, it is assumed that the probability of successfully establishing a new connection is very low once the hop count reaches a value of 3, at which point the process terminates unsuccessfully ("*N1=1;M2+1;M2<4*"). The use of both the *Repeat* and *Or* rule ensures the respective replication and continuation of the process as needed.

5.6 Simulation, Implementation and Programming Details

5.6.1 Simulation

We now describe relevant implementation and programming details of WISEMAN as a discrete time simulation in OMNeT++ that helped to identify potential problems before actual hardware implementation. First, the WISEMAN interpreter allows only a single agent to be processed at a time. Although some of the currently existing hardware devices for WSNs might

support concurrent multi-agent processing, others have very limited capacity. Therefore, we aim for a system that can be easily ported between different WSN platforms in support of device heterogeneity. A simple signalling system is implemented, in which the dispatcher instructs the incoming agent queue to release the next agent to the parser if one is available. Therefore, the incoming queue works as a container that releases the next agent (if any) for subsequent processing and dispatching.

When the parser receives an agent for processing, it first detaches its accompanying *Environmental* variables, as well as its mobile variables if they exist. It then proceeds to separate the agent into its HEAD and TAIL as explained before. However, the parser may have to remove code embraced by brackets delimiting one of the rule constructs. Once the embraced code has been extracted, each segment is atomically executed. In addition, we only allow a certain degree of nesting in programs. In particular, the interpreter can process code within round brackets embraced by an *And/Or* rule construct (e.g., "*O*[*...;*(*...*)*;...*]"), or a *Repeat* rule (e.g., "*R*{*...;*(*...*)*;...*}"). Similarly, the interpreter can process code within an *And/Or* construct embraced by a *Repeat* rule (e.g., "*R*{*...;O*[*...*]*;...*}"). In other words, the *Repeat* rule has precedence over the *And/Or* rules, and the *And/Or* rules have precedence over the simple round brackets, indicating atomic execution of the code within. Simple round brackets must not contain any other rule, and the *And/Or* rule must not contain a *Repeat* rule. This limitation simplifies the structure of the parser, while providing enough functionality for processing agents. As for simple atomic segments, the parser tokenizes the left and right operands, as well as the operator, which are then passed on to the execution block for processing.

Another aspect worth mentioning is the virtual links feature of WISEMAN. Hop operations shall contain null operand parameters when no semantic relationships between nodes have been established yet, as the agents are assumed to learn about the current network

beforehand. This implies that agents are initially broadcast from the local node to all neighbouring nodes. When an interpreter receives a broadcast agent, the value of the *Predecessor* environmental variable is added to the local table of neighbouring nodes. This entry has a null value label for the semantic identifier. At this point, interpreters are now able to process hop operations to specific nodes (e.g., "*#3*"). While some systems do provide built-in multi-hop routing capabilities, we see no merit in sending agents to a node from which no other information is known aside from its network identifier.

The interpreter is ready to process labelled-hop operations once the corresponding relationships have been created. In doing so, the interpreter employs a "next-hop" table to temporarily store values obtained from the general node-forwarding table following the indications of the hop operation. Thus, only the entries in the forwarding table that match the semantic characteristics indicated by the hop operations are copied into the next-hop table. This table is then employed by the dispatcher block during the actual forwarding process, at which point the agent is cloned as many times as entries in this table are, minus 1. Finally, this table is erased once all of the agents have been forwarded.

We simulated the WISEMAN program described in the previous section as an exercise to evaluate its performance in a WSN of random topology comprised by 50 hosts dispersed in an area of 1000 m^2, yielding a network diameter of 15 hops. The program is injected into the network from a gateway located halfway between two corners of the simulation area at the perimeter. The data rate employed by the WSN nodes is fixed at 250 Kbps. In our simulation, an event (e.g., fire, seismic activity, etc.) is detected by six of the WSN's nodes, which register a greater than usual reading that is subsequently employed by the agents as a reference to create a perimeter around the affected area. This logical network might be later employed by other agents to continue monitoring the area of interest. We employed a generic Medium Access Control

(MAC) layer since our proposed scheme is agnostic to the underlying WSN system, and we are primarily interested in the characteristics relevant to the WISEMAN agents. In addition, processing time is not measured; only the time spent by the nodes during agent forwarding is registered. In practice, actual processing times vary depending on the sensor hardware being employed, ranging from a few tens of microseconds for simple operations in the high-end devices [61], to a few hundreds of microseconds in the lower-end devices [62]. Measurements of this simulation run confirm the benefits of employing WISEMAN. For instance, only 201 agents were created for the completion of this process, incurring a total of only 24918 bytes of bandwidth observed in the WSN. A perimeter formation time of 1.49 ms, and a process completion time of 7.43 ms can be regarded as reasonably short.

5.6.2 Implementation*

Once WISEMAN's design and operational correctness was tested in a computer simulation, we proceeded to its implementation over commercially available devices popularly known as MICAz motes [107]. These devices allow users to create WSN applications that are initially programmed in the NesC language, and then compiled and linked to the corresponding TinyOS system libraries to generate binary images [108]. These images are then embedded into the WSN nodes for custom deployment over the target area.

In essence, WISEMAN's C++ simulation was ported to a NesC modular design, whose modules were named in accordance to the original implementation. The classes necessary for operation were translated to NesC modules and interfaces. Public methods were converted to NesC commands, and private/protected methods were adopted as static functions. The following are the most important modules implemented for WISEMAN. The VirtualMachineM module is

* This subsection is based in part on an EECE496 project report that refers to WISEMAN's code porting from C++ to NesC in Crossbow MICAz motes. NesC codes and accompanying files can be downloaded at http://www.ece.ubc.ca/~sergiog/wiseman.

positioned in the topmost hierarchical level, and it receives WISEMAN scripts directly. This module encapsulates the system from external modules and provides a clean interface for handling messages to the AgentQueueM module. The PacketGenM module realizes the external interface of the virtual machine, and it is used to send WISEMAN agents to the local mote's transmitter. The ParserM module receives and processes messages removed from AgentQueueM whenever an agent needs to be processed. This is performed by means of the handleMessage() command. The EngineM module is a generic interface through which the ProcessorM and ParserM modules are called to tokenize text strings that comprise the WISEMAN agents, and to execute arithmetic and logical operations respectively. In particular, the ProcessorM module relies heavily on the StringsM module, which had to be created from the ground up to process text strings appropriately since NesC lacks the capabilities of regular C++ libraries to process text strings. Finally, the DispatcherM module is utilized by ProcessorM to evaluate link assignments, and perform virtual network maintenance. To comply with WISEMAN's specifications, local function calls were implemented by passing simple text strings encoded as keywords that the interpreter knew how to process (see Table 5.2). For simplicity, parameters to function are passed within the text string after a period '.' character delimiter. WISEMAN's modular diagram for the MICAz motes is shown in Figure 5.10.

Table 5.2 Built-in Function Calls in WISEMAN

Function	Parameter	Description
F1	None	Test function; always returns \1"; used for debugging purposes.
led	[y|r|g]	Lights up a LED of the specifed colour (y = yellow, r = red, g = green). Returns: nothing.
lit	none	Returns a string of raw light sensor value in decimal (0 - 255).
s	[0-9]	Delays script execution by the given number of seconds. Returns: nothing.
sd	[moteid]:[msg]	Sends the message msg to the specifed mote id. Returns: nothing.

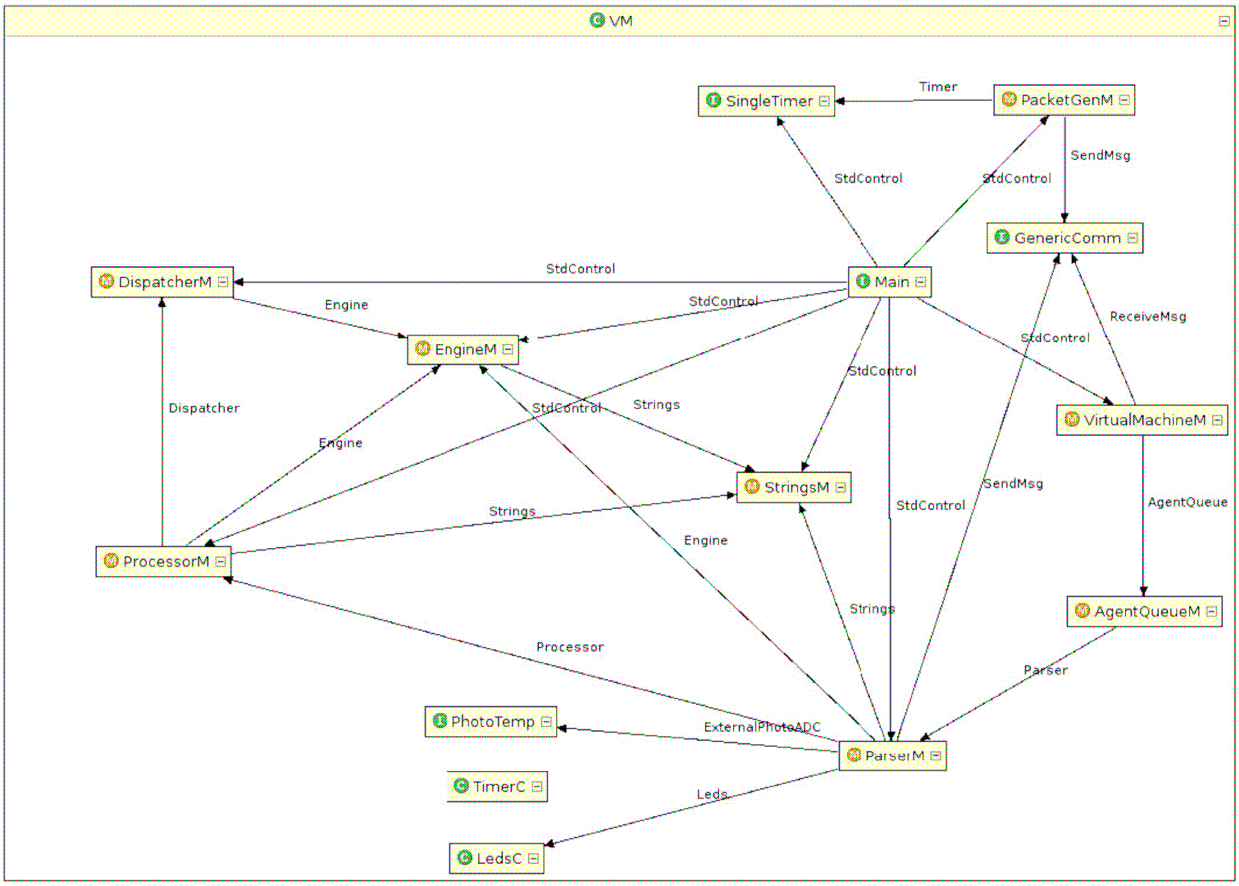

Figure 5.10 WISEMAN's modular implmentation in MICAz

We tested WISEMAN's hardware implementation with simple and short programs that are currently limited to 29-byte characters. In the future, a separate transport layer module will allow larger agents to be transmitted. The design of this module relies on a separate research effort that is out of the scope of this thesis. In addition, only single-hop functionality is currently available. The multi-hop functionality depends on the implementation of the transport layer as well. We deployed a simple WISEMAN program that hopped from a gateway mote by creating a virtual link to another mote and executed a "led" function call to switch on one of the mote's LED (Figure 5.11). A secondary agent was then injected and transmitted through the virtual link in order to read the light value of a sensor, and then hop back to the gateway mote to display it. Similar programs were successfully executed to test arithmetic and rules-constructs defined by the WISEMAN specification.

```
#;L=a;"led.r"
a#;M1=lit;#P
```

Figure 5.11 WISEMAN agent test program

5.7 Summary

We have presented WISEMAN, a mobile code approach for WSNs. We described WISEMAN's architecture, its language constructs, and its features as a suitable system for the support of service-oriented tasks in WSNs. We also described important implementation and programming details of our scheme. The design of our system is based on an earlier implementation of the Wave system for mobile processing in wired networks. However, a number of changes were introduced as required by the scarcity of hardware and energy resources that characterize WSN devices. Among these changes are: elimination of dynamic memory allocation requirements, redefinition of the language constructs, simplification of the interpreter's architecture, and simplification of the agents' program structure. WISEMAN has been implemented and tested using a computer simulation with the objective of finding potential problem spots and areas of improvement before the system is actually put into hardware devices. The benefits of employing WISEMAN's agents were readily evident as shown by their ultra-compact size, which lead to very low bandwidth usage. In addition, WISEMAN facilitates the creation of overlay networks and process coordination in WSNs, which we regard here as essential service-oriented tasks. The following table presents a feature comparison of WISEMAN with Agilla [62], which is a popular agent system for WSNs that is often cited in the literature. This comparison includes values of a preliminary WISEMAN hardware implementation for proof-of-concept. As we can see, WISEMAN either meets or exceeds Agilla's functionalities and hardware requirements.

Table 5.3 Feature Comparison of Agilla vs. WISEMAN

Feature	Agilla	WISEMAN
Interpeter's memory footprint	41.6 KB	20 KB
RAM requirement	3.6 KB	2 KB
Program style	Bytecode	Text strings
Distributed process coordination	No	Yes
Virtual network creation	No	Yes
Semantic matching	Yes (local)	Yes (local and distributed)
Metamorphism	No	Yes
Execution granularity	Fine	Coarse
Agent cooperation	Yes	Yes
Migration strategy	Strong	Strong

Chapter 6. Conclusions

Through our investigations we have introduced and evaluated innovative improvements in the fields of topology formation and service discovery in wireless/mobile ad-hoc networks. We now describe the highlights of our technical achievements, followed by suggestions on possible directions for future work.

6.1 Summary of Technical Achievements

The first part of our work enabled us to justify the importance of incorporating service-oriented considerations as an integral element in the design of wireless network protocols. We did so by presenting a novel scatternet formation protocol based on mobile agent technology that overcomes several issues seen in existing schemes. *Bluescouts* benefits from the characteristics of agent-based systems in various ways. First, the mobile agent approach enables the protocol to execute in a completely distributed and asynchronous fashion, as BDs need not execute the protocol concurrently when new devices attempt to join an existing piconet/scatternet. This capability clearly helps to reduce the processing load on individual devices compared with centralized approaches, enables a dynamic and gradual organic-like growth of a scatternet as users arrive, and scales well to networks of moderately large size. Second, our programmable approach enables a flexible scatternet formation, in which BDs become logically tethered according to fully customizable policies. For testing and comparison purposes, the protocol used for our simulations followed a predefined policy that enabled new BDs to join an existing scatternet as quickly as possible. However, other policies may be introduced so that BD reconfiguration takes into account context-dependent factors such as the type of device, link capacity, or even available services, according to the needs of the user. Third, our protocol is much more flexible than existing ones, since it decouples the actual scatternet formation

procedure from device discovery. *Bluescouts* relies on existing APIs that employ either mandatory schemes or variations thereof in order to discover new devices [115] - [117]. Thus, in contrast to many other approaches, no changes or enhancements of the existing device discovery scheme are necessary.

In the second part of our thesis, we presented the Service Directory Placement Algorithm, which in conjunction with SLPv2 enables efficient directory-based service discovery in mobile ad-hoc networks. The formulation of SDPA's decision scheme as a Semi-Markov Decision Process, and its solution through the Q-Learning technique, proved helpful in defining a policy that saves bandwidth, as opposed to relying on a purely heuristic method. Quantitative measurements obtained through computer simulations revealed the usefulness of the mobile directory approach, with average bandwidth savings often close to 50% for MANETs in which hosts move at low speeds. SDPA's performance gain was observed to degrade gracefully at higher host speeds, consistently yielding performance improvements of over 40%. The actual performance of a directory-moving scheme is expected to depend on a combination of factors, such as: true user mobility pattern(s), the directory-moving strategy employed (e.g., as messages, as an object, etc.), and the type/number of services and applications being run by the users. Additionally, factors such as the topology configuration protocol, the MAC technology employed, and radio signal propagation issues would also play an influential role in the overall packet overhead of the network. However, our simulation results suggest that the single-directory approach promoted by SDPA is a feasible service advertisement technique that can be employed when small MANETs are deployed, the number of services available is limited, and users move at pedestrian speeds. Issues affecting the practical implementation of our approach are discussed in the following paragraph.

In the third part of our thesis, we introduced SDPP, which is a multi-directory enhancement to the single-directory approach of SDPA. SDPP also relies on Q-learning to find efficient directory relocation polices that maximize performance on an averaged basis. Several changes to the original learning system were introduced in response to the needs of the multi-directory approach. Our simulations revealed that the performance enhancements introduced by using mobile directories are considerable, yet highly dependent on both the directory memory space available in user hosts, and the number and physical distribution of service providers covering the deployment area. In particular, we observed that SDPP performed best when the coverage area of existing service providers spans only a portion of the whole MANET deployment area. In such cases, the use of mobile directories effectively reduces the amount of data traffic observed in the MANET by dynamically relocating service information to network areas that become physically distant from the service providers. However, SDPP also relies on users' hosts having a certain amount of memory available that can be employed for directory use. For instance, larger MANETs increase the memory requirements that make the concept of mobile directories feasible, especially if the memory footprint of the services being advertised increases. We noted that, incidentally, this is a current tendency in web services, where information-rich descriptions are becoming pervasive. Therefore, it is reasonable to expect service advertisements in MANET environments characterized by highly detailed attributes, possibly including multimedia (audio/video clips, short animations and product/service images). We believe that ongoing technological advances and commercialization strategies offset this constraint, as evidenced by the features seen in portable electronic devices that are commercially available at present.

Another constraint previously discussed for SDPA that also affects SDPP is the practical implementation of the learning system, in particular, the implementation of the Q-table in user

devices. We have proposed a simple way to address this issue, which we believe is practicable given the relatively limited number of system states observed. To avoid possible performance degradations introduced by employing function-approximation approaches, reasonably sized Q-tables can be feasibly stored in the memory heap space in users' hosts, which is usually plentiful in devices such as digital audio players, cell phones, PDAs and laptop computers. Conversely, service directories can be kept in RAM memory to curb possible delay access issues. Some other issues relevant to our findings are also described in the next section.

In the fourth and final part of the thesis we have proposed a software agent system – WISEMAN – for use in WSNs. Unlike existing MAS for WSN, WISEMAN enables programmability of distributed tasks with a service-oriented approach. Unlike existing schemes, WISEMAN allows the creation of semantic relationships between WSN nodes that provide an aggregate service. As a result, WSN programmers do not need to know in advance which subset of nodes to send one or more agents to. Another important feature of WISEMAN lies in the compactness of its code, which reduces the size of agents being sent onto the WSN, while at the same time being human-readable (i.e., is not based on a byte-code system). In contrast to existing MAS for WSNs, WISEMAN's architecture is targeted at the distributed coordination of tasks, further enabling code compactness by employing a coarse-grained language construct. WISEMAN exploits the intrinsic characteristics of WSNs and relies on existing programs available at the local nodes that execute in native form, also promoting faster data-processing time. As an added benefit, it can be reasonably argued that WISEMAN's architecture may improve system security aspects by de-emphasizing fine-grained control of local tasks, relying instead on what WSN programmers have made available at the nodes beforehand. As a whole, WISEMAN provides a good MAS solution for WSNs where coarse-grained task coordination is favoured over fine-grained task execution.

6.2 A Unified View

From our previous explanations, we see that although the research subjects in this thesis address distinct issues, they are all complementary. For instance, *Bluescouts* was defined to support Bluetooth-based scatternet formation. However, our proposed decoupling of the topology formation process from the link setup procedures enables the deployment of *Bluescouts* in ad-hoc networks based on other type of networking technologies (e.g., WLAN). To this effect, mobile agents would only have to access the corresponding API to access the underlying link controller sub-system and perform the desired topology configuration. It is safe to assume that *Bluescouts* can even be employed in wireless networking systems where users' devices might belong to more than one network through separate hardware interfaces (e.g., one for Bluetooth, and one for IEEE 802.11). Therefore, Wave agents can effectively hop from one network plane to another within a hybrid networking system in order to perform the desired network maintenance task. Minor changes to the Wave agent would suffice to accomplish this, which can be realized as supported by the metamorphic code feature explained in the case of WISEMAN. In fact, WISEMAN code could be employed for the same task, although to a somewhat limited extent. The reason for this is that WISEMAN's architecture does not incorporate the Track Layer originally available in Wave as explained in Chapter 5. This would translate into WISEMAN's added overhead due to additional link utilization if it were employed in its current form for Bluetooth scatternet formation (i.e., due to additional code necessary to compensate for the functionality otherwise provided by the Track Layer).

In addition to the above, we believe that our proposed SDPA/SDPP schemes could conceivably be employed to reduce the number of service discovery packets in Bluetooth scatternets. In fact, Bluetooth technology only supports service discovery between devices directly linked (i.e., no multi-hop functionality). In this case, SDPP could be employed to

distribute service information in the scatternet by registering information at select piconet master BDs, making it convenient for slave BDs to discover available services in the scatternet. In this case, Bluescouts' scatternet formation policy could be defined so as to have slave BDs linked to master BDs that possess a service directory. Existing SFPs do not support this degree of flexibility.

The main lesson learned in this thesis is that the formulation of novel solutions that take into consideration service-oriented policies requires the use of techniques that detach from the otherwise conventional. In the case of Bluescouts, the use of active networking though mobile codes represents a significant departure from the existing schemes, and yet, it produces comparable performance results. Although the proposed approach also introduces overhead in terms of higher memory and code processing requirements, we believe that ongoing advancements in computer hardware will afford this the use of this technology in the very near future. A similar argument can be made for the case of SDPA/SDPP, which significantly differs from existing schemes that are formulated as sophisticated query routing and packet forwarding. The use of reinforcement learning allowed us to estimate the extent to which heuristic approaches can be made more efficient. In addition, we observed that data mobility is feasible to cope with the difficulties introduced by mobile devices in wireless ad-hoc networks. Finally, WSNs are a particular instance of a networked system where a service provision is carried out through the interaction of its member devices, as mentioned before. The main lesson here, and perhaps in any type of network characterized by this feature, is that physical memory and bandwidth usage can be leveraged by designing management system centered on the idea of supporting distributed process coordination. WISEMAN accomplished this objective, while achieving re-tasking programmability. As a whole the novel schemes proposed in this thesis can be combined to facilitate service discovery and provision in MANETs.

6.3 Suggestions for Future Research

Additional improvements to *Bluescouts* include mechanisms to avoid deadlocks when piconets are being reconfigured, which is a potential issue if the instability of the wireless channel causes the wireless packet carrying the agent to be dropped. In addition, the protocol can be extended to allow the concurrent reconfiguration of more than a single device at a time when dealing with batched BD arrivals. Security concerns normally attributed to agent systems at the application level could also be investigated to prevent malicious agents from reconfiguring the scatternet, so that one or more BDs connect to a malicious host. Two additional suggestions for future research include: (1) the characterization of service-oriented scatternets, which would yield the configuration policies implemented by the agents to improve service provision, and (2) the applicability of an agent-based service discovery mechanism that overcomes the constraints of Bluetooth's current SDP.

SDPA can also benefit from a number of enhancements. First, the system needs to incorporate mechanisms for survivability and recovery to ensure that it continues to operate even if: (1) the wireless packet that carries the mobile directory is dropped, (2) the host that carries the mobile directory moves out of the MANET's radio range, or (3) this same host is suddenly powered off. In any of these cases, it seems reasonable for the system to rely on the fixed AP to recover from temporary failure. Additional suggestions that are applicable to both SDPA and SDPP can be made. One of the most important is to further investigate the practical implementation of the value function (Q-function/table), so that additional memory space can be preserved, as previously discussed. Another suggestion is to run more simulations with both SDPA and SDPP individually to further explore the systems' performance as the parameter space is manipulated. Variations to consider would include host speed, mobility models, query inter-arrival time, movement pause time, directory memory space, and service entry memory

footprint. Further studies into additional factors that can be employed to augment the system-space representation are also a priority, as per the results obtained for the 50-host case in SDPP. Similarly, distinct reinforcement learning approaches that incorporate multi-agent cooperation could be explored [118] - [122]. Additionally, in our simulations we have assumed that service-related information changes infrequently. However, if this situation arises, there needs to be an update mechanism in place so that DAs do not contain stale information.

As for WISEMAN, a preliminary hardware implementation that supports single-hop agent navigation has already been produced, yielding a memory footprint of roughly 20KB. Further efforts are underway to support multi-hop capabilities, which will allow testing its full operational correctness. Also, the interpreter can be further extended to ensure that mismatches in the semantic context of numeric variables do not affect the system. This situation may arise if a programmer creates two or more types of agents that employ the same variable for different purposes. Fault tolerance mechanisms should always be incorporated into the architecture of the system as comprehensively as possible, depending on the amount of memory available at the nodes. In addition, WISEMAN should ideally be able to allow upgrades to the local libraries. One way to solve this issue would be for WISEMAN the distribution of these libraries also as interpreted code. However, an alternative mechanism would be required in order to distribute binary images as performed by existing approaches (e.g., Deluge). This functionality would be beneficial if the WSN user/administrator wanted to upgrade binary code that executes improved algorithms for more efficient memory use, or reduced energy consumption. Finally, an automated garbage collection/deletion subsystem would also be a sensible enhancement for WISEMAN, to release memory at the nodes and erase virtual links between them when they are no longer required.

References

[1] M. El-Sayed and J. Jafee, "A View of Telecommunications Network Evolution," *IEEE Communications Magazine*, vol. 40, issue 12, pp. 74-81, December 2002.

[2] C. Park and T.S. Rappaport, "Short-Range Wireless Communications for Next-Generation Networks: UWB, 60 GHz Millimeter-Wave WPAN, And ZigBee," *IEEE Wireless Communications*, vol. 14, issue 4, pp. 70-78, August 2007.

[3] S. Stroh, "Wideband: Multimedia Unplugged," *IEEE Spectrum*, vol. 40, issue 9, pp. 23-27, September 2003.

[4] D. Qiao and S. Choi, "New 802.11h mechanisms can reduce power consumption," *IEEE IT Professional*, vol. 8, issue 2, pp. 43-48, March-April 2006.

[5] S. Staab, P. Domingos, P. Mika, J. Golbeck, L. Ding, T. Finin, A. Joshi, A. Nowak and R.R. Vallacher, "Social Networks Applied," *IEEE Intelligent Systems*, vol. 20, no. 1, pp. 80-93, January-February 2005.

[6] N. Eagle and A. Pentland, "Social Serendipity: mobilizing social software," *IEEE Pervasive Computing*, vol. 4, issue 2, pp. 28-34, January-March 2005.

[7] M. Motani, V. Srinivasan and P. Nuggehalli, "PeopleNet: Engineering a Wireless Virtual Social Network," in *Proceedings of ACM MobiCom*, Cologne, Germany, August 2005.

[8] S. Androutsellis-Theotokis and D. Spinellis, "A Survey of Peer-to-Peer Content Distribution Technologies," *ACM Computing Surveys*, vol. 36 no. 4, pp. 335-371, December 2004.

[9] P.Johansson, M. Kazantzidis, R. Kapoor and M. Gerla, "Bluetooth – An Enabler of Personal Area Networking," *IEEE Network*, vol. 15, issue 5, pp. 28-37, September-October, 2001.

[10] C. Bisdikian, "An Overview of the Bluetooth Wireless Technology," *IEEE Communications Magazine,* vol. 39, no. 12, pp. 86-94, December 2001.

[11] R. Schneiderman, "Bluetooth's Slow Dawn," *IEEE Spectrum Online*, January 2001.

[12] Bluetooth Special Interest Group Specification, http://www.bluetooth.com.

[13] C. Law, A.K. Mehta and K.-Y. Siu, “A New Bluetooth Scatternet Formation Protocol,” *Mobile Networks and Applications*, vol.8, no. 5, pp. 485-498, October 2003.

[14] G. Tan, A. Miu, J. Guttag and H. Balakrishnan, “An Efficient Scatternet Formation Algorithm for Dynamic Environments,” in *Proceedings of IASTED Communications and Computer Networks (CCN)*, Cambridge, USA, November 2002.

[15] G.V. Zaruba, S. Basagni and I. Chlamtac, "Bluetrees - Scatternet Formation to Enable Bluetooth-Based Ad Hoc Networks," in *Proceedings of IEEE International Conference on Communications (ICC)*, Helsinki, Finland, June 2001.

[16] L. Ramachandran, M. Kapoor, A. Sarkar and A. Aggarwal, “Clustering Algorithms for Wireless Ad-hoc Networks,” in *Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, Boston, USA, August 2000.

[17] T. Salonidis, P. Bhagwat, L. Tassiulas and R. LaMaire, “Distributed Topology Construction of Bluetooth Wireless Personal Area Networks,” *IEEE Journal on Selected Areas in Communications*, vol. 23, issue 3, pp. 633 – 643, March 2005.

[18] S. Basagni and C. Petrioli, “A Scatternet Formation Protocol for Ad Hoc Networks of Bluetooth Devices,” in *Proceedings of the IEEE Semi-annual Vehicular Technology Conference (VTC)*, Birmingham, USA, May 2002.

[19] S. Basagni and C. Petrioli, "Degree-Constrained Multihop Scatternet Formation for Bluetooth Networks," in *Proceedings of the IEEE Globecom*, Taipei, Taiwan, November 2002.

[20] Z. Wang, R. Thomas and Z. Haas, “Bluenet – A New Scatternet Formation Scheme,” in *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, Hawaii, January, 2002.

[21] C.C. Foo and K.C. Chua, "BlueRings: Bluetooth scatternets with ring structures," in *Proceedings of 2nd IASTED International Conference on Wireless and Optical Communication (WOC 2002)*, Banff, Canada, July 2002.

[22] S. Avancha, A. Joshi and T. Finin, “Enhanced service discovery in Bluetooth,” *IEEE Computer*, vol. 35, issue 6, pp. 96-99, June 2002.

[23] O. Ratsimor, D. Chakraborty, A. Joshi and T. Finin, “Allia: Alliance-Based Service Discovery for Ad-hoc Environments,” in *Proceedings of the 2nd international workshop on Mobile Commerce*, Atlanta, USA, September 2002.

[24] N. A. Nordbotten, T. Skeie and N. D. Aakvaag. “Service Discovery in Bluetooth Scatternets,” in *Proceedings of Workshop on Mobile Ad Hoc Networking* (Institut EURECOM), Sophia-Antipolis, France 2003.

[25] C. Schwingenschlögl and A. Heigl, “Development of a Service Discovery Architecture for the Bluetooth Radio System,” in *Proceedings of the 6th EUNICE Open European Summer School*, Twente, Netherlands, September 2000.

[26] IEEE 802.15 WPAN Task Group 3 (TG3). http://www.ieee802.org/15/pub/TG3.html.

[27] R. Handorean and G.-C. Roman, “Service Provision in Ad Hoc Networks,” in *Proceedings of the 5th International Conference on Coordination Models and Languages (Coordination 2002)*, F. Arbab and C. Talcott, (editors), Lecture Notes in Computer Science 2315, Springer-Verlag, York, UK, April 2002.

[28] U. C. Kozat and L. Tassiulas, "Service Discovery in Mobile Ad-hoc Networks: An Overall Perspective on Architectural Choices and Network Later Support Issues," *Ad-Hoc Networks*, vol. 2, no. 1, pp. 23-44, January 2004.

[29] L. M. Feeney, "An Energy Consumption Model for Performance Analysis of Routing Protocols for Mobile Ad-hoc Networks," *Journal on Mobile Networks and Applications*, vol. 6, no 3, pp. 239-249, Kluwer Academic Publishers, June 2001.

[30] P. Wei, and L. Xi-Cheng, "On the Reduction of Broadcast Redundancy in Mobile Ad hoc Networks," in *Proceedings of Mobile and Ad Hoc Networking and Computing*, Boston, USA, August 2000.

[31] G. Cornuejols, G. L. Nemhauser and L.A. Wolsey, "The Uncapacitated Facility Location Problem," in *Discrete Location Theory*, P. Mirchandani, and R. Francis (editors), pp.119-171, John Wiley and Sons, New York, 1990.

[32] V. Arya, N. Garg, R. Khandekar, A. Meyerson and V. Pandit, "Local Search Heuristics for k-Median and Facility Location Problems," *SIAM Journal on Computing Online*, vol. 33, no. 3, pp. 544-562, 2004.

[33] C. Papadimitriou, "Probabilistic Analysis of a Geometric Location Problems," *Springer,* vol. 1, no. 3, pp. 544-562, October, 2004.

[34] L. Qiu, V.N. Padmanabhan and G.M. Voelker, "On the Placement of Web Server Replicas," in *Proceedings of IEEE INFOCOM*, Anchorage, USA, April 2001.

[35] L. Bo, M.J. Golin, G.F. Italiano, D. Xin and K. Sohraby, "On the Optimal Placement of Web Proxies in the Internet," in *Proceedings of IEEE INFOCOM*, pp.1282-1290, New York, USA, March 1999.

[36] The UPnP Forum. http://www.upnp.org.

[37] Jini Network Technology, http://www.sun.com/jini.

[38] Service Location Protocol v2, IETF RFC2608, http://www.apps.ietf.org/rfc/rfc2608.html.

[39] J. Tyan and Q. H. Mahmoud, "A Comprehensive Service Discovery Solution for Mobile Ad-hoc Networks," *Mobile Networks and Applications*, vol. 10, no. 4, pp. 423-434, August 2005.

[40] P. Grace, G. S. Blair and S. Samuel, "A Reflective Framework for Discovery and Interaction in Heterogeneous Mobile Environments," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 9, no. 1, pp. 2-14, January 2005.

[41] O.V. Ratsimor, V. Korolev, A. Joshi, T. Finin and Y. Yesha, "Agents2Go: An Infrastructure for Location-Dependent Service Discovery in the Mobile Electronic Commerce Environment," in *Proceedings of ACM Mobile Commerce Workshop*, Rome, Italy, July 2001.

[42] R. Hermann, D. Husemann, M. Moser, M. Nidd, C. Rohner and A. Schade, "DEAPspace — Transient Ad-hoc Networking of Pervasive Devices," *Computer Networks*, vol. 35, issue 4, pp. 411–428, March 2001.

[43] M. Klein, B. König-Ries and P. Obreiter "Lanes - A Lightweight Overlay for Service Discovery in Mobile Ad Hoc Network," in *Proceedings of 3rd Workshop on Applications and Services in Wireless Networks (ASWN)*, Berne, Switzerland, July 2-4 2003.

[44] U. Saif, J.M. Paluska and V. Chauhan, "Practical Experience with Adaptive Service Access," in *ACM Mobile Computing and Communication Review Journal (MC2R)*, vol. 9, pp. 27-40, January 2005.

[45] F. Sailhan and V. Issarny, "Scalable Service Discovery for MANET," in *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, March 2005.

[46] A. Friday, N. Davies, N. Wallbank, E. Catterall and S. Pink, "Supporting Service Discovery, Querying and Interaction in Ubiquitous Computing Environments," *Wireless Networks*, vol.10, no. 6, pp. 631-641, November 2004.

[47] D. Charlet, V. Issarny and R. Chibout, "Service Discovery in Multi-radio Networks: An Assessment of Existing Protocols," in *Proceedings of the 9th ACM international Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems*, Terromolinos, Spain, October 2006.

[48] D. Chakraborty, A. Joshi, Y. Yesha and T. Finin, "Toward Distributed Service Discovery in Pervasive Computing Environments," *IEEE Transactions on Mobile Computing*, vol. 5 issue 2, pp. 97-112, February 2006.

[49] V. Lenders, M. May and B. Plattner, "Service Discovery in Mobile Ad Hoc Networks: A Field Theoretic Approach," in *Journal on Pervasive and Mobile Computing*, vol. 1, issue 3, pp. 343-370, Elsevier, September 2005.

[50] M.J. Kima, M. Kumara and B.A. Shirazib, "Service discovery using volunteer nodes in heterogeneous pervasive computing environments," *Journal of Pervasive and Mobile Computing*, vol. 2, no. 3, pp. 313-343, September 2006.

[51] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees and M. Welsh, "Deploying a Wireless Sensor Network on an Active Volcano," *IEEE Internet Computing*, vol. 10, no. 2, pp. 18-25, March-April 2006.

[52] K. Lorincz, D. J. Malan, T.R.F. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh and S. Moulton, "Sensor Networks for Emergency Response: Challenges and Opportunities," *IEEE Pervasive Computing*, vol. 3, issue 4, pp. 16-23, October-December 2004.

[53] C. Chee-Yee and S.P. Kumar, "Sensor networks: Evolution, opportunities, and challenges," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1247 – 1256, August 2003.

[54] I.F. Akyildiz, Y.W.S. Sankarasubramaniam and E. Cayirci, "A Survey on Sensor Networks," *IEEE Communications Magazine*, vol 40, issue 8, pp 102 – 114, August 2002.

[55] M. Chen, T.K. Kwon, Y. Yuan, Y.H. Choi and V.C.M. Leung, "Mobile-agent-based Directed Diffusion (MADD) in Wireless Sensor Networks," *EURASIP Journal on Applied Signal Processing*, vol. 2007, issue 1, January 2007.

[56] H. Qi, S. S. Iyengar and K. Chakrabarty, "Multiresolution Data Integration Using Mobile Agents in Distributed Sensor Networks," *IEEE Transactions on Systems, Man and Cybernetics* – Part C: Applications and Reviews, vol. 31, no. 3, pp. 383-391, August 2001.

[57] P. Levis and D. Culler, "Maté: A Tiny Virtual Machine for Sensor Networks," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, USA, Oct. 2002.

[58] T. Liu and M. Martonosi, "Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems," in *Proceedings of ACM SIGPLAN: Symposium on Principles and Practice of Parallel Programming*, San Diego, USA, June 2003.

[59] J. Hui and D. Culler, "The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, Baltimore, USA, November 2004.

[60] A. Boulis, C.-C. Han and M. Srivastava, "Design and Implementation of a Framework for Efficient and Programmable Sensor Networks," in *Proceedings of the First International ACM Conference on Mobile Systems, Applications and Services*, San Francisco, USA May, 2003.

[61] P. Kang, C. Borcea, G. Xu, A. Saxena, U. Kremer, and L. Iftode, "Smart Messages: A Distributed Computing Platform for Networks of Embedded Systems," *The Computer Journal*, Special Issue on Mobile and Pervasive Computing, Oxford Journals, vol 47, no. 4, pp. 475-494, 2004.

[62] C.-L. Fok, G.-C. Roman and C. Lu, "Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications," in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS)*, Columbus, USA, June 2005.

[63] JXTA Technology, http://sun.com/jxta.

[64] M. Pirker, M. Berger and M. Watzke, "An Approach for FIPA Agent Service Discovery in Mobile Ad Hoc Environments," in *Proceedings of Workshop on Agents for Ubiquitous Computing*, New York, USA, July 2004.

[65] S. Gonzalez-Valenzuela, S.T. Vuong, and V.C.M. Leung, "A Reinforcement-Learning Approach to Service Directory Placement in Wireless Ad-hoc Networks," in *Proceedings of the 5th IEEE Workshop on Applications and Services in Wireless Networks*, Paris, France, June-July, 2005.

[66] S. Gonzalez-Valenzuela, S.T. Vuong, and V.C.M. Leung, "A Mobile-Directory Approach to Service Discovery in Wireless Ad-hoc Networks," in press, *IEEE Transactions on Mobile Computing*.

[67] S. González-Valenzuela, S.T. Vuong and V.C.M. Leung, "Programmable Agents for Efficient Topology Formation of Bluetooth Scatternets," accepted for publication, *International Journal of Wireless and Mobile Computing*, Inderscience Publishers, November 2004.

[68] IEEE 802.15 Working Group for WPAN standards. http://grouper.ieee.org/groups/802/15.

[69] Y. Kawamoto, V.W.S. Wong and V.C.M. Leung, "A Two-phase Scatternet Formation Protocol for Bluetooth Wireless Personal Area Networks," *IEEE Wireless Communications and Networking Conference (WCNC)*, New Orleans, Louisiana, March 2003.

[70] P. Sapaty, "Mobile Processing in Distributed and Open Environments," *John Willey & Sons*, 2000.

[71] S.T. Vuong and I. Ivanov, "Mobile Intelligent Agent Systems: Wave Vs. Java," in *Proceedings of IEEE Emerging Technologies and Applications in Communications*, Portland, USA, March 1996.

[72] S. González-Valenzuela and V.C.M. Leung, "QoS-Routing for MPLS Networks Employing Mobile Agents," *IEEE-Network*, vol. 16, issue 3, pp. 16-21 May-June 2002.

[73] S. González-Valenzuela, S.T. Vuong and V.C.M. Leung, "BlueScouts - A Scatternet Formation Protocol Based on Mobile Agents," in *Proceedings of the IEEE 4th Workshop on Aplications and Services in Wireless Networks*, Boston, USA, August 2004.

[74] P. Borst, "An Architecture for Distributed Interpretation of Mobile Programs," PhD dissertation, Faculty of Informatics, University of Karlsruhe, 2001.

[75] F. Yu, V.W.S. Wong and V.C.M. Leung, "Efficient QoS Provisioning for Adaptive Multimedia in Mobile Communication Networks by Reinforcement Learning," *ACM/Springer J. Mobile Networks and Applications*, vol. 11, no. 1, pp. 101-110, February 2006.

[76] V.W.S. Wong, M.E. Lewis and V.C.M. Leung, "Stochastic Control of Path Optimization for Inter-switch Handoffs in Wireless ATM Networks," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 336-350, June 2001.

[77] C. K. Tham and J. C. Renaud, "Multi-Agent Systems on Sensor Networks: A Distributed Reinforcement Learning Approach," in *Proceedings of the 2nd International Conference on*

Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Melbourne, Australia, December 2005.

[78] T. Camp, J. Boleng and V. Davies, "A Survey of Mobility Models for Ad Hoc Network Research," in *Wireless Communication & Mobile Computing (WCMC)*, Special Issue on Mobile Ad Hoc Networking Research, Trends and Applications, vol. 2, no. 5, pp. 483-502, 2002.

[79] F. Bai, N. Sadagopan and A. Helmy, "IMPORTANT: A Framework to Systematically Analyze the Impact of Mobility on Performance of RouTing protocols for Adhoc NeTworks," in *Proceedings of IEEE INFOCOM*, San Francisco, USA, April 2003.

[80] X. Hong, M. Gerla, G. Pei and C.-C. Chiang, "A Group Mobility Model for Ad Hoc Wireless Networks," in *ACM/IEEE MSWiM*, August 1999.

[81] G. Lin, G. Noubir and R. Rajaraman, "Mobility Models for Ad hoc Network Simulation," in *Proceedings of IEEE INFOCOM*, Hong Kong, China, March 2004.

[82] S. Ross, "Introduction to Probability Models," *Academic Press*, 7th Ed., New York, 2000.

[83] C. Bettstetter, G. Resta and P. Santi, "The Node Distribution of the Random Waypoint Mobility Model for Wireless Ad Hoc Networks," *IEEE Transactions On Mobile Computing*, vol. 2, no. 3, pp. 257-269, July-September 2003.

[84] P. Nain, D. Towsley, L. Benyuan, Z. Liu, "Properties of Random Direction Models," in *Proceedings of the 24th Conference of the IEEE INFOCOM*, Miami, USA, March 2005.

[85] M. L, Puterman, "Markov Decision Processes," *Wiley Interscience*, New York, USA, 1994.

[86] R.S. Sutton and A.G. Barto, "Reinforcement Learning – An Introduction," MIT Press, Cambridge, USA, March 1998.

[87] Y. Pointurier and F. Heidari, "Reinforcement Learning Based Routing in All-optical Networks with Physical Impairments," in *Proceedings of the IEEE Conference on*

Broadband Communications, Networks, and Systems (Broadnets), Raleigh, USA, September 2007.

[88] Z. Liu and I. Elhanany. "RL-MAC: A Reinforcement Learning Based MAC Protocol for Wireless Sensor Networks," *International Journal of Sensor Networks* vol. 1, no.3/4, pp. 117 – 124, Inderscience Publishers 2006.

[89] V. Charvillata and R. Grigoras, "Reinforcement Learning for Dynamic Multimedia Adaptation," *Journal of Network and Computer Applications*, vol. 30, issue 3, pp 1034-1058, August 2007.

[90] Y.-H. Chang, T. Ho, L. P. Kaelbling, "Mobilized Ad-Hoc Networks: A Reinforcement Learning Approach," in *Proceedings of the First International Conference on Autonomic Computing (ICAC'04)*, New York, USA, May 2004.

[91] C. J. C. H. Watkins, "Learning from Delayed Rewards," PhD thesis, Cambridge University, Cambridge, England, 1989.

[92] S. J. Bradtke and M. O. Duff, "Reinforcement Learning Methods for Continuous-Time Markov Decision Problems," in *Advances in Neural Information Processing Systems 7*, G. Tesauro, D. S. Touretzky and T. K. Leen (editors), MIT Press, Cambridge, USA 1995.

[93] The OMNeT++ Discrete Event Simulator Environment: http://www.omnetpp.org.

[94] D. Johnson, D. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," in *Mobile Computing*, Imielinski and Korth (editors), Kluwer Academic Publishers, 1996.

[95] C. Perkins, E. Belding-Royer and S. Das, "Ad-hoc On-Demand Distance Vector (AODV) Routing," *IETF RFC3561*, July 2003.

[96] J. Yoon, M. Liu and B. Noble, "Random Waypoint Considered Harmful," in *Proceedings of IEEE INFOCOM*, San Francisco, USA, April 2003.

[97] A. Nedos, K. Singh, R. Cunningham and S. Clarke, "A Gossip Protocol to Support Service Discovery with Heterogeneous Ontologies in MANETs," in *Proceedings of the Third IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, White Plains, USA, October 2007.

[98] S. B. Mokhtar, A. Kaul, N. Georgantas and V. Issarny, "Efficient Semantic Service Discovery in Pervasive Computing Environments," *Lecture Notes in Computer Science (LNCS)*, Springer, vol. 4290, pp. 240-259, 2006.

[99] H. Artail, H. Al-Asadi, "A Cooperative and Adaptive System for Caching Web Service Responses in MANETs," in *Proceedings of the IEEE International Conference on Web Services*, pp. 339-346, Chicago, USA, September 2006.

[100] D. Bianchini, V. D. Antonellis, M. Melchiori and D. Salvi, "Lightweight Ontology-Based Service Discovery in Mobile Environments," in *Proceedings of the 17th International Conference on Database and Expert Systems Applications DEXA*, Krakow, Poland, September 2006.

[101] M. Wagner and M. Paolucci, "mobiXPL – Semantic-based Service Discovery on Tiny Mobile Devices," in *Proceedings of the 1st Workshop on Semantic Web Technology for Mobile and Ubiquitous Applications*, Hiroshima, Japan, November 2004.

[102] B. Liang and Z. J. Haas, "Virtual Backbone Generation and Maintenance for Ad-hoc Network Mobility Management," in *Proceedings of 19th IEEE INFOCOM*, Tel Aviv, Israel, March 2000.

[103] J. Wu and F. Dai. "A Distributed Formation of a Virtual Backbone in MANETs Using Adjustable Transmission Ranges," in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS)*, Tokio, Japan, March 2004.

[104] I. Hokelek, M.U. Uyar and M.A. Fecko, “Random-Walk Based Analysis of Virtual Backbone in MANETs,” in *IASTED International Conference on Communications and Computer Networks (CCN)*, Marina del Rey, USA, October 2005.

[105] O. Dagdeviren and K. Erciyes, “A Distributed Backbone Formation Algorithm for Mobile Ad Hoc Networks,” *Lecture Notes in Computer Science*, vol. 4330/2006, pp. 219-230, Springer Berlin / Heidelberg, 2006.

[106] K. Xu, X. Hong and Mario Gerla, “An Ad Hoc Network with Mobile Backbones,” *IEEE International Conference on Communications (ICC),* New York, USA, May 2002.

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

[108] The TinyOS Operating System for Embedded Sensor Networks. http://www.tinyos.net.

[109] D. Kotz, R. Gray and D Rus, “Future Directions for Mobile-Agent Research,” IEEE Distributed Systems Online, vol. 3, no 8, August 2002.

[110] S. Hadim and N. Mohamed, “Middleware Challenges and Approaches for Wireless Sensor Networks,” *IEEE Distributed Systems Online*, vol. 7, no. 3, March 2006.

[111] L. W. McKnight, J. Howison and S. Bradner, “Wireless Grids - Distributed Resource Sharing by Mobile, Nomadic and Fixed Devices,” *IEEE Internet Computing*, vol. 8, no. 4, pp. 24-31, July-August 2004.

[112] S. González-Valenzuela, S.T. Vuong and V.C.M. Leung, "A Mobile Code Platform for Distributed Task Control in Wireless Sensor Networks," in *Proceedings of 6th ACM MobiDE*, Chicago, USA, June 2006.

[113] M. Chen, S. Gonzalez-Valenzuela and V.C.M. Leung, "Applications and Design Issues of Mobile Agents in Wireless Sensor Networks," *IEEE Wireless Communications*, vol 14, issue 6, pp. 20-26, December 2007.

[114] P. Sapaty, "A Wave Language for Parallel Processing of Semantic Networks," Computers and Artificial Intelligence, vol. 5, no. 4, 1986.

[115] K. Cheolgi, M. Joongsoo and L. Joonwon, "A Random Inquiry Procedure Using Bluetooth," *IEICE Transactions on Communications*, vol. no. 9, pp. 2672-2683, September 2003.

[116] G.V. Zaruba and V. Gupta, "Simplified Bluetooth Device Discovery — Analysis and Simulation," in *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)* - Track 9, January 2004.

[117] B.S. Peterson, R.O. Baldwin and J.P. Kharoufeh, "Bluetooth Inquiry Time Characterization and Selection," *IEEE Transactions on Mobile Computing*, vol. 5, no. 9, pp. 1173-1187, September 2006.

[118] M. Lauer and M. Riedmiller, "An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-agent Systems," in *Proceedings of 17th International Conference on Machine Learning*, San Francisco, USA, 2000.

[119] S. Kapetanakis and D. Kudenko, "Reinforcement Learning of Coordination in Cooperative Multi-agent Systems," in *Proceedings of the 8th national conference on Artificial intelligence*, Edmonton, Canada, July 2002.

[120] C. Guestrin, M.G. Lagoudakis and R. Parr, "Coordinated Reinforcement Learning," in *Proceedings of the 19th International Conference on Machine Learning*, San Francisco, USA 2002.

[121] L. Panait, K. Sullivan and S. Luke, "Lenient Learners in Cooperative Multiagent systems," in *Proceedings of the fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, Hakodate, Japan, May 2006.

[122] J. Huang, B. Yang and D.-Y. Liu, "A Distributed Q-Learning Algorithm for Multi-Agent Team Coordination," in *Proceedings of the 4th International Conference on Machine Learning and Cybernetics*, Guangzhou, China, August 2005.