High-Efficiency Service Discovery in Wireless Mobile Ad Hoc Networks

by

Yu Yang

A thesis submitted to the

School of Computing

in conformity with the requirements for

the degree of Master of Science

Queen's University

Kingston, Ontario, Canada

April 2005

Your file Votre référence
ISBN: 978-0-494-38736-8
Our file Notre référence
ISBN: 978-0-494-38736-8

ABSTRACT

Being infrastructure less mobile networks, MANETs (Mobile Ad hoc NETworks) have no central control and are self organized. This makes service discovery more difficult than in wired networks. Unfortunately, most of the potential applications for MANETs must invoke some service requests during their runtime. Traditional service discovery protocols such as JINI, UPnP and SLP cannot be directly applied to MANETs because of their message exchange requirements and the heterogeneity and mobility of MANETs. In this thesis, we propose a new high efficiency service discovery (HESED) scheme, which is based on multicast query and multicast reply and hence is fundamentally different from existing service discovery protocols. In this scheme, after a client node sends a service query, all the matched servers advertise their information to all the nodes. Clients cache the received service information, and are capable of evaluating and utilizing the cached information. This can reduce many potential clients' queries. The message complexity of HESED is shown to be O(N) for N-node MANETs as opposed to O(N^2) for traditional schemes. Furthermore, HESED eliminates the effect of wireless asymmetric links and provides reliable connectivity.

ACKNOWLEDGEMENTS

Deepest thanks to my supervisor, Dr. Hossam Hassanein, for his supervision and encouragement of my research and the completion of this thesis. I would also like to thank all my friends in the Telecommunication Research Lab (TRL) at Queen's University for giving me so much information in the area of wireless communication and networking during our weekly group seminar. Special thanks go to Abdelhamid Taha and Afzal Mawji for their help in proofreading the thesis.

I would also like to acknowledge the financial support provided by the School of Computing and Communications and Information Technology Ontario (CITO). Thanks should also be given to Debby Robertson, Gary Powley, Tom Bradshaw and Richard Linley for their help.

I am extremely thankful to my wife Tianwen He and my parents for their long-term support and encouragement in all aspects of my life.

TABLE OF CONTENTS

ABSTRACT....ii
ACKNOWLEDGEMENTS....iii
LIST OF TABLES....vii
LIST OF FIGURES....viii
LIST OF ACRONYMS....x
Chapter 1 Introduction....1
1.1 Service Discovery....1
1.2 MANETs....2
1.3 Routing in MANETs....4
1.4 Motivation And Importance....6
1.5 Contributions....8
1.6 Thesis Organization....10
Chapter 2 Existing Service Discovery Protocols....11
2.1 JINI and UDDI....11
2.1.1 Architecture....11
2.1.2 How JINI Works....12
2.1.3 JINI and UDDI in MANETs....13
2.2 Universal Plug and Play (UPnP)....13
2.2.1 Architecture....13
2.2.2 UPnP Procedures....14
2.2.3 UPnP in MANETs....16
2.3 Service Location Protocol (SLP)....16
2.3.1 Architecture....16
2.3.2 SLP In MANETs....17
2.4 Post-Query Strategies....18
2.4.1 Introduction....18
2.4.2 Request-Reply Strategies....18
2.4.3 Comments....19
2.5 Konark – A Service Discovery and Delivery Protocol....20
2.5.1 Architecture....20
2.5.2 Service Discovery....20
2.5.3 Comments....21
2.6 Network Layer Support for Service Discovery....21
2.6.1 Architecture....21
2.6.2 Backbone Management (BBM)....22
2.6.3 Distributed Service Discovery....22
2.6.4 Comments....23
2.7 Other MANET Service Discovery Research....23
Chapter 3 Proposed Service Discovery Scheme (HESED)....25
3.1 Objective....25
3.2 Framework Overview....27
3.3 Multicast Query and Multicast Reply Algorithm....28
3.4 Long Beacon Neighbour Detection....31

3.4.1 MANET Organization........32
3.4.2 Proposed Neighbour Detection Algorithm........33
3.5 Edge Node Forwarding Algorithm........36
3.6 Cache Algorithm........39
3.6.1 Half Neighbour Changing Time T_α........39
3.6.2 Connection Validation Probability (CVP)........40
3.6.3 Route Validation Probability (RVP)........42
3.6.4 Cost Function for Service Discovery........42
3.6.5 Cache Selection Algorithm........45
3.7 Backward Learning Routing Algorithm........46
3.8 XML Packet Format........47
3.9 HESED Analysis........47
3.9.1 Performance of Multicast Mesh........47
3.9.2 Message Complexity of HESED........50
3.9.3 Other Performance Issues........54
Chapter 4 HESED Design and Implementation........56
4.1 Overview........56
4.2 Graphical User Interface........58
4.2.1 Join Panel........58
4.2.2 Search Panel........59
4.2.3 Result Panel........60
4.3 Experimental Tests........61
4.3.1 Asymmetric Link Scenario........61
4.3.2 Short Linear Chain Scenario........62
4.3.3 Long Linear Chain Scenario........63
4.3.4 Multiple Routes Scenario........63
4.3.5 Bottleneck Scenario........64
4.3.6 Printer Server Test........65
4.3.7 Test Results........66
Chapter 5 HESED Simulation and Result Analysis........70
5.1 Simulation Setting........70
5.1.1 Basic Parameters........70
5.1.2 Mobility Model........72
5.1.3 Medium Access Control Model........73
5.2 Test Scenarios........74
5.2.1 Scenarios Descriptions........74
5.2.2 Performance Metrics........75
5.3 Simulation Analysis........77
5.3.1 Effect of Number of Packets........77
5.3.2 Cache Hit Rate........80
5.3.3 Delay........82
Chapter 6 Conclusions and Future Work........85
6.1 Summary........85

6.2 Future Work 86
6.2.1 Reliable Service Discovery 86
6.2.2 Tree-based Service Search 87
6.2.3 Security Over Service Discovery 88
6.2.4 XML Templates 88
References 89
Appendix A XML Packet Format 94
A.1 Client's Multicast Query 94
A.2 Server's Multicast Reply 94
A.3 Client's Unicast Confirmation 95
A.4 Server's Unicast Acknowledgement 95
A.5 Client's Service Request 96
A.6 Server's Result Report 97
Appendix B Packets Format 98
B.1 Hello Message 98
B.2 Multicast Packet Format 98
B.3 Unicast Packet Format 100
Appendix C HESED Implementation Architecture 103
C.1 Client Package 103
C.1.1 Overview 103
C.1.2 Algorithms for Client Package 105
C.2 Server Package 106
C.3 Hello Package 107
C.3.1 Overview 107
C.3.3 Algorithms for Hello Package 108
C.4 Multicast Package 112
C.4.1 Overview 112
C.4.2 Flowchart For Multicast Packet Handler 113
C.5 Unicast Package 114
C.5.1 Unicast Package Architecture 114
C.5.2 Flowchart for Unicast Packet Handler 115

LIST OF TABLES

Table 3.1 HESED Framework... 28
Table 5.1 Simulation Environment Parameters...72
Table 5.2 Mobility Model Parameters.. 73
Table 5.3 Parameters for Pattern A 76
Table 5.4 Parameters for Pattern B 77
Table 5.5 Parameters for Pattern C 77

LIST OF FIGURES

Figure 1.1 Wireless Station Mode....3
Figure 1.2 Wireless Ad Hoc Mode....3
Figure 2.1 A Typical JINI System 13
Figure 2.2 UA/SA/DA Architecture....16
Figure 2.3 Konark Architecture....20
Figure 3.1 Algorithms for Client and Server Nodes.... 29
Figure 3.2 Multicast Query and Multicast Reply Algorithm....30
Figure 3.3 Symmetric/Asymmetric Link Detection....35
Figure 3.4 Route Recovery....36
Figure 3.5 Edge Node Set....37
Figure 3.6 CVP Calculation Diagram....40
Figure 3.7 Average Distance Calculation Diagram....43
Figure 3.8 Number of Edge Node Analysis....49
Figure 4.1 HESED Implementation Architecture....57
Figure 4.2 Join Panel....58
Figure 4.3 Search Panel....59
Figure 4.4 Result Panel....62
Figure 4.5 Asymmetric Link Scenario....62
Figure 4.6 Short Linear Chain Scenario....62
Figure 4.7 Long Linear Chain Scenario....63
Figure 4.8 Multiple Route Scenario....64
Figure 4.9 Bottleneck Scenario....64
Figure 4.10 MANET Printer Project....65
Figure 4.11 Test Result(1) - Join Panel.... 66
Figure 4.12 Test Result(2) - Search Panel 67
Figure 4.13 Test Result(3) – Server's Reply in Result Panel 67
Figure 4.14 Test Result(4) – Server's ACK in Result Panel 68
Figure 4.15 Test Result(5) – Server Output 68
Figure 4.16 Test Result(6) – Client Output 69
Figure 5.1 Two Schemes in the Simulation....74
Figure 5.2 Average Numbers of Packets Per Query....79
Figure 5.3 Comparisons for Cache Hit Rate....81
Figure 5.4 Comparison for delay....84
Figure 6.1 Service Searching Tree....87
Figure A.1 Multicast Query Message Format 94
Figure A.2 Multicast Reply Message Format 94
Figure A.3 Client Confirmation Message Format 95
Figure A.4 Server Acknowledge Message Format 96
Figure A.5 Service Request Message Format.... 96
Figure A.6 Service Result Message Format 97
Figure B.1Hello Message Format....98
Figure B.2 Multicast Packet Header Position 98
Figure B.3 Multicast Packet Header.... 99

Figure B.4 Multicast Packet State Field 100
Figure B.5 Unicast Packet Header Position 100
Figure B.6 Unicast Packet Header 101
Figure B.7 Unicast Packet State Field...102
Figure C.1 Client Package Architecture...103
Figure C.2 Algorithm for Client Package..105
Figure C.3 Server Package Architecture..106
Figure C.4 Hello Package Architecture..107
Figure C.5 Algorithm for Hello Package...108
Figure C.6 Disappeared Neighbour Problem...110
Figure C.7 Algorithm for Edge Node Selection.....................................111
Figure C.8 Algorithm for Half Neighbour Change Time.......................112
Figure C.9 Flowchart for Multicast Handler..113
Figure C.10 Unicast Package Architecture..115
Figure C.11 Flowchart for Unicast Packet Handler...............................116

LIST OF ACRONYMS

ACK	Acknowledgment
ABR	Associativity Based Routing
AODV	Ad Hoc On Demand Distance Vector
BBM	BackBone Management
CAN	Content Addressable Networks
CGSR	Clusterhead Gateway Switch Routing Protocol
CORBA	Common Object Request Broker Architecture
CVP	Connection Validation Probability
DA	Directory Agent
DHCP	Dynamic Host Configuration Protocol
DSDV	Destination Sequenced Distance Vector
DSR	Dynamic Source Routing
FTP	File Transfer Protocol
GSR	Global State Routing
GUI	Graphical User Interface
HESED	High Efficient SEvice Discovery
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
ISO/OSI	International Standard Organization's Open System Interconnect
JINI	Java Intelligence Naming Interface
LAN	Local Area Network
LAR	Location-Aided Routing
LBAR	Load-Balanced wireless Ad hoc Routing
IETF	Internet Engineering Task Force
MAC	Multiple Access Control
MANET	Mobile Ad hoc NETwork
ODMRP	On-Demand Multicast Routing Protocol
OLSR	Optimized Link State Routing Protocol
OS	Operating System
PDA	Personal Data Assistant
RMI	Remote Method Invocation
RDT	Reliable Data Transportation
RVP	Route Validation Probability
RTS/CTS	Require To Send/Clear To Send
SA	Service Agent
SLP	Service Location Protocol
SOAP	Simple Object Access Protocol
STAR	Source Tree Adaptive Routing protocol
TCP	Transmission Control Protocol

TORA	Temporally Ordered Routing Algorithm
TOSD	Traditional On-demand Service Discovery
TTL	Time-to-Live
UA	User Agent
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
UPnP	Universal Plug and Play
URL	Uniform Resource Locator
VAP	Virtual Access Point
VBS	Virtual Base Station Protocol
WRP	Wireless Routing Protocol
XML	Extensible Makeup Language
ZRP	Zone Routing Protocol

Chapter 1 Introduction

A service is an entity that can be used by a person, a program or another service. It has a general definition in service discovery and may be software objects, information access via the Internet, music on demand or classical services such as those offered by printers, scanners, fax machines, HTTP servers [1] and FTP servers [1].

1.1 Service Discovery

Service discovery finds a desired service over some network coverage that includes two important classes of entities: servers and clients. Clients need services and servers are willing to provide services to other nodes.

In general, there are two basic modes to find a service: *passive* mode and *active* mode. In passive mode, clients do not send any queries and servers advertise the available services periodically. When a client needs a service, and it does not have any information on it, it has to wait until it receives service information from the server. In active mode, clients proactively try to find the service. If a client needs a service, the client will send a service query to other nodes that can forward this query to the server. When a server receives a service request from a client directly or indirectly, the server will respond. Finally the client can setup a connection with the server and utilize the service after it receives the server's response. Many existing protocols support both modes.

In general, service discovery has three goals:

1. To search and browse for services
2. Choose the right service
3. Utilize the service

In this thesis, we focus on the first goal, which is the way to find out the server's information. Choosing and utilizing the service is application dependent and is beyond the scope of this thesis.

1.2 MANETs

Most wireless devices have two communication modes: station mode and peer-to-peer mode. In station mode (Figure 1.1), mobile terminals must find a station and register themselves to that station, which is usually the wireless access point. Mobile terminals can communicate with outside computers through this access point. In this mode, clients have to remain within the station's radio range.

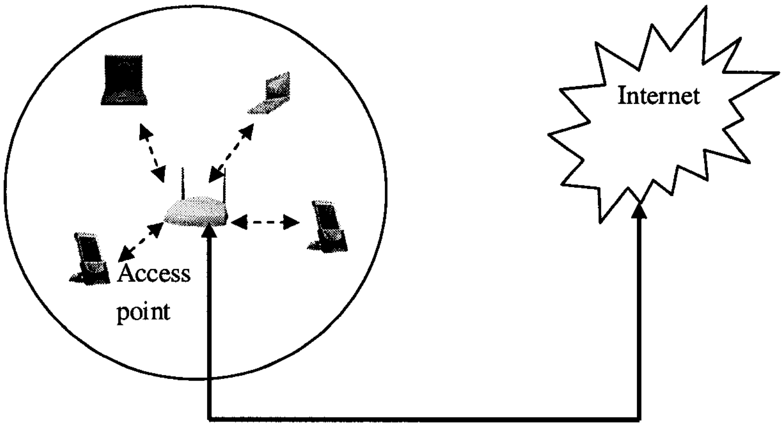


Figure 1.1 Wireless Station Mode

In ad hoc mode (Figure 1.2), there is no centralized management and nodes communicate with each other through multi-hop mode [4]. If the destination node is outside the sender node's radio range, then the sender has to find some other nodes to act as relay nodes for forwarding data packets to the destination node. In this mode, nodes may cooperate in routing each other's data packets and this enhances reliability and extensibility. For example, mobile nodes can select a good route to a destination and bypass an obstacle such as a concrete wall or an iron door, which is not possible in station mode.

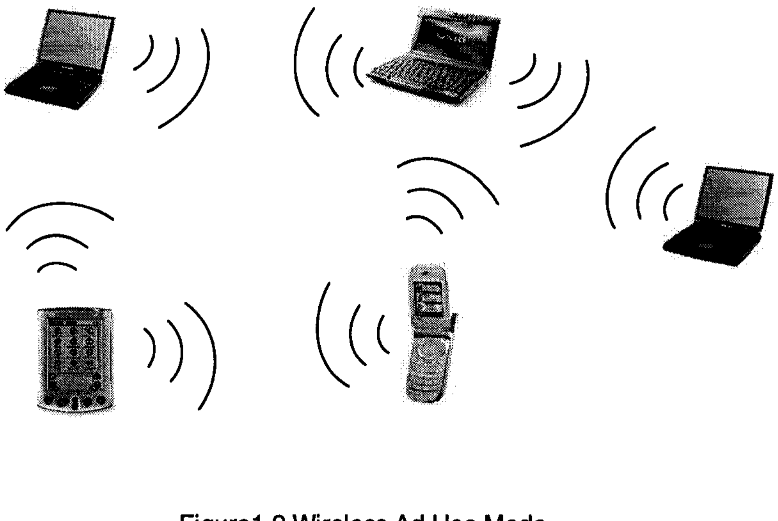


Figure1.2 Wireless Ad Hoc Mode

A MANET (Mobile Ad hoc NETwork) is a collection of wireless mobile nodes without an existing network infrastructure or centralized administration, and in which all mobile nodes can communicate with each other in multi-hop ad hoc mode [2][3]. Since MANETs do not need any backbone infrastructure, they can be widely used in many areas where such an infrastructure is not viable. For example, if we want to setup a wireless LAN to reliably cover the area inside a building that has obstacles, we can use many access points to bypass all the obstacles. If we use ad hoc mode, we only need a few access points and all computers can reach these access points through multi-hop connections, which effectually bypass obstacles.

1.3 Routing in MANETs

Routing is a critical factor for MANETs because a node must find a multihop route to the destination node before it can send a packet. In general, routing algorithms belong to the network layer and service discovery is in the application layer, so they are independent of each other. Because our proposed service discovery scheme is a cross-layer design, meaning the routing algorithm is implemented in the application layer, some MANET routing algorithms are presented in this section as background.

Routing protocols in conventional wired networks generally use either distance-vector or link-state routing algorithms [1], neither of which can be applied in a dynamic environment such as MANETs. Numerous multi-hop MANET routing protocols have been proposed [5-19] in the past few years. They can be classified into three categories: reactive, proactive and hybrid [20].

On-demand reactive protocols (DSR [5], LAR [6], ABR [7], AODV [8], TORA [9] and LBAR [10]) start route discovery only when needed. Whenever a source node wants to send a packet to a destination node, the source node has to send route request packets (control packets) and start a route discovery process. After a route is found, the source and the destination can communicate with each other through this route. The on-demand protocols significantly reduce the number of control packets because control packets are sent only when it is necessary. However, the first data packet cannot be sent until the route is ready, hence, the delay for the first packet can be very long [21].

Table-driven proactive protocols (CBA [11], GSR [12], OLSR [13], STAR [14], WRP [15], DSDV [16] and CGSR [17]) discover routes before they are needed. Each node maintains a global network-topology table by periodically exchanging information with its neighbours. When a source node wants to send a packet to the destination, it can read the local topology table and find out a route to the destination. Hence, the source node does not suffer a long delay before the packet is sent. Since nodes have to exchange the topology information packets (control packets) even if there is no route request, the number of control packets exchanged can be very large.

Hybrid methods (ZRP [18] and VBS [19]) utilize both proactive and reactive procedures and have all the advantages of both approaches. However, the hybrid methods also increase the complexity and have extra costs.

1.4 Motivation And Importance

With the advances in wireless technology, the number of wireless devices is increasing at a fast rate. At the same time, there are more and more services available in MANETs. However, the lack of centralized control in MANETs complicates service discovery. In this thesis, we present a new high-efficiency service discovery scheme for MANETs, which has low communication cost and short delay.

Most of the service discovery protocols have two common phases. The first involves the discovery of the service provider, and the second utilizes the service. There are many challenges in both phases. Because MANETs are vastly different from the traditional wired or wireless networks, it is always a big challenge to find a proper way to organize a MANET because of its mobility and heterogeneity. In general, structured networks are always easier to deal with than infrastructure-less networks. Organizing services and discovering them in MANETs are, therefore, a fundamental challenge for any MANET application.

Energy consumption is another important issue in MANETs. Service discovery must query enough nodes in order to find a server and the route to that server, so it may create a large amount of network traffic. Since mobile devices have very limited energy, managing the use of energy to run service discovery is a big challenge that must be faced. Our approach is energy efficient because we can significantly reduce the number of packets exchanged compared to traditional schemes.

Another issue is delay. Because packet transmission must go through a multi-hop connection, we have to control the delay from client to server below an acceptable level. There are three kinds of delays for multi-hop MANETs: processing delay at each node, request-propagation delay among the different nodes and medium access delay. Processing delay is the time for processing a packet at a node, such as the time for a client node to prepare a query, for intermediate nodes to update and forward packets or for a server node to create a reply. Propagation delay is the delay for a node sending or receiving packets. Medium access delay in wireless networks may take a much longer time than it does in wired networks because all the wireless equipment has to share the limited media resource. There are three ways to reduce delay: decrease the number of hops, prevent communication bursts in a small area and reduce the computation load at each relay node. Our approach reduces delay in the following three ways:

(a) Reduce computation load for intermediate nodes thus shortening the processing delay.

(b) Prevent adjacent nodes from requesting medium access within a short time period thus decreasing packet collisions and reducing the medium access delay

(c) Use cached service information to reduce the overall delay.

In MANETs, the communication radii for different nodes may be different and asymmetric links are universal phenomena in wireless connections. All nodes can be mobile and connections between different nodes are not always available. Because each node has only limited energy, there is no such thing as a permanent node in MANETs. Service discovery should be capable of eliminating such adverse effects and providing reliable service to the users.

In general, we need a fast, energy-efficient, reliable and extensible service discovery scheme among the non-permanent nodes using the unreliable connections of MANETs. Developing a good scheme is very difficult, but will prove useful for wireless applications. Most wireless applications need exterior services and the performance of service discovery affects the overall application performance. Without a high-efficiency service discovery scheme, we cannot have wireless application access with acceptable performance.

1.5 Contributions

The main contributions of this thesis are as follows.

(1) We have analyzed the problems and provided solutions for service discovery over MANETs. Finding a service provider among mobile nodes is a difficult task. Traditional on-demand solutions use multicast query and unicast reply, in which client nodes send their service queries to all nodes and server nodes respond by sending a reply to clients by unicast. In this thesis, we propose a new algorithm, which is based on multicast query and multicast reply, which is fundamentally different from existing service discovery protocols. In this algorithm, after a client node sends a service query, all the matched servers advertise their information to all nodes. This can reduce many potential client queries. The scheme is an on-demand service discovery protocol, but also has the advantages of passive service discovery.

(2) We propose new routing algorithms for multicast communication and unicast communication, called the "Edge Node Forwarding Algorithm" and the

"Backward Learning Algorithm". Edge Node Forwarding is a selective forwarding algorithm in which only nodes selected forward multicast packets. With different selection criteria, we could build different kinds of multicast meshes for various user requirements, such as a fast mesh, a minimum size mesh or a minimum energy consumption mesh. Backward Learning Routing has no routing phase and unicast routing is just a by-product of the multicast procedure. It can shorten the delay and reduce energy consumption at the same time.

(3) We develop Java-based service discovery software, which can be used either as a client to detect, select and confirm a service, or as a service provider to respond to and accept clients. We provide a client GUI interface, which is very convenient for novice users. In our implementation, we also solve the asymmetric link problem, which is common in wireless LANs.

(4) We design a simulation model to measure the performance of our proposed scheme. Simulation results show our scheme is faster and more energy efficient than traditional schemes.

(5) We have set up a mathematical model for message complexity of the proposed scheme and used this model to analyze energy consumption.

(6) Other MANET protocols can benefit from the work in this thesis. Our service discovery algorithms can serve as middleware for other higher level MANET protocols such as data replication and task scheduling since they must find and utilize services from other mobile nodes.

1.6 Thesis Organization

This thesis is organized as follows:

- Chapter 2 reviews existing service discovery protocols, such as JINI, UPnP, SLP, Post and Query strategies and Konark.

- Chapter 3 presents our service discovery scheme and the algorithms to realize it. A framework overview is used to show the relationship among these algorithms.

- Chapter 4 discusses the implementation issues, such as the project architecture, class diagram, critical components and user interface. We also describe our test cases in this chapter, which include a linear chain scenario, a multiple route scenario and a bottleneck scenario.

- Chapter 5 presents the simulation results, which include basic parameter settings, test case descriptions, and numerical result and their analysis.

- Chapter 6 provides conclusions of the research and directions for future work.

Chapter 2 Existing Service Discovery Protocols

Service discovery protocols in wired networks can be either agent-based or without agents. We introduce several existing service discovery protocols and post-query strategies on MANETs. After a brief introduction of each protocol, we provide a short analysis of their viability for MANETs.

2.1 JINI and UDDI

2.1.1 Architecture

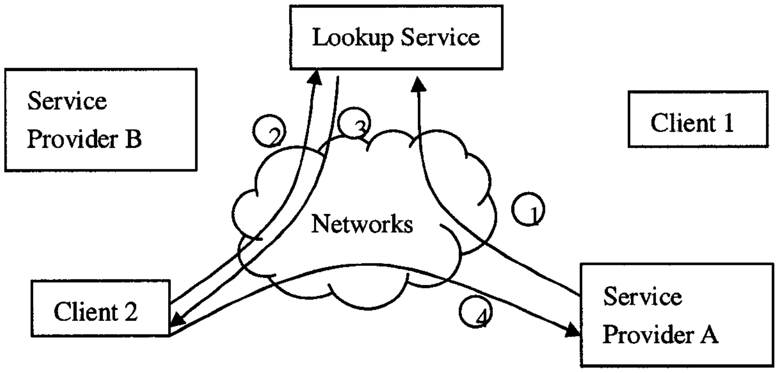
JINI[22] (Java Intelligent Network Interface) is a Java-based service discovery protocol, which is good for programming service discovery applications. UDDI[23] (Universal Description, Discovery and Integration) is another industry standard which focuses on Web services. The main architectures of JINI/UDDI are quite similar because both of them are agent-based service discovery protocols.

JINI has three components: service provider, lookup service and client. The service provider can provide service to clients over a published interface. Services can be physical devices (printer, digital camera, coffee machine, etc) or software objects. In JINI, a service can be implemented in any programming language, but it must have a Java interface available to clients, which is used for finding other JINI devices and services. Service implementations can be written using a variety of platforms, such as SOAP [24], RMI [25] or CORBA [26]. The lookup service acts as a broker between services and clients.

UDDI has a Web-based distributed directory system which is called as "UDDI Business Registry". It allows businesses to list themselves on the internet and discover each other.

2.1.2 How JINI Works

The lookup service acts as a service registrar in JINI (Figure 2.1). When a server or a client enters a JINI network, the first thing it does is to get a "service registrar object" (Java object) from the lookup service. The server can use the service registrar object for registration. During its registration, the server will send a copy of its service proxy (Java object) to the lookup service. On the other hand, when a client receives the service registrar object from the lookup service, it can use that service registrar object to search for an expected service. If such a service provider has registered in the lookup service, the lookup service should send a service proxy to the client, and then the client can use the service proxy to communicate with the service directly.



(1) Service provider A registers itself to lookup service
(2) Client 2 requests service provider information from lookup
(3) Lookup service sends service provider's interface to client 2
(4) Client 2 sets up a connection with service provider A

Figure 2.1 A Typical JINI System

2.1.3 JINI and UDDI in MANETs

JINI and UDDI cannot be applied in MANETs due to the difficulty of finding a node that can work as a lookup service center (in JINI) or a business registry (in UDDI) since each node only has limited lifetime and the energy consumption of such nodes is much higher than the normal level, which can cause depletion of the broker node.

2.2 Universal Plug and Play (UPnP)

2.2.1 Architecture

UPnP[29] stands for Universal Plug and Play, and is promoted by Microsoft. Universal plug and play means that whenever a device is connected to a network, it can be used by other network devices automatically without the need for driver installation or set up. It looks more like a large and loose operating system, and any device can connect and disconnect at any time.

There are three main components in UPnP: control points, devices and services. The service is the smallest unit in UPnP networks. A service in UPnP device includes a state table, a control server and an event server. The state table has the state of the current service. For instance, a clock server has a state “current time”, a printer server has states “idle”, “busy”, ”data transporting”, etc. The control server can change the states in the state table and the event server will announce the changed state to the whole network.

A UPnP device consists of services and nested devices. Control points can retrieve the device and service description, invoke actions to control the service and receive the event announcement from a service’s event server.

2.2.2 UPnP Procedures

There are six steps for invoking UPnP service discovery: addressing, discovery, description, control, eventing and presentation.

A. Addressing

When a device enters an UPnP network, it needs an IP address [30] to identify itself. There are two ways of getting an IP address: DHCP (Dynamic Host Configuration Protocol) [31] and Auto IP [32].

B. Discovery

After a new device gets an IP address, it will advertise its existence to all the control points and the discovery message contains a few, essential specifics about this device. If a new control point enters a UPnP network, it will search for all the interested devices. After this stage, control points and devices have basic knowledge about each other.

C. Description

If the control point wants to know more about a device, it can retrieve the device's detailed description from the URL provided by the device in the discovery message. This description includes all the information for the device, such as model name, serial number, manufacturer information, and a list of embedded devices or services, URLs for control, eventing and presentation.

D. Control

Control points can extract the necessary control information from the service description, such as command or action lists, parameters for each command, variables, data types, etc. Control points can send an action request to a device's control URL [1] for using this service.

E. Eventing

Each service has a state table that contains one or more state variables. An event changes one of the state variables. A service or device should announce the state change to all control points by sending an event message. An event message contains the name and current value of the changed variable.

F. Presentation

A device should have URL page for presenting itself to the users. A user can get the current device status and control the device by browsing the URL page.

2.2.3 UPnP in MANETs

UPnP cannot be applied to MANETs because most of the searching, registration and event updating packets in UPnP have to be broadcast to the entire network. This can be too expensive in MANETs due to bandwidth and energy concerns.

2.3 Service Location Protocol (SLP)

2.3.1 Architecture

Service Location Protocol (SLP)[33] (Figure 2.2), developed by the IETF, is a vendor-independent standard on TCP/IP networks. SLP has three main components: the User Agent (UA), the Service Agent (SA) and the Directory Agent (DA).

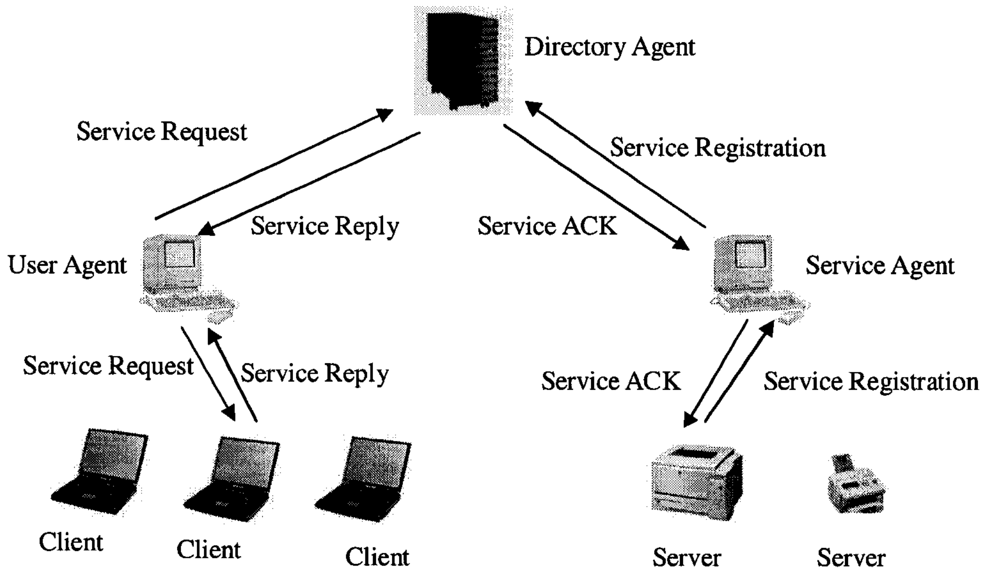


Figure 2.2 UA/SA/DA Architecture

A. User Agents (UA):

When a client needs a service, it sends a service discovery request to its UA. The UA passes this request to its DA, and the DA searches all the registered services. If there is a service available in the DA, the DA sends the service information to the UA, and the UA forwards this information to the client.

B. Service Agents (SA):

When a new server enters an SLP network, it must first register itself with DAs. The server sends a registration request to an SA, and the SA forwards it to DAs. DAs perform the registration, and send an ACK to the SA. After the SA receives the ACK, it sends an ACK to the server, so that server knows the registration is finished. A server also can send a cancellation message to its SA when it leaves this network.

C. Directory Agent (DA):

The DA saves all the registered service information in its database and responds to the UA's queries.

2.3.2 SLP In MANETs

It is obvious that we could not apply SLP to MANETs. SLP needs many different agents and it is very difficult to implement an agent on unreliable mobile nodes. Even if we have all these agents and a client can get the desired server's information such as IP address, the client still has to query the entire network to find out a route to that server. These costs are not affordable by the mobile nodes in MANETs who only have limited battery-energy.

2.4 Post-Query Strategies

2.4.1 Introduction

A Post-Query protocol[34] is a time-dependent protocol executed in rounds and modeling request-reply interactions between clients and servers on MANETs.

The network is modeled as a graph G = (N,L), where N = {1, 2, 3,...n} is a set of mobile nodes, L is the set of directed links between all the nodes. Let us assume P(s) is the post protocol, Q(c) is the query protocol. There is one pair (P, Q) for each service discovery round. In each round, a server posts service information to some nodes, and a client queries service information from some other nodes. A client can find a service if and only if P∩Q is not empty in the current round. The main idea of this protocol is that the client receives service replies from intermediate nodes.

2.4.2 Request-Reply Strategies

There are four different strategies for request-reply in this protocol which are "Post-to-all", "Query-to-all", "Uniform memory less" and "Incremental post-query".

In the "Post-to-all" strategy, the server advertises service information to all the clients, and clients save the service information into their local cache. Servers should update the service information periodically before it expires. When a client wants a service, it does not need to query any other nodes because the service information is already in its local cache. The algorithm is good for small and static networks. In a small and static network, the server posts the service information only once, and the service information can be used by clients for a long time, thus there is no querying delay for each client.

In the “Query-to-all” strategy, servers do not advertise their information, and clients should query all the nodes to find out the service. This strategy is basically a greedy on-demand algorithm.

In the “Uniform memory less” strategy, each server posts its advertisements to exactly k nodes, and each client queries exactly k’ nodes where k and k’ are fixed. This algorithm does not guarantee that the client will find the server, but it has less network traffic than greedy algorithms.

In the “Incremental post-query” strategy, there is a sequence (Pr, Qr), r= 1, 2, 3,…R of Post-Query protocols. The posting or querying node set starts at a small value, and it will be increased slowly until the client can find the server. The network traffic may be reduced to minimum, but the querying delay is very large since the client has to wait for many steps to get a result. Two variants of this strategy are the “post-incremental” and “query-incremental”.

2.4.3 Comments

The Post-Query protocol almost covers all the cases of infrastructure less MANET service discovery. Different strategies are suitable for different kinds of networks. In general, “Query-to-all” is good for high mobility MANETs; it is expensive but quick. “Incremental post-query” is good for high mobility MANETs as well, and it has the minimum network traffic but the maximum delay. Users can select different strategies according to their requirements. This protocol is very comprehensive and general, but it is not enough for implementing service discovery for real applications. One reason is that it does not take into account the routing cost. Indeed, the route discovery cost can be higher than the service discovery cost and it should not be ignored.

2.5 Konark – A Service Discovery and Delivery Protocol

2.5.1 Architecture

The Konark[35] architecture (Figure 2.3) is operating system and programming language independent because it is built above IP network connectivity and is XML based. Konark allows each device to act as a server and a client simultaneously. There is a "Konark SDP manager" in each node. Konark applications are built above SDP managers, and a Konark application can facilitate human interaction to initiate and control advertising, discovery and service usage. Konark has two major parts: service discovery and service delivery.

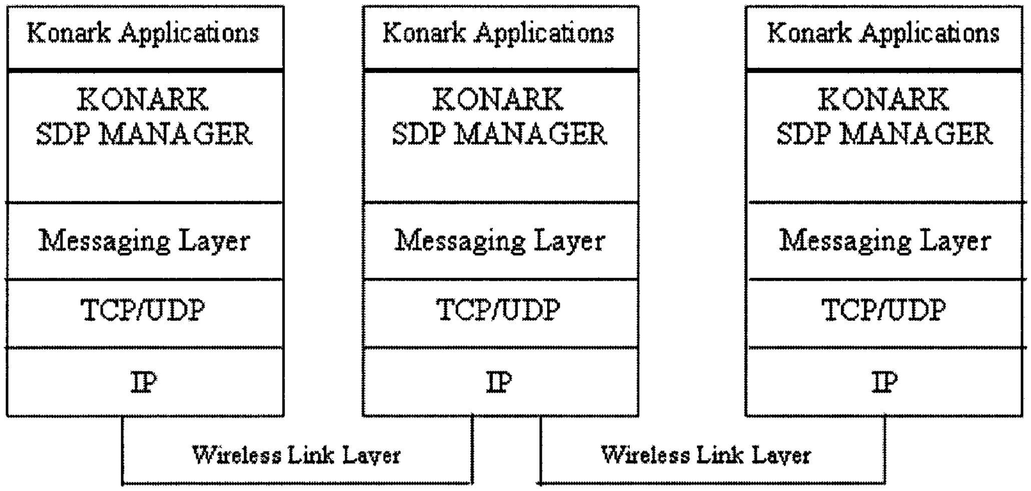


Figure 2.3 Konark Architecture

2.5.2 Service Discovery

A Konark manager maintains a tree-based structure registry, called the "service tree". The root and internal nodes of this service tree are the service types, and the leaf nodes are the service name. From top to bottom, the service may become more specific and the service name will eventually be found at the leaf node. When a node receives a service request, it will search for the requested service in the service tree from root to leaf, when it will know whether the requested service is at current node or not.

Konark supports "active pull" and "passive push" modes for discovering a service in MANETs. In "active pull", a client sends a query to all of the nodes in the MANET to find the service location. In "passive push", a server advertises the service information to the entire network periodically. The discovery process has two steps. In the first step, a client sends a discovery message on a fixed multicast group. In the second step, each receiver will search for the local service tree, and check if it matches the service request. If it matches, the node will send a service response.

2.5.3 Comments

Konark provides a solution for service discovery on MANETs, but it does not consider energy consumption or delay. Without consideration of energy, it cannot be applied to real ad hoc applications and is limited to research issues only.

2.6 Network Layer Support for Service Discovery

2.6.1 Architecture

This is a directory-agent based service discovery [36]. The first step for this algorithm is to set up a backbone structure on the MANET, and the backbone nodes serve as Directory Agents (DA). In the second step, a server registers its service with the DAs and a client queries for a desired service from the DAs.

2.6.2 Backbone Management (BBM)

The goal of BBM is to obtain a small and relatively stable backbone. Basically, it has three steps. The first step is to select the backbone nodes, and then find a route between the backbone nodes. After forming backbones, it should maintain the topology changes in the MANETs.

Initially, each node is assigned a “white” color. All nodes keep sending hello messages to each other, so that they know their neighbouring nodes. After a time interval, each node should have collected enough information about its nearby environment. Only the nodes that have fewer changed neighbours become a backbone node whose color is changed to black. In this stage, some nodes are assigned as green nodes, which are the backup nodes for backbone nodes. After the virtual backbone is set up, all nodes in a MANET must find a backbone node as its VAP (Virtual Access Point).

In the next step, the links between backbone nodes are found and a routing table is prepared at each backbone node. The maintenance of the backbone nodes is also very important for MANETs. If green nodes do not receive their VAP’s hello messages for a time period, they know that their backbone node is down. An election is started to vote for a new backbone node.

2.6.3 Distributed Service Discovery

The server must register itself to its VAP first, and its VAP then multicasts the service information to some other VAP nodes or all the VAP nodes. The TTL (time-to-live) field is used to control the VAP multicast range. Clients can search for service from their VAP directly. The VAP serves as Directory Agent (DA) in this algorithm.

2.6.4 Comments

These algorithms use agent-based MANET service discovery, and the major problem for agent-based algorithms is that the agent node may fail. Furthermore, it is quite energy consuming to rebuild a new agent, which contains all the service information. This is especially true since mobile nodes have limited battery energy in MANETs.

2.7 Other MANET Service Discovery Research

Other service discovery research for MANETs can be separated into three classes: directory agent based, without directory agents and security oriented.

Directory agent based service discovery in MANETs [37-41] generates an infrastructure by the mobile nodes. Some nodes are core nodes which control or help other nodes. The infrastructures can vary for different researches such as Service Ring [37], Content Addressable Networks (CAN) [38], Multi-Layer Clusters [39], hash indexing self-organized directory agent [40], and Bluetooth centralized structure [41]. In general, the directory agent based algorithms create an infrastructure for MANETs first. The nodes in that infrastructure are management nodes, which manage some other nodes. Other nodes query and receive information through the management nodes.

Some service discovery protocols do not have a directory agent [42-45]. In these cases, client nodes have to discover services themselves [42][45], from intermediate nodes [43] or by service groups [44]. In typical proactive service discovery solutions, client nodes send multicast or broadcast queries and server node respond with unicast replies [45].

Other research directions in this field related to security or location-aware design include service representation (coordination-based design) [46], asynchronous home agent based [47], secure location-aware service discovery SPLENDOR [48] and Bluetooth based schemes [49].

Chapter 3 Proposed Service Discovery Scheme (HESED)

In this chapter, we present the features of a good service discovery scheme for MANETs. We then describe our service discovery scheme HESED (High Efficiency SErvice Discovery) and the algorithms for realizing HESED. At the end of this chapter, a theoretical analysis proves two major advantages of HESED, which are energy efficiency and short delay.

3.1 Objective

Before we can design an effective service discovery scheme, let us first look at the features of a good service discovery scheme.

Low energy consumption is a critical, and possibly most the important, feature. Since most of the mobile devices only have limited energy supply, high energy consuming applications cannot run on these devices. Wireless communication is a major source of energy consumption, and depends on the number of packets transmitted. For most wireless adapters, the signal strength is set during installation. Although the user can manually change it later, it must stay at a fixed level during the communication. This means the energy consumption only depends on the packet size and it is not related to the actual distance between two nodes (if they are in range of each other). Therefore, we can roughly control the energy consumption level by controlling the number and size of packets sent. Hence, we make this an objective of our proposed scheme in order to lower the total energy consumption.

The second feature is shortening the delay. Any application that utilizes service discovery cannot afford extended delays. It should be noted that delay and energy consumption may be two contradicting factors. Reducing delay may mean sending more control packets, hence increasing energy consumption. We show later how our proposed scheme balances the two requirements.

We remark that wireless connection and ad hoc nodes are not reliable because of the mobility in MANETs. On the contrary, practical wireless applications need reliable communication between different nodes. Our proposed scheme can partially archive this goal by sensing the nodal neighbourhood information.

Because of the heterogeneity of MANETs, compatibility with various operating systems (OS) is a necessary factor. We, therefore, propose application-level routing, to complement lower level wireless routing support that is compatible to any OS. Hence, and in contrast to the ISO/OSI model (International Standard Organization's Open System Interconnect) and *IP layered* approaches, which may not be best suited for wireless ad hoc communication, we deploy a *multi-layer* approach which combines an application-level support protocol (viz. service discovery) with a networking layer protocol (viz. routing). According to our proposal, there should be four sub-layers within the application/networking layer: application-level routing, application-level multi-hop relaying, application-level RDT (reliable data transportation) and application-level beacon. Naturally, these four sub-layers are not totally separate in our implementation.

3.2 Framework Overview

In HESED, when a client node requires a service, it evaluates the cached services using a *Cache algorithm* first. If it cannot find any valid cached service, the client node sends a query and waits for replies according to a *Multicast Query and Multicast Reply algorithm.* The multicast packet is sent by an *Edge Node Forwarding algorithm* and the unicast packet is routed by a *Backward Learning Routing algorithm. Long Beacon Neighbour Detection* provides the necessary information and eliminates asymmetric links for the Edge Node Forwarding algorithm. Table 3.1 summarizes the features of the main algorithms used in HESED.

Algorithm Name	Brief Description
Cache	Evaluate client node's cached information by using the neighbour node mobility information, which is obtained from the Long Beacon Neighbour Detection algorithm.
Multicast Query and Multicast Reply	Exchange information among client nodes and server nodes with the Edge Node Forwarding Multicast algorithm.
Edge Node Forwarding Multicast Routing	Forward multicast packet by using an auto-generated multicast mesh which is formed by edge nodes. Edge nodes are selected by using the information collected from the Long Beacon Neighbour Detection algorithm.
Backward Learning Routing	Propagate unicast packet along the reversed multicast route. This is a by-product of Multicast Query and Multicast Reply algorithm.
Long Beacon Neighbour Detection	Detect neighbourhood nodes. It is the lowest layer of HESED, and provides the necessary information to other algorithms.

Table 3.1 HESED Framework

3.3 Multicast Query and Multicast Reply Algorithm

This algorithm is the core part of HESED because it defines how to exchange service discovery messages. Indeed, HESED is fundamentally different than any of the post-query strategies in that it uses query-all and reply-all mode. Query-all means clients send multicast queries to all reachable nodes with a limited TTL. Service providers also use multicast mode with limited TTL to send their reply and none of the intermediate nodes send any reply even if they have some knowledge about the query. All the nodes that receive the service reply should cache the service information in their local memory. Since there are many different servers available, a client may receive more than one reply. Clients should send a unicast confirmation to the most desirable server. After a server receives this confirmation, the server sends the client an acknowledgment (ACK) message, which includes detailed information about the service and how to utilize it. All packets use the pre-defined XML [50] template. Figures 3.1 and 3.2 describe the algorithms for a client and server and the interaction between them.

```
Algorithm for Client Node
Main()
Upon receiving a multicast packet p1
    If client did not receive p1 before and it is required to re-send, then
        multicast(p1)
     If p1 is a service discovery reply, then
        updateCache(p1)
        if current node = p1.destination then
            generate unicast packet p2
            unicastForward(p2)
  else if receive a unicast packet p3, then
    if current node is the destination then
        Output final result to user
    else
        unicastForward(p3)
else if current node need a service
    generate a new service discovery query packet p4
    multicast(p4)
```

```
Algorithm for Server Node
Main()
Upon receiving a multicast packet p1
    If server did not receive p1 before and it is required to re-send, then
        multicast(p1)
     If p1 is a service discovery query, then
        Generate a service reply packet p2
        multicast(p2)
  else if receive a unicast packet p3, then
    if current node is the destination then
        generate a service confirmation packet p4
        unicastForward (p4)
    else
        unicastForward (p3)
```

Figure 3.1 Algorithms for Client and Server Nodes

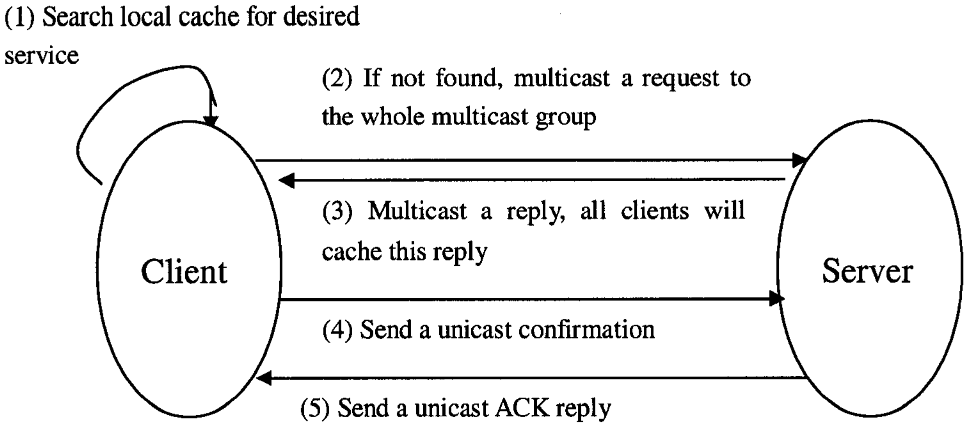

(1) Search local cache for desired service
(2) If not found, multicast a request to the whole multicast group
(3) Multicast a reply, all clients will cache this reply
Client
Server
(4) Send a unicast confirmation
(5) Send a unicast ACK reply

Figure 3.2 Multicast Query and Multicast Reply Algorithm

It should be noted that when an intermediate node receives a client's query, the intermediate node may have some service providers' information in its local cache. In such a case, the intermediate node is able to respond to the query and it need not forward packets to the service providers. While this may reduce some potential network traffic, we do not allow such a scenario because:

- There are potentially a number of similar servers available in large MANETs. It is difficult for an intermediate node to have full knowledge of all of them and hence select the best. If the intermediate node replies to the query and does not forward it, clients may lose access to some servers that exist but which are unknown to the intermediate node.

- Even if some intermediate nodes know all the service information (it may be very expensive to maintain), intermediate nodes still have to send this information to the client, and the size of the response message could be very large. If every intermediate node sends this large response, the client may receive many large duplicate responses.

- Each intermediate node has to check whether the services in its cache can be matched with the client's query. The intermediate node can do packet forwarding only after it cannot find any cached information. This may create a long processing delay at each relay node.

- Some information may be outdated. This not only leads to misinforming clients, but also to having conflicting information about services arrive at clients.

For the reasons listed above, only service providers are allowed to answer client queries and the intermediate nodes just cache the service information for their own use.

Service discovery includes two phases: a searching phase and a utilization phase. After the client receives a server's reply, the searching phase of the service discovery is over and it then goes into the utilization phase. In the utilization phase, the client node sends a unicast confirmation first and the server sends a unicast acknowledge if it can accept this client node. The server's unicast acknowledgement could be converted into a GUI and the users could control the server through this GUI. After the server finishes the assigned task, the server generates a service report to the client and ends the whole service discovery process. All the messages are presented in XML; the detailed format is described in Appendix A.

3.4 Long Beacon Neighbour Detection

This neighbour detection algorithm is a fundamental part of service discovery because it organizes MANETs in an efficient way and supports the multicast algorithm. Each node maintains local topology information so that the node may find a better solution for a given task such as multicasting. This algorithm requires each node to send beacon messages periodically to all its neighbours. The beacon message includes its one-hop neighbours' information so that each node can know all the nearby nodes within two hops.

3.4.1 MANET Organization

There are many ways to organize wireless MANETs and we can separate them into three classes according to each node's neighbour knowledge. Let us assume that each node knows its nearby nodes inside K hops. We can separate the MANETs into three categories according to the different K values in which K equals to 0, D and $1 \leq K \leq D-1$ (where D is the network diameter).

The first category is K=0 where each node knows nothing about its neighbours. For setting up a connection between two nodes, the source node has to query the whole network to find a route to the destination node even if the distance between the source and the destination is only one-hop. Many well-known routing algorithms such as DSR [5] use this method because it is the simplest mode and requires no cost.

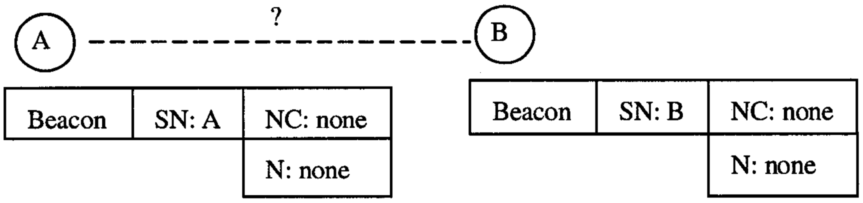
The second category is K=D, which means nodes have complete knowledge about the whole network-topology. This is very expensive to maintain and update. It is very efficient for small networks but it is impractical for large networks.

The third category is when $1 \leq K \leq D-1$, which means each node knows the network topology up to K hops away K = 1, 2 ...D-1. HESED belongs to this class where K=2. Most of the current wireless devices belong to this category where K=1. It is our view though that this is not a good solution for MANETs. For example, consider two nodes in a MANET – A and B. If node A receives a beacon message from node B, then node A cannot determine if node B can also receive node A's beacon message. Node A cannot trust this link for reliable data transport because node A does not know whether this is a bi-directional link or not. To solve this problem, node A must ping all its neighbours to confirm these links before node A can actually use them. This may create a long delay because node A has to wait for these unreachable nodes until a time-out. It may lead to a disaster if every node does this before it can send a packet. This may even be worse than ignoring such information and setting up a new route. This short beacon is very useful to detect a wireless access point in infrastructure wireless LAN, but it cannot be used in MANETs.

The choice of particular importance in MANETs is K=2 because the asymmetric link problem is resolved at the neighbour detection stage and it reduces the difficulty of route discovery or service discovery. Users enjoy much better performance such as shorter delay and lower energy consumption with affordable beacon cost.

3.4.2 Proposed Neighbour Detection Algorithm

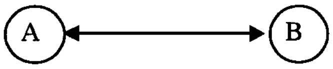
Because communication range varies for different nodes, asymmetric links are inevitable in MANETs and they are also a main source for unreliable wireless connections. We alleviate this problem by "long beacon neighbour detection". In our algorithm, only the nodes that are connected by symmetric links, are considered as neighbours of one another. There is an example in Figure 3.3 of a symmetric link and an asymmetric link. In Figure (2a), (3a), (4a), there is a symmetric link between node A and B, and they can receive each other's beacons. At step 1, when node A receives node B's beacon, node A adds node B as its neighbour candidate and send the updated beacon. Node B also list node A as its neighbour candidate for the same reason. At step 2, node A finds itself appeared in node B's beacon, and node A knows that node B has received node A's beacons. Node A can conclude that the link between node A and B is symmetric link and Node A updates Node B from its neighbour candidate list to its neighbour list. In Figure (2b)(3b)(4b), the link between nodes A and B is an asymmetric link. Since node A does not receive beacons from node B, node A cannot list node B as its neighbour or as a neighbour candidate. Node B cannot find itself in node A's beacon, and node B can conclude that node A cannot receive node B's beacon and that there is an asymmetric link between the two nodes. Because finding an asymmetric link needs feedback from one's neighbour nodes, this task cannot be done by short beacons.



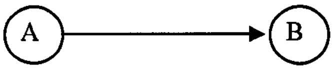
Beacon	SN: A	NC: none
		N: none

Beacon	SN: B	NC: none
		N: none

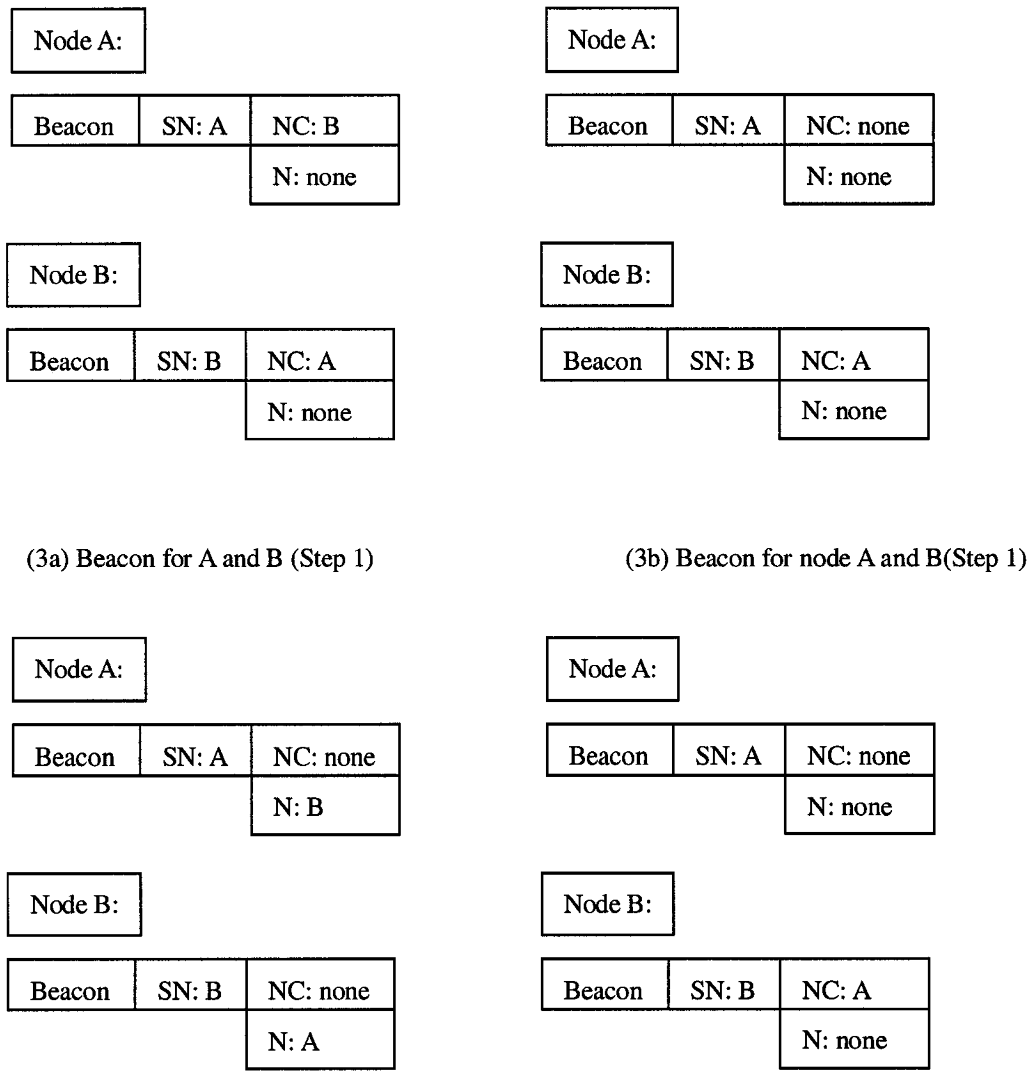
(1) Node A and B do not detect each other at the initial stage



(2a) Link AB is a symmetric link



(2b) Link AB is an asymmetric Link



(3a) Beacon for A and B (Step 1)

(3b) Beacon for node A and B(Step 1)

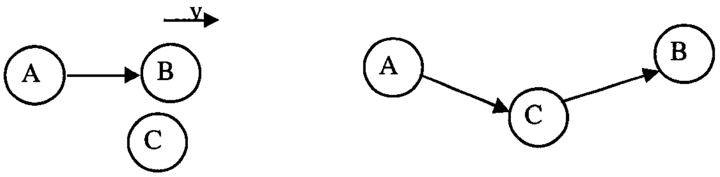
(4a) Beacon from A and B (Step 2)

(4b) Beacon from node A and B (Step 2)

Where:
SN: Source Node
NC: Neighbour Candidate
N: Neighbour

Figure 3.3 Symmetric/Asymmetric Link Detection

This algorithm also has one advantage for route recovery. Because nodal movements are continuous, a 1-hop neighbour node may become a 2-hop neighbour node for a period during its departure. This provides an opportunity for the sender node to rebuild a route without querying others. For example (Figure 3.4), node A and B are the relay nodes on some connections and node B is node A's next relay node at time T_0. But node B is leaving node A's radio range at time T_1, so node A has to find a new route to the destination. Fortunately, node A can find node B's existence from node C's beacon. Node A can use node C as the relay node to forward packets to node B. In this case, it is not necessary to redo the route discovery from the source to the destination. This reduces route re-discovery costs and improves network performance.



(a) Time T_0

(b) Time T_1

Figure 3.4 Route Recovery

Since each node has some knowledge about the mobility of its neighbours, it can calculate the average neighbour residence time T_α, which can be used in the cache algorithm. Details of the cache algorithm are described later.

3.5 Edge Node Forwarding Algorithm

The Edge Node Forwarding algorithm can forward the multicast messages which are created by the Multicast Query and Multicast Reply Algorithm. Many multicast algorithms (such as ODMRP [51]) use flooding to forward the initialization packet. In flooding mode, every node has to forward the packet so that it can be propagated to the whole network. In the Edge Node Forwarding algorithm, only selected nodes forward packets while achieving the same coverage as flooding mode.

After the neighbour detection stage, each node knows all the nearby nodes within a 2-hop distance, and we separate these into two sets: the one-hop set including all nodes that can be reached within a single hop and the two-hop set including all nodes whose distance to the source node is two hops. We separate the one-hop set into two subsets: an *edge-node set,* which includes the nodes that can cover the entire two-hop set nodes in their communication range, and an *internal-node set,* which includes the remaining nodes.

Each node asks all its edge nodes to forward the multicast packets, while internal nodes do not forward any packets. On the other hand, a node will forward a packet only when it first receives the packet and the sender declares it as an edge node.

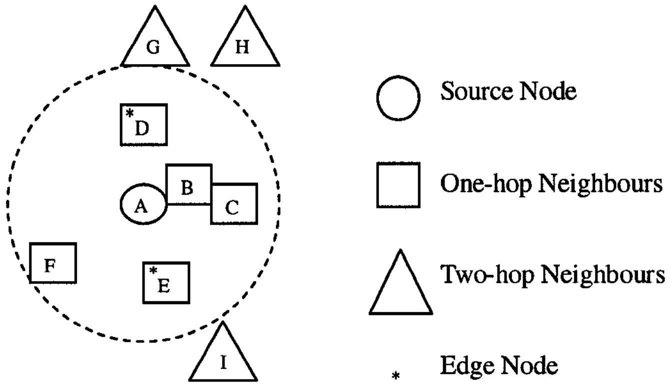


Figure 3.5 Edge Node Set

Figure 3.5 illustrates the operation of the multicast packet forwarding algorithm. In Figure 3.5, the dashed-line circle is node A's radio range. Source node (A) has one-hop neighbours (nodes B-F) and two-hop neighbours (nodes G-I). Node D and E can cover all the second-hop neighbours, so node D and E are edge nodes and all other nodes, namely nodes B, C and F, are the internal nodes. If node A wants to send a multicast packet, it will ask node D and E to relay the packet.

Edge nodes are selected by a greedy algorithm. A source node sorts all its one-hop neighbours by the number of one-hop neighbours that node has, and the source node selects edge nodes from top to bottom on the list until the selected edge node set can cover all the two-hop set nodes. The algorithm always selects the same set of nodes, those which have the most neighbours as their edge nodes, so they always ask the same node to cover the same area and this can prevent duplicate resending to the same area by different nodes. All of above analysis is based on the assumption that local topology does not change during one multicast communication. The duration for one multicast communication is approx. 10-100ms, and the nodal movement is less than 1 meter during this time which is not comparable with the average communication radius (200m). Thus, ignoring nodal movement during a multicast communication should not affect the final result.

Edge node selection can vary based on different criteria. The above selection could form a multicast mesh where the number of mesh nodes is minimized because edge nodes are selected by the maximum number of neighbour nodes. If we select the edge nodes based on minimum response time, then we can create a fast multicast mesh. If the edge nodes are selected based on minimum energy consumption or maximum remaining energy, then an energy-aware multicast mesh will result.

3.6 Cache Algorithm

This algorithm is mostly independent from the other algorithms, except for the Neighbour Detection algorithm. The algorithm uses a time period known as the "half neighbour changing" time, T_α, which is measured by the neighbour detection algorithm, and can be used for calculating the connection validation probability (CVP). Since one route consists of many connections, the route validation probability (RVP) can be calculated by accumulating CVPs. The RVP can then be used to calculate the cost function to check if we can take advantage of using cached service information.

3.6.1 Half Neighbour Changing Time T_α

Let r_m be the ratio between the number of one node's remaining neighbours and the number of all its neighbours after time T_0, it is given by $r_m = |S_1 \cap S_2| / |S_1|$ where S_1 is the neighbour node set at time T_0, and S_2 is the neighbour node set at time T_m. T_α is the time for half of the neighbouring nodes to change, which means r_m=0.5. Assuming neighbouring nodes leave at a fixed rate, the average leaving rate r_l per time unit is $r_l = (1/2)/\ T_\alpha = 1/(2*T_\alpha)$. Let the proportion of remaining nodes be r_n, hence $1-r_n$ nodes have left. Based on the properties of uniform distribution, the time is $t = (1- r_n)/\ r_l = 2 * T_\alpha * (1- r_n)$. Thus, the proportion of remaining neighbours, which is our defined as CVP, after time t is given by:

$$\text{CVP} = r_n = (1 - \frac{t}{2T_\alpha}).$$

3.6.2 Connection Validation Probability (CVP)

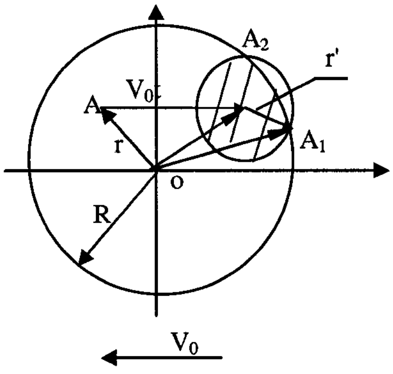


Figure 3.6 CVP Calculation Diagram

The CVP can be calculated in the following way. In Figure 3.6, the large circle is the communication range of node O, which is at the origin of the coordinate system. Assume the size of the MANET is relatively large and all nodes are evenly distributed in this network with density ρ nodes per communication range. The communication radius R for each node is the same. All nodes are moving with a velocity v' and we assume that v' is normally distributed with probability density function $G(v')=\frac{1}{v\sqrt{2\pi}}e^{-(v'-u)^2/2v^2}$, where u is the mean value and v^2 is the variance.

Assume node O is moving with velocity v_0 towards direction –x. If the reference frame moves with node O, a relative velocity $-v_0$ should be applied to all the other nodes. There is a node A at time t_0 with position (r,θ) whose absolute velocity is v' and it can move along any direction. So the position of node A at time t_1 can be any point on the circumference of the small circle. Only those points on the circumference of the small circle's shadow area can be treated as node O's neighbour at time t_1. Thus the probability that node A, whose velocity is v', is in node O's communication range at time t_1 is as follows:

$P_1(v')$=(length of arc A_1A_2)/(perimeter of the small circle)

$$=(\pi-\arccos\frac{d^2+(v't)^2-R^2}{2dv't})/\pi \quad (1)$$

Where $d=\sqrt{(v_0t)^2+r^2+2rv_0t\cos\theta}$ and $t=t_1-t_0$

If the speed of the node is less than (R-d)/t, this node is moving too slowly to move out of the communication range. If the speed of the node is larger than (R+d)/t, this node is moving too fast to stay in the communication range. Thus, the probability of a node A with variable velocity being in node O's communication range at time t_1 is:

$$P_2(r,\theta)=\int_0^{\frac{R-d}{t}} G(v)dv+\int_{\frac{R-d}{t}}^{\frac{R+d}{t}} G(v)P_1(v)dv \quad (2)$$

The probability that all the nodes which were in the communication range at t_0, will remain in communication range at t_1 is as follows:

$$P_3(V_0)=\frac{\int_0^R\int_0^{2\pi} p_2(r,\theta)\cdot\rho\cdot r\cdot d\theta\cdot dr}{\pi R^2\rho}$$

$$=\frac{\int_0^R\int_0^{2\pi} p_2(r,\theta)\cdot r\cdot d\theta\cdot dr}{\pi R^2} \quad (3)$$

After integrating (3), the only variable in P_3 is nodal velocity and all other parameters are the network attributes, which are constants. If one node's velocity is fixed, for any given time period, its neighbour departure rate is the same.

3.6.3 Route Validation Probability (RVP)

Let RVP be the probability of a route remaining valid after time t. The RVP depends on the CVPs of all its intermediate nodes and the source node. Thus RVP = $P_1 * P_2 * \ldots$ *Pn (P1, P2,.. Pn are the CVPs for all the intermediate nodes and the source node)

$$\mathrm{RVP} = (1 - \frac{t}{2T^{(1)}\alpha}) * (1 - \frac{t}{2T^{(2)}\alpha}) * \ldots\ldots (1 - \frac{t}{2T^{(n)}{}_{\alpha}})$$

For two hops, we have

$$\mathrm{RVP} = (1 - \frac{t}{2T^{(1)}\alpha}) * (1 - \frac{t}{2T^{(2)}\alpha}) \approx 1 - \frac{t}{2T_{\alpha}^{'}} \qquad \text{where } T'_{\alpha} = \frac{T^{(1)}{}_{\alpha}T^{(2)}{}_{\alpha}}{T^{(1)}{}_{\alpha} + T^{(2)}{}_{\alpha}}$$

Recall T'_{α} is the accumulated half neighbor changing time. We can accumulate T'_{α} at each intermediate node so that we can get the T_{α} for the entire route.

3.6.4 Cost Function for Service Discovery

Let M_{normal}, T_{normal} be the number of packets and the delay, respectively, for a service discovery process without using caching. If cached information is used and the service provider is successfully found, only M_{cache} packets are sent and the reply is obtained in time T_{cache}. If it fails, we have to start a new service discovery process, thus the total number of packets sent is M_{cache}+ M_{normal}, and the total time is T_{cache}+ T_{normal}.

So, the expected cost of using the cache is

T_{exp} = RVP * T_{cache} + (1-RVP)*(T_{cache}+ T_{normal}) = T_{cache} + (1-RVP)*T_{normal}

M_{exp} = RVP * M_{cache} + (1-RVP)*(M_{cache}+ M_{normal}) = M_{cache} + (1-RVP)*M_{normal}

where M_{exp}, T_{exp} is the expected number of packets and delay, respectively, for a service discovery process.

Let D be the network diameter, which is the maximum number of hops from one end to another end, and let d_h be one-hop delay. If TTL is used for multicast packet forwarding, the network diameter D should be considered as 2*TTL because this is the actual network size from a node's point of view.

Since edge nodes are selected by maximizing the communication coverage, the average distance for one-hop is close to the communication radius. In our analysis, there is one server and many clients evenly distributed in the network whose shape is a circle with diameter D and the clients and the server are evenly distributed in this circle.

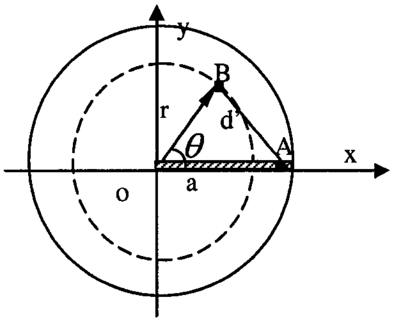


Figure 3.7 Average Distance Calculation Diagram

In Figure 3.7, let us randomly select two points A and B whose polar coordinates are (a,0) and (r, θ), respectively; d' is the distance from A to B which is:

$$d' = \sqrt{a^2 + r^2 - 2ar\cos\theta}$$

We can get the average distance d", which is from A to all the points inside this circle by integrating r from 0 to D and θ from 0 to 2π.

$$d'' = \int_0^{\frac{D}{2}} \frac{1}{2\pi} \int_{-\pi}^{\pi} d' \cdot d\theta \cdot dr$$

Thus, the average distance $\overline{d}$, which is between two randomly selected points, is the integral of d",

$$\bar{d} = \int_0^{\frac{D}{2}} \frac{4}{\pi D^2} d''\cdot 2\pi a \cdot da \approx 0.62\text{D}$$

The approximate value of $\bar{d}$ was calculated by a small Java program and it can be approximated as 0.62D for ease of computation. Since D is the number of hops from one end to another, $\bar{d}$ is also counted as number of hops which is used as the average number of hops between the server node and the client node.

Let T_{cache} and M_{cache} be the delay and number of packets for using caching which are the cost of two unicast (confirmation and ACK) communication since we do not need to send and receive multicast queries or replies. Here are the equations for T_{cache} and M_{cache}:

T_{cache} =(number of hops * delay per hop) *2

= (0.62D* d_h)*2

= 1.24D* d_h

M_{cache} = (number of hops * 2)

=(0.62D)*2

= 1.24D.

Where d_h is the average delay per hop.

Let T_{normal} and M_{normal} be the delay and number of packets for a normal service discovery process. T_{normal} and M_{normal} are the cost of two multicasts and two unicasts (service request and service reply). Similarly, the equations for T_{normal} and M_{normal} are as follows:

T_{normal} = (multicast cost)+(unicast cost)

=(number of hops * delay per hop) *2+(number of hops * delay per hop) *2

= (0.62D* dh)*2+(0.62D* dh)*2 = 2.48*D* dh

M_{normal} = (number of multicast packets)+(number of unicast packets)

= (Area of the whole network / average coverage per packet) *2
+ (average number of hops from a server to a client) *2

$=[\pi D^2/(\pi r^2/2)]*2 + (0.62D)*2$

$=(2D^2) *2 + (0.62D)*2$

$= 4*D^2+1.24D$

Note that the coverage area of each multicast packet is $\pi r^2/2$ on average.

Two strategies can be used for evaluating RVP. The first is delay-sensitive, which selects P by forcing $T_{exp} < T_{normal}$. Another is energy-aware, which selects P by using $M_{exp} < M_{normal}$. For the delay-sensitive strategy, we can get the cost function as RVP>1/2. For the energy-aware strategy, the cost function is RVP>1.24/(4D+1.24). In the simulation and implementation, we use the delay-sensitive strategy.

3.6.5 Cache Selection Algorithm

Each node joins one or more multicast groups to receive all the service information in these groups. Each newly received service reply is attached with a timestamp and saved into local memory.

When a new service reply comes and the cache is full, the RVP for all the services in the cache will be calculated. The cached service that has the minimum RVP will be replaced with the new information.

When a client needs a service, it will search the local cache first. If the service is in the cache, the RVP will be calculated, and a cost function is used to determine if it can be used. Sometimes information for more than one service can pass the evaluation, and all of the passed information should be reported to the user. The user can decide which one should be used.

3.7 Backward Learning Routing Algorithm

Routing in HESED is integrated within the service discovery process. Each multicast packet contains the source node's IP address and the previous node's IP address. Each node keeps these two IP addresses to form a routing table and updates this routing table whenever it forwards a multicast packet. For example, if node A receives a multicast packet whose source is node B and the previous node is Node C. Node A should know that all packets whose destination is node B should be sent to node C first.

Our algorithm is similar to backward learning [51], with the following additional features:

- Routing tables can be updated whenever the current node receives a multicast packet, thus the tables are always up-to-date.
- Since each node only sends a multicast packet once, there is no loop in the multicast mesh. For this reason unicast routing is able to avoid the looping problem.
- The asymmetric link problem is resolved because we detect and remove asymmetric links during the neighbour detection phase and we can guarantee the backward unicast route to the source to be bi-directional and reliable. Using asymmetric links would increase the connectivity of MANETs, but some applications may need an ACK at each hop which cannot use asymmetric links. Since HESED is designed to provide a general solution for any ad hoc application, we must sacrifice the asymmetric links for reliable communication.
- Route recovery is aided by the neighbour detection phase. When a node is departing from a certain communication range, it may switch relaying from a

1-hop neighbour to a 2-hop neighbour. Neighbour detection can check whether the departing node belongs to its 2-hop neighbours. If it does, the current node can still communicate with this departing node.

3.8 XML Packet Format

As wireless technology develops, more and more services will become available in MANETs. Our ultimate goal is to make services on the MANETs Internet-compatible, hence the service description language should be able to describe any service. For this reason, we select XML [50] as the description language because of its extensibility and flexibility. We define six packet types, namely the client's multicast query, the server's multicast reply, the client's unicast confirmation, the server's unicast acknowledge, the client's service request and the server's result report. The detailed XML packet format can be found in Appendix A.

3.9 HESED Analysis

3.9.1 Performance of Multicast Mesh

A. Up-to-date multicast mesh

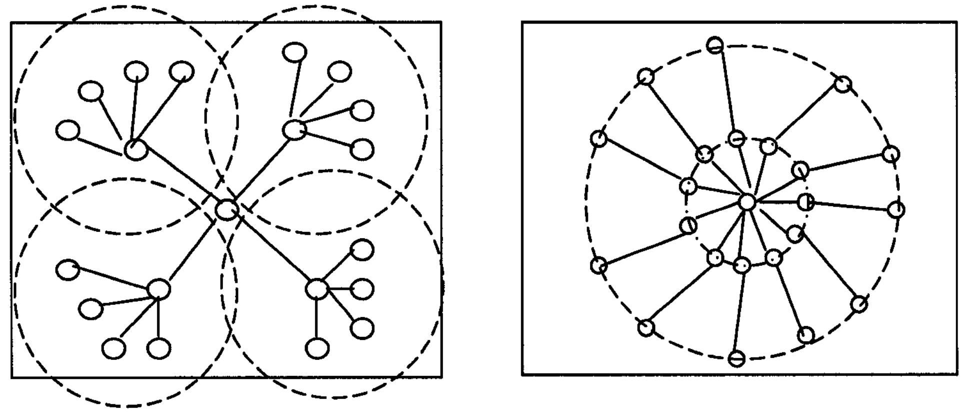
Edge node forwarding should exhibit good performance in highly dynamic environments such as MANETs because the multicast mesh can be created automatically and it is always up-to-date, which is not always the case in other multicast algorithms. In highly dynamic large MANETs, the network-topology is always changing. Each node can adjust its edge node list to adapt to changes and there is no need to rebuild the whole multicast mesh. This self adapting process has limited cost, since it is a by-product of the neighbour detection. Many other multicast algorithms (viz. ODMRP [51]) create the multicast mesh at the initial stage and they assume the multicast backbone does not change during the communication. The adaptation property of edge node forwarding is much better than ODMRP because ODMRP's multicast mesh is hard to change once it is created.

B. Comparison with flooding mode

Compared to the flooding mode, edge node forwarding has three major advantages: low network traffic, short medium access delay and high reliability. Since only a small portion of nodes forward packets, the network traffic can be reduced. Lower network traffic can eliminate many potential medium access collisions, thus it also can reduce the medium access delay. Edge node forwarding has a good potential to be extended to a reliable multicast algorithm if we apply ACKs from the edge nodes to the source node. In flooding mode, using ACKs is not possible because sender nodes do not have the receiver nodes' list and they cannot decide whether the packet should be resent.

C. Number of edge nodes (N_e)

The number of edge nodes naturally depends on the geometry and size of the MANET. Figure 3.8 presents the best case and the worst case of the edge node selection. In the best case, N_e (the number of edge nodes) is approximately $S/\pi R^2$, where S is the area of the MANET and R is the radio radius. In the worst case, N_e is N/2, where N is the total number of nodes. The actual value for N_e should be between these two extremes.



A. Best Case

B. Worst Case

Figure 3.8 Number of Edge Node Analysis

In case A, the duplicated radio coverage is reduced to a minimum and the edge nodes' radio range can cover the most of the MANET. N_e depends on the MANET field size S and edge node's radio radius R and is close to $S/\pi R^2$. In case B, all the neighbour nodes are distributed at the boundary of the two circles whose radius is R and 2R, respectively. Each two hop neighbour can only be reached by a unique 1-hop node, thus N_e is N/2.

Let us create a general equation for N_e.

For case A,

$N_e = S/\pi R^2 = (N/D)/\pi R^2 = (1/D\pi R^2)*N = k_1*N$

where: S is the simulation field area.

R is the radio radius

D is the node density per area unit

k1 is a constant which is equal to $(1/D\pi R^2)$

N is the total number of nodes

For case B:

$N_e = N/2 = k_2*N$

Where k_2=1/2

For any kind of node distribution, the number of edge node should be between the maximum and the minimum value:

$k_1*N < N_e < k_2*N$

Thus we have:

$N_e = k*N$, where $k_1 < k < k_2$

So the number of edge nodes is directly related to the total number of nodes. Simulation results show the value of N_e in a random MANET is about N/3, which is closer to the worst case. Nevertheless the performance of HESED is still superior to that of the traditional scheme.

3.9.2 Message Complexity of HESED

We have so far claimed that the advantages of HESED are low energy consumption and low delay in large-scale MANETs. Although a multicast reply is more expensive than a unicast reply, the multicast reply can deliver the service information to all the reachable nodes, which can reduce many potential multicast queries.

If the query results in a cache hit, this will save time and energy for searching and replying. The maximum number of queries depends only on the average cache expiry time, which is independent of the size of the MANET. For one query, the number of packets is O(N) because each node must receive a multicast packet. Thus the message complexity in HESED is O(N). In the following we compare the message complexity of HESED to that of traditional service discovery protocols.

A. Basic parameters and assumptions

The following are the parameters used in the comparison. These represent typical MANETs with randomly distributed nodal locations and mobility.

Size:	N nodes
Density:	n nodes/per communication range
Communication radius:	R
Average cache expiration time:	T_{exp}
Average network diameter (approx.):	$D=\sqrt{N/n}$
Number of regions:	N/n
Time period:	1 unit of time
Service request rate:	k queries/ (per node, per unit of time)
Total number of requests:	N*k (per unit of time)

We assume query and response packets are the same size. We also assume that all nodes have the same transmission range (or equivalently, use the same signal strength). This is typical in MANETS using existing air interfaces. Hence the energy consumption for each packet is the same.

B. Cost of traditional algorithms

The typical behaviour of on-demand service discovery is that a client sends a multicast query and a server sends a unicast reply to the client. The client may receive more than one reply and the client selects one and sends a confirmation to that server. The server sends an ACK to the client to end this service discovery. Let us assume that ODMRP [51] is used as the multicast algorithm. The multicast source node sends a join query and every node re-sends this join query so that every node in the network may receive this join query. All the multicast group members send an ACK by unicast through the same route back to the source to form the multicast mesh. The multicast packets are sent through this multicast mesh. We calculate the number of packets for each step as follows:

Step1: Multicast join query. The number of packets transmitted equals to the number of the nodes in the network because each node has to resend this join query, $M_1 = N$

Step 2: multicast ACK. Because flooding forward is used in ODMRP, all the nodes whose distance from the source ranges from 0 to R have the same probability of becoming a relay node. Thus the average distance between two relays is $l = \frac{\int_0^R 2\pi r \cdot r dr}{\pi R^2} = \frac{2}{3} R$. The number of relays in the whole network is $(N/n)/(2/3)^2$= (9N)/4n. There is at least one ACK for each relay node, so the minimum number of ACKs is M_2= (9N)/4n.

Step 3: Multicast client's query. Each relay node has to re-send this query, so the number of packets in this step is the same as the number of relays, which is M_3 = (9N)/4n.

Step 4: Server sends a unicast reply. Assume there is only one server in this network. The average distance from client to server is approximately 0.62D where D is the network diameter. The total data delivered is computed by aggregating all the point-to-point distances inside a circle whose diameter is D. The number of hops is M_4=(0.62D)/(2/3) = 0.93D

The total number of packet (M_{trad}) in MANETs during time t is

M_{trad} =(N*k)*($M_1+M_2+M_3+M_4$) = $N^2*C_1+N*\sqrt{N}*C_2$, which is $O(N^2)$

$$where: C_1 = k*(1+\frac{9}{2n}), C_2 = k*0.93*\sqrt{\frac{1}{n}}$$

C. Cost for HESED

(1) Cost for service discovery queries

Because there are service caches in our algorithm, the worst case is that the clients have queries when the cached item expires. The interval of multicasting a service query is the average cache expiration time, and the maximum number of queries in one unit of time is $1/T_{exp}$, where T_{exp} is the average cache expiration time.

The cost per query (M_{per_query}) is the cost for one multicast query and one multicast reply. One packet can cover one region in the HESED algorithm and if 1/2 of this region is new, it would require (N/n)*2 packets in total.

$$M_{per\_query} = \frac{N}{n}*2+\frac{N}{n}*2=\frac{N}{n}*4$$

The total queries' cost for HESED ($M_{queries}$) is:

$$M_{queries} = \frac{1}{T_{exp}}* \frac{N}{n}*4$$

(2) Cost for beacon messages

Assume T_{int} is the interval for sending hello message periodically. Since each node must send the beacon message periodically, the number of beacon packets (M_{beacon}) is

$$M_{beacon} = \frac{1}{T_{int}}*N$$

(3) Total Cost

M = $M_{queries}$ + M_{beacon} = C_1*N, which is O(N) where: $C_1 = \frac{1}{T_{exp}}*\frac{4}{n}+\frac{1}{T_{int}}$

In general, HESED has an extra overhead cost which is the beacon message cost. But this cost is O(N) and is considerably lower than the cost of the traditional mode, which is $O(N^2)$.

3.9.3 Other Performance Issues

A. Memory Usage

Comparing with the traditional mode, HESED uses more memory in order to improve the performance. The memory used in HESED can be separated into three parts: routing table storage, neighbour information storage and service information storage.

- The memory size of each entry in the routing table is only 8 bytes which includes a source node IP address (4 bytes) and next hop node IP address (4 bytes). Even if a node serves for 100 different routes, the size of route table is only 800 bytes.

- Neighbour information storage is also very small. Suppose we have a crowded MANET, one node has 100 one-hop neighbours and each of its one-hop neighbours also has another 100 one-hop neighbours, so this node has 100 one-hop neighbours plus 100*100=10000 two-hop neighbours. Because each node uses 4 bytes for its own IP address, the total memory usage for neighbour information storage is only (10000+100)*4≈40k.

- The size of service information storage depends on the number of available servers. If each item of service information is 2000 bytes and there are 1000 servers available, then the service information storage needs 2000*1000=

2M bytes.

Based on the above estimations, the memory usage in HESED is affordable to all users.

B. CPU Resource and Communication Cost

HESED uses more CPU resources than the traditional mode because HESED has to evaluate the cached information, maintain local topology information and interpret XML packets. Without doing these computations, a client has to send out a service request throughout the whole MANET in the traditional mode. For a large MANET, the energy consumption for doing computation at the local node is much less than the communication cost of propagating a multicast request to all the nodes.

Chapter 4 HESED Design and Implementation

In this chapter, we review the overall architecture of HESED and discuss its implementation. We design several test scenarios for service discovery in MANETs, test the viability of these scenarios and demonstrate the usability of the system.

4.1 Overview

The implementation consists of six Java Packages: a client package, a server package, a multicast package, a unicast package, a hello package, a XML package and a global package (see Figure 4.1). The client package includes several GUI classes and their event handler classes, and the event handler classes are in charge of sending and receiving multicast, unicast packets for service queries and service confirmation and managing the service information cache. The server package includes the service response classes, which can respond to clients' service queries and confirmations. The client and server packages are supported by three other packages, which are the multicast, unicast and hello packages. The multicast package can create, open and modify a multicast packet header, and the unicast package can create, open and modify a unicast packet header. The hello package can create, send, analyze and receive the hello messages, update the current node's local topology information and keep track of the half neighbour changing time T_α (see section 3.6.1). All packets are built according to an XML grammar, which can be created and opened by the XML package. The global package includes all the global variables, which are used by all other packages. Details of the HESED implementation architecture are described in Appendix C.

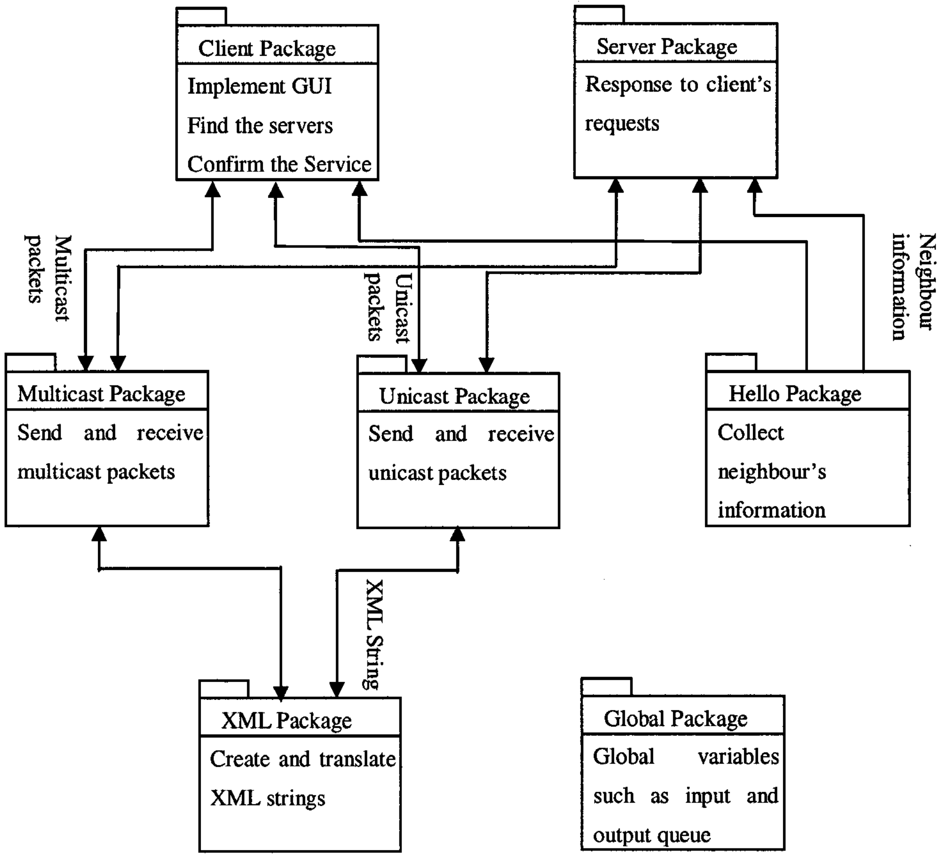


Figure 4.1 HESED Implementation Architecture

4.2 Graphical User Interface

The client program provides a simple GUI as a convenience to users who may not have knowledge about service discovery and/or MANETs. The GUI has three panels: a join panel (Figure 4.3), a search panel (Figure 4.4) and a result panel (Figure 4.5).

4.2.1 Join Panel

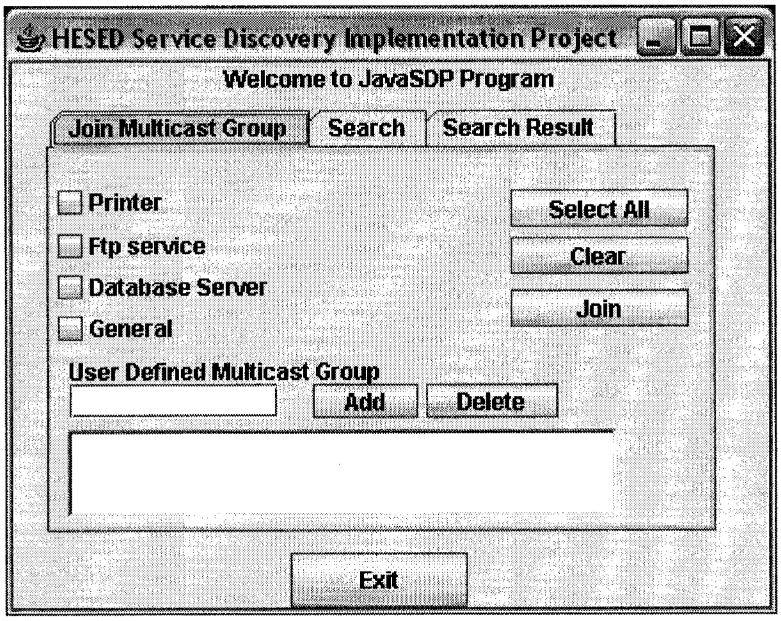


Figure 4.2 Join Panel

The join panel (Figure 4.2) allows a user to join different multicast groups such as printer groups, ftp service groups, database server groups or user defined groups. If a user selects the "General" group then the user will accept all kinds of service information. At the network layer, all multicast groups use the same multicast socket and the user drops some multicast packets at the application layer if he did not join that group.

Each node is responsible for forwarding multicast or unicast packets for others even if that node does not belong to that packet's multicast group. Small multicast groups may not be able to form a multicast mesh to deliver their information if packets are forwarded only by their group members. For this reason, each node is required to help other multicast groups if it is an edge node. This way, each node can attain reliable communication.

After a user selects a desired multicast group, the user should click the "join" button and a packet filter will be created according to the user's selection. All the packets which belong to these desired multicast groups will be delivered to the client's program and these packets are saved into the local service information cache if necessary.

4.2.2 Search Panel

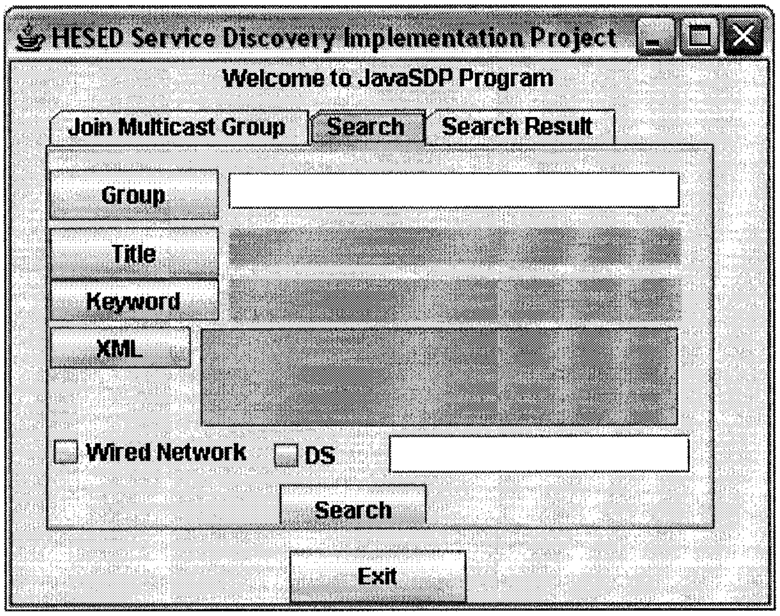


Figure 4.3 Search Panel

The search panel (Figure 4.3) allows a user to specify their search criteria and convert these criteria to XML text whose format is presented in Appendix A.

Searching may be based on the service title or service keyword, which can be found in the service description. We only provide these two criteria but advanced users can also edit their own XML search query in the XML code text field. When the user clicks on the "search" button, the client program starts searching and checks the local cache first. If the desired service information is found in the local cache, it shows the service information at the result panel. If none of the cached information matches the user's criteria, the client has to send a multicast service request and waits for the service reply.

4.2.3 Result Panel

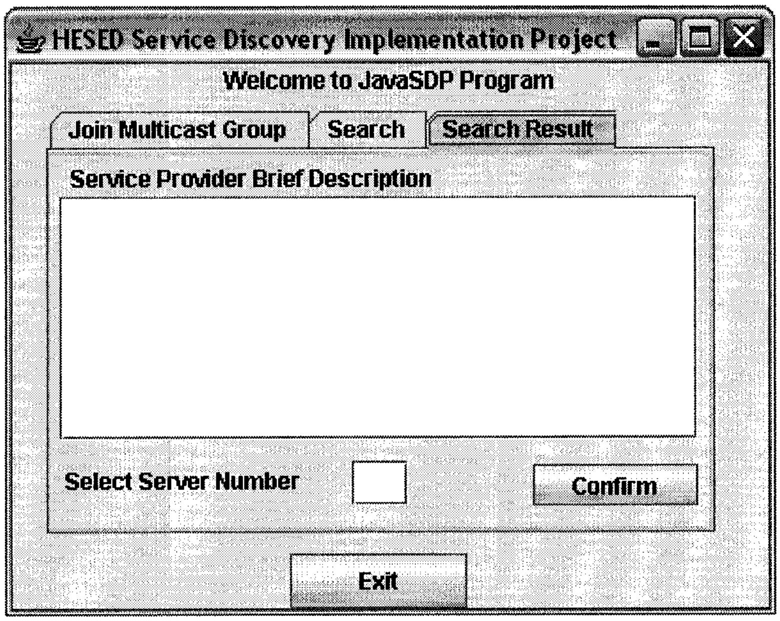

HESED Service Discovery Implementation Project
Welcome to JavaSDP Program
Join Multicast Group
Search
Search Result
Service Provider Brief Description
Select Server Number
Confirm
Exit

Figure 4.4 Result Panel

The result panel (Figure 4.4) shows all received service replies. The client may receive more than one service multicast reply and the user selects the favorite server and sends a confirmation to it. The client should input the selected server number in the textfield and press the "confirm" button. The client program will send a unicast confirmation to the server and the server will respond to it with a detailed service information packet. When the client program receives this unicast response, the client will display the detailed information in the "service provider brief description" text area. Then the user can invoke the program to use the service.

This user interface is designed for novice users who may not have any knowledge about service discovery or MANETs. Our ultimate goal is that the service discovery delay is short enough that users do not realize there is a MANET service discovery phase at the beginning of applications running on the MANET. Since the delay in our scheme is relatively low, we expect that we can achieve this goal in our implementation.

4.3 Experimental Tests

We have designed several scenarios for this service discovery algorithm which include an asymmetric link scenario, a short linear chain scenario, a long linear chain scenario, a multiple route scenario and a bottleneck scenario. Since this is application layer software, all the packets' sending and receiving should go through the regular operating system's socket. Its performance depends very much on the operating system. Hence, we do not measure the specific performance data and we will present our simulation for the performance evaluation in the following chapter. The goal for this implementation is to prove the feasibility and usability of our algorithm. HESED shows very good performance for the following scenarios.

4.3.1 Asymmetric Link Scenario

As we know the asymmetric link problem must be solved in wireless ad hoc applications. In our scheme, we have the neighbour detection algorithm, which can detect asymmetric links and mark these links as invalid. This scenario is designed for demonstrating the asymmetric link detection.

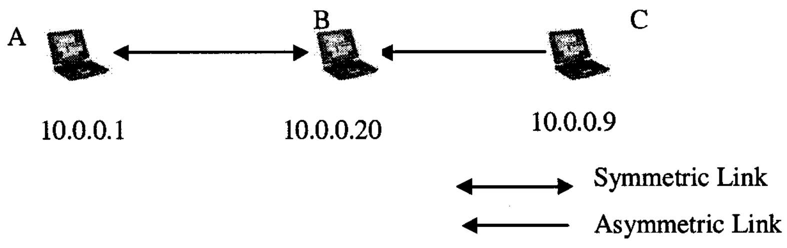


Figure 4.5 Asymmetric Link Scenario

Node A is the source node which has a symmetric link with node B, and node B has an asymmetric link with node C. In this scenario, nodes A and B can detect each other and node B knows of node C's existence but node C cannot receive node B's hello messages. When node A sends a multicast query, node A will not list B as one of its edge nodes, since there is no second hop node connected to B. When B receives a multicast packet, it will not forward it to C.

4.3.2 Short Linear Chain Scenario

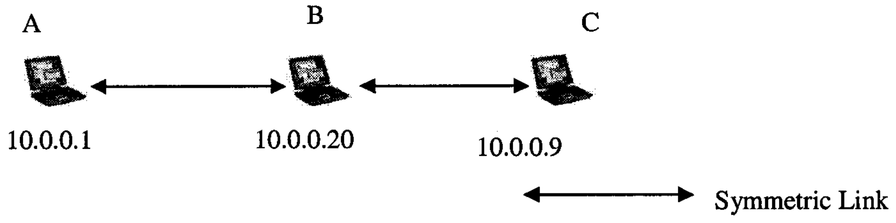


Figure 4.6 Short Linear Chain Scenario

Node A is the client node and node C is the service provider node. Node A sends a multicast service query and node C may respond with multicast service information. After this phase, node A and node C exchange a unicast confirmation and ACK.

4.3.3 Long Linear Chain Scenario

In this test case, we have 5 nodes which are arranged as a line. Node A is the client node, and Node E is the server node. All the packets have to go through four hops to reach the destination. The scenario can test the data transport reliability on longer distances.

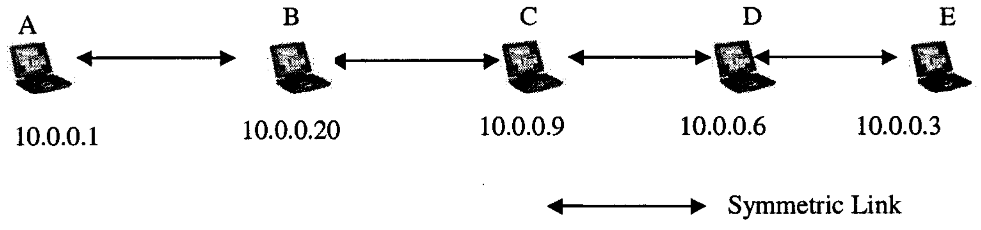


Figure 4.7 Long Linear Chain Scenario

4.3.4 Multiple Routes Scenario

This scenario[52] (Figure 4.8) is used to test if the routing implementation can choose a proper route from multiple alternative routes. Node A is the client node and node E is the server node. The route can be any of A-B-E, A-C-E or A-D-E. According to our edge node selection criterion, node C has the most neighbours and should be selected as the edge node. Thus, the route A-C-E is selected in this scenario.

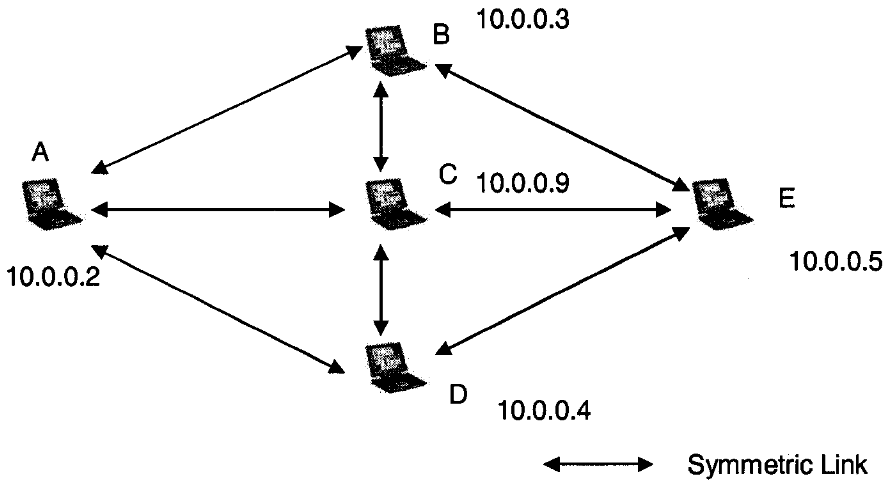


Figure 4.8 Multiple Route Scenario

4.3.5 Bottleneck Scenario

This scenario[52] (Figure 4.9) is used to test several concurrent excessive FTP server accesses through an intermediate node, which will be the bottleneck of communication. In this scenario, Node A, B and C are the client nodes, Node E, F, G are the FTP server nodes. Node D serves as a bridge between the clients and servers. Node D should maintain six routing entries in its unicast routing table, and HESED is very stable in this test.

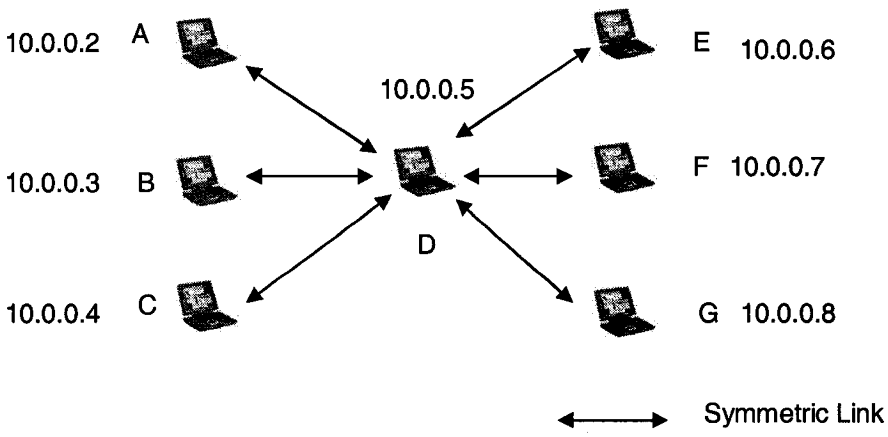


Figure 4.9 Bottleneck Scenario

4.3.6 Printer Server Test

As an instance of HESED, we have designed a MANET printer service (Figure 4.10), which has been implemented by an undergraduate student [53]. This Visual Basic based project will be integrated with HESED. HESED discovers the printer server's information such as name, IP address, port number, and this information is manually input into the client program of this project. This project has two parts: one is the printer server and another is the client program. The client program can send a print task to the printer server through a multi-hop wireless ad hoc connection by using our MANET testbed, and the server can receive and perform the printing task.

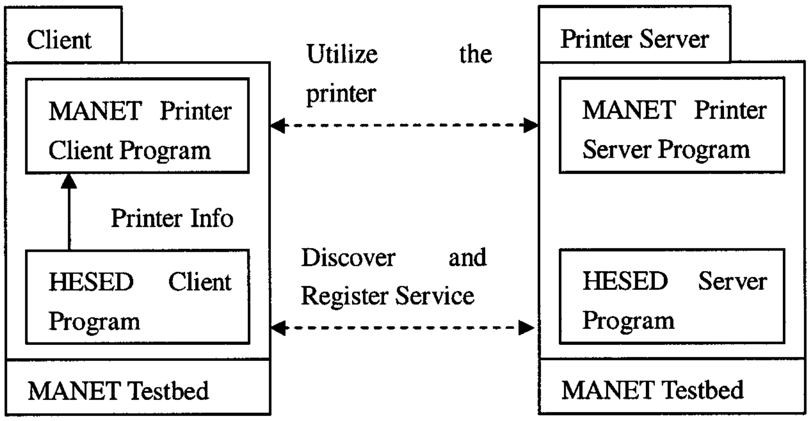


Figure 4.10 MANET Printer Project

The printer server project has three steps:

- At first, the HESED client program sends its multicast query and HESED server responds to it with a multicast reply. This reply includes the server's IP address, port number and a brief description for the printer.
- Then, the HESED client program sends a unicast confirmation to the server. This confirmation includes client's printer account information such as account number and password. The HESED server can create a new access code and send the code to the client if the account information is correct.

- Finally, the MANET client can send print jobs to the MANET printer server by using the access code and the printer can print the task later.

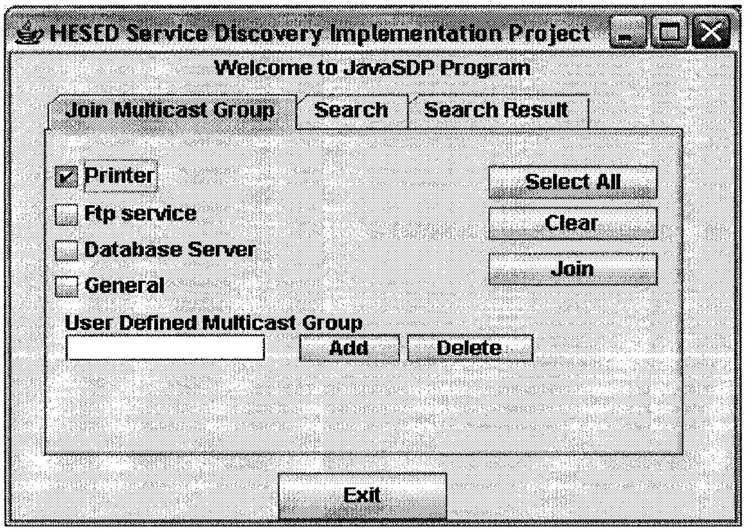
HESED implements the first two steps and the MANET printer project is in charge of the last step.

4.3.7 Test Results

All of the above tests were very successful and this section presents a typical result for the long linear chain scenario (see section 4.3.3) in which the IP addresses for the server and the client are 10.0.0.3 and 10.0.0.1, respectively.

First, the client joins the "printer" multicast group (see Figure 4.11). Then, the client inputs the search criteria and starts the search process (see Figure 4.12). After the server responds to the client's query, the client displays the server's response on the result panel (see Figure 4.13). In this test, the client picks the server whose number is "1" and sends a unicast confirmation to that server. Figure 4.14 presents the acknowledgement received from the server. Figure 4.15 and Figure 4.16 are the command prompt output of the server and the client, respectively.

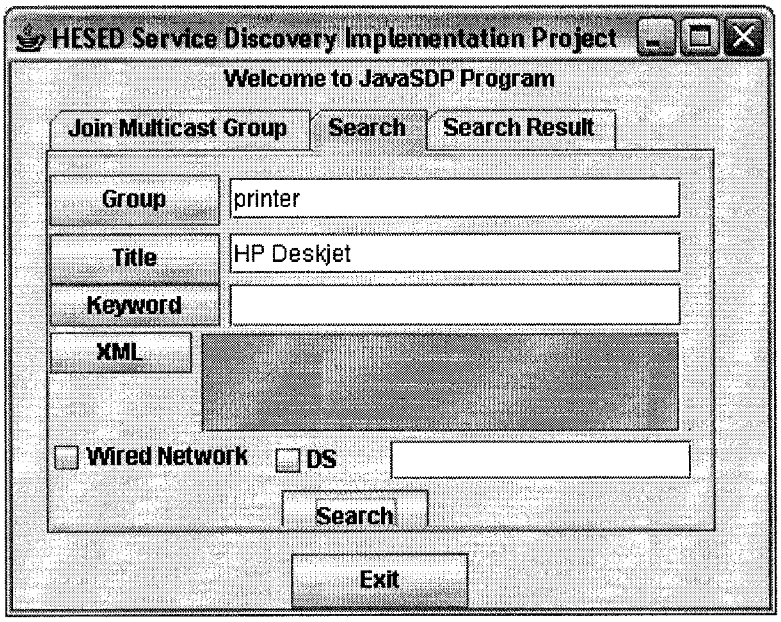
(A) The client joins a multicast group


HESED Service Discovery Implementation Project
Welcome to JavaSDP Program
Join Multicast Group
Search
Search Result
Select All
Clear
Join
User Defined Multicast Group
Add
Delete
Exit

☑ Printer
□ Ftp service
□ Database Server
□ General

Figure 4.11 Test Result(1) - Join Panel

(B) The client inputs search criteria



Group: printer

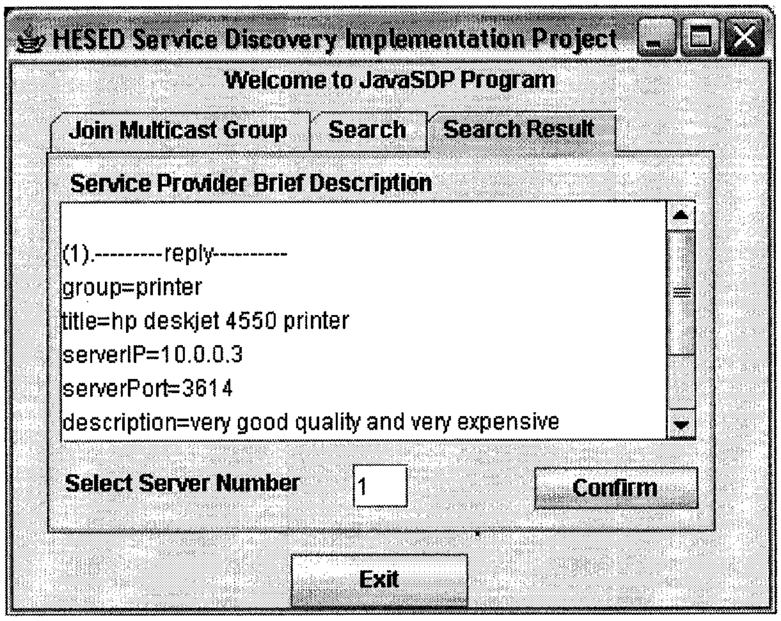
Title: HP Deskjet

Keyword:

□ Wired Network □ DS

Figure 4.12 Test Result(2) - Search Panel

(C) The client displays the search result on the result panel.


HESED Service Discovery Implementation Project
Welcome to JavaSDP Program
Join Multicast Group
Search
Search Result
Confirm
Exit

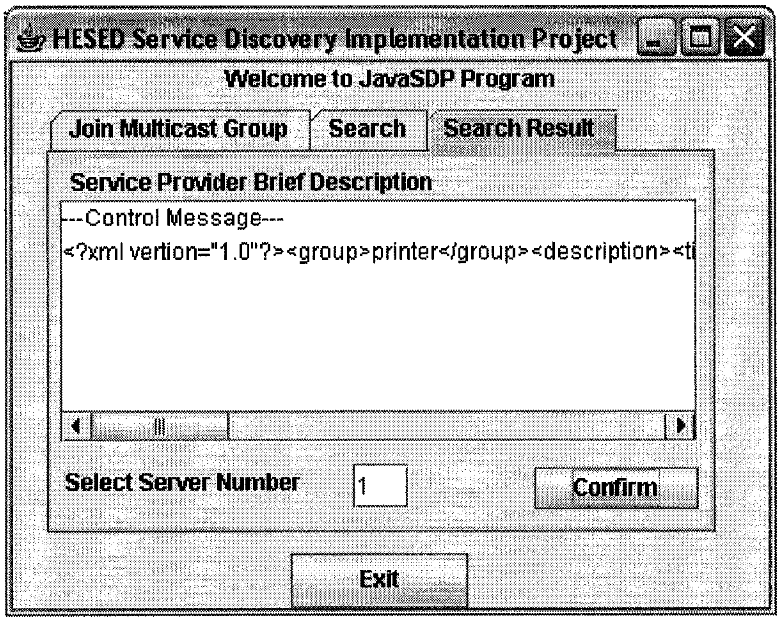
Service Provider Brief Description

```
(1).---------reply----------
group=printer
title=hp deskjet 4550 printer
serverIP=10.0.0.3
serverPort=3614
description=very good quality and very expensive
```

Select Server Number: 1

Figure 4.13 Test Result(3) – Server's Reply on Result Panel

(D) This is the server's unicast ACK which is received by the client.


HESED Service Discovery Implementation Project
Welcome to JavaSDP Program
Join Multicast Group
Search
Search Result
Service Provider Brief Description
Confirm
Exit

```
---Control Message---
<?xml vertion="1.0"?><group>printer</group><description><ti
```

Select Server Number 1

Figure 4.14 Test Result(4) – Server's ACK on Result Panel

(E) The server's command prompt output:

```
C:\test\Javasdp>Java WA_SDP/server/SDPAHServer
IP=10.0.0.3
Port=3614
Start Hello Receiver
Start Hello Sender
Start listening on 239.0.0.101

(1) Add One Neighbour:10.0.0.6@3217
No other neighbours

(1) Add One Neighbour:10.0.0.6@3217
Bi -Conn:10.0.0.3@3594

(1) Add One Neighbour:10.0.0.6@3217
```

Figure 4.15 Test Result(5) – Server Output

(F) The client's command prompt output

```
C:\test\Javasdp>Java WA_SDP/client/SDPClient
IP=10.0.0.1
Port=3592
Start Hello Receiver
Start Hello Sender
Start listening on 239.0.0.101

(1) Add One Neighbour:10.0.0.20@4485
Uni-Conn:10.0.0.1@3592

(1) Add One Neighbour:10.0.0.20@4485
Bi -Conn:10.0.0.1@3592

Client Sends multicast Query*****   Multicast Packet Header    *****
Packet ID = '000002af'
Source IP = '10.0.0.1'
PreviousIP= '0.0.0.0'
Payload   =   '<?XML   version="1.0"><group>printer</group><Query><title>HP
Deskjet</title><keyword></keyword></Query>'

Receive and Add to ResultPool *****   Multicast Packet Header    *****
Packet ID = '00000237'
Source IP = '10.0.0.3'
PreviousIP= '10.0.0.20'
Payload = '<?xml vertion="1.0"?><group>printer</group><reply><title>HP Deskjet
4550      printer</title><serverip>10.0.0.3</serverip><serverport>3614</serverport>
<description>Very good quality and very expensive</description><load>95%</load
></reply>'


Sender: send it to 10.0.0.3 @3608*****   Unicast Packet Header    *****
Packet ID = '000002b0'
Source IP = '10.0.0.1'
Source Port= '3592'
Dest IP = '10.0.0.3'
Dest Port= '3614'
Payload  =  '<?xml  vertion="1.0"?><group>printer</group><confirmation><title>hp
deskjet 4550 printer</title><clientip>10.0.0.1</clientip><clientport>3592</cl
ientport></confirmation>'
```

Figure 4.16 Test Result(6) – Client Output

Chapter 5 HESED Simulation and Result Analysis

We evaluate the performance of HESED by comparing it to the traditional service discovery using simulation. The performance of HESED is tested under three patterns, which are nodal density pattern, field size pattern and mobility pattern. We also change the network traffic of the above three scenarios. Simulation is also used to verify the theoretical analysis of Section 3.9.

5.1 Simulation Setting

5.1.1 Basic Parameters

We simulate MANETs with an area of 1000-1600m in length and 300-480m in width, which has on average 0.00005-0.00014 nodes per square meter (15-63 nodes in total). All nodes are evenly distributed in the simulation area and service discovery queries are generated at each node according to a Poisson distribution with an arrival rate of 0.1-0.2 queries per second. The communication range for each node is a

Gaussian random variable whose mean value is 200m and variance is 50m. The packet processing time for each node is evenly distributed from 1 to 100ms. Each node sends the hello message for neighbour detection every 3 seconds. The simulation time period is 300 seconds and each simulation is run 10 times and the average is obtained and reported. We assume real MANETs are large scale and contain many servers and each server is in charge of one small area, and we only simulate one such small area in our simulation.

The service discovery query rate is a critical parameter in our simulation. Since our ultimate goal is to make MANETs have the same functionality as wired networks, we selected the query rate according to that seen in wired networks. In wired networks, when a user is surfing the Internet, the user could open many web pages, which contain quite a number of different objects. Due to the mobility of MANETs, we should consider each webpage object as a new service request, even if they come from the same client. The actual number of queries used in our simulation was thus selected from 0.1 to 0.2 queries per second per node. Because we use cached information, HESED can achieve better performance when the query rate is ever higher than in the simulation.

Since wireless cards vary and the transmission rate could be quite different, we assume there are three kinds of wireless adapters used in the simulation, whose transmission rate are 56kbps, 11Mbps and 54Mbps with probability 20%, 40% and 40%, respectively.

Parameter	Nominal value
Simulation field size	(1000m-1600m) * (300m-480m)
Nodal density	0.00005-0.00014 nodes/ m^2
Number of nodes	15-63
Node placement	Evenly random
Query arrival rate	0.1-0.2 queries/second
Communication range	Mean is 200m, variance is 50m
Processing delay at each node	1-100 ms
Period of sending hello messages	3 seconds
Simulation time	300ms
Number of available servers	1
Bandwidth	56kbps, 11Mbps and 54Mbps

Table 5.1 Simulation Environment Parameters

5.1.2 Mobility Model

The random waypoint model [54][55] is the most commonly used mobility mode for MANETs. Each node randomly selects a destination, moves to that location with a randomly selected speed and stays at that point for a randomly selected time period. This mode will cause the node density at the central area to be higher than that at border areas, nodes are not evenly distributed in the simulation area, and the simulation mode would not represent the real world very well. The model is adapted in the simulation by requiring nodes to bounce at the boundary thus all the nodes are evenly distributed within the simulation field at any time.

We consider the velocity as an attribute of a mobile node and one mobile node cannot change its velocity during its life time, which means all the nodes are moving at their average velocity all the time. A node can move along one direction for a random selected time period, pause for a fixed time period and change its direction after pausing. In our simulation, the nodal velocity is distributed according to a Gaussian distribution whose mean is 5 m/s and variance is 1 m/s. The pause time is two seconds. Node moving time is randomly selected from 1000ms to 5000ms.

Parameter	Nominal value
Nodal movement model	Revised random waypoint mode
Speed	Mean 5m/s, variance 1/m
Node moving time	1000 ms -5000 ms
Pause time	2 seconds

Table 5.2 Mobility Model Parameters

5.1.3 Medium Access Control Model

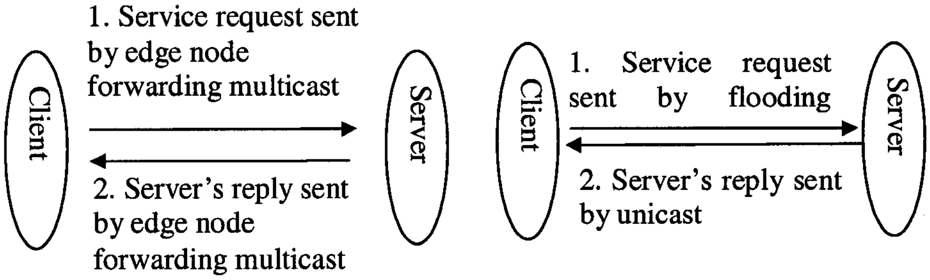
We adopt a simplified multiple access control (MAC) scheme. We did not implement the IEEE 802.11 [56] RTS/CTS MAC protocol in our simulation because our service discovery algorithm is aimed for all kinds of wireless devices such as cell phones, PDAs, laptop computers or Bluetooth devices. We do not want to be limited by any existing MAC protocol. Another reason is that IEEE 802.11 does not define the MAC mode for broadcasting, and more than 50% of the packets in HESED are multicast packets. In our simulation, we assume there is a perfect MAC protocol, which means a node can send a packet as soon as the medium is available and there is no communication overhead, as in RTS/CTS etc. Medium availability is when all nodes inside the sender node's communication range are not receiving or sending packets at that moment. If a node wants to send a packet but the medium is busy, it must wait and this event is called a collision in our simulation. We implemented a global medium access queue and all the packets must enter this queue first before they can be sent. This queue is sorted by sending time and all the packets have to check the medium availability before they can leave the queue.

5.2 Test Scenarios

5.2.1 Scenarios Descriptions

We compared two schemes in this simulation (see Figure 5.1), one is the HESED scheme and another is the Traditional On-demand Service Discovery mode, which is abbreviated as TOSD in the rest of this chapter.

In HESED mode, client nodes will check whether there is cached information available when they need a service. In the event of a cache hit, a route validation check is started for checking whether the route can be reused. A new service discovery process is started if the route is invalid.


Client
1. Service request sent by edge node forwarding multicast
2. Server's reply sent by edge node forwarding multicast
Server
Client
1. Service request sent by flooding
2. Server's reply sent by unicast
Server

(A) HESED Scheme (B) TOSD scheme

Figure 5.1 Two Schemes in the Simulation

In the TOSD scheme (viz. [35]), clients use the flooding mode to broadcast the service request. The service providers send a unicast reply after they receive the query. This TOSD mode in our simulation is improved in two aspects. First, we assume there is no route discovery cost. Because route discovery belongs to the network layer and the service discovery is in the application layer, there is no cost for the network layer to discover a route to destination when it is doing a unicast. HESED does not have such a cost because the backward learning algorithm can find route without additional cost. The TOSD scheme in the simulation also uses the backward learning routing algorithm and there is no extra communication cost or delay for the routing phase. Second, we assume the flooding mode can solve the asymmetric link problem. For example, each node maintains a bi-directional neighbour list and only forwards the packets received from the nodes that are on the list. We cannot reuse the cached information because there is no cache evaluating algorithm comparable to our research.

We do not perform a comparison between our scheme and other agent-based or cluster-based service discovery algorithm because it is not practical to implement an agent or a cluster header with unreliable nodes or links. The performance of the agent-based scheme depends on the stability of the agent node and the result may vary under different node stability assumptions. Agent-based or cluster-based service discovery algorithms can attain very good performance if they assume the agent nodes are super-nodes, who have the unlimited energy supply and are always alive. Since it is very hard to verify the correctness of such an assumption, we donot compare our algorithm with agent-based or cluster-based algorithms.

5.2.2 Performance Metrics

Our performance metrics include three parameters for each test case which are delay, number of packets and cache hit rate. The delay is the average delay for all successful queries. In HESED, the delay includes the delay for a multicast service request and that for a multicast service reply. The delay in TOSD includes the delay for one multicast service request and that for another unicast service reply. The number of packets is the average number of packets for each successful query and this data is directly related to energy consumption. In HESED, these packets include two multicast communications and the cost of hello messages. In TOSD mode, the total packets include one multicast communication and one unicast communication. The cache hit rate is the probability that cached service information can be used by a client node, and this can show how often the cached service information can be reused by a new query.

In the analysis, we use three simulation patterns, which we call pattern A, B and C. In pattern A (nodal density pattern), the average velocity and the simulation area are fixed and we increase the nodal density from 0.00005 to 0.00014 nodes per square meter and the service discovery query rate from 0.1 to 0.19 queries per second per node. This pattern is designed for comparing the performance when the network traffic is high.

Parameter	Nominal value
Simulation field size	1500*300
Average velocity	5 m/s
Query arrival rate	0.1, 0.13, 0.16, 0.19 queries/second
Nodal density	0.00005, 0.00008, 0.00011, 0.00014 nodes/m^2
Number of nodes	22, 36, 49, 63

Table 5.3 Parameters for Pattern A

In pattern B (field size pattern), the nodal density is fixed and the simulation field is changed from 1000m * 300m to 1600m * 480m and the service discovery rate from 0.1 to 0.19 queries per second per node. Since the nodal density does not change, the number of nodes increases when the simulation field becomes larger. This pattern is designed for comparing the performance of the above two schemes when one service provider has to provide services to a large area.

Parameter	Nominal value
Simulation field size	1000*300, 1200*360, 1400*420, 1600*800
Average velocity	5 m/s
Query arrival rate	0.1, 0.13, 0.16, 0.19 queries/second
Nodal density	0.00005 nodes/m^2
Number of nodes	15, 21, 29, 38

Table 5.4 Parameters for Pattern B

In pattern C (mobility pattern), the average velocity is changed from 0-6 m/s and the service discovery query rate is also from 0.1 to 0.19 queries per second per node. In this pattern, we try to evaluate our scheme's performance under different nodal mobility.

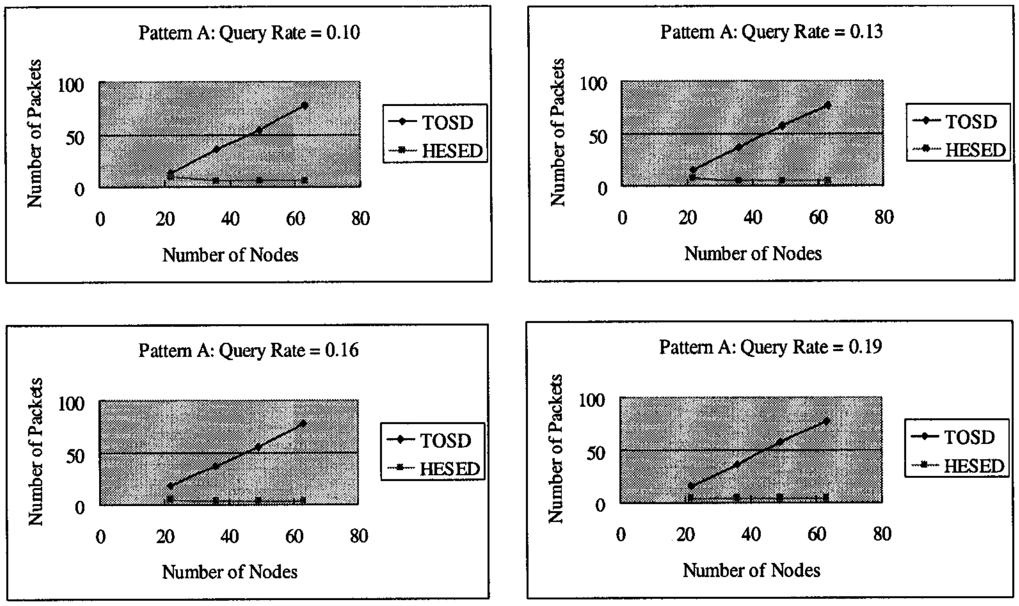
Parameter	Nominal value
Simulation field size	1500*300
Average velocity	0, 2, 4, 6m/s
Query arrival rate	0.1, 0.13, 0.16, 0.19 queries/second
Nodal density	0.00005 nodes/m^2
Number of nodes	22

Table 5.5 Parameters for Pattern C

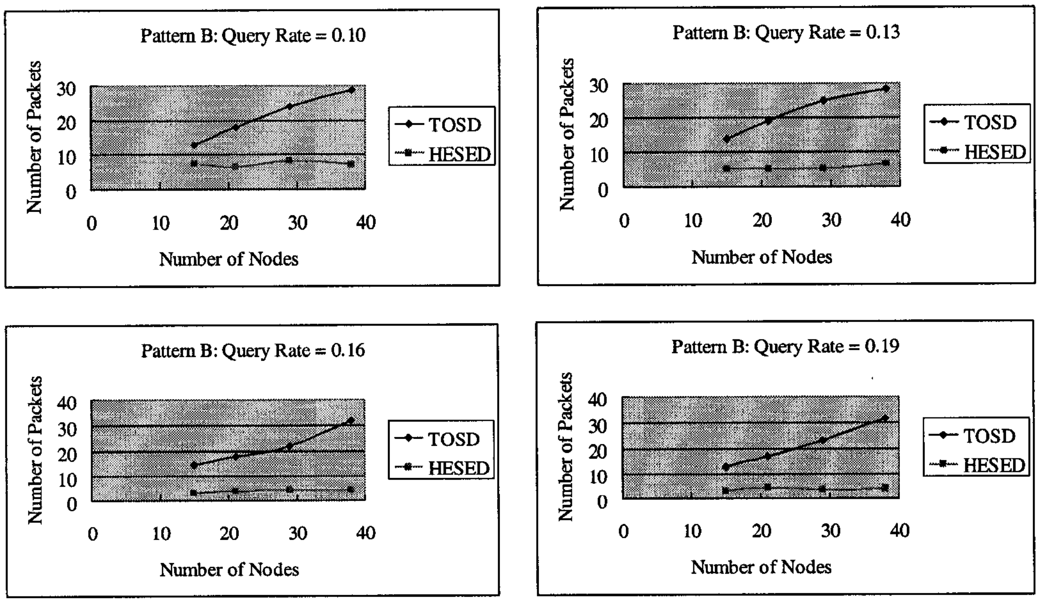
5.3 Simulation Analysis

5.3.1 Effect of Number of Packets

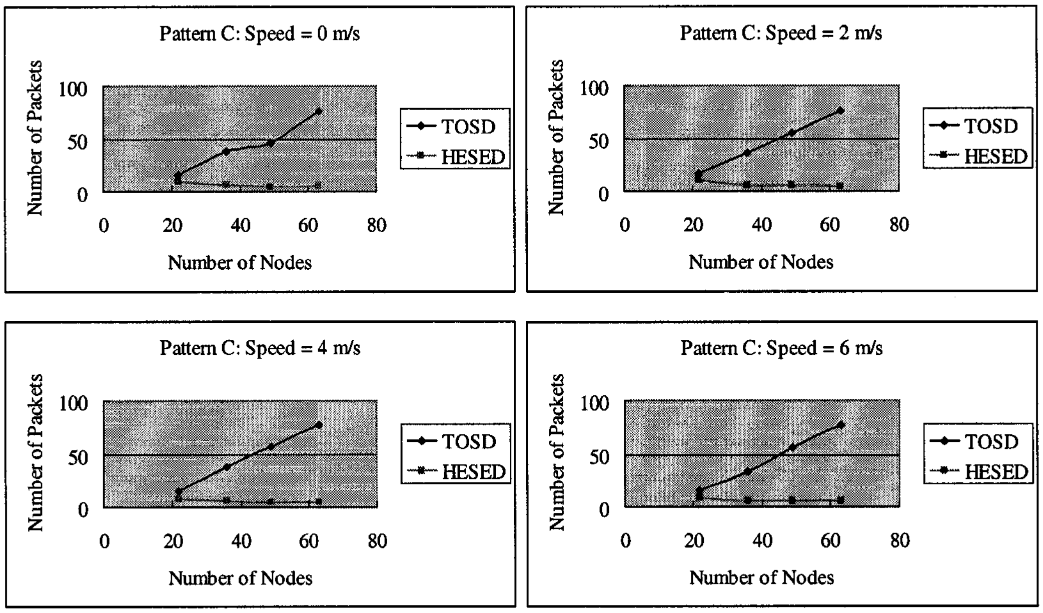
The number of packets is directly related to energy consumption. Figure 5.2 is a comparison of number of packets per query for the HESED and TOSD schemes.



(E) Pattern A (Nodal Density Mode)



(B) Pattern B (field size pattern)



(C) Pattern C (mobility pattern)

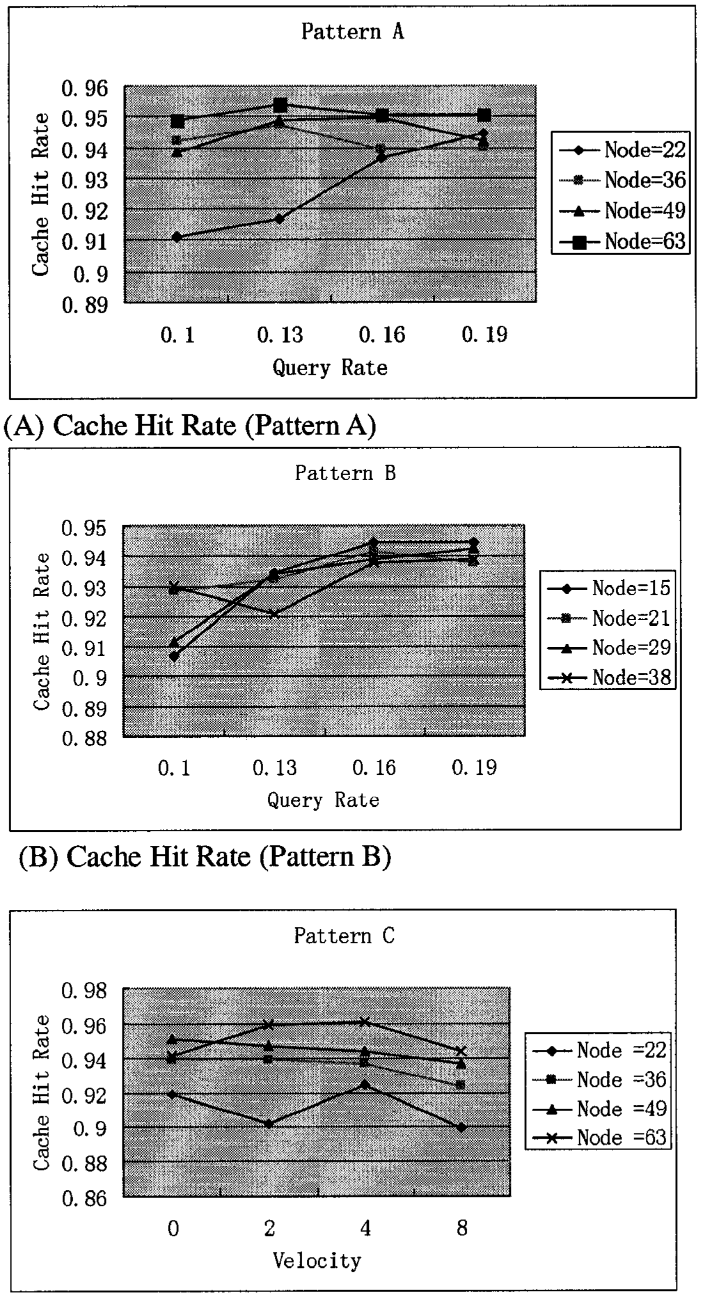
Figure 5.2 Average Numbers of Packets Per Query

Figure 5.2 (A)(B)(C) correspond to pattern A, B and C, respectively. The curves for TOSD are the number of packets per query for the traditional flooding mode. The curves for HESED are the numbers of packets which included the query cost and beacon cost. In, short, it is the average for all the queries with all the costs. In Figure 5.2 (A) and (B), the four parts represent the query rate as 0.1, 0.13, 0.16 and 0.19, respectively. These four figures in Figure 5.2 (C) represent different node velocity as 0, 2, 4, 6 m/s, respectively.

In TOSD mode, the average number of packets per query includes the flooding broadcast packets and the multi-hop unicast packets. In HESED mode, the number of packets includes the hello message cost and two edge forwarding cost, which is one for the client's query and another for the server's response. Theoretically, the number of packets for a flooding broadcast should be equal to the number of nodes in the whole simulation area. Since there always are some nodes isolated from others, the number of packets for a flooding broadcast is always lower than the number of nodes. The edge node forwarding algorithm only queries selected nodes to send packets thus the average number of packets of the implemented queries is lower than that of the flooding broadcast. When we count all the queries, which include those queries using the cached information, the average number of packets hardly changes when the number of nodes is increasing. We can see from the figures that the curves of the TOSD scheme are going up with the number of nodes and the curves for HESED are flat. Recall that the average number of packets per query for TOSD scheme is O(N) and that for HESED is O(1). Recall that when we calculate the energy consumption (message complexity) for the whole network, it is $O(N^2)$ for TOSD and O(N) for HESED.

5.3.2 Cache Hit Rate

The cache hit rate is the ratio between the number of queries using the cache and the total number of queries. Cache hit rate is a very important parameter for indicating the adaptability of HESED on different kinds of MANETs, thus we measure and analyse it in this section.



(A) Cache Hit Rate (Pattern A)

(B) Cache Hit Rate (Pattern B)

(C) Cache Hit Rate (Pattern C)

Figure 5.3 Comparisons for Cache Hit Rate

Figure 5.3 (A)(B)(C) are the cache hit rate for patterns A,B and C, respectively. Since the server node always multicast its information to all clients, those nodes who are closest to the server always receive the newest information and can use the cached information. There are two events which can cause most of the new queries: one event is that the node's cached information expires when it needs service, and another is when a node just comes into the server node's connectivity area and has not received any service information yet.

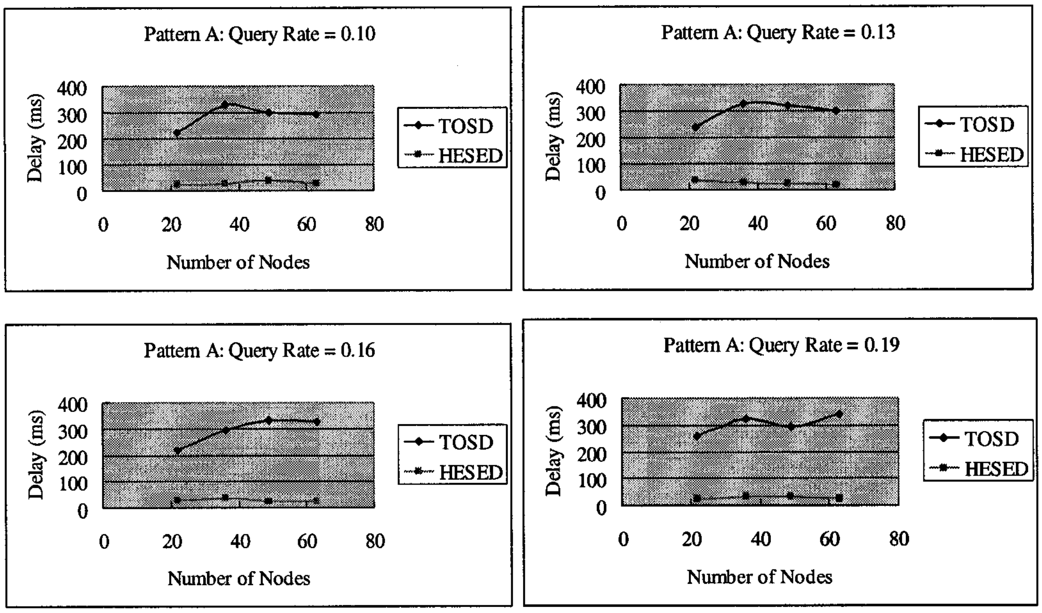
The first event usually occurs when the node is very far from the server and the lifetime for the cached information is very short. This kind of event increases in frequency when the query rate becomes low. In such cases, the interval between two service responses is enlarged and there is a greater probability that remote nodes are timing out. This can be illustrated using Figure 5.3 (A) and (B) in which the cache hit rate increases with the number of nodes.

The likelihood of the second event increases when the nodal mobility is high and more nodes may come and leave the server's connectivity area during the simulation time period. Thus the cache hit rate decreases when the average velocity increases (see Figure 5.3 (C)).

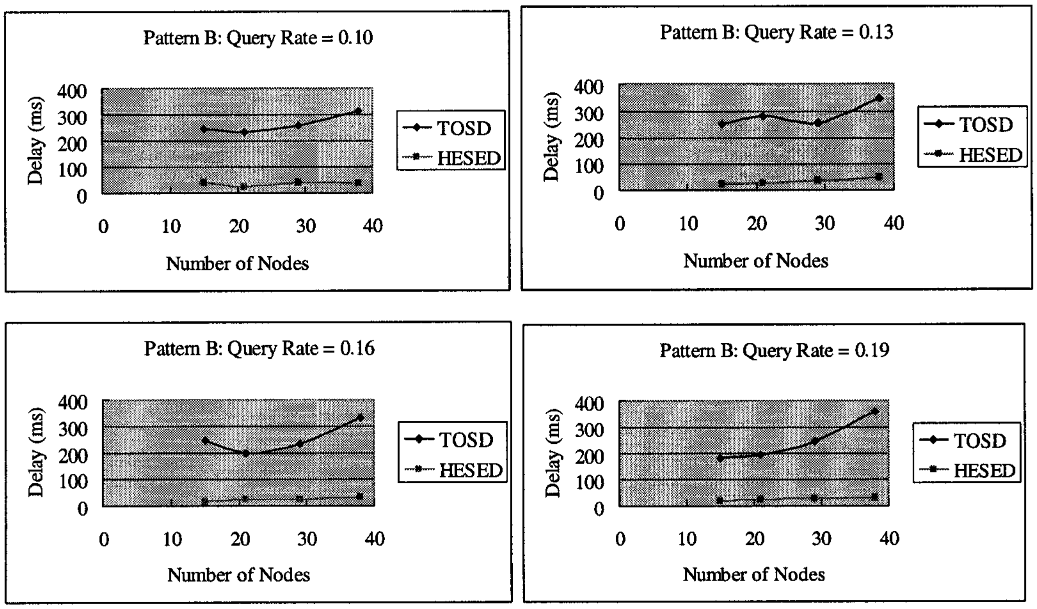
The cache hit rate can present the cache information usage and this result can help to improve our service discovery scheme and maximize the usage of the cached information in the future.

5.3.3 Delay

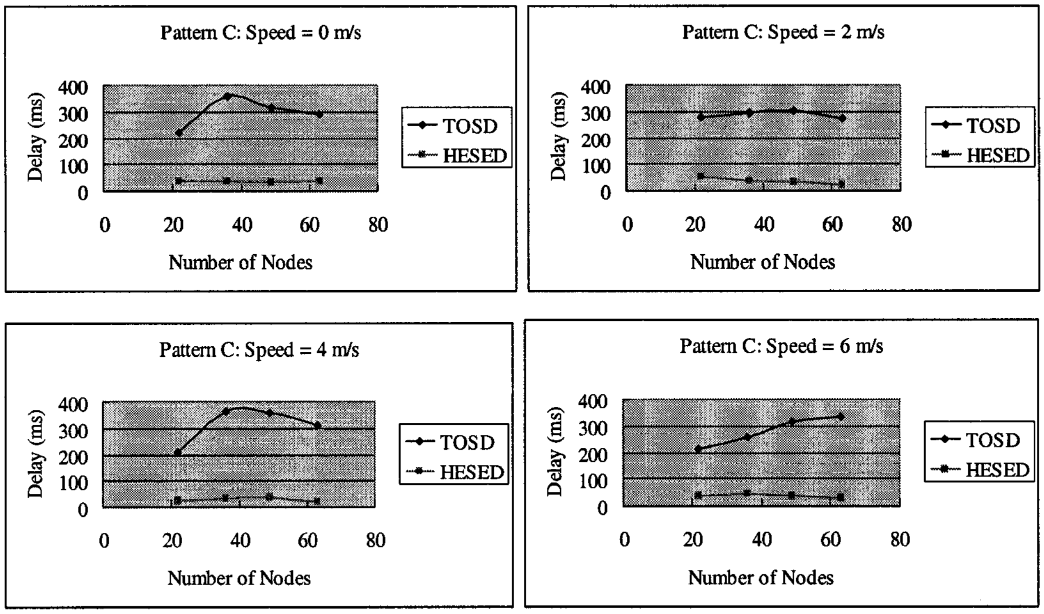
Delay is a critical performance metric for any service discovery scheme. Figure 5.3 indicates that the average delay for our HESED scheme is much lower than the TOSD scheme.



(A) Pattern A (nodal density pattern)



(B) Pattern B (field size pattern)



(C) Pattern C (mobility pattern)

Figure 5.4 Comparison of delay

Figure 5.4 (A) (B) (C) are the delay for pattern A, B and C, respectively. The TOSD curves are the delay for the traditional scheme and the HESED curves are the average delay per query for our scheme. The four figures in sections (A) and (B) represent different query rates as 0.1, 0.13, 0.16, 0.19 queries per second per node, respectively. In fact, the number of queries did not affect the average delay per query too much. Section C shows the average delay per query for different nodal mobility and the velocity for these four figures are 0, 2, 4 and 6 m/s respectively.

The above figures indicate the performance comparison under different cases such as different node density, different simulation field and different node mobility. HESED has a better performance than that of TOSD in all of the above scenarios.

Chapter 6 Conclusions and Future Work

This chapter summarizes the results of the thesis work and provides a number of directions for future work.

6.1 Summary

In this thesis, we introduce a high efficiency service discovery (HESED) scheme for MANETs, which has low communication cost and short delay. HESED is fundamentally different from earlier post-query strategies in that it uses query-all and reply-all mode. Query-all means clients send multicast queries to all reachable nodes with a limited TTL (time to live). Service providers also use multicast mode with limited TTL to send their reply and the intermediate nodes do not send any reply even if they have some knowledge about the query. All the nodes which receive the service reply should cache the service information in their local memory. The packet complexity of HESED is O(N) for N-node MANETs as opposed to O(N^2) for traditional schemes. HESED also eliminates the effect of asymmetric links and provides reliable connectivity. Our theoretical analysis, simulation and implementation demonstrate the superiority of HESED over existing schemes.

6.2 Future Work

HESED provides a highly efficient framework for service discovery on MANETs. There are still several issues to be resolved before it can be widely used, including:

A. Reliability: reliable multicast algorithm in MANETs.

B. Security: a third party cannot steal the user's information

C. Commercializing the product: service discovery library for developers

D. Service classification and matching algorithm: tree-based service matching can greatly improve the searching performance.

E. Standard XML templates.

6.2.1 Reliable Service Discovery

Since most of the communications in HESED are multicast, a reliable multicast algorithm is necessary for a reliable service discovery scheme.

In our edge node forwarding mode, all the edge nodes form a multicast backbone. We are planning to setup an acknowledgement mechanism for all the backbone nodes and make sure all backbone nodes can receive and forward multicast packets. In this algorithm, all the backbone nodes should send ACKs to their previous hop node. The previous hop node should count the number of received ACKs. If it fails to collect all the ACKs, it has to resend the packet until it receives all the ACKs or reaches the maximum retry limitation. We have reserved a flag in the multicast packet header, which is in the state field (Appendix B.2). If the ACK style is "10" in the state field, then the edge nodes should send ACKs to its previous node.

6.2.2 Tree-based Service Search

When a server receives a client's query, the server should compare itself against the client's criteria and check if it matches. In HESED, we only do text comparison using brute force. If the MANET is quite large, the number of available services can be huge and the client's criteria can be very large. This can take a high computation cost if brute force algorithm is used. The time complexity for brute force is O(N) as opposed to O(logN) for a tree-based algorithm, which improves the search performance.

A service searching tree (Figure 6.1) can have different levels, which represent service classification. When moving down from top to bottom, the service will become more specific. A leaf node represents a specific service.

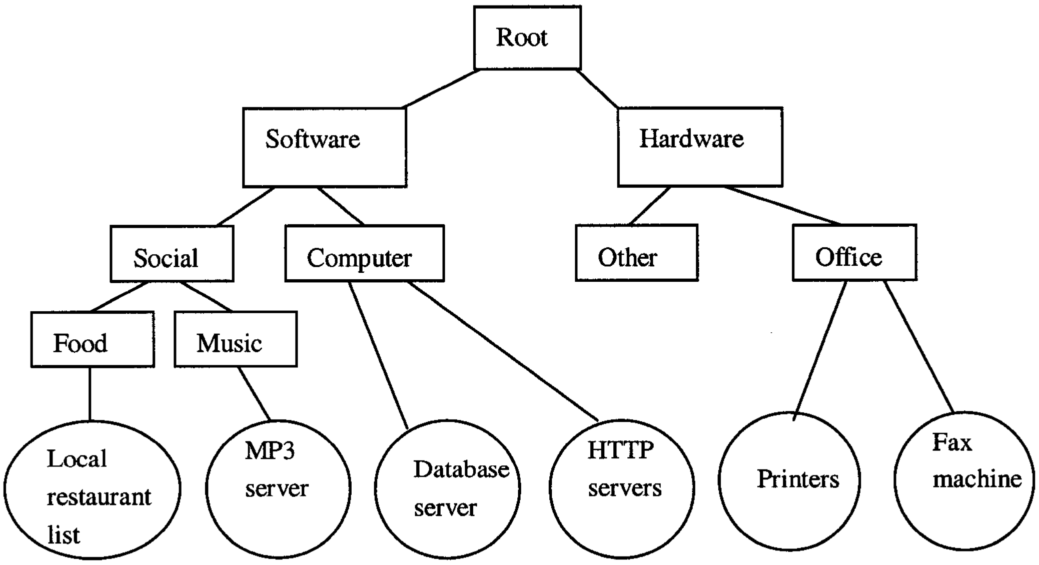


Figure 6.1 Service Searching Tree

The service searching tree is predefined, and can be stored at the client nodes and the server nodes. Client nodes could use this tree structure to create XML queries and server nodes could check whether the service query belongs to their own groups.

6.2.3 Security Over Service Discovery

Security is always a problem in wireless LANs because a hacker could receive the radio signal between the sender and the receiver. Service discovery has to use multicast for advertising the service information so all the clients could receive this information and there is no secret code among the client nodes. A technique like the Public-Key Cryptography can be used for solving the security problem. Such a solution would have the following steps:

Step1: A client node sends a plain-text service confirmation to the server after it receives the server's multicast reply.

Step2: The server sends an acknowledgement packet to the client, which includes the server's public key k_1.

Step3: The client node sends a service request to the server, which is encrypted using k_1 and this packet has the client's public key k_2.

Step4: All the packets sent by the client or the server should be encrypted with k1 and k2, respectively.

6.2.4 XML Templates

More specific XML templates should be developed for different kinds of services and devices. We cannot develop these currently because we still do not have large MANETs and the services exist only in our imagination. After MANETs have been in use for a while, we can analyze the service properties and propose more suitable XML templates.

References

[1] J. F.Kurose and K.W.Ross, Computer Networking: A Top-Down Approach Featuring the Internet, Second Edition. Addison Wesley, 2003.

[2] I. Chlamtac, M. Conti, and J.Liu, "Mobile Ad Hoc Netowrking: Imperatives and Challenges", *Ad Hoc Network Journal*, Volume 1, No.1, pp.13-64, January 2003.

[3] C. E. Perkings, AD HOC Networking. Addison Wesley, 2001.

[4] F.Y. Loo, "Ad Hoc Networks: Prospects and Challenges", Rinkou Paper, http://www.mlab.t.u-tokyo.ca.jp/~ylfoo/Research/MANET_RinKou.pdf, January 2004.

[5] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks", *Mobile Computing*, Kluwer Academic Publishers, pp.153-181, 1996.

[6] Y.B. Ko and N. H. Vaidya, "Location-Aided Routing (LAR) in Mobile Ad Hoc Networks", *Mobicom '98. 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 66-75, October 1998.

[7] C.-K. Toh, "A Novel Distributed Routing Protocol to Support Ad Hoc Mobile Computing", *15th IEEE Annual International Phoenix Conference on Computers and Communications*, pp. 480-486, 1996.

[8] C.E. Perkins and E.M. Royer, "Ad-hoc On Demand Distance Vector Routing", *the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90-100. , February 1999.

[9] V. Park and S. Corson, "Temporally-Ordered Routing Algorithm (TORA) Version 1" Internet Draft, draft-ietf-manet-tora-spec-03.txt, work in progress, June 2001.

[10] H.S. Hassanein and A. Zhou, "Load Aware Destination Controlled Routing for MANETs", *Computer Communications*, Volume 26, Issue 14, Septmeber 2003.

[11] P. Krishna, N.H. Vaidya, M. Chatterjee and D.K. Pradhan, "A Cluster-Based Approach for Routing in Dynamic Networks", *ACM SIGCOMM Computer Communications Review*, 1997.

[12] T.-W. Chen and M. Gerla, "Global State Routing: A New Routing Scheme for Ad-hoc Wireless Networks", *IEEE ICC '98*, pp. 171-175, Jun.1998.

[13] P. Jacquet, P. Muhlethaler, A. Qayyum, A. Laouiti, L. Viennot and H. Clausen, "Optimized Link State Routing Protocol Internet Draft", draft-ietf-manet-olsr-04.txt, work in progress, June 2001.

[14] J. Garcia-Luna and M. Spohn, "Tree Adaptive Routing Internet Draft", draftietf-manet-star-00.txt, work in progress, October 1999.

[15] S. Murthy and J.J. Garcia-Luna-Aceves, "An Efficient Routing Protocol for wireless Networks", *ACM Mobile Networks*, Special Issue on Routing in Mobile Communication Networks, pp.183-97, October 1996.

[16] C. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers", *Computer Communications Review*, pp. 234-244, October 1994.

[17] C. Chiang, "Routing in Clustered Multihop, MobileWireless Networks with Fading Channel", *IEEE SICON*, pp. 197-211., Apr.1997.

[18] Z. J. Haas, M. Pearlman and P. Samar, "The Bordercast Resolution Protocol (BRP) Internet Draft", draft-ietf-manet-zone-zrp-04.txt, work in progress, July 2002.

[19] H.S. Hassanein and A.M. Safwat, "Virtual Base Stations for Wireless Mobile Ad Hoc Networks – Infrastructure for the Infrastructure-less," *International Journal of Communication Systems*, Volume 14, Issue 8, October 2001.

[20] C. Ying, Q. Lu, Y. Liu and M. Shi, "routing protocols overview and design issues for self-organized network", *International Conference on Communication Technology (WCC - ICCT)*, Volume 2,pp.1298-1303, 2000.

[21] A. Boukerche, "A simulation based study of on-demand routing protocols for ad hoc wireless networks", *34th Annual Simulation Symposium*, pp.85-92, April 2001.

[22] Sun Microsystems, "JINI technology", http://www.sun.com/jini, January1999.

[23] OASIS, "UDDI Specifications Version 3.0.2", ttp://uddi.org/pubs/uddi_v3.htm, 2005.

[24] World Wide Web Consortium (W3C), "SOAP Specification", http://www.w3.org/TR/soap/, 2005.

[25] Sun Microsystems, "Java Remote Method Invocation (Java RMI)", http://Java.sun.com/products/jdk/rmi/, 2005.

[26] The Object Management Group (OMG), "Catalog of OMG CORBA®/IIOP® Specifications", http://www.omg.org/technology/documents/corba_spec_catalog.htm, 2005.

[27] A. S. Tanenbaum, "Computer Networks", Prentice Hall Inc., USA, 1996.

[28] W. Stallings, Data & Computer Communications, Sixth Edition. Prentice Hall, 2000.

[29] UPnP Forum, "UPnP^{TM} Documents", http://www.upnp.org/standardizeddcps/upnpresource20050331.zip, 2005.

[30] Internet Engineering Task Force (IETF), "RFC 791: Internet Protocol", http://www.ietf.org/rfc/rfc0791.txt, 1981.

[31] Internet Engineering Task Force (IETF), "RFC 2131: Dynamic Host Configuration Protocol ", http://www.ietf.org/rfc/rfc2131.txt, 1997.

[32] C. Toh, AD HOC Mobile Wireless Networks Protocols and Systems, Prentice Hall PTR, 2002.

[33] Internet Engineering Task Force (IETF), "RFC 2608: Service Location Protocol, Version 2 ", http://www.ietf.org/rfc/rfc2608.txt, 1999.

[34] M. Barbeau and E. Kranakis, "Modeling and Performance Analysis of Service Discovery Strategies in Ad Hoc Networks", *International Conference on Wireless Networks (ICWN)*, 2003.

[35] C. Lee, A.Helal, N.Desai, V. Verma and B. Arslan, "Konark: A system and protocols for device independent, peer-to-peer discovery and delivery of mobile services", *Systems, Man and Cybernetics*, Part A, IEEE Transactions on, Volume 33 , Issue: 6, pp.682 – 696, Nov. 2003.

[36] U. Kozat and L. Tassiulas, "Network Layer Support For Service Discovery In Mobile Ad Hoc Networks", *INFOCOM, Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, IEEE Volume 3, pp. 1965 - 1975, 2003.

[37] M. Klein B. Konig-Ries and P. Obreiter "Service Rings – A Semantic Overlay For Service Discovery In Ad Hoc Networks" *14th International Workshop on Database and Expert Systems Applications*, 2003.

[38] M. Klein B. Konig-Ries and P. Obreiter, "Lanes – A Lightweight Overlay For

Service Discovery In Mobile Ad Hoc Networks", 3rd *Workshop on Applications and Services in Wireless Networks (ASWN)*, 2003.

[39] M. Klein and B.Onig-Ries, "Multi-Layer Clusters In Ad-Hoc Networks – An Approach To Service Discovery", *International Workshop on Peer-to-Peer Computing, co-located with Networking 2002 Conference*, 2002.

[40] J. Liu, K. Sohraby and Q. Zhang, "Resource Discovery In Mobile Ad Hoc Networks", *The Electrical Engineering Handbook Series: The handbook of ad hoc wireless networks*, pp. 431 – 441, 2003.

[41] M. Haase, I. Sedov, S. Preuss, C. Cap and D. Timmermann, "Time and Energy Efficient Service Discovery in Bluetooth", *57th IEEE Semiannual Vehicular Technology Conference*, pp.736 – 742, 2003.

[42] J. Wu and M. Zitterbart "Service Awareness And Its Challenges In Mobile Ad Hoc Networks", *Workshop der Informatik*, Wien, Österreich, 2001.

[43] L. Cheng and I. Marsic, "Service Discovery And Invocation For Mobile Ad Hoc Networked Appliances", *2nd International Workshop on Networked Appliances (IWNA)*, 2000.

[44] L.Cheng, "Service Advertisement and Discovery In Mobile Ad Hoc Networks", *ACM Conference on Computer Supported Cooperative Work (CSCW)*, 2002.

[45] S. Motegi, K. Yshihara and H. Horiuchi, "Service Discovery For Wireless Ad Hoc Networks", *The 5th International Symposium on Wireless Personal Multimedia Communications*, pp.232- 236, 2002.

[46] R. Handorean and G. Roman, "Service Provision In Ad Hoc Networks", *5th International Conference on Coordination Models and Languages*, pp. 207-219, 2002.

[47] K. Nagi and B. König-Ries "Asynchronous Service Discovery In Mobile Ad-Hoc Networks", *Database Mechanisms for Mobile Applications*, pp.69-78, 2003.

[48] F. Zhu and M. Mutka and L. Ni "Splendor: A Secure, Private, And Location-Aware Service Discovery Protocol Supporting Mobile Services" *IEEE Annual Conference on Pervasive Computing and Communications (Percom)*,pp.235-242, 2003.

[49] I. Sedov, S.Preuss and C.Cap, "Time and energy efficient service discovery in

Bluetooth", *Vehicular Technology Conference*, pp.418 – 422, 2003.

[50] World Wide Web Consortium (W3C), "Extensible Makeup Language (XML) 1.0", http://www.w3.org/TR/REC-xml, 2004.

[51] Y. Zhao, L. Xu and M. Shi "On-demand multicast routing protocol with multipoint relay (ODMRP-MPR) in mobile ad-hoc network" *Communication Technology Proceedings, ICCT. International Conference* on, Volume.2, pp.1295 - 1300, 2003.

[52] Y. Xu and H. Hassanein, "Ad Hoc Layer for Seamless Wireless Hopping", Queen's Univeristy, PARTEQ, IP Disclosure (Patent in preparation), 2003.

[53] R. Au, "Wireless Printer Server Project", CISC 499, 2005.

[54] G.Lin, G. Noubir, and R. Rajaraman "Mobility models for ad hoc network simulation", *INFOCOM. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies* Volume 1, 7-11, pp.454 – 463, 2004.

[55] C.Bettstetter, G.Resta and P.Santi, "The node distribution of the random waypoint mobility model for wireless ad hoc networks" *Mobile Computing, IEEE Transactions* on, Volume 2, Issue: 3, pp. 257 – 269, 2003.

[56] Matthew Gast, 802.11 Wireless Networks: The Definitive Guide, O'Reilly, 2002.

Appendix A XML Packet Format

A.1 Client's Multicast Query

A client may send this multicast query when it needs a service and it cannot find that service in its cache. This query includes the search criteria such as title or keyword and the server can use such criteria to check if a match exists.

```
<?xml version=”1.0”?>
<group></group>
<query>
<title></title>
<keyword></keyword>
</query>
```

Figure A.1 Multicast Query Message Format

A.2 Server's Multicast Reply

When a server receives a client's multicast query and it matches the search term, the server should send a brief service description and server's location information such as IP address and port by multicast. All clients cache this service information for future use. If load balancing is required, we can add the server's load information such as capacity or current load into this packet.

```
<?xml version=”1.0”?>
<group></group>
<reply>
<title></title>
<serverIP></serverIP>
<serverPort></serverPort>
<description></description>
</reply>
```

Figure A.2 Multicast Reply Message Format

A.3 Client's Unicast Confirmation

When a client selects a server, it should send a confirmation to that server for registration purposes. This confirmation includes the client's information such as IP address and port number through which the server can communicate with the client.

```
<?xml version="1.0"?>
<group></group>
<confirmation>
<title></title>
<clientIP></clientIP>
<clientPort></clientPort>
<description></description>
</confirmation>
```

Figure A.3 Client Confirmation Message Format

A.4 Server's Unicast Acknowledgement

When a server decides to accept a new client, it sends an ACK to that client. This ACK can be translated into a webpage through which the clients utilize the service.

```
<?xml version=”1.0”?>
<group></group>
<details>
  <title></title>
  <serverIP></serverIP>
  <serverPort></serverPort>
  <description></description>
  <panel>
    <label>
        <position></position>
        <text></text>
        <name></name>
    </label>
    <inputFile>
        <position></position>
        <text></text>
        <name></name>
    </inputFile >
    <textField>
        <position></position>
        <text></text>
        <name></name>
    </textField >
  </panel>
</details>
```

Figure A.4 Server Acknowledge Message Format

A.5 Client's Service Request

Clients fill out and submit the service webpage when they want to utilize the service. In this request, clients inform the server of the necessary values or provide an input file so that the server can provide the service accordingly.

```
<?xml version=”1.0”?>
<group></group>
<request>
<variable name>value</variable name>

</request>
```

Figure A.5 Service Request Message Format

A.6 Server's Result Report

After the server finishes the requested service, this report should be sent to the client indicating the final result of the requested service. This packet ends the service discovery process.

```
<?xml version=”1.0”?>
<group></group>
<result></result>
```

Figure A.6 Service Result Message Format

Appendix B Packets Format

B.1 Hello Message

A hello message contains information for the current node and the uni-directional and bi-directional neighbours list.

Cur_IP	Cur_Port	Num	Neigh_IP (1)	Neigh_Port (1)	Neigh_IP (2)	Neigh_Port (2)	…	Properites
4	2	2	4	2	4	2		n

Figure B.1 Hello Message Format

Note:

Cur_IP: current node's IP address, 4 bytes

Cur_Port: current node's UDP port number, 2 bytes

Num: number of neighbour nodes, 2 bytes which is maximum 65536.

Neigh_IP: one of the known neighbour's IP address, 4 bytes.

Neigh_Port: one of the known neighbour's port, 2 bytes

Properties: property of each neighbour node which is one bit per node. The bit is 1 for symmetric link and 0 for asymmetric link.

B.2 Multicast Packet Format

(1) Multicast Header position

Mac Header	IP header	UDP Header	Multicast Header	Data

Figure B.2 Multicast Packet Header Position

Since this is an application level multicast that takes place above the transport layer, the multicast header should be put after the UDP header.

(2) Multicast Header Format

Packet ID (4)	Source IP (4)	Previous Node IP (4)	State (1)	TTL (2)
Number of Edge Nodes (1)	Edge Node IP (4)	Edge Node IP (4)	QoS (1)	T 1/2 (3)
Length (4)	Check sum (4)	Option		

Figure B.3 Multicast Packet Header

Notes:

PacketID : a unique number for a multicast packet

SourceIP: multicast source node's IP address

Previous node IP: because this is a multi-hop multicast, this field is the previous node's IP address. This field and the source IP field can be used for backward learning routing.

State: the type and properties for the current multicast. See next section.

TTL: Time-To-Live. Number of maximum multicast forwarding hops

Number of edge nodes: only edge nodes should forward the received packets.

Edge node IP: list all the edge nodes' IP addresses so that the receiver knows whether it needs to forward the packet.

QoS: reserved field for Quality of Service

T1/2: half neighbour changing time. It is changed at each forwarding node.

Length: length of packet header

Checksum: checksum for the whole packet

Options: reserved field.

(3) State field

The state field defines several different modes for the current multicast and is 8 bits.

multicast reply(1)	Forward style(2)	ACK style(2)					

Figure B.4 State field

Multicast Reply (1/0): All the multicast group members must send a unicast reply to group leader (Sender)

Forward Style: 00: only the mesh nodes forward the packet

01: only the edge nodes forward the packet

10: flood forward, every node must forward the packet once.

11: reserved

ACK style: 00: no ACK

01: ACK to direct sender by unicast

10: ACK to all neighbours by one-hop multicast

11: reserved

B.3 Unicast Packet Format

(1)Unicast Header position

Mac Header	IP header	UDP Header	Unicast Header	Data

Figure B.5 Unicast Packet Header Position

Since this is an application level multi-hop unicast that takes place above the transport layer, the unicast header should be put after the UDP header.

(2) Unicast Header Format

Packet ID(4)	Source IP (4)	Source Port(2)
Dest_IP(4)	Dest_Port(2)	TTL(2)
QoS(1)	State(1)	Length(4)
CheckSum(4)	Optional(var)	Data

Figure B.6 Unicast Packet Header

Notes:

PacketID : unique number for a unicast packet

SourceIP: source node's IP address

SourcePort: source node's UDP port number

Dest_IP: destination node's IP address

Dest_Port: destination node's UDP port number

Previous node IP: because this is a multi-hop multicast, this field is the latest node IP address. This field and the source IP field can be used for backward learning routing.

TTL: Time-To-Live. Number of maximum multicast forwarding hops

QoS: reserved field for Quality of Service

State: the type and properties for the current multicast. See next section.

Length: length of packet header

Checksum: checksum for the whole packets

Options: reserved field.

(3) State Field

FLAG	ACK	RESEND	FAILURE				

Figure B.7 Unicast Packet State Field

FLAG:	1: Control packet, 0: data packet
ACK:	1: ACK, 0: No-ACK
RESEND:	1: should resend it later, 0: ok, received
FAILURE:	1: failure notice, cannot find a route to destination
	0: successful delivery

Appendix C HESED Implementation Architecture

C.1 Client Package

C.1.1 Overview

The client package includes 9 classes, and it can create a GUI for the user and handle all the events coming from this GUI (see Figure C.1).

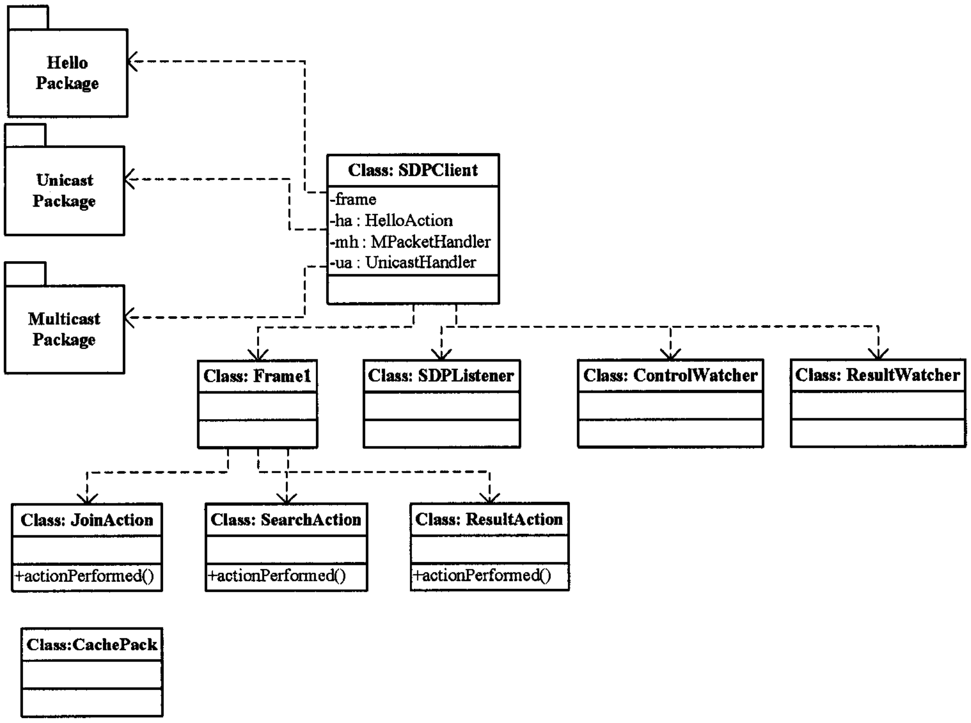


Figure C.1 Client Package Architecture

Class SDPClient has the main method and it invokes all of the other classes and methods in this package (see Figure C.1). It creates 4 threads: the GUI implementation thread, the hello message thread, the multicast handler thread and the unicast handler thread, using the class Frame1, HelloAction, MPacketHandler and

UnicastHandler, respectively. Class Frame1 is the GUI implementation and it has three panels: the join panel, the search panel and the result panel. There are three other classes, the event managers for those panels, called class JoinAction, SearchAction and ResultAction. SDPListener opens a multicast socket and receives packets. SDPListener adds the received multicast packets to a global multicast packet queue that is checked by the multicast handler. A multicast handler processes the packets and saves the result into a multicast result queue. There is also a similar queue called the unicast result queue which is a result for processed unicast packets by UnicastHandler. ControlWatcher and ResultWatcher check the unicast result queue and multicast result queue, and write the result to the search panel and the result panel, respectively.

C.1.2 Algorithms for Client Package

```
Algorithm for main()
  Start GUI Process
  Start SDPListener Thread
  Start HelloAction Thread
  Start MPacketHandler Thread
  Start UnicastAction Thread
End of Main

Algorithm for SDPListener
   Open a multicast socket
   While(true):
     Waiting for coming packets
     If a packet arrives, then:
        Add the packet to the input multicast packets queue
End of SDPListener

Algorithm for GUI and actionPerformed methods

  If "join" button is pressed, then:
    Switch to search panel
      Update the multicast packet filter
  If "search" button is pressed, then:
    Switch to result panel
    Send the multicast query
      Start ControlWatcher thread
      If a server reply is coming, then:
        ControlWatcher picks up the reply from queue.
          The server reply is displayed at the result panel
  If "confirm" button is pressed, then:
      Send a unicast confirmation
      Start ResultWatcher thread
      If a server's ACK is coming:
         ResultWatcher remove the ACK from the ACK queue
         The ACK is shown at the result panel
End of GUI
```

Figure C.2 Algorithm for Client Package

C.2 Server Package

The server package includes four classes: SDPAHServer, MulticastListener, ServerMulticastHandler and ServerUnicastHandler. SDPAHServer is the main class, MulticastListener receives and saves multicast packets into the input multicast packet queue. ServerMulticastHandler opens and replies to the received multicast packets and ServerUnicastHandler opens and replies to the unicast packets. We also used the unicast package, multicast package and hello package for multicast, unicast and neighbour detection, respectively. The architecture is shown in Figure C.3.

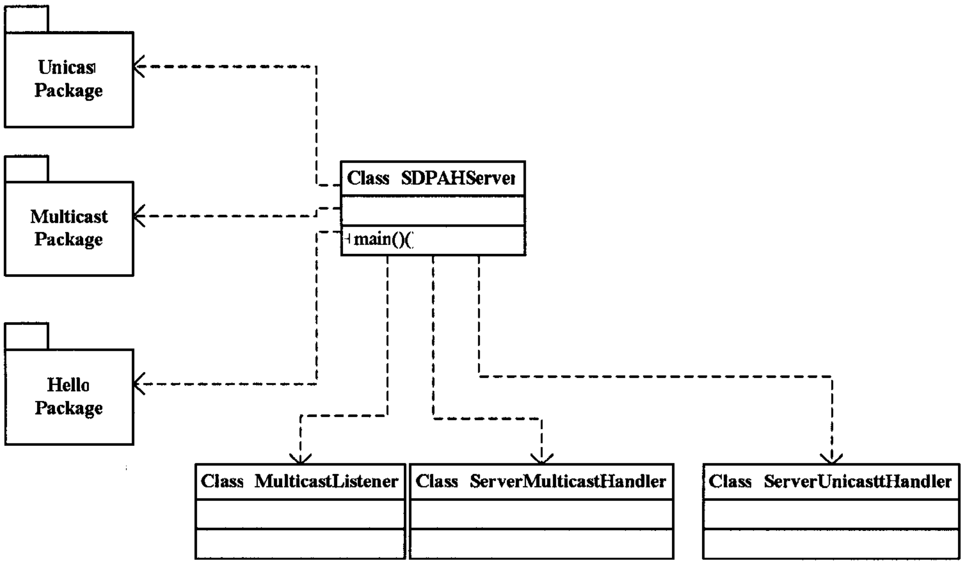


Figure C.3 Server Package Architecture

The server opens two sockets: one is a multicast socket for listening to clients' requests and another is a unicast socket for listening to these clients' confirmations. When a server receives a service request (Appendix A.1), the server creates and replies with a service's reply (Appendix A.2). If the server receives a unicast confirmation (Appendix A.3) , the server makes an ACK packet (Appendix A.4) and sends the ACK to that client. Since we also invoke the Hello package, Multicast package and Unicast package, the server also can perform neighbour detection, multicast and unicast packet forwarding.

C.3 Hello Package

C.3.1 Overview

The Hello package performs neighbour detection collecting all the nodes' information within two hops, eliminating asymmetric links, maintaining the edge node list and calculating the half neighbour changing time. This package includes five classes: HelloAction, HelloMsg, OneHopObject, OneHopSet and TS_half (see Figure 4.8).

HelloAction is the main class in this package and has two threads: HelloSender and HelloReceiver. HelloSender sends the hello message periodically for reporting its local topology information. HelloReceiver receives hello messages from the nearby nodes and updates the local topology information. The other 4 classes are data structure classes providing data structures for the hello message, one-hop node information and half neighbour changing time.

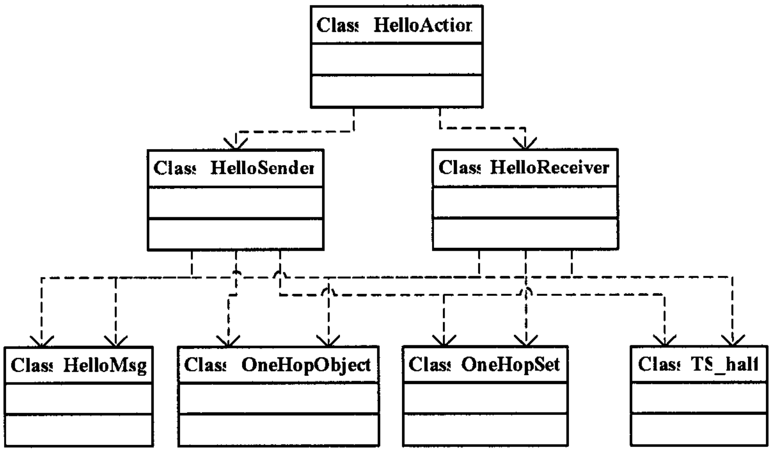


Figure C.4 Hello Package Architecture

C.3.2 Hello Message Format

The hello message (Appendix B.1) has all of the one-hop neighbour nodes' information and it also shows whether there is a symmetric link between a node and its neighbour or not. A node's ID includes the node's IP address and its UDP socket number so that there can be more than one client and server running on the same computer if they use different UDP socket number and have their own ID in HESED.

C.3.3 Algorithms for Hello Package

A. Beacon Handling Algorithm

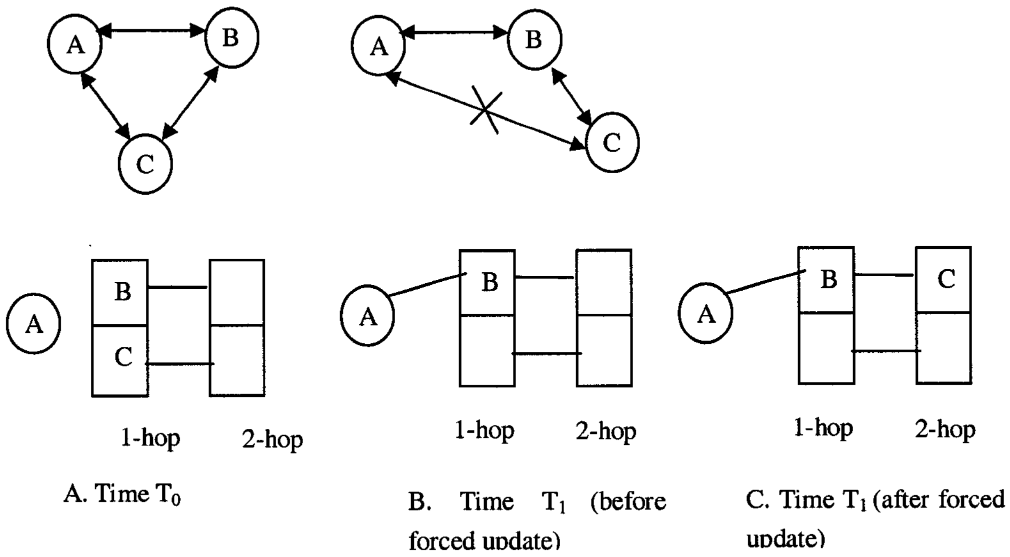
```
Set p1← new received hello message
Set cur_ID ← current node's ID
Set pattern ← p1's host node ID

If pattern= cur_ID, then return.
Set forcedUpdate ← (forcedUpdate+1)%10
If   forcedUpdate !=0, then
     if p1 has been registered in the hashtable, then
          update the time stamp for p1
          return
open packet p1
neigh[]← neighbour node's ID list in p1
for i←0 to neigh.length, do
     if neigh[i] = cur_ID, then
          there is bi-direction connection between cur node and p1
     else if neigh[i] is a uni-direction neighbour of cur_ID and it is bi-direction to pattern
          add neigh[i] as one of 2-hop neighbour of pattern
     else if neigh[i] is a bi-direction neighbour of pattern
          add neigh[i] as one of 2-hop neighbour of pattern
end
```

Figure C.5 Algorithm for Hello Package

Each node has two neighbour sets: the 1-hop neighbour set and the 2-hop neighbour set. The 1-hop neighbour set contains all the uni-directional and bi-directional neighbouring nodes. If a node (Node A) finds itself in another node's (Node B) neighbour list, Node A knows that Node B received Node A's hello message. Since Node A has received node B's hello message, Node A concludes that there is a bi-directional connection between nodes A and B. Using feedback from node B is the only way to decide whether the connection is bi-directional or uni-directional, therefore a long beacon algorithm is necessary for all reliable MANET applications. We expect this algorithm to be implemented by the wireless equipment manufacturers in the future because the asymmetric link problem is quite common for wireless ad hoc communication. We also have a checking validation algorithm, which can check the timestamp for each received beacon. If a neighbour node is time out, then it is removed from the neighbour list.

Since the neighbour nodes are not changed as frequently as the beacon message, each node should check the received beacon duplication. If one hello message is the same as the previous received message, we can sometimes ignore it for reducing the computation cost. In some specific cases, we have to update the local topology information even if the hello message is the same as before. To solve this problem, we have a variable called forcedUpdate which can update the local topology information even if one node receives the same hello message. It can solve the following problem which we call the disappeared neighbour problem.



A. Time T_0

B. Time T_1 (before forced update)

C. Time T_1 (after forced update)

Figure C.6 Disappeared Neighbour Problem

At time T_0, node A, B and C in Figure C.6, can detect each other, thus Node A finds it has two one-hop neighbours, Node B and C. At time T_1, Node C moves out of Node A's radio range and Node A cannot receive Node C's hello message. After Node C's previous hello message times out, Node A may delete Node C from its 1-hop neighbour's list. Node B still can sense Node C and A so node B thinks the network-topology did not change and its neighbours are still node A and node C, and node B will not change its hello message. Node A always receives the same hello message from Node B, and Node A thinks it does not need to change the two hop neighbours of node B, Node C then disappears from Node A's neighbours list, where it should be listed as a two hop neighbour. If Node A intends to send a multicast packet at this time, Node A does not think Node B can forward this packet and node C may never receive it.

If we force node A to update B's 2-hop neighbours even if node B's hello message does not change, then node A can become aware that Node C has become a neighbour of Node B and list Node C as one its 2-hop neighbour. In this manner, node C will not lose any multicast packets from node A, since node A has listed node C as one of its neighbours.

B. Edge Node Selection Algorithm

```
finishedOneHop ← new hashset
finishedTwoHop ← new hashset
keyList ← one hop bi-direction neighbours's list
do
    for i←0 to keyList.length
        curSize ← keyList[i]'s neighbour number
        if curSize = 0, then
            add keyList[i] to finisedOneHop
        if curSize is the max,then
            index ← i
    add keyList[index] to finishedOneHop
    if all of keyList[index]'s neighbours are in the finishedTwoHop, then
        continue
    else
        add keylist[index]'s neighbours into finishedTwoHop
while finishedOneHop's size != one hop neighbour's size
```

Figure C.7 Algorithm for Edge Node Selection

At the beginning of the algorithm, we create two new hash sets, which are finishedOneHop and finishedTwoHop. After that, we sort the 1-hop neighbours by the number of its neighbours, select these nodes on the ordered list one by one, and register all the 1-hop neighbours for the selected nodes. After we have registered all the 2-hop neighbours or finished all the 1-hop neighbours, then all the selected nodes form the edge node set.

C. Half Neighbour Changing Time Estimation Algorithm

A node should check the neighbour validation before it sends a hello beacon each time. If it finds some nodes are timed out, it will remove and count such nodes. This is called a T_half event. Each such event creates a T_half value, and we use the average of the latest 10 T_half value.

```
Algorithm for data collection
  numCh ← number of new removed neighbour nodes
  totalNeighbour ← number of all the 1-hop neighbours
  curTS← get from operating system
  Interval ← curTS – lastTS
  lastTS ← curTS
  T_half =   (interval * totalNeighbour)/(2*numCh)
  Add T_half to a TS queue

Algorithm for T_half calculation
  When need this T_half:
  If TS queue size >10
      Get the last 10 data from the TS queue
      Return the average for this 10 data
  Else
    Get all data from the TS queue
    Return the average for these data
```

Figure C.8 Algorithm for Half Neighbour Change Time

C.4 Multicast Package

C.4.1 Overview

The multicast package does not interact with other threads directly and it always gets or puts these packets to multicast input or output queues. The major functionality of this package is to receive and forward multicast packets. This package only includes two classes, which are MPacket and MPacketHandler. MPacket is the data structure class for multicast packets and MPacketHandler is a thread class which can receive and forward multicast packets. The multicast packet format can be found at Appendix B.2.

C.4.2 Flowchart For Multicast Packet Handler

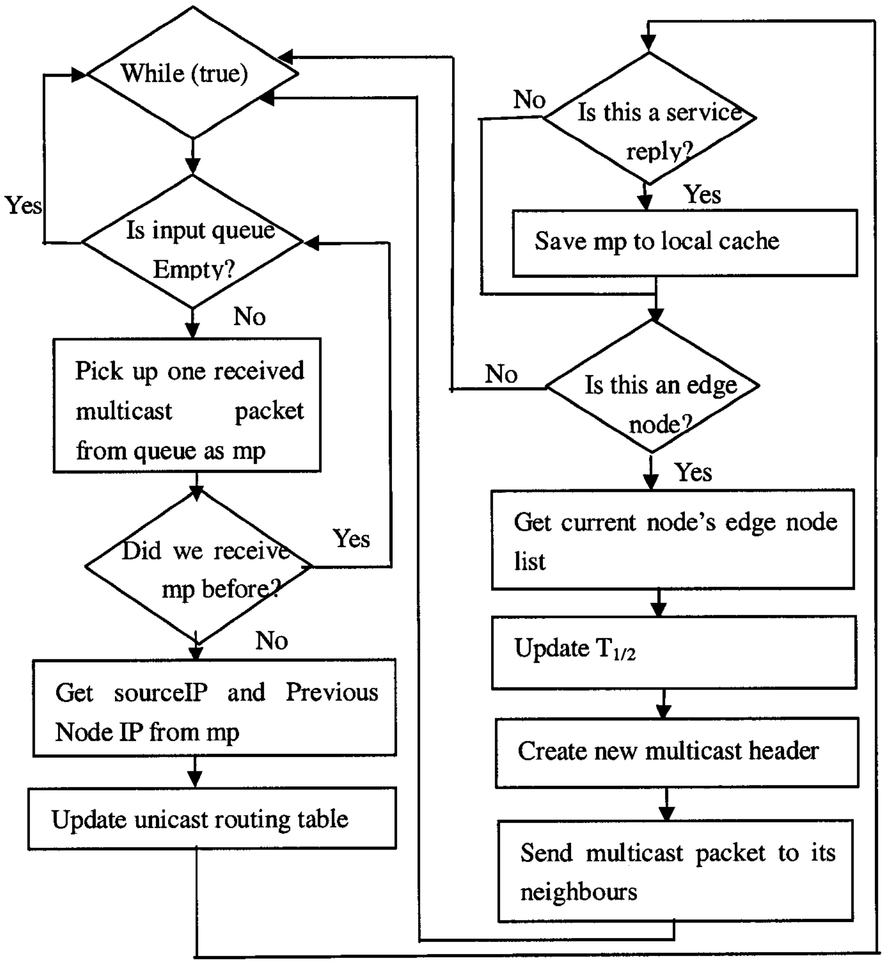

While (true)
Is input queue Empty?
Yes
No
Pick up one received multicast packet from queue as mp
Did we receive mp before?
Yes
No
Get sourceIP and Previous Node IP from mp
Update unicast routing table
Is this a service reply?
No
Yes
Save mp to local cache
Is this an edge node?
No
Yes
Get current node's edge node list
Update T1/2
Create new multicast header
Send multicast packet to its neighbours

Figure C.9 Flowchart for Multicast Handler

The multicast handler includes three functionalities: the unicast routing table update, the service cache update and multicast packet forwarding. When a node (node A) receives a multicast packet (MP), it checks whether it has received this packet before. If it is a duplicate multicast packet, then node A should drop this packet. Otherwise node A should open it and extract the source node (B) IP address and previous node(C) IP address. Node A can update the unicast routing table with these two IP addresses, all the unicast packets, whose destination is node B should henceforth be sent to node C first. If this is a service reply and node A is a client node, then node A should save this service information to its local cache. If node A is listed as one of the edge node in the received packet, then node A should rebuild the multicast packet header and resend it. The rebuilding work includes TTL decreasing, changing the previous node IP field to current node's IP address, creating a new edge node list and updating T_α.

C.5 Unicast Package

C.5.1 Unicast Package Architecture

This package includes five classes: UnicastAction, UnicastHandler, UnicastListener, UnicastSender and UPacket (Figure C.10). UnicastAction is the main class which starts three threads: UnicastListener, UnicastHander, UnicastSender. UnicastListener receives unicast packet and saves it to an input packet queue. UnicastHandler gets the packet from the input packet queue, deals with it and puts the modified packet into the output packet queue for UnicastSender. UnicastSender gets the packet from the output packet queue and sends it out. UnicastListener sends an ACK for all received unicast packets and UnicastSender checks ACKs and resends the packet if there is no ACK. UPacket is the data structure class for unicast packet. The unicast packet format can be found in Appendix B.3.

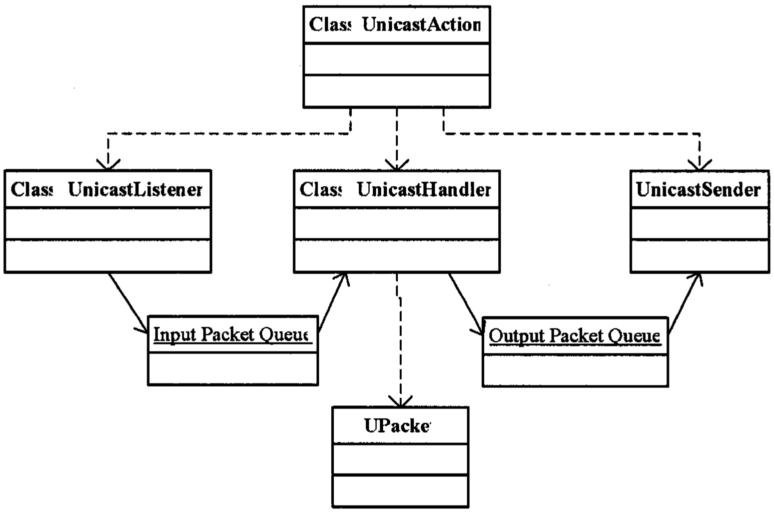


Figure C.10 Unicast Package Architecture

C.5.2 Flowchart for Unicast Packet Handler

When a node receives a unicast packet, it should check whether this packet's destination and port number match the current process. If it is the same then this packet should be put into the response queue so that other threads such as the client or server's main thread can get this packet. Otherwise, the current node should forward it to its destination. The current node should find out the next relay node's information from its routing table, decrease the TTL and send it out. The current node should also wait for an ACK from the next relay node. If it does not get an ACK by the time out, then this packet should be resent. Because we have the neighbour detection phase, if we can find the next relay node from the routing table then the relay node must be reachable by the current node. It is highly probable that packets can be delivered to next hop because of the ACK and resend mechanism. If the current node cannot find the relay node, then it should report a route failure to the source node, and the source node must repeat the service discovery process.

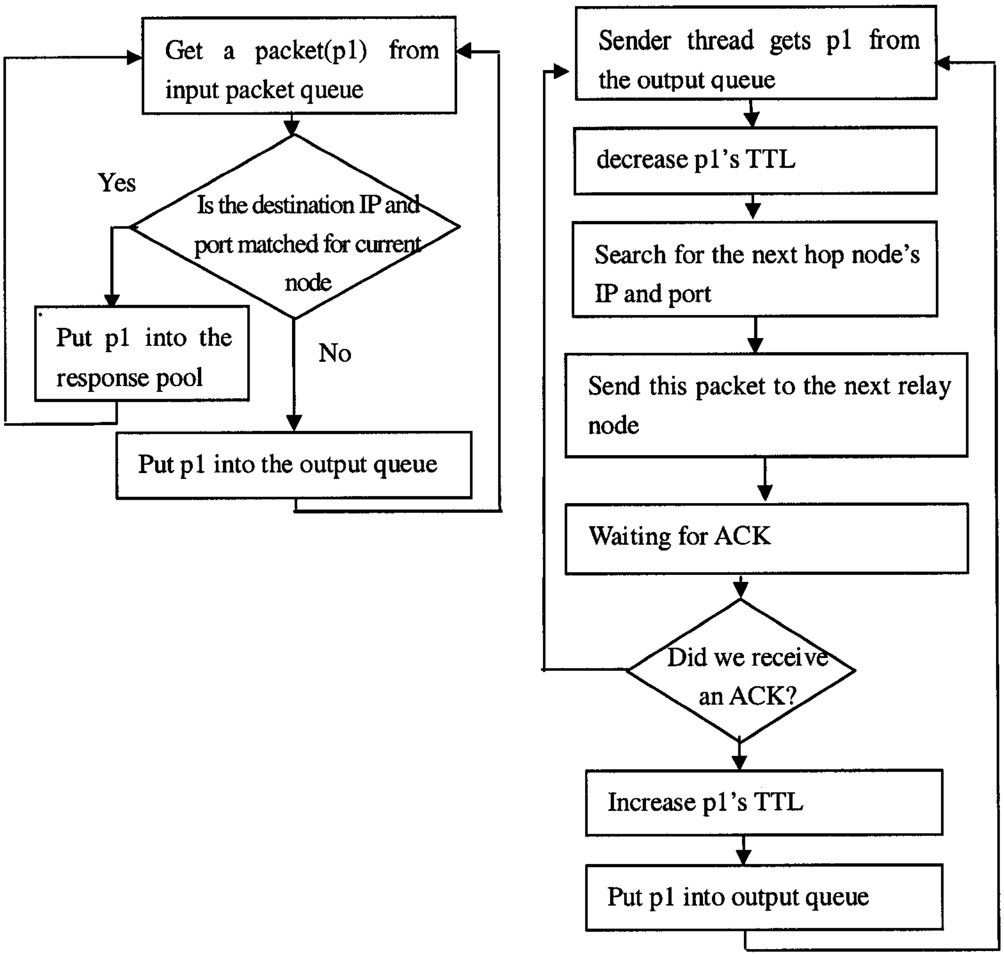

Get a packet(p1) from input packet queue
Is the destination IP and port matched for current node
Yes
No
Put p1 into the response pool
Put p1 into the output queue
Sender thread gets p1 from the output queue
decrease p1's TTL
Search for the next hop node's IP and port
Send this packet to the next relay node
Waiting for ACK
Did we receive an ACK?
Increase p1's TTL
Put p1 into output queue

Figure C.11 Flowchart for Unicast Packet Handler