The Design and Simulation of Routing Protocols for Mobile Ad hoc Networks

By

Mieso Denko Kabeto

BSc., Addis Ababa University, Ethiopia, 1988
MSc., University of Wales, United Kingdom, 1994

Submitted in fulfilment of the academic
requirements for the degree of
Doctor of Philosophy
in the
Department of Computer Science
University of Natal, Durban

December 2000

Dedicated to the Memory of My Father

Denko Kabeto

1948–1999

Preface

The work described in this thesis was carried out in the Department of Computer Science, University of Natal, Durban, from September 1997 to February 2000, under the supervision of Professor Wayne Goddard.

This thesis represents original work by the author and has not otherwise been submitted in any form for any degree or diploma to any other tertiary institution. Where use has been made of the work of others it is duly acknowledged.

Acknowledgements

My greatest debt is to professor Wayne Goddard for his dedicated supervision, support and kindness during this research work. He motivated and inspired me during every stage of my research. He also sponsored me to participate in conferences where I presented some of my work.

I am grateful to Dr. Ismail Ikram, Dr. Hugh Murrell, Prof. Alan G. Sartori-Angus and Mrs. Jane Meyerowitz of the Department of Computer Science for their support and encouragement during my studies at the University of Natal.

I would like to thank the German Academic Exchange Service (DAAD) for sponsoring my studies and the University of Natal for granting me a Graduate Award during the 1998 and 1999 academic years.

I would also like to thank my wife, Hana Elimo, whose encouragement and love has always supported me, and my son, Naol Mieso, for his patience during the write-up process.

Also, I wish to thank my mother Jehi Tuffa and my brother Kedir Denko whose faithful support and encouragement continued over the years of my studies.

Contents

1 Introduction 1

1.1 Mobile Computing and Networking 1

1.2 Mobile Ad hoc Networking and the Routing Problem 2

1.3 Research Overview and Contributions 3

1.3.1 Limited flooding protocols 3

1.3.2 Hierarchical clustering architecture 3

1.4 Organisation of the Thesis 4

2 Mobile Networks 5

2.1 Mobile Computing Environment 5

2.2 Infrastructure-based Mobile Networks 6

2.2.1 Cellular networks 7

2.2.2 Wireless Local Area Networks (WLANs) 8

2.2.3 Wireless Wide Area Networks (WWANs) 8

2.2.4 Paging networks 8

2.2.5 Satellite networks 9

2.3 Infrastructureless Mobile Networks 9

2.3.1 Packet radio networks 9

2.3.2 Mobile ad hoc networks 10

2.4 Challenges and Tasks in Mobile Networks 10

2.4.1 The impact on the conventional layered model 11

2.4.2 Network access and wireless communication 12

2.4.3 Mobility related problems in mobile networks 13

2.4.4 Mobility management 15

2.4.5 Approaches to handle node mobility 16

2.5 Mobility and the Internet 18

2.5.1 Mobile IP 19
2.5.2 Cellular IP 20

3 Mobile Ad hoc Networks **22**
3.1 Description of Mobile Ad hoc Networks 22
3.2 Applications of Mobile Ad hoc Networks 24
3.3 Challenges 25
3.3.1 Mobile ad hoc network routing protocols 26
3.3.2 Mobility management 26
3.3.3 Medium Access Control protocols 27
3.3.4 Data management 27
3.4 Mobile Ad hoc Networks and the Internet 28

4 Routing Problems in Networks **30**
4.1 Routing in Traditional Static Networks 30
4.1.1 Static routing protocols 31
4.1.2 Adaptive routing protocols 31
4.1.3 Distance Vector and Link State Algorithms 33
4.2 Routing in Mobile Ad hoc Networks 35
4.2.1 Proactive routing protocols 35
4.2.2 Reactive routing protocols 39
4.2.3 Hybrid routing protocols 43
4.2.4 Some comparisons of mobile ad hoc routing protocols . 43
4.2.5 Use of clustering 44

5 Limited Flooding for Mobile Ad hoc Networks **46**
5.1 Flooding 46
5.1.1 Broadcast and point-to-point networks 46
5.1.2 The basic flooding algorithm 47
5.1.3 Flooding control mechanisms 47
5.2 Description of Limited Flooding Protocols 48
5.2.1 Motivation 48
5.2.2 Protocol overview 49
5.2.3 Variations 50
5.2.4 Protocol applicability and assumptions 51
5.3 Related Protocols 52

6 Simulation Environment for Limited Flooding 54

6.1 An Overview of the Simulation Environment 54

6.1.1 The Network Model 54

6.1.2 Assumptions 55

6.2 The Simulation Language 55

6.3 The Simulation Entities 56

6.3.1 The mobility model 56

6.3.2 The Mobility entity 57

6.3.3 The Communication entity 57

6.3.4 The Node entity 58

6.3.5 The Channel entity 58

6.3.6 Simulation parameters 59

6.4 Discussion of the Results 59

6.4.1 Performance metrics 60

6.4.2 Deterministic Limited Flooding 61

6.4.3 Prioritised Limited Flooding 64

6.4.4 Randomised Limited Flooding 67

6.4.5 Comparisons of limited flooding: Randomised paths . . 71

6.4.6 Comparisons of limited flooding: Likely paths 74

6.4.7 Comparisons of all protocols 77

6.5 Summary and Conclusions 80

7 Hybrid Cluster-based Hierarchical Routing 81

7.1 Benefits of Hierarchical Clustering 81

7.2 Description of a hybrid Hierarchical Cluster-based Routing (HCR) 82

7.2.1 Definition of some terminology 83

7.2.2 Routing overview 84

7.2.3 Assumptions 84

7.2.4 The main operations of the protocol 85

7.3 Intra-cluster Routing Mechanisms 87

7.3.1 Route computation at nodes 87

7.3.2 Route computation at clusterheads 88

7.3.3 Routing on demand 89

7.4 Inter-cluster Routing 89
7.4.1 Packet forwarding 91
7.4.2 Multiple paths for inter-cluster routing 91

8 Simulation of Clustering Algorithms 93
8.1 Cluster Computation 93
8.2 Proposals 95
8.3 Simulation 98
8.3.1 Performance metrics 99
8.4 Discussion of the Simulation Results 99
8.4.1 Lowest ID Clustering algorithm(LID) 100
8.4.2 Node Mobility-based Clustering algorithm (NMC) . . . 104
8.4.3 Signal Strength-based Clustering algorithm (SSC) . . 108
8.4.4 Comparison between the clustering algorithms 112
8.5 Summary and Conclusions 115

9 Conclusions and Future Work 116
9.1 Summary and Conclusions 116
9.1.1 Limited flooding protocols 116
9.1.2 Hierarchical clustering architecture 117
9.2 Future Work 118

10 Bibliography 119

A Abbreviations and Terminology 130

B Limited Flooding Algorithms Source Code 132

C Clustering Algorithms Source Code 145

List of Figures

2.1 Mobile Computing: layered approach 5
2.2 Mobile Computing: technology components 6
2.3 Mobile IP entities 20

3.1 Topology changes in mobile ad hoc networks 24
3.2 Mobile ad hoc network and the Internet 28
3.3 Mobile Internet layers 29

4.1 Routing protocol classifications 36

6.1 Simulation entities and their relationships 58
6.2 Packets relayed: N = 15 62
6.3 Channel utilisation: N = 15 62
6.4 Normalised packet delivery ratio: N = 15 63
6.5 Packets relayed: Tx = 30% 63
6.6 Channel utilisation: Tx = 30%. 63
6.7 Normalised packet delivery ratio: Tx = 30%. 64
6.8 Packets relayed: N = 15 65
6.9 Channel utilisation: N = 15 65
6.10 Normalised packet delivery ratio: N = 15 66
6.11 Packets relayed: Tx = 40%. 66
6.12 Normalised packet delivery ratio: Tx = 40%. 66
6.13 Channel utilisation: Tx = 40%. 67
6.14 Packets relayed: N = 20 69
6.15 Channel utilisation: N =20 69
6.16 Normalised packet delivery ratio: N = 20 69
6.17 Packets relayed: Tx = 30%. 70

6.18 Channel utilisation: Tx = 30%. 70
6.19 Normalised packet delivery ratio: Tx = 30%. 70
6.20 Packets relayed: Tx = 40% 72
6.21 Channel utilisation: Tx = 40% 72
6.22 Normalised packet delivery ratio: Tx = 40% 72
6.23 Packets relayed: N = 10 73
6.24 Channel utilisation: N = 10 73
6.25 Normalised packet delivery ratio: N = 10 73
6.26 Packets relayed: Tx = 30% 75
6.27 Channel utilisation: Tx = 30% 75
6.28 Normalised packet delivery ratio : Tx = 30% 75
6.29 Packets relayed: N = 10 76
6.30 Channel utilisation: N = 10 76
6.31 Normalised packet delivery ratio: N = 10 76
6.32 Packets relayed: Tx = 40% 78
6.33 Channel utilisation: Tx = 40% 78
6.34 Packet Delivery ratio: Tx = 40% 79
6.35 Packets relayed: N = 10 79
6.36 Channel utilisation: N = 10 79
6.37 Normalised packet delivery ratio: N = 10 80

7.1 Network Architecture 83
7.2 Clusterhead Border Node Routing 91

8.1 Topology Before and After Clustering 95
8.2 Clustering Metrics: N = 10 101
8.3 Number of Clusters: N = 10, 20 101
8.4 Clusterhead Changes: N = 10, 20 101
8.5 Cluster Membership Changes: N = 10, 20 102
8.6 Clustering Metrics P = 20 102
8.7 Number of Clusters: P = 10, 20, 50 102
8.8 Clusterhead Changes: P = 10, 20, 50 103
8.9 Cluster Membership Changes: P = 10, 20, 50 103
8.10 Clustering Metrics: N = 20 105

8.11 Number of Clusters: N = 5, 10, 20 105
8.12 Clusterhead Changes: N = 5, 10, 20 105
8.13 Cluster Membership Changes: N =5, 10, 20 106
8.14 Clustering Metrics P = 20 106
8.15 Number of Clusters: P = 10, 20, 50 106
8.16 Clusterhead Changes: P = 10, 20, 50 107
8.17 Cluster Membership Changes: P = 10, 20, 50 107
8.18 Clustering Metrics: N = 10 109
8.19 Number of Clusters: N = 5, 10 109
8.20 Clusterhead Changes: N = 5, 10 109
8.21 Cluster Membership Changes: N = 5, 10 110
8.22 Clustering Metrics: P = 10 110
8.23 Number of Clusters: P = 10, 20, 50 110
8.24 Clusterhead Changes: P = 10, 20, 50 111
8.25 Cluster Membership Changes: P = 10, 20, 50 111
8.26 Number of clusters: N = 10 113
8.27 Clusterhead Changes: N = 10 113
8.28 Cluster Membership Changes: N = 10 113
8.29 Number of Clusters: P = 20 114
8.30 Clusterhead Changes: P = 20 114
8.31 Cluster Membership Changes: P = 20 114

simulation experiment. The simulation results have confirmed that the proposed routing schemes and clustering algorithms are suitable for the intended application environments.

Abstract

This thesis addresses a novel type of network known as a mobile ad hoc network. A mobile ad hoc network is a collection of entirely mobile nodes that can establish communication in the absence of any fixed infrastructure. Envisioned applications of these networks include virtual classrooms, emergency relief operations, military tactical communications, sensor networks and community networking.

Mobile ad hoc networking poses several new challenges in the design of network protocols. This thesis focuses on the routing problem. The main challenges in the design of a routing protocol for mobile ad hoc networks result from them having limited resources and there being frequent topological changes that occur unpredictably. Moreover, there is no fixed infrastructure that supports routing. The conventional routing protocols are not generally suitable for mobile ad hoc networks, as they cannot react quickly to the changing network topology, cause excessive communication and computation, or converge very slowly creating routing loops.

In this thesis we propose two classes of routing schemes for mobile ad hoc networks. The first class is known as Limited Flooding Protocol. The protocol is fully reactive and does not require the computation of routing tables. It uses some basic principles of flooding, but reduces the communication overhead by restricting packet propagation through the network. Several variations of limited flooding are considered including deterministic, randomised and priority-based mechanisms. The main advantage of this protocol is that it can be used in networks with unpredictable topological changes and highly mobile nodes, since maintaining routing table at the intermediate nodes is not required.

The second class of routing protocols is based on hierarchical clustering architecture and is intended for use in a relatively low mobility environment. The basic idea of this protocol is to partition the entire network into smaller units known as clusters and define routing mechanisms both within and between clusters using a hierarchical architecture. The main advantage of this architecture is reduction of storage requirements of routing information, communication overhead and computational overhead at each node.

Discrete-event simulation is used for modelling and performance evaluation. Various options and variations of the protocols are examined in the

Chapter 1

Introduction

1.1 Mobile Computing and Networking

Recent advances in wireless communication technology and affordable mobile computing and communication devices have brought about new requirements for computing and communication networks. Various portable computing devices such as laptop, notebook and palmtop computers are commercially available which allow one to be mobile while communicating and to communicate while mobile. The communication capability of these portable computers is increasing with network access ranging from high speed modems to gigabit satellite access. These technological advances together with users' demands for anytime and anywhere information access introduced the era of mobile computing and networking, along with new challenges.

Mobile networks can be classified into infrastructure-based networks and infrastructureless networks. The conventional cellular wireless networks are based on the existing wireline infrastructure and are examples of infrastructure-based networks. Mobile ad hoc networks require no infrastructure (wired or base station) and are examples of infrastructureless networks.

A mobile computing system consists of mobile computing terminals, workstations, PDAs etc., each participating in a Wireless Local Area Network or a Wireless Wide Area Network. Each of these devices and terminals can also be connected to fixed networks including the Internet or to other mobile networks. Thus mobile computing and networking can be viewed as a combination of three components: wireless communication, mobility and computational power.

Wireless communication. Communication within a network is common to both fixed and mobile networks. But mobile networks have an additional

communication component, the wireless link, which does not exist in the traditional fixed networks. This has a bandwidth constraint due to the limitation of the available spectrum, which is regulated by authorities such as the Federal Communication Commission in the USA.

Mobility. Mobility is another component of mobile networks that does not exist in the traditional fixed networks. It is the ability to send and receive messages regardless of the location of the sender or receiver. In a mobile network environment, both the source and destination devices, application and people can be mobile. The essence of the nomadic environment is to integrate and adjust all aspects of computing, communications and storage functionality transparently [Bag95]. Transparency in mobility refers to user or location, available bandwidth, computing platform, or user's current status while in active communication. Mobility scenarios range from low mobility applications such as a wireless LAN in an office environment to high mobility applications such as mobile nodes moving at vehicular speed.

Computational power. Computation exists both in fixed and mobile networks. It includes both hardware and software aspects. Laptop and notebook computers currently provide about the same services as their desktop counterparts, but they have limited computational power due to limited battery power.

1.2 Mobile Ad hoc Networking and the Routing Problem

Mobile ad hoc networks are formed by a group of fully mobile nodes that communicate over wireless channel. They have no central co-ordinating mechanism for packet routing or mobility management. In mobile ad hoc networks, two nodes can communicate directly while within the communication range of each other, and rely on intermediate nodes for forwarding packets otherwise. Thus, each mobile node functions not only as a host but also as a router.

One of the main challenges in mobile ad hoc networks is routing packets to the mobile nodes. Due to mobility and wireless transmission characteristics, conventional routing protocols cannot be used directly in mobile ad hoc networks. Several routing protocols have been proposed for these networks. These protocols are based on either reactive or proactive algorithms. In reactive algorithms, routes are established when needed using a route discovery mechanism. Route update overhead is low at the expense of possibly stale routes. In proactive protocols, routes are pre-computed using shortest path

algorithms and are available for use when needed. However, these protocols generate route updates whenever a node moves thereby creating high route update overhead. A combination of both approaches is the least investigated.

1.3 Research Overview and Contributions

Most proposed routing protocols are fully distributed with a flat routing architecture. In these protocols, each node maintains a complete routing table to each possible destination. Flat routing has a good performance for relatively small number of nodes but incurs high overhead as number of nodes increase.

Though no commercial products are available yet, possible applications of mobile ad hoc networks include scenarios requiring low to high mobility ranges. No single routing protocol can "fit" all applications. Thus, new routing protocols and routing architectures are proposed in this thesis.

1.3.1 Limited flooding protocols

When nodes are highly mobile, neither proactive nor reactive protocols nor their combination are suitable for mobile ad hoc networks. Proactive protocols cannot cope with topological changes while the explicit routes discovered in reactive protocols might be unusable due to unpredictable mobility. Hence, the use of some form of flooding is proposed in this network environment. We propose a class of routing protocols known as Limited Flooding Protocols [Den99]. These protocols are discussed in chapter 5 and simulation results comparing them with pure flooding are presented in chapter 6.

1.3.2 Hierarchical clustering architecture

The use of a hybrid of routing on-demand and shortest-path algorithms together with a clustering architecture can reduce routing overhead and increase efficiency. We propose hybrid routing protocols based on hierarchical clustering architecture [Den98] to achieve better performance. The protocols overcome routing and communication overhead through clustering and address scalability through the use of a hierarchical architecture. The detailed discussions on intra-cluster and inter-cluster routing mechanisms are presented in chapter 7. We propose two clustering algorithms [Den01] for clustering the ad hoc network. Simulation results comparing the suitability of the clustering algorithms are presented in chapter 8.

1.4 Organisation of the Thesis

Chapter 2 provides a general discussion about mobile networks including the new challenges that they introduce to traditional computing and networking. Chapter 3 introduces mobile ad hoc networks and discusses their benefits and possible application environment. Chapter 4 discusses the routing problems in computer networks in general and in mobile ad hoc networks in particular. Here, a brief discussion of the recently proposed routing protocols for mobile ad hoc networks is presented.

Chapter 5 through 8 discuss our proposals for the routing problem. Chapter 5 presents *Limited Flooding Protocols* and various forms of limited flooding protocols are investigated. Chapter 6 discusses the simulation environment for these protocols and presents results of the simulation experiments. Chapter 7 presents another class of routing protocols known as *Hierarchical Cluster Routing*. The inter-cluster and intra-cluster routing mechanisms are discussed there. Chapter 8 discusses clustering mechanisms and a simulation environment for clustering architecture and presents the simulation results.

Chapter 9 presents conclusions and directions for future research. Chapter 10 presents the bibliography of the consulted reference materials. Finally, supplementary information is included in the appendices.

Chapter 2

Mobile Networks

2.1 Mobile Computing Environment

The mobile computing environment can be organised into a four-layered architecture as shown in figure 2.1 [Day97]. In the layered approach, the wireless networks layer consists of wireless LANs for local area wireless network services and wireless WANs for wide area wireless services. The devices layer consists of mobile computing devices that can be carried (notebooks or notepads), kept in a pocket (palmtops, cellular phones or PDAs) and worn (smart badges and watch-sized pagers).

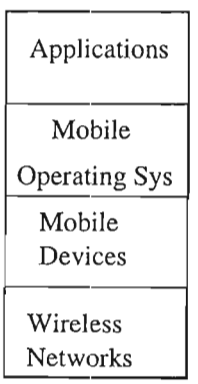

Figure 2.1: Mobile Computing: layered approach

The mobile operating systems level provides tools for application programmers to access different mobile computing devices and different wireless networks. Mobile operating systems are useful for the development of mobile applications such as mobile agents and mobile application interfaces. Such

operating systems must consume less energy, as mobile devices are portable and constrained by energy requirements.

In the applications layer, two types of applications can be provided: Horizontal applications and vertical applications. Horizontal applications have wider users across most markets whereas vertical applications address functions associated with a small market segment. Applications such as messaging, fax & email services, network and database access are examples of horizontal applications. Specific market segment applications such as telemetry, point-of-sale, airline systems, emergency services and hospitals are examples of vertical applications.

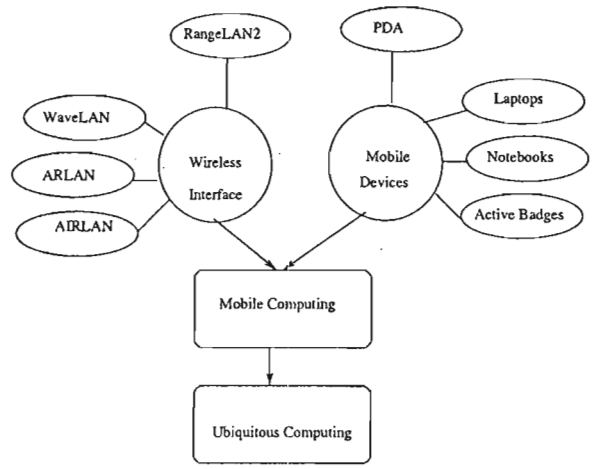

Figure 2.2: Mobile Computing: technology components

Another approach to mobile computing is to view it as a combination of various technologies as shown in figure 2.2. The future of computing and communication will bring Ubiquitous Computing [Wei91] into reality. We are already witnessing commercial availability of devices that integrate the functions of a personal computer, a PDA and a wireless phone. Such integration will decrease the degree of dependence on infrastructure by enabling networking and communications to be achieved ubiquitously and easily.

2.2 Infrastructure-based Mobile Networks

Mobile networks can be classified into two categories: infrastructure-based and infrastructureless networks. In this section we discuss the former.

Most infrastructure-based networks are two-tiered networks consisting of a wireline backbone and wireless networks. The wireline backbone is the conventional network that consists of static hosts known as Mobile Support

Stations (MSS) and communication links between the mobile nodes. The MSS have wireless interfaces and act as a gateway between the wireline and wireless networks. The wireless end consists of mobile units that are capable of moving within the range of wireless radio communication links.

Mobile nodes communicate with an entity within its communication range called a **base station**. They can move within or between base stations while communicating. When a mobile node moves out of range of one base station, it connects with a new base station and starts communicating through it. This process is called hand-off. Infrastructure-based networks can consist of heterogeneous networks that are based on different standards, bandwidth and application services. They can provide both local and wide area network services.

In general, an infrastructure-based network can have mobile or static end-points known as nodes, and mobile or static relaying nodes known as routers. The mobile nodes act as a source or destination for communication while the routers act as gateways. Examples include cellular networks, wireless local areas networks, wireless wide area networks, paging systems and satellite networks.

2.2.1 Cellular networks

Cellular networks are organized into small geographical areas called **cells** each with a particular radio frequency coverage. The main reason for using a cell is that it allows frequency reuse. This increases the system capacity which is much constrained by the available bandwidth.

A cellular network consists of mobile nodes, base stations and Mobile Switching Centres (MSC). A base station is located at the centre of a cell and services the mobile nodes in the cell. Base stations are connected to the mobile switching centres. The mobile switching centres act as an interface between base stations and backbone networks and they are connected to Public Switched Telephone Networks or Integrated Systems Digital Networks. The main design problem in cellular networks is partitioning the coverage area into cells, microcells or picocells [Cox92] and associating base stations with the MSCs.

Another problem in cellular networks, as in most mobile networks, is how to keep track of the location of the mobile node. In cellular systems, mobility management is handled by Home Location Registers (HLRs) and Visiting Location Registers (VLRs) located on the fixed network. The HLR contains the permanent subscriber parameters for a group of subscribers and a pointer to the VLR for each mobile unit. The VLR is a dynamic database whose entries change as mobile units move. Each MSC has an attached VLR

containing information about all the mobile units in the coverage area of the MSC. The time when, or the location where update occurs, depends on the mobility management strategy.

2.2.2 Wireless Local Area Networks (WLANs)

These are the traditional LANs extended to support wireless communication through wireless interfaces. They differ in the data rates they provide, the transmission technology they use (infrared, narrow band RF or Spread Spectrum RF) and the communication ranges they support.

The WLAN can function independently or be connected to conventional wireline networks. WLANs that are connected to the wireline networks require Mobile Access Stations which act as a router between the mobile nodes and fixed station on the wireline network. Independent WLANs are standalone networks that can be constructed in places where there is no wired infrastructure for communication. These WLANs can be configured using the standard LAN topologies viz. bus, ring or star.

There are several existing and future application scenarios for WLAN. First, WLANs can be used when limited mobility support is needed. One can roam within the coverage area of the installed wireless LAN interface such as a building and gain network access using a notebook computer. Second, WLANs can provide temporary network services in temporary classrooms, a conference or disaster recovery. Third, WLANs can be constructed where building wired networks is not possible due to cost, geographical location or being a historically significant building.

2.2.3 Wireless Wide Area Networks (WWANs)

Wireless Wide Area Networks cover relatively larger geographical area providing wireless data services. These services include RMD, ARDIS, CDPD Networks, satellite networks and paging services. These wide area data services require lower data rates than wireless Local Area Networks. The RMD and ARDIS networks support applications such as remote host access, dispatch and field services. CDPD uses digital transmission on an analog cellular channel to provide mobile packet-switched data services. It utilises the unused voice capacity for data communication.

2.2.4 Paging networks

The emerging PCN services and paging systems are based on a cellular type of infrastructure. Paging networks are low-cost, long-battery-life networks that provide alert messaging services. Most paging networks broadcast the

page to all local controllers in an attempt to find the user. Current paging supports two-way asymmetrical and symmetrical messaging services. Two-way asymmetrical paging networks need two separate networks, one for each direction. Two-way symmetrical pagers are similar to mobile ad hoc networks. With respect to costs, one-way paging services are much cheaper than two-way paging services. The former provides best-effort service with no acknowledgement service capability. The future expectation of paging is the development of audio-based pager over narrow-band frequency networks.

2.2.5 Satellite networks

Satellite networks provide wide area network coverage including remote areas where packet radio is uneconomical. Satellites can offer land-based applications such as positioning services for locating vehicle fleets on the ground. Satellite networks differ from cellular systems in that the end points are fixed whereas the routers are mobile. Unlike mobile ad hoc networks, the topology of satellite networks is predictable. The challenge in satellite networks is the design of efficient topological layout including the parameters such as number of satellites, their positions, coverage areas etc., [Gru91].

2.3 Infrastructureless Mobile Networks

Infrastructureless networks are collections of mobile nodes that can communicate over wireless links without the use of any wireline or wireless network infrastructure. In these networks both the mobile nodes (hosts) and the switches (routers) can be mobile. Examples of infrastructureless networks are mobile ad hoc networks and the traditional packet radio networks.

2.3.1 Packet radio networks

Interest in dynamic networks dates back to the 1970's when the U.S. Defense Advanced Research Projects Agency (DARPA) began to work on Packet Radio Network (PRNET) [Jub87, Kah78] and Survival Adaptive Networks (SURAN) [Sch87] projects.

The PRNET network system consists of terminals with attached packet radios, repeaters (routers), and stations that provide centralized administration. All the system components can be mobile. PRNET supports automatic route set up and maintenance of packet switched communication routes in a network with moderate mobility. SURAN extended the PRNET technology to a large hierarchical network with the first hierarchy consisting of packet

radios organised into clusters. The second and third levels consist of network and superclusters respectively.

Packet radio networks support both broadcast and point-to-point routing. The former is used for a slow moving node and the latter is for a fast moving node. In broadcast routing, packets are flooded through the entire network with the intermediate nodes storing already seen packets so that they cannot be forwarded again. The advantages of broadcast routing is that it relieves stations from computing continuously changing topology information.

In point-to-point routing, stations maintain all the routing information. They know about all radios in the network, they compute network topology and all the routing information, and they distribute the routing information to repeaters or source packet radios. Repeaters perform store-and-forward packet routing. Each node has a lookup table to forward packets. Routing in PRNET uses a form of distance vector routing, with each node broadcasting a routing update packet every 7.5 seconds. The data link protocol is based on hop-by-hop acknowledgement using active and passive acknowledgements from the received packets.

2.3.2 Mobile ad hoc networks

Mobile ad hoc networks are multi-hop dynamic mobile networks that are formed cooperatively by a group of mobile nodes sharing a common transmission channel. Unlike the traditional packet radio networks, which were designed for tactical military communications in a hostile environment, mobile ad hoc networks are intended for use in civilian applications. Mobile ad hoc networks are fully discussed in chapter 3.

2.4 Challenges and Tasks in Mobile Networks

Mobile networks encounter a number of limitations that do not exist in traditional static networks. These challenges emerge from mobility on one hand and the wireless nature of communication on the other hand. This impacts the conventional layered network models which were designed based on fixed network architecture.

The two main issues in mobile networks can be summarised as network access and mobility management. Network access deals with the mechanisms for getting connected to the communication network whereas mobility management deals with keeping track of the mobile node's location so that getting data to the mobile is done efficiently. In this section, we consider the challenges of mobile networks and their impact.

2.4.1 The impact on the conventional layered model

The existing communications protocols in OSI and the Internet were developed prior to the emergence of mobile networking. Hence, neither the Internet protocol suite TCP/IP nor the ISO/OSI layered model supports mobility. In both, the end points are assumed to be stationary. But in mobile networks, nodes can change the point of their network attachment from time to time. Thus mobility affects the decision for protocol development at various layers of the Internet network protocol stack.

The Physical layer. At the physical layer, two directly communicating entities interface with each other. The main design problem at this layer is the hardware and electromechanical characteristic of the medium between the communicating entities. Due to mobility, the characteristics of a communication channel varies not only with the location of the user but also with time. Signal propagation mechanisms, such as multipath fading, attenuation and shadowing by obstacles can hinder radio channels for mobile communications. While the traditional wireless applications operate at high power, the emerging mobile communications devices are light-weight, small and inexpensive requiring low-power technology suitable for data communications over air link.

The Data Link Layer. The main function of the link layer is to reliably transmit a bit stream between two directly connected layer-two entities. In wireless media, transmitter power level is usually high and not all transmitting stations can detect the occurrence of collisions during transmission. Due to this, the Ethernet-like Carrier Sense Multiple Access/Collision Detection (CSMA/CD) cannot be used as a medium access protocol. Moreover, token-passing-based access protocols are also inappropriate since a mobile station can get connected to or disconnected from the network at any time.

The Network Layer. The network layer deals with communication between two entities which are not directly connected. The key assumption in network protocols such as IP [Pos81] is that the end systems or hosts are stationary and can be reached at static network addresses. In mobile networks, network access points may change as the hosts move from one network to another. The network layer defines mechanisms for routing and forwarding packets from source to the final destination. These include naming and addressing schemes for network entities and a mechanism for routing packets in the network. We discuss this below under mobility management.

The Transport Layer. The transport layer provides end-to-end communication between the two communicating entities without the aid of any intermediate layer. Node mobility and wirelessness should be transparent

to the transport layer. But this is possible only at the cost of throughput degradation [Bak95]. Node mobility and the use of wireless channels for communication violate some basic design assumptions in transport protocols. For example, packet losses in wireless communication links are interpreted by the transport protocols as congestion. The transport layer protocols then react to this by retransmitting using an exponential back-off and reduce the subsequent throughput, which reduces the efficiency of the wireless link. New mechanisms are needed to avoid breaking connections on changes in network access points.

The Application Layer. The services are directly accessible to an application via Application Program Interfaces (API). The impact of node mobility and the wireless channel on the application layer depends on whether the applications perform well when used in a mobile environment. It also depends on whether the client-sever model is the right choice for developing mobile applications. Unfortunately, mobile computers and hand-held communication devices have limited resources and do not operate well in the presence of mobility. Hence, new software or hardware systems may be needed.

2.4.2 Network access and wireless communication

Mobile network access is the first stage in getting mobile network services. Network access is gained through wireless interfaces though mobile networks use different mechanisms. For example, in the CDPD system, mobile nodes attach to the network infrastructure via radio frequency channel. CDPD uses different modulation techniques to enable digital transmission that provides higher data rates than cellular systems. CDPD adopts the LAN mode of operation by using a shared media. Hence, unlike cellular networks, a radio frequency channel is not dedicated to a call for its duration.

The use of wireless channels for communication results in lower network performance due to limited resources and wireless transmission characteristics. Some constraints are discussed below.

1. *Low bandwidth.* The bandwidth available for a wireless medium is much lower than the wired medium due to limited spectrum availability. While the Ethernet LAN can provide up to 10 Mbps, FDDI 100 Mbps and ATM typically about 155 Mbps, most wireless transmission media generally provide less than 2 Mbps. However, recent advances in laser communication technology has resulted in the availability of wireless laser communication systems that can support the wireline data rates.

2. *High link error rates.* Wireless communication is easily affected by channel interference, multipath fading and direction of propagation.

So data communication over wireless media has a higher error rate than over wired media. The bit error rate for wired links is 10^{-10} to 10^{-12} and 10^{-2} to 10^{-6} for wireless links [Kri96].

3. *Frequent disconnection.* Disconnected operation is the mode of operation where a client continues to have read and write access to data in its cache during temporary network disconnection [Sat93]. A mobile node is very susceptible to disconnection from the network. But disconnection must be treated as a planned failure, and different from system failures, though both increase latency. To handle disconnection, protocol designers should either spend available resources on the network to reduce disconnection or design mechanisms to cope with it.

4. *Limited available power.* Storage space on mobile devices is limited by physical size and power requirements. Batteries are the main source of weight in mobile devices. Battery size can be reduced but this may result in frequent recharging, replacing or reducing the frequency of using the portable devices, which undermines portability.

5. *Heterogeneous networks.* The mobile networks may consist of heterogeneous networks using various wireless technologies each possibly with different bandwidth. It may even be connected to a wireline backbone having heterogeneous networks. Applications may assume a high bandwidth link and operate only when plugged in, or assume low bandwidth and fail to utilize a high bandwidth when available. This heterogeneity makes mobile networks more complex than conventional networks.

6. *Security risks.* Mobile devices are liable to physical security problems such as equipment theft, data loss or change due to interference on shared transmission media and access by unauthorised personnel. Authentication and data encryption techniques designed for fixed networks are not appropriate since they require multiple exchange of information. Nevertheless, LAN networks have started incorporating interface cards that provide encryption mechanisms for both the entire traffic on wireless LANs and for specific applications.

2.4.3 Mobility related problems in mobile networks

Mobility brings new challenges to data networking. These include address migration, name-to-address mapping, location-to-topology mapping, store-and-forward capability, location-sensitive network resources, application transparency and scalability.

1. *Address migration.* As a mobile node moves from one location to another location, it may end up with a different network access point or address. The mobile node should be able to receive data at its new location. The mechanisms proposed for getting data to the mobile node can be grouped into three main categories. In informed routing, the mobile node transmits its current location to the correspondent hosts so that they can directly send it data. In triangle routing, the correspondent host sends data to a home location and the home router forwards it to the mobile node. The mobile notifies its current location to the home location. In broadcast routing, a packet is sent to all nodes in the network with the assumption that the destination node will receive the packet with minimum delay. Each technique has advantages and disadvantages in terms of the processing and routing overhead.

2. *Location-sensitive network resources.* Resources such as bandwidth, communication latency, program and data availability, network and service availability, peripheral devices access, etc., are location-sensitive [Liu95]. Thus mechanisms for location-sensitive information management are needed. There is a trade-off between pre-allocating resources (via a prediction mechanism) and determining them only when they are needed. Given the resource constraints in mobile networks, the combined strategies that involve resource prefetching and caching may be useful.

3. *Location-to-topology mapping.* A small movement of a mobile node can result in crossing a network administration boundary. One challenge posed by mobility is the mapping of geographic coordinates to network addresses. Since a mobile node should be able to communicate with the other host from anywhere, there should be a mapping of the geographic location of a mobile node and the network connectivity either by the network system or the communicating nodes.

4. *Name-to-address mapping.* In the Internet, the DNS stores name-to-address mappings in a distributed data structure [Tan96]. A host address is found by making a directory lookup operation and this name-to-address binding is assumed to be static throughout the network connection session. For a mobile node, the address may change as it moves and thus name-to-address mapping does not function well.

5. *Scalability.* Mobility in wide area networks should be scalable to support a million hosts. But, given the limited computing power of mobile nodes, existing routing protocols such as RIP [Hed88] and OSPF [Moy94a] may not scale well due to routing overhead in mobile nodes.

6. *Store-and-forward capability.* Mobile networks are characterised by occasionally connected nodes and hence efficient and timely packet delivery is hardly possible. Continuous pooling of unavailable host stretches scarce resources. Instead, the capability for efficient storing and forwarding of data to the corresponding application entities is needed. Since network connectivity is a network layer function and correspondence between communicating entities is handled at application layer, a store-and-forward mechanism that bridges both layers is necessary.

7. *Application transparency.* In static networks, packets are sent based on the fixed address on the destination network. In mobile networks there is no fixed address for a destination host since it can be anywhere. But mobility of a node should be transparent to both the applications running on it and other hosts with which it communicates. It has been shown in [Ioa93] that even if the network layer hides the addressing aspect of mobility from higher layers, applications may function differently as the service provided by the network link changes due to granularity of communication. Hence some mechanisms must be designed at lower layers including the Medium Access and Logical Link Control sub-layers to improve network performance.

2.4.4 Mobility management

Mobility management is the set of mechanisms by which location information is gathered, updated and disseminated in the presence of mobility. There are two distinct tasks common to all mobility management strategies in mobile networks. The first is finding the location of the mobile node at any time. The second is routing location information or data packets to the mobile nodes in the network.

Finding the location of the mobile node. There are two strategies used for tracking a mobile node: location updating and location searching [Moa94, Ros96, Bar95]. In location updating, a mobile node registers: that is, the node sends its new location to the network for updating the location database. In location searching, the network pages: that is, the network initiates a location finding request by sending pooling signals in predefined or most likely locations based on the recent update [Bha99]. In either case, location information must be maintained in a database in a manner similar to routing tables for network routing.

There are two categories of location update techniques: static update and dynamic update. In a static strategy, there is a predetermined set of locations at which location updates are carried out. In some networks, the cell

is partitioned into Location Areas and a mobile node updates its location whenever it crosses the boundary [Bar95]. Or a subset of the cells is designated as reporting centres where the mobiles must update their locations [Bar93]. The main drawback of the static update strategy is that it does not really take user mobility into account. For example, excessive update messages can be generated if a user enters and exits from reporting centres frequently.

In a dynamic strategy, location update is based on user's mobility but not on predetermined cell locations. A mobile node decides when the update should be carried out. Three dynamic update schemes can be identified [Tab93, Bar95, Mad95]: distance-based, movement-based and time-based.

In the time-based scheme, the mobile nodes send periodic updates (every T time units) to the network. This results in redundant updates if the mobile node is stationary for a long time. In the distance-based scheme, the mobile nodes send update messages whenever the distance covered exceeds the defined maximum distance threshold D. The updated distance can be computed using the Euclidean distance formula from the location of the previous update. The distance can be specified in terms of units such as kilometer or number of cellls between two locations. *Movement-based* schemes generate update messages whenever a mobile node makes M cell crossings. The mobile node counts the number of cell boundaries crossed and updates when the count exceeds the movement threshold M. In [Bar95], these three dynamic update schemes have been compared in terms of the paging cost with varying update rates to previous one. It has been found that the distance-based scheme consistently performs the best.

Routing information to the mobile node. The other aspect of mobility management is routing location data to the mobile node. This requires first determining routing paths based on the collected location information or location probability distribution [Ros96] and then forwarding the packet along the chosen paths. Thus, location information dissemination is carried out in a manner similar to packet routing and forwarding in the conventional datagram networks. As routing is a network layer problem, mobility management techniques are best handled if integrated at the network layer.

2.4.5 Approaches to handle node mobility

A number of mechanisms have been proposed to handle node mobility [Per96, Ioa91, Bla96, Maa97, Wad93]. These mechanisms are based on the approaches described below.

Application awareness. In this approach, a mobile node informs all peers about its location so that it can receive messages from them. Any peer that does not have this address cannot communicate with the mobile node. The advantage of this approach is that network infrastructure is not aware of the mobility. The main drawback of this approach is that end users and applications must perform routing and this requires the development of new applications. Also, maintaining routing information at the end systems for the mobile nodes violates the functionality of the layered network architecture in addition to being inefficient.

Directory lookup. In this approach, a mobile node informs a mobility directory of its current location. Application peers query the mobility directory before sending messages. This approach is similar to DNS, but for a mobile node, this information gets obsolete very quickly. There is also a high overhead to keep this information up-to-date. The advantage of this method is that a peer application that wishes to communicate with the mobile sends query to the mobility directory and the mobile node only informs the mobility directory about its current location. But the mobility directory requires application participation, as the application must understand mobility to maintain association between the mobile node and the mobility directory.

Mobility mailbox. In this approach, application peers leave messages in a mobility mailbox and the mobile node application retrieves the messages from the mobility mailbox. The peer application does not have to do mobility tracking— it only sends the information to the mobility mailbox. This approach does not support real-time communication and is not appropriate for interactive applications. Also, it requires the modification of both the network and application layer protocols.

Administrative redirection. Administrative redirection is based on the concept of a mobility agent, a system or server on the home network that keeps track of the mobility information about the mobile node. Thus, a mobile node sends information about its current location to a home mobility agent whenever it changes its network connection. The home agent then redirects the packets to the mobile node in real time. The advantages of this approach is that it supports interactive communication between the mobile node and peer application transparently. The peer application is not aware of routing and mobility management during the communication. This approach requires the home mobility agent to know the current location of the mobile node. This approach to mobility handling has been adopted by the existing mobile technologies including the IETF's Mobile IP protocol, Novell's Mobile IPX protocol and the CDPD networks.

2.5 Mobility and the Internet

In the Internet, a datagram is routed to a host based on the network number contained in the host's Internet address. If the destination host changes its current network attachment point, the datagram will no longer be delivered correctly. The existing Internet protocol stack (TCP/IP) does not support host mobility and a new mobile networking infrastructure is needed. To ensure the inter-operability with the existing network infrastructure, mobility should be transparent to the protocols and applications running on the static host. Also, it must be transparent to the transport and higher layers, so that no modifications need to be made to the existing applications. So the aim is providing mobility support features in the existing Internet infrastructure without changing the entire protocol stack. See [Ioa93, John95, Per96].

The Internet uses a two-level hierarchical addressing scheme, with the hosts forming the lower layer and the routers the higher layer. Each host in the Internet is assigned a unique 32-bit IP address which consists of network-ID and host-ID [Com95, Tan96]. The Internet router maintains only the network part of the destination address and forwards the datagram using this address. The destination network router delivers the datagram to the host using the host-ID part. In the absence of mobility this hierarchical routing scheme in the Internet is simple and effective. But the Internet address is valid only when the host remains connected to the network at a fixed point. When it moves to a new network it must obtain a new IP address. Since host name is a location-independent identifier of the host whereas address is its location, the DNS, which performs an address translation service, should be modified to allow changes of network connection point. But, when name-to-address binding fails to function well, the protocols such as TCP become ineffective.

In fact, the real problem is that in the existing network infrastructures such as TCP/IP, ISO/OSI and Novell's IPX, the network address serves as a host identifier for application and transport layers and as a routing directive for the network layer. When a host moves and gets connected to a foreign network, the old address cannot be used to deliver packets. On the other hand, the original host address must be preserved in order that the active transport layer connection be kept [Com95]. The main issue in extending the Internet to have mobility functionality is to resolve this conflict. Several proposals have been made along this line. We discuss mobile IP and cellular IP.

2.5.1 Mobile IP

Mobile IP is the protocol developed by the Mobile IP Working Group of the IETF to support node mobility in the Internet environment [Per96]. It extends the existing Internet protocol to allow a mobile node to move from one point of network connection to another without changing its IP address and without losing network connection. Mobile IP handles the mobility problem at the network layer. It provides transparent and seamless routing mechanisms for mobile nodes. Protocols above the network layer continue to function as if they were on ordinary static networks.

The Mobile IP protocol assigns a unique IP address to the mobile node within its home network as any ordinary Internet host. The entities that form the new network layer architecture in Mobile IP are described below. Figure 2.3 shows these entities.

- *Mobile Node (MN)*: This is a host that moves and can receive network services regardless of its current location.
- *Correspondent Node (CN)*: This is any host that communicates with the mobile node. A correspondent host can itself be mobile or static. Multiple correspondent hosts can also communicate with a mobile node.
- *Home Agent (HA)*: Each mobile node has a home agent on the home network. The HA maintains the location of the mobile node and tunnels a packet to the foreign network which is later delivered to the mobile node. The mobile location information is identified as a care-of address. The link between a mobile node's home address and its current address is called mobility binding and it is updated whenever the mobile node changes its care-of address.
- *Foreign Agent (FA)*: This is a mobile support entity on the foreign network. Often, the FA assigns a care-of address to the mobile node. Alternatively, a mobile node can itself obtain a temporary care-of address through Dynamic Host Configuration Protocol, called a co-located care-of address. The foreign agent also maintains and performs mapping between the care-of address and the home address of the mobile node. A foreign agent decapsulates and forwards packets to the mobile node.

When a node moves, it detects whether it is in its home network or a foreign network. If the latter, it registers with the foreign agent, which provides a care-of address. Then it notifies its home agent of this. In Mobile IP, a correspondent host always sends a packet to the home agent. The home agent encapsulates the packet and sends it to the mobile node's care-of

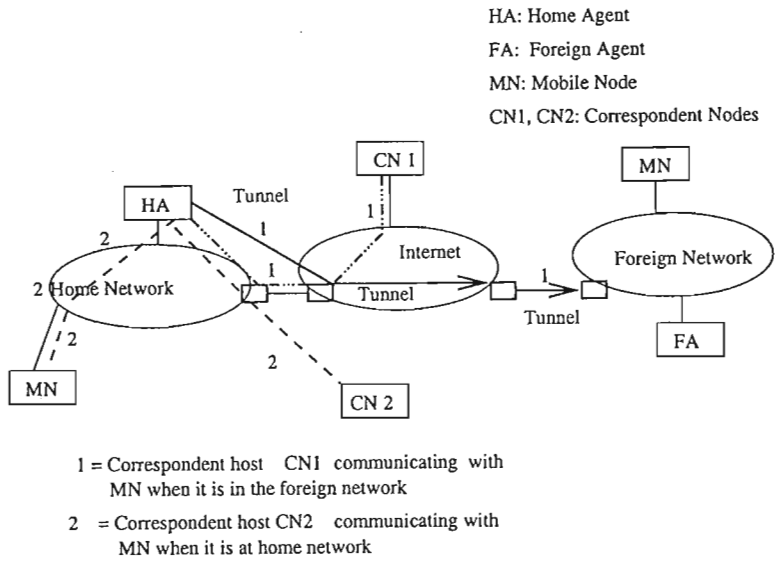

Figure 2.3: Mobile IP entities

address via the foreign agent using tunneling. The foreign agent decapsulates the packet and delivers it to the mobile node.

The assignment of mobility agents is implementation dependent [John95]. The functionality of home agent and foreign agent may be provided by the existing IP routers in the Internet or may be provided by a different host in the network. Also a single node may act both as a home agent for its mobile nodes and as a foreign agent for visiting mobile nodes.

2.5.2 Cellular IP

In Mobile IP, a change of access point during active data communication (sending or receiving) results in a hand-off. During a hand-off, packet losses may occur due to delayed propagation of new location information. So a mechanism is desired that ensures the reliable delivery of packets addressed to a moving node. The cellular IP protocol proposed in [Val99] attempts to provide mobility and hand-off support for frequently moving nodes with high hand-off rate. Cellular IP protocol can be used on a local level, for LANs and MANs, or can internetwork with Mobile IP for wide area mobility support.

Cellular IP routing uses cached mappings information for packet routing. IP packets transmitted by mobile nodes are routed from the base station to the Gateway using regular hop-by-hop shortest path routing. Cellular IP nodes update routing cache mappings by monitoring these passing packets. These map mobile node's IP addresses to Cellular IP node interfaces. Packets addressed to the mobile node are routed along the reverse path based on

cached mappings. Mobile nodes that do not have data to send must send periodic route-update packets to keep their routing cache mappings up to date.

In Cellular IP, hand-off is initiated by the mobile node. As a mobile node moves and approaches a new base station, it redirects its packets from the old base station to the new one. The redirected packets update the routing caches along the way from the new base station to the Gateway. A mobile node entering the service area can immediately start transmitting paging-update packets configuring paging caches. After registration with the new Cellular IP network, the mobile node uses the Gateway's IP address to register with its home agent.

Chapter 3

Mobile Ad hoc Networks

The traditional, packet radio networks were designed for tactical military communications in hostile environments. The advancement in wireless technology and portable computing brought about the demand for civilian networks that support full mobility without depending on any existing wireline infrastructure. Also, the operation of these portable devices in license-free frequencies has accelerated the demand. This has led to the emergence of fully mobile networks. These networks have recently received the attention of both industry and academia. The formation of a new working group named MANET [MAN97] by the IETF is part of this development.

3.1 Description of Mobile Ad hoc Networks

Mobile ad hoc networks are collections of mobile nodes which are capable of communicating with each other in the absence of any existing infrastructure. In this network environment, all hosts can act as routers so that routes are built quickly adapting to changes in the network topology.

There is no single standard term for these networks. They are also known by other names (by many authors) each describing different aspects of the network.

- *Multi-hop Networks*: This emphasizes the fact that the network is organized in a cooperative manner such that nodes can forward packets for other nodes that are not within the transmission range. This functionality is not possible in a single-hop network such as a cellular network.
- *Fully Mobile Networks*: This emphasizes the fact that the network is fully autonomous and does not depend on any existing infrastructure.

All the nodes can be mobile relative to each other and there is no base station to connect to a static network.

- *Peer-to-Peer Networks*: This emphasizes the fact that mobile nodes are capable of direct mobile-to-mobile communications. In cellular networks and mobile network technologies such as Mobile IP [Per96] and Cellular IP [Val99], mobile nodes communicate only via their assigned base stations.

The term *mobile ad hoc networks*, indicates the fact that the network can be temporarily set up anywhere, at anytime, to operate without the need of any established infrastructure or central coordinating mechanism. The MANET working group uses the term mobile ad hoc networks and we adopted this name in this thesis.

Physically, mobile ad hoc networks consist of multiple mobile nodes and wireless communication devices with each node logically consisting of a router and one or more wireless interfaces. In reality, a node may consist of separate networked devices or may be integrated into a single unit such as laptop or palmtop computers [Mac98]. When multiple network devices are used, some of the interfaces can be hardwired while the rest are wireless interfaces.

The nodes may use omnidirectional or highly directional transceivers or a combination of both. The omnidirectional transceivers support broadcast type of communications whereas directional transceivers support-point-to point communication. While routers in infrastructure-based mobile networks and the Internet are static, those in ad hoc networks are mobile forming a random topology. In this architecture, router-to-router or router-to-mobile node connectivity is highly dynamic and may change while in operation.

The architecture of mobile ad hoc networks is a hybrid of traditional packet radio networks and cellular networks. Like packet radio networks it involves datagram traffic, multi-hop routing and dynamic network reconfiguration. Cluster-based network organisation, with each cluster being coordinated by a clusterhead as in the architecture proposed in chapter 7, creates an architecture similar to cellular networks. However, clusterhead in ad hoc networks and base stations in cellular networks are very different entities—clusterheads have no special hardware components and are dynamically selected from nodes in the networks via some clustering algorithms.

Mobile ad hoc networks thus have several distinguishing characteristics from other mobile networks:

1. *Infrastructureless operation.* Unlike cellular mobile networks which depend on base stations, mobile ad hoc networks have no stable communication infrastructure and central controlling mechanism.

2. *Distributed operation.* Ad hoc network connectivity is based on proximity and nodes within transmission range can communicate directly. Each node is responsible for routing and mobility management in a distributed fashion. However, a single node or subset of nodes may be used to perform route computations.

3. *Relaying capability.* Nodes in mobile ad hoc networks can communicate directly when they are reachable. Those that are outside the wireless transmission range, depend on intermediate nodes to relay packets to each other.

4. *Dynamic topologies.* A mobile ad hoc network experiences rapid changes in the network topology. Nodes move unpredictably and can get disconnected from or be added to the network at any time (see figure 3.1). This makes existing routing protocols based on route computation and periodic route update inadequate.

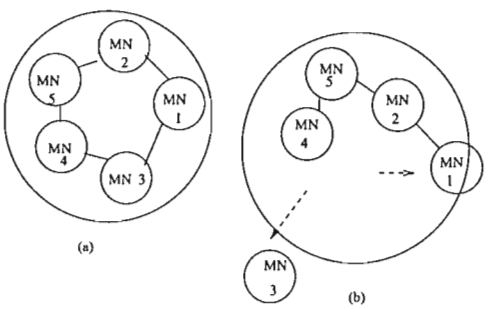

Figure 3.1: Topology changes in mobile ad hoc networks

5. *Limited resources.* Mobile ad hoc networks have limited resources such as bandwidth and power. The network can be congested due to limited capacity. Also, dependence on battery power for communication makes disconnected operation more common. For energy conservation, nodes may need to postpone transmitting or receiving for a while, which may degrade network performance.

3.2 Applications of Mobile Ad hoc Networks

Mobile ad hoc networks have both commercial and non-commercial applications. Some examples of these are described below.

Emergency relief operations. Mobile ad hoc networks can be used for rapid deployment in rescue missions in areas where wiring is not available or

base stations, mobile switching centers, home and visitor location registe
On the other hand, mobile ad hoc networks need sophisticated distributed
gorithms to perform routing and mobility management operations. Mobile
hoc networks design requires making decisions on the network organisati
media access protocols, network topology, bandwidth, energy requireme
and routing protocols.

3.3.1 Mobile ad hoc network routing protocols

Unicasting refers to a point-to-point communication between source and
tination pairs. Multicasting refers to sending a single packet to a grou
nodes which are small compared to the entire network. Due to the na
of node connectivity and network resource constraints, unicast and multi
routing protocols developed for the wireline networks cannot be directly
plied to mobile ad hoc networks. Design of unicast routing protocols is
of the main challenging topics in mobile ad hoc networks.

Several unicast routing protocols have been recently proposed for m
ad hoc networks, but there is no standard routing protocol available f
hoc routing, though such effort is currently under way. The existing ro
protocols for mobile ad hoc networks routes are discussed in chapter 4.
proposals are discussed in chapters 5–8.

In ad hoc networks hosts may move in a group or work in a grou
cessitating the need for sending multicast messages. The existing mul
routing protocols for fixed networks described in [Moy94b, Dee90, I
Dee96] are not suitable for mobile ad hoc networks for several reasons.
multicast servers or multicast members may move, making source-ini
protocols inefficient. Second, most current protocols are based on a mul
tree. A tree-based multicasting is not suitable for mobile ad hoc net
since the multicast tree can easily get broken and transient loops m
formed during tree reconfiguration. Recently, some multicast protocol
been proposed for mobile networks [Bom98, Lee99, Gra99, Wu98]. De
ing an adaptive and reliable multicast technique for mobile ad hoc ne
still requires further investigation.

3.3.2 Mobility management

In mobile ad hoc networks there is no reference point like in cellul
works. Thus, the main difference between mobility management tech
developed for personal communication systems and hierarchically or
ad hoc networks are the absence of pre-established infrastructure a

inapplicability of area-based updating mobility management strategies. Mobility management for hierarchical organised mobile ad hoc networks is a new area that needs further investigation.

3.3.3 Medium Access Control protocols

The traditional CSMA/CD-based MAC protocols are not adequate for mobile ad hoc networks due to mobility and the wireless nature of communication. The two main problems of wireless communication are *hidden terminal* and *exposed terminal* problems. The hidden terminal problem occurs when two nodes that cannot hear each other try to transmit to another node that can hear both of them. The exposed terminal problem occurs when two nodes that can hear each other transmit to another node that can hear only one of them. A mobile ad hoc network requires the design of an appropriate medium access control protocol to reduce or avoid these problems.

Since the CSMA/CD protocol does not function well in a mobile network environment, the IEEE 802.11 Wireless LAN standard [IEE97] that uses Carrier Sense Multiple Access for Collision Avoidance (CSMA/CA) protocol has been proposed for mobile networks. Other protocols that attempt to overcome the shortcomings of CSMA/CD are discussed in [Bah94, Ful97, Jin98, Kar90, Lin97].

Most of these protocols are designed for infrastructure-based mobile networks and they assume that the interfering nodes can hear Request To Send/ Clear To Send (RTS/CTS) dialogs, but this assumption is not valid in a highly mobile network environment. A new media access control protocol called Dual Busy Tone Multiple Access [DBTMA] [Jin98] has recently been proposed for mobile ad hoc networks. In this protocol, a single common channel is split into two sub-channels. The first is a data channel which is used to transmit data packets. The second is a control channel which is used to transmit control packets such as RTS and CTS. Two busy tones are assigned to the control channel: the receive busy tone and the transmit busy tone. The protocol improves channel capacity while avoiding most of the collisions that occur due to hidden terminal problem. Efficient medium access control protocols that avoid collisions and also solve the fairness problem are needed for mobile ad hoc networks.

3.3.4 Data management

Data management includes operations such as data dissemination over wireless link, location-dependent querying of data, transaction management, information retrieval and advanced interfaces for mobile computers. Data management in cellular networks is based on asymmetric data transmission.

Since packet receiving consumes less power than transmitting and mobile nodes have scarce resources, data is sent to a mobile node with periodic packet broadcasts made from the fixed networks [Imi94]. Such transmission mechanism leads to communication latency. A joint broadcasting scheduling approach has recently been proposed in [Su98] where the server broadcasts the information and the user caches such information. In mobile ad hoc networks, data management poses a new challenge. Each mobile node is responsible for maintaining a database for user data and control information.

3.4 Mobile Ad hoc Networks and the Internet

Although a mobile ad hoc network is an autonomous system of mobile nodes that can operate in isolation, its integration with fixed networks provide greater network flexibility and increases its application ranges. Thus, the network should interoperate with existing static networks such as the Internet. This can be achieved by connecting the mobile ad hoc network to the static network via a gateway router. Figure 3.2 shows one possible configuration of mobile ad hoc networks and Internet connectivity.

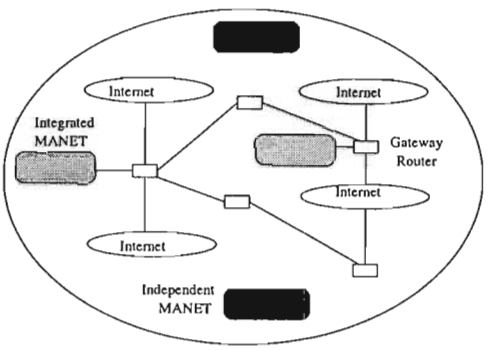

Figure 3.2: Mobile ad hoc network and the Internet

The integration of mobile ad hoc networks and the Internet provides fully mobile Internet services. Relative to static networks, the emerging mobile Internet can be divided into two layers [Mac98] as shown in figure 3.3. The first layer is a mobile nodes layer where nodes are connected to the fixed network. This layer consists of hosts that are connected to the routers on the static network temporarily. Mobile nodes at this layer are like hosts in cellular networks and can be supported by protocols such as Mobile IP. With dependence on fixed networks, both mobility management and addressing issues are handled at this layer.

The second layer is a mobile routers layer which consists of mobile routers and mobile nodes. Like ordinary ad hoc network nodes, these mobile nodes

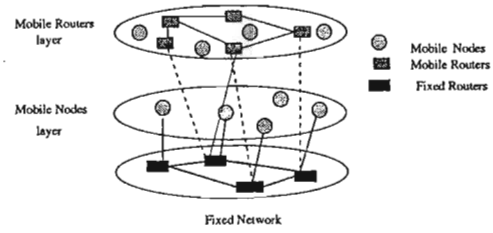

Figure 3.3: Mobile Internet layers

are free to move and can be connected to or disconnected from the network as well as routers. A node can act as a mobile router or mobile node. Also, like ad hoc networks, the mobile router layer gains no support from the fixed network and runs independently. While host-to-routers connection may be wireline or wireless, the connections between the mobile routers are assumed to be wireless.

In view of the future mobile Internet's layering architecture, the MANET working group is developing routing standards for the mobile router layer. For supporting multiple wireless technologies, ad hoc network addressing should be interoperable with the existing IP addressing architecture. Policies and protocols for Router ID and IP address assignment should be developed as needed and the existing approach does not specify how IP addresses are assigned to interfaces on hosts or routers.

Chapter 4

Routing Problems in Networks

Routing is a process that selects the paths along which Network Protocol Data Units known as packets propagate from the source to the destination. It is the main function of the network layer in both the ISO/OSI and TCP/IP protocol architectures.

Route selection involves a complex collection of algorithms that work more or less independently and yet support each other by sharing or exchanging routing information [Ber92]. This complexity emerges from the fact that, first, unlike lower and higher layer protocols, routing requires coordination between all nodes in the network. If a single node fails or transmits incorrect routing data, the entire network can be affected. Second, the routing algorithms must adapt to link or node failures and react fast enough to avoid data loss. This may be achieved by redirecting packets or updating routing information. Third, periodic routing update is essential for routing efficiency. For example, a routing algorithm may need to update its route when there is congestion within the network.

4.1 Routing in Traditional Static Networks

Routing algorithms are used to set up routing tables which specify the neighbour or list of nodes via which the packet propagates to reach the destination. Routing algorithms should perform correctly and be free from looping. They should remain globally optimal by adapting to the changes in the network topology, making efficient use of processing, memory and bandwidth resources.

Many routing algorithms have been proposed for fixed networks. The design of these algorithms involve three main tasks:

- *Defining routing metrics and measurement techniques*: This is a measurement process for determining the network characteristics. It requires deciding on link cost measurement techniques, such as the use of average delay, number of hops, available bandwidth or communication costs. Cost metrics can be fixed or change with the network condition. One also decides on mechanisms for gathering the node or link information used for route computation.

- *Specifying route generation and forwarding mechanisms*: This determines the mechanisms for computing the routing paths. Based on adjacency information and traffic requirements, routing tables are constructed.

- *Deciding route maintenance and update policy*: This determines how the routes affected by changes in network topology are updated. It replaces stale routes with fresh route information to avoid loop creation or packet loss. A mechanism for distribution of this update information should also be specified.

A good routing algorithm is essential to achieve successful network operation and routing performance. Performance criteria for routing algorithms include delay, throughput, reliability and routing costs [McQ74].

There are several ways of classifying routing algorithms, though most protocols differ only in their implementations. The two major classifications are: Adaptive routing protocols and static routing protocols.

4.1.1 Static routing protocols

In static routing protocols, communication paths are fixed regardless of changes in the network condition. The route between each pair of node is computed in advance and installed into the router during startup. Routing tables change very slowly, often in response to human intervention. Thus, static routing can provide the optimal routes for a network that has no fluctuations or failures. The advantage of static protocols is computational simplicity, loop freedom, suitability for network design and evaluation through analytical methods. But the assumptions of network stability, reliability and sufficient resources make them unimportant for the existing networks.

4.1.2 Adaptive routing protocols

Adaptive routing protocols adapt to the dynamic nature of the network such as changes in the network topology, link qualities and traffic conditions. The routing tables entries may change over time reflecting the current network

status. Due to their flexibility and better routing performance, dynamic routing protocols are used in the existing networks.

Adaptive routing protocols can be further classified.

1. *Distributed or centralised.* In centralised routing, a central site computes and disseminates the routes to its neighbours or the entire nodes in the network. The main problem with this technique is that the central router can be overloaded or even fail resulting in the loss of routing information. But since it has global knowledge of traffic load and network topology, looping can be avoided and routing can be more efficient in the long run. In distributed routing, each router computes routes locally based on information gathered from its neighbours. This route information is also exchanged with neighbours. Distributed routing allows a node to react more rapidly to changes in its locality than in centralised routing.

2. *Proactive or reactive.* Routing protocols can be classified based on whether routes to all nodes are computed and kept in the routing tables, or generated only when they are needed. The former is proactive and the latter reactive. Flooding can be considered as a reactive mechanism where packets are sent to the entire network. Several mechanisms that rely on flooding for route discovery have been proposed for mobile networks.

3. *Single path or multipath.* Single path (also known as shortest-path) routing is the most commonly used routing scheme in the existing networks including the Internet. A drawback is that it may result in network congestion. These protocols react to congestion by only changing the route used to arrive at the target destination, but this can lead to an oscillatory behaviour [Ber92]. In multipath routing, multiple routes are maintained for each destination. Multipath routing provides better congestion control and load balancing mechanisms.

4. *Flat or hierarchical.* In flat routing, each node maintains the full routing information and can directly communicate with other nodes in the network. In hierarchical routing, each node maintains a subset of the complete routing information. Hierarchical routing reduces computation, communication and storage overhead while optimality of the generated routes and adaptivity to local changes are preserved [Ame88]. Hierarchical routing protocols are highly scalable to larger networks.

5. *Source-, hop- or destination-oriented.* Most routing algorithms construct routing tables for hop-by-hop routing. Here a routing decision is

made at each intermediate node which forwards packets according to its routing table entries. But the routing decision can also be made at the source or destination node.

4.1.3 Distance Vector and Link State Algorithms

Most common routing algorithms are adaptive, distributed and use single path. They are often based on distance vector or link state algorithms. In link state routing, packets are small in size since the information each contains is link status of only a node's neighbours. But route updates are flooded to all nodes in the network. On the other hand, for the distance vector algorithm, packets are large in size since the information it contains is distances of a source node to all destinations. But route update packets are sent only to a node's neighbours.

Distance Vector Algorithm (DVA). Distance vector routing is based on the Distributed Bellman-Ford algorithm. In the DVA, a router knows the cost of the preferred path through each of its neighbours to all destinations and uses this information to compute the shortest path and the next hop in the path to each destination.

For a given destination j, each node keeps track of its best known distance to the destination. Initially, the distance is 0 to the node itself and infinity to all other nodes. Also, a node i knows the cost $N(i,k)_t$ of its link to any neighbour k, and the distance $D(k,j)_t$ from k to any node j, as reported by its neighbour k at time t. After receiving an update message from k, node i updates its distance $D(i,j)_{t+1}$ to j using the formula:

$$D(i,j)_{t+1} = Min[D(i,j)_t, D(k,j)_t] + N(i,k)_t$$

A node sends an update to its neighbours either periodically or when its distance is changed. To be able to react to changes in the topology, nodes store the distances as reported by their neighbours. A node computes its best route based on neighbour information using the formula:

$$D(i,j) = \underset{k}{Min}\,[D(k,j) + N(i,k)]$$

The neighbour through which this shortest path goes is chosen as the next-hop entry in the routing table. The recomputed distance information is broadcasted to all neighbours. Since all nodes perform the same computation and forwarding process, after some time all routers will have consistent information.

Compared to the link state algorithm, DVA is computationally more efficient, easier to implement and requires much less storage space [Gra95, Per94]. But there are routing loops and count-to-infinity problems. Count-to-infinity is a problem that arises when a node in the network increments its distance to a destination until it reaches a predefined maximum routing distance value. This problem occurs when there is a network partition, a node failure or a network congestion. The basic DVA has no mechanism to determine when a network node should stop incrementing its distance to a given destination.

There are many proposals for solving the count-to-infinity problem. But they are not always effective because they are either too complex to use or not scalable when a large number of nodes are involved in the count-to-infinity problem [Hui95, Tan96, Tho96].

RIP [Hed88], BGP [Lou91], and Cisco's IGRP [Bos92] are some of the protocols based on the distance vector algorithm.

Link State Algorithm (LSA). In link state routing, a topology database containing information about every link is maintained at every node. Each node broadcasts the network topology information to all its neighbours. Thus, the complete topology information is replicated at every node and used to compute the up-to-date routing table. Since the link-state protocol keeps complete topology information at routers, multiple routes can be computed and Quality Of Service based routing is possible. This has the advantage that the long-term looping problem that exists in the distance vector protocol does not exist in link state routing. This protocol exchanges information regarding link characteristics unlike the distance information in distance vector protocols. OSPF [Moy94a], IDPR [Est93], ISO IS-IS [ISO89] are some of the protocols based on the link state algorithm.

Link Vector Algorithm. Both DVA and LSA have scalability problems. The distribution of information on a per path basis in DVA leads to a combinatorial explosion with the number of service types and policies [Gra94]. On the other hand, LSA's complete topology replication consumes excessive communication and processing resources in large networks.

To address these scalability problems, a new algorithm known as the Link Vector Algorithm was proposed in [Beh97a]. In this algorithm, each router reports to its neighbours the characteristics of those links that it uses in its preferred paths. Based on this information, each router constructs a source graph that consists of a partial topology and is used to compute its preferred paths. By disseminating only relevant information, the algorithm significantly reduces the communication and storage overhead.

4.2 Routing in Mobile Ad hoc Networks

The main challenges in designing a routing protocol for mobile ad hoc networks arise from the fact that the network has limited resources and encounters frequent and unpredictable topological changes. Also, the number of nodes in the network can be large resulting in considerable data and frequent exchange of routing information. In mobile ad hoc networks, all nodes in the network can act as routers and take part in the generation and maintenance of routes to other nodes in the network.

Routing protocols in traditional fixed networks are generally based on either distance vector or link state routing algorithms. These routing algorithms are not suitable for mobile ad hoc networks [Kri96, John96, Per94]. First, the protocols are not designed for networks that are subject to frequent and unpredictable topological changes. Second, the protocols put high computational overhead on mobile nodes and high communication overhead on wireless channels. Third, the protocols converge slowly upon topological changes. Also, the routing protocols based on cellular architecture are not suitable as they rely on slowly-changing networks and fixed home agents.

In this section we describe the recently proposed routing protocols for mobile ad hoc networks. We have classified them into three categories: Proactive routing protocols, reactive routing protocols and hybrid routing protocols. The hybrid protocols are based on some features of reactive and proactive protocols. Each of these protocols can further be classified into cluster-based and non-clustered routing protocols as shown in figure 4.1. In cluster-based routing protocols, the mobile nodes are grouped into small units known as clusters. Some of these protocols use flat routing while others organise clusters hierarchically to reduce routing overhead in the network.

In non-clustered routing schemes, there is a flat architecture where each mobile node has complete routing information either by computing it or discovering on demand. Most recently proposed routing protocols for mobile ad hoc networks are not based on a clustering architecture. For most of these protocols, route maintenance, computation or discovery mechanisms incur high overhead. These overheads also increase considerably with increase in the network size.

4.2.1 Proactive routing protocols

In proactive routing protocols, each node computes and stores routing information to every other node in the network. When the network topology changes, the nodes propagate update messages through the network in order to maintain consistent and up-to-date routing information about the whole

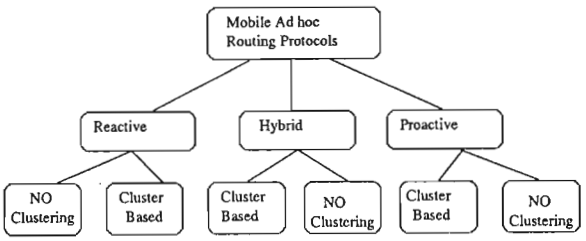

Figure 4.1: Routing protocol classifications

network. These routing protocols differ in the method by which the routes are computed and disseminated and the number of routing-related tables constructed. Below we describe some specific routing protocols.

A) Destination-Sequenced Distance Vector protocol (DSDV)

The Destination-Sequenced Distance Vector Routing protocol described in [Per94] is a proactive protocol based on the Distributed Bellman-Ford Routing Algorithm (DBF). The main improvement is avoiding loop formation. Every mobile node maintains a routing table that lists all available destinations, the number of hops to reach the destination and a sequence number which is originated by the destination node. The main purpose of the sequence number is to distinguish stale routes from fresh routes and thus avoid the creation of routing loops.

The mobile nodes periodically transmit their routing tables to their immediate neighbours. Each route update packet contains routing table information and a unique sequence number. Whenever route update information is received, comparison is made with the existing routing information. At any time, a route labelled with the highest sequence number is the most recent and is used. If two routes have the same sequence number then the route with the shortest length is used. In order to avoid inconsistency in routing information updates, each node estimates the settling time of routes.

B) Cluster-Based Routing protocol (CBR)

A cluster-based routing algorithm was proposed in [Kri96]. However, it does not require election of a clusterhead. The protocol divides the network into a number of overlapping small clusters where a change in the network topology corresponds to a change in cluster membership.

This routing protocol has two phases: route construction and route maintenance. In the former, routes are constructed between all pairs of nodes.

Each node computes and maintains route information in a way similar to the distance vector algorithm. In the latter phase, loop-free routes are recreated when the existing routes are destroyed due to topology changes. Route update is propagated to all other nodes in the network by boundary nodes. This routing protocol is based on clustering but with a flat architecture. Though only a single node propagates route updates, since cluster size is small there may be many propagating nodes. Thus, route updates result in a complete network maintenance making inefficient utilisation of wireless resources.

C) Cluster MCDS routing (CMCDS)

A Cluster Minimum Connected Dominating Set (MCDS) routing algorithm was proposed in [Das97a] for routing in mobile ad hoc networks. It uses a two-level hierarchical architecture with a virtual backbone. The first level groups nodes into clusters while the second level maintains the virtual backbone for each cluster. In this protocol, for routing within clusters, topology information is gathered into all MCDS nodes and shortest path routes are computed in a manner similar to link state but restricted to MCDS nodes.

Based on this work, a two-level hierarchical routing architecture with a dynamic structure called a spine was proposed in [Das97b]. The within-cluster routes are computed by gathering topology information into all spine nodes in a manner similar to the link state routing approach but restricted to spine nodes. Inter-cluster routing is performed using link state routing algorithms.

D) Global State Routing (GSR)

Global State Routing presented in [Che98] is based on the link state routing algorithm. It maintains the full network topology but avoids flooding of routing information. In this algorithm, each node maintains four tables: a neighbour list, a topology table, a next hop table and a distance table. The neighbour list contains the list of its neighbours. For each destination, the topology table contains the link state information as reported by the destination and the timestamp of the information. For each destination, the next hop table contains the next hop to which the packets for this destination must be forwarded. The distance table contains the shortest distance to each destination node.

The routing messages are generated on a link change as in link state protocols. But GSR does not flood the link state packets to the entire network; instead, it periodically exchanges them with neighbours. When routing information is received, a node updates its topology table if the sequence number

of the message is larger than the sequence number stored in the table, as in DSDV [Per94]. The shortcomings of GSR are wasting bandwidth due to the large size of update messages and latency of the link state change dissemination which depends on the length of update period.

E) The Wireless Routing Protocol (WRP)

The Wireless Routing Protocol described in [Mur96] is a proactive protocol based on a path-finding algorithm. It eliminates the count-to-infinity problem of distance vector algorithms and reduces the occurrence of routing loops. In WRP, each node in the network maintains four tables: a Distance table, a Routing table, a Link-Cost table and a Message Retransmission list table.

The Distance table of a node i contains, for each destination j and each neighbour of i (say k) the distance to j and the predecessor as reported by k. The Routing table contains for each known destination j, the distance of j from node i, and the successor of i on this path. It also contains a marker used to identify if the entry is a simple path or a loop. The Link-Cost table of node i contains the cost of relaying a packet through each neighbour node k, and the number of periodic updates that have elapsed since node i received any error-free message from node k. The Message Retransmission List table contains the sequence number of the update message, a retransmission counter, an acknowledgment-required flag vector with one entry per neighbour and a list of update messages sent. This information is used to keep track of which updates in an update message need to be retransmitted and which neighbours should acknowledge the retransmission.

Mobile nodes exchange routing tables with their neighbours using update messages periodically as well as on link changes. On receiving an update message, the node modifies its distance table and looks for better paths. Any new path is relayed back to the original nodes so that they can update their tables. If there is no change in routing table since last update, the recipient is required to send a Hello message to confirm connectivity.

WRP uses a novel mechanism to avoid loops. Nodes communicate the distance and second-to-last hop information for each destination in the wireless network. Each node checks the consistency of all its neighbours whenever it detects a change in link of any of its neighbours. This ultimately eliminates loops and provides faster route convergence upon link failure.

F) Hierarchical State Routing (HSR)

Hierarchical State Routing described in [Iwa99] uses multilevel clustering and logical partitioning of mobile nodes. The network is partitioned into clusters and a clusterhead elected for each cluster. In HSR, the clusterheads again organize themselves into clusters. The nodes of a cluster broadcast their link information to each other. The clusterhead summarizes the cluster information and sends it to neighbouring clusterheads via gateways. Clusterheads are members of the cluster on level higher and they exchange their link information as well as the summarized lower-level information among each other. A node at each level floods to its lower level the information that it obtains after the algorithm has run at that level. Each node has a hierarchical address. Gateways can communicate with multiple clusterheads and can be reached from the top hierarchy via more than one path and thus can have more than one hierarchical address.

In HSR nodes are also partitioned into logical subnetworks and each node is assigned a logical address of (subnet, host). Each subnetwork has a location management server (LMS). All the nodes of that subnet register their logical address with the LMS. The LMS advertises its hierarchical address to the top levels and the information is sent down to all LMS too. Thus the LMS translates between logical addresses and hierarchical addresses.

The main advantage of HSR is that the hierarchical architecture makes the routing table more scalable to a larger network. The drawbacks of this routing approach are: first, the use of link state routing algorithms in route construction in a longer hierarchical clustering makes the cost of continuously route update high. Second, the protocol has high complexity compared to most existing routing protocols for ad hoc networks.

4.2.2 Reactive routing protocols

In reactive routing protocols, up-to-date routes are not maintained; rather routes are established on demand. Reactive routing is also known as on-demand routing. In these protocols, when a source wants to send a packet to a destination, it invokes a route discovery mechanism to find the path to the destination. The route remains valid so long as the destination is reachable or until the route timeout occurs.

In general, two different mechanisms are used for route discovery in reactive protocols. Some algorithms use the *backward learning* process where the intermediate nodes receiving the route query packet learn the path to the source and record the route in the forwarding table. The destination node upon receiving the query packet uses the path traced by the query for reply.

Others use a *source routing* mechanism where the query packet records in its header the identities of the intermediate nodes that it traverses. The destination then extracts the route information from the query packet header and sends it back using source routing mechanism.

A) Ad hoc On-demand Distance Vector Routing (AODV)

The Ad hoc On-demand Distance Vector Routing proposed in [Per99] is an on-demand version of the distance vector routing algorithm which adopts back tracking mechanism.

To find a path to the destination, the source broadcasts a route request packet to its neighbours. The neighbours in turn broadcast the packet to their neighbours until it reaches an intermediate node that has recent route information about the destination or the desired destination is reached. The route request packet uses sequence numbers and a node discards a route request packet that it has already seen. Thus if a node replies to a route request, it replies with the up to date routes. AODV uses only symmetric links since the route reply packet follows the reverse path of route request packet. As the route reply packet returns to the source, the intermediate nodes insert the route into their tables.

Route maintenance is performed as follows: If the source moves then it can re-initiate route discovery. If one of the intermediate nodes moves, then the moved node's neighbour realizes the link failure and sends a link failure notification to its upstream neighbours. The process continuous until it reaches the source upon which the source can re-initiate route discovery if necessary.

B) Dynamic Source Routing Protocol (DSR)

The Dynamic Source Routing Protocol proposed in [John96] is an on-demand routing protocol based on source routing. In DSR, each mobile node maintains route caches that contain the complete routes from source to destination. The two major phases of the protocol are route discovery and route maintenance.

When a node wants to send a packet to a destination, it checks its route cache for a route to the destination. If it finds an unexpired route, it uses this route. Otherwise, it initiates the route discovery process by broadcasting a route request packet. The route request packet contains the source and destination, and a unique identification number. Each intermediate node checks whether it knows a route to the destination. If it does not, it appends

its address to the route record of the packet and forwards the packet to its neighbours. To limit the number of route requests propagated, a node processes the route request packet only if it has not already seen the packet. A route reply is generated when a node with current information about the destination receives the route request packet. It replies to the source with a reply packet that copies the route from the query packet.

The DSR protocol uses two types of packets for route maintenance: Route Error packet and acknowledgement packet. When a node receives a route error packet, it removes the hop in error from its route and all routes containing the hop are truncated at that point. Acknowledgment packets are also used to verify the correct operation of the route links. Since DSR uses source routing, both short-lived and long lived loops can be easily eliminated.

C) Temporally Ordered Routing Algorithm (TORA)

The Temporally Ordered Routing Algorithm proposed in [Par97b] is a highly adaptive, efficient and scalable distributed routing algorithm based on the concept of link reversal. TORA is a source-initiated on-demand routing protocol that finds multiple routes from a source node to a destination node. The main feature is that the control messages are localized to a very small set of nodes near the occurrence of a topological change. To achieve this, the nodes maintain routing information about adjacent nodes.

TORA performs three basic functions: Route creation, route maintenance, and route erasure. Route Creation is done using query and update packets. The route creation algorithm starts with the height of destination set to 0 and all other node's height set to NULL (i.e. undefined). The source broadcasts a query packet with the destination node's ID in it. A node with a defined height responds with an update packet that has its height in it. A node receiving an update packet sets its height to one more than that of the node that generated the update. A node with higher height is considered upstream and a node with lower height downstream. This yields a Directed Acyclic Graph (DAG).

When a node moves, the DAG route is broken. When the last downstream link of a node fails, it generates a new reference level. Links are reversed to reflect the change in adapting to the new reference level. This has the same effect as reversing the direction of one or more links when a node has no downstream links. In the route erasure phase, TORA floods a broadcast clear packet throughout the network to erase invalid routes. When multiple nodes concurrently detect partitions, erase routes and build new routes based on each other, oscillations may occur, but route convergence will ultimately occur.

D) Associativity Based Routing (ABR)

The Associativity Based Routing proposed in [Toh96] is a demand-based routing protocol free from loop, deadlock and packet duplication. The basic objective of ABR is to find longer-lived routes for ad hoc mobile networks. Association stability means connection stability of one node with respect to another node over time. Based on the degree of association stability, the protocol defines longevity of a route, relaying load of the intermediate node and link capacity as a new routing metric.

The three phases of ABR are route discovery, route reconstruction and route deletion. The route discovery phase is a broadcast query and await-reply. On receiving a message, an intermediate node appends its address and its association stability to the query packet. Each packet arriving at the destination will contain the association stabilities of the nodes along the route from source to the destination. The destination selects the best route by examining the association stabilities along each path. Once a path has been chosen, the destination sends a reply packet back to the source along this path. The nodes on the path that the reply packet follows mark their routes as valid.

When the node moves and the selected route becomes invalid, route reconstruction is invoked. When a discovered route is no longer needed, the source node initiates a route delete broadcast. In ABR, each node maintains a routing table to store routing information, a neighbour table to record the degree of association stability and a seen table to stop it forwarding the same packet again.

E) Signal Stability Routing (SSR)

The Signal Stability based Adaptive Routing protocol proposed in [Dub97] is an on-demand routing protocol that selects long-lived routes based on signal strength and location stability. Routes that have "stronger" connectivity and have been valid for longer times are chosen.

Intermediate nodes maintain a Routing Table and Signal Strength Table which stores the signal strength of neighbouring nodes (obtained by periodic beacons from them). On receiving a route-search packet, a node updates the appropriate table entries and forwards the packet if necessary. The destination chooses the first arriving packet, reverses the route and sends a route-reply message back to the initiator. The intermediate nodes along the path update their routing tables. Route-search packets arriving at the destination have necessarily arrived on the path of strongest signal stability

because packets arriving over a weak channel are dropped at intermediate nodes.

In SSR, route maintenance is event-driven. When a link failure is detected within the network, the intermediate nodes send an error message to the source indicating which channel has failed. The source then sends an erase message to notify all nodes of the broken link and initiates a new route-search process to find a new path to the destination.

4.2.3 Hybrid routing protocols

We have seen that in reactive protocols, the route update overhead is low but routes may not be immediately available when needed or they may be stale and sub-optimal. On the other hand, in proactive protocols, routes are precomputed and used whever needed but updates can lead to excessive overhead.

A) Zone Routing Protocol (ZRP)

A hybrid of routing on demand and shortest path algorithm known as Zone Routing Protocol (ZRP) was proposed in [Haa97, Haa98]. ZRP uses routing zones. A routing zone for a node is defined as all nodes whose distance is at most some parameter (the Zone Radius). Each node proactively maintains routes within its routing zone, and carries out a global search by initiating a query packet when the destination is not in its routing table. The route query propagates using a packet delivery service called bordercasting. ZRP is a flat routing protocol where each node maintains routes for its own routing zone. There is a one-to-one correspondence between nodes and routing zones.

B) Hierarchical Cluster-based Routing Protocols (HCR)

In chapter 7, we propose a hybrid routing scheme based on a hierarchical clustering architecture. The main features of the protocols are dynamic formation of clusters using clustering algorithms and the use of hybrid routing mechanisms for inter-cluster and intra-cluster routing. Unlike ZRP, our proposal is based on a hierarchical routing mechanism and uses clusterheads to coordinate routing.

4.2.4 Some comparisons of mobile ad hoc routing protocols

Table 4.1 and 4.2 below give summary of some of the proactive and reactive routing protocols discussed in this chapter. The summary is made in terms

of some common parameters. In table 4.2, Route Cache (RC), Routing Table (RT), Shortest Path (ShP), Location Stability (Stab), degree of Association (Assoc), Fresh routes (Fresh) and Relaying load (Load) are used as routing related metrics.

Parameters	DSDV	WRP	CBR
Routing hierarchy	Flat	Flat	Flat
Base protocol	DBF	DBF	DBF
Loop freedom	Yes	Yes	Yes
Number of tables	Two	Four	Four
Update rate	Event/Periodical	Event/Periodical	Event/Periodical
Update sent to	Neighbour	Neighbour	Neighbour
Sequence number	Yes	Yes	Yes
Hello messages	Yes	Yes	Yes
Routing Metric	Shortest Path	Shortest Path	Shortest Path
Critical Node	No	No	Boundary node
Routing Overhead	High	High	High

Table 4.1: Proactive Routing Protocols

Parameters	AODV	DSR	ABR	SSR
Routing type	Flat	Flat	Flat	Flat
Loop freedom	Yes	Yes	Yes	Yes
Route maintainer	RT	RC	RT	RT
Route Expiration timer	yes	no	no	no
Hello messages	Yes	No	Yes	Yes
Routing Metric	ShP/Fresh	ShP	ShP/Load/Assoc	Assoc/Stab
Routing Overhead	Low	Low	Low	low

Table 4.2: Reactive Routing Protocols

4.2.5 Use of clustering

Finally in this section we expand on the benefits of clustering. The concept of a coordinator or a leader is commonly used in parallel processing and communication networks. In multiprocessor systems, leader election algorithms are used to select a co-ordinator for efficient utilization of CPU. Also

in multi-access networks, a designated router is selected to reduce the size of link state database so that it becomes manageable. Clustering architecture with clusterhead can be used to function similarly.

Clustering with each cluster coordinated by the clusterhead was proposed for mobile packet radio networks in [Bak84]. Similarly, a multicluster mobile multimedia packet radio network architecture for wireless mobile information systems was proposed in [Ger95b]. By using a distributed clustering algorithm, nodes are organised into clusters each with a clusterhead. Clusterheads are used as coordinators for channel scheduling and code assignment. Also, we have seen CBR and CMCDS clustering above.

Clustering provides several advantages over non-clustered ad hoc networks. First, it stabilizes the network topology. Clustering provides some kind of infrastructure to the otherwise disorganised and dynamically changing ad hoc networks. In a clustering architecture, a node can be elected as regional coordinator thus establishing a controlling mechanism for a network without a base station.

Second, network partitioning improves functions such as routing and mobility management while reducing signalling or control overhead and minimising congestion [Sha96]. Having many small networks rather than one large network can reduce communication overheads in the network. Furthermore, a clustering architecture with a clusterhead can also assist in efficiently tracking the mobile nodes.

Third, clustering improves MAC resource management and provides efficient utilisation of wireless channels by assigning different codes to each cluster and also provides power control mechanisms [Ger95b].

Given the above advantages of clustering architecture, we use clustering as a mechanism for providing an infrastructure to the network in order to improve routing performance. Clustering architecture reduces power interference from neighbouring nodes including hidden nodes and increases control efficiency [Bak84]. Using suitable inter-cluster and intra-cluster routing mechanisms, clustering architecture can reduce routing overhead in mobile ad hoc networks.

Chapter 5

Limited Flooding for Mobile Ad hoc Networks

5.1 Flooding

5.1.1 Broadcast and point-to-point networks

Point-to-point in connection-oriented and broadcasting in connectionless networks are the two main modes of data communication in computer networks. Broadcasting disseminates information in a system from one node to all other nodes. Applications such as database transactions, network management, tactical military communication, etc., depend on broadcast services. In this chapter we describe a proposal for broadcast in mobile ad hoc networks. It is also a possibility for unicast in very dynamic networks.

Several broadcasting techniques have been proposed. *Flooding* is a distributed procedure for data broadcast where packets are duplicated at each intermediate node and sent to other nodes. In broadcasting on a *spanning tree*, a tree-based broadcast is made where packets are disseminated on the spanning tree. It makes better use of bandwidth but requires maintenance of the spanning tree. In *multidestination routing*, several packets can traverse the same path for some time until they reach their desired destinations. Those that arrive at their destination are delivered while the others continue propagating in the network. *Reverse path forwarding* requires packet forwarding only if it arrives on the preferred paths. We propose a variation of flooding as a routing mechanism in ad hoc networks due to its simplicity, speed, reliability and popularity.

5.1.2 The basic flooding algorithm

In pure flooding, every incoming packet is sent out on every outgoing link except the one it arrived on. It is a simple and effective routing technique that does not require the computation and maintenance of a routing table. The originator of the packet sends it to all neighbours, the neighbours relay it to their neighbours except the originator, and so on until the packet is delivered or dropped. Thus, flooding guarantees fast packet delivery with minimum computation at the intermediate nodes. However, pure flooding uses considerable network resources.

Despite its overhead, flooding is popular in both traditional wireline networks and mobile wireless networks. Flooding can be used on its own or as part of other routing mechanisms. As an independent routing mechanism, flooding is used in networks with frequent topological changes such as military communication networks and applications that require entire network database updates [Top89].

Flooding is used with other routing protocols for broadcasting topology update information or determining the possible paths for demand-based routing algorithms. For instance, routing protocols based on link state algorithms use flooding for broadcasting topology information to the entire nodes in the network. Existing standard Internet routing protocols that use flooding include OSPF, IS-IS, NLSP and IDPR (see [Est93, ISO89, Moy94b, Wyc95]). Most currently proposed routing protocols for mobile ad hoc networks use flooding for propagating control messages (such as route discovery) and update messages (for route maintenance). For example, see [Mar97, John96, Dub97].

5.1.3 Flooding control mechanisms

Pure flooding generates many packets which may degrade network performance. To limit the growth of flooding traffic and avoid packet duplication, various flooding control mechanisms have been proposed. These include sequence numbers, number of hops and an age field (also known as Time To Live).

Sequence numbers. Sequence numbers can be used to control flooding traffic as well as to validate update information in the existing routing protocols. To avoid duplicates in flooding, the source node ID and the sequence number are included on each packet generated. The sequence number is incremented with each new packet sent. The sequence number field is finite but is large enough so that the maximum sequence number is reached only rarely. By keeping the highest sequence number received for each source node and

not forwarding packets with sequence numbers that are less than or equal to the one stored in the forwarding table, a node can avoid duplicates. Thus, sequence numbers guarantee that each packet is transmitted to neighbours at most once by each intermediate node.

The age field. Each node visited by a message records its arrival time and increments its age field before transmitting it to its neighbours. A message whose age exceeds a pre-specified limit is not transmitted. This technique is often combined with sequence numbers. But the use of sequence numbers alone can prevent the transmission of aged packets [Beh97b].

Hop count. A hop count shows the length of a path from the source node to the destination node. Each packet's hop count is decremented by one when it is forwarded to another node. Flooding terminates when the hop count reaches zero. The hop count is initialised to the distance between source and destination nodes. If this figure is not available, the diameter of the entire network can be used as a worst-case estimate.

5.2 Description of Limited Flooding Protocols

5.2.1 Motivation

If the rate of topological change in a mobile ad hoc network is low, the traditional routing algorithms can be used for packet routing. For, when the host movement is very slow or infrequent, no overhead is incurred in routing since routing information changes slowly.

In a very highly mobile environment, the traditional routing protocols fail to provide reliable communication between the mobile hosts. For example, in proactive routing protocols the computed routes might not be utilised due to frequent changes in network topology. So both route computation and storage of computed routes at the nodes waste resources. On the other hand, reactive protocols may incur high communication overhead if the rate of topological change is high, since the algorithm will be executed too often. Moreover, the discovered routes may even not be usable since the mobile might have moved to another location by the time the source sends the packet. Thus when hosts move quickly enough and frequently enough, the best strategy is to flood data packets through the network [John96].

In this chapter we consider a relatively highly mobile network environment with unpredictable changes in the network topology and network resources. We propose a protocol which lies between the two extremes, pure flooding and shortest path routing, but closer to flooding. Flooding is robust

to operate in a mobile network environment regardless of mobility, since forwarding packets to destinations is made through all possible routes at the same time. But some improvements are needed to further limit the flooding traffic.

We introduce additional flooding control mechanisms to minimise the excessive use of network resources. We call our protocol *Limited Flooding* and it is described below. For relatively highly mobile environments, the limited flooding protocol for infrequently communicating nodes is expected to provide better performance than periodic or event driven updates of link state information in link-state-based protocols. While the network overhead in limited flooding is per communication request, the overhead of link-state-based protocols is per link-state updates. Also for link-state-based protocols, link-state packets are transmitted throughout the network through pure flooding every time the link-state is updated whereas in limited flooding it is transmitted only along few selected paths. The protocol also consumes less resources than pure flooding algorithm while enabling more reliable communication and in some cases resulting in the balanced use of the scarce wireless resources.

5.2.2 Protocol overview

To gain the advantages of both randomisation and multiple paths, we propose a randomised multipath routing protocol. The protocol uses the basic flooding algorithm to limit the extent of flooding. It further reduces communication overhead by using only a few selected links from each node for message propagation.

```
The Limited Flooding Algorithm

On receiving a packet, if destination is in the
neighbour's list then deliver the packet.
Otherwise, perform the following steps.
Step 1.  Determine the  degree of node;
step 2:  Determine the number_of_links chosen;
          (see section 5.2.3)
Step 3:  Determine the actual links;
          (see section 5.2.3)
Step 4:  For all chosen links
          forward the packet via the intermediate nodes
```

For optimum bandwidth utilisation, it is necessary to determine the number of links needed for packet forwarding at each intermediate node. The use of too many links may decrease latency at the expense of bandwidth utilisation since there will be high message propagation. On the other hand, using too few links may increase latency with less bandwidth utilisation due to slow propagation of packets in the network. Thus, there is a trade-off between reduction of message propagation and ensuring message delivery.

The features common to all the limited flooding protocols are: First, route computation or maintenance is not required. Each node only knows its neighbours that it uses to forward packets along and no other information such as a routing table is needed. Second, no central controlling mechanism is needed for the operation of the protocol. Each node assigns a sequence number to each packet it generates to avoid duplicates. Third, the intermediate nodes do not perform any complex operation. They only forward packets to all other nodes selected by the protocol, but at most once.

5.2.3 Variations

The proposed protocol involves two stages. In stage one, the number of outgoing links are determined. In stage two, the actual links are selected along which the packets are forwarded. The number of outgoing links are obtained from the node's degree based on some deterministic, random or priority based method as follows. We propose three categories of the Limited Flooding Protocol.

1. *Randomised Limited Flooding Protocol.*

   In this protocol, the number of outgoing links is randomly determined at run time. It is chosen between one and the total number of neighbours.

2. *Deterministic Limited Flooding Protocol.*

   In this protocol, a proportion of outgoing links are chosen at each intermediate node based on the node's degree. This proportion is a fixed parameter that depends only on the degree of the intermediate node. The choice of larger number of links enables faster and more reliable delivery than fewer links.

3. *Prioritised Limited Flooding Protocol*

   In this protocol, the number of links selected depends on whether the packet has a higher priority or lower priority. For high priority packets, more links are used. High priority packets may overload the network

but there is reliable delivery. In some cases one may even use pure flooding.

All the protocols are further divided into two depending on whether the actual links are determined randomly or deterministically. So there are six variations: Randomised Limited Flooding with randomised paths (RLF-R), Randomised Limited Flooding with likely paths (RLF-L), Deterministic Limited Flooding with randomised paths (DLF-R), Deterministic Limited Flooding with likely paths (DLF-L), Prioritised Limited Flooding with randomised paths (PLF-R) and Prioritised Limited Flooding with likely paths (PLF-L). The most recently used links are used to determine the likely paths. Comparison results of all limited flooding protocols are discussed in chapter 6 using simulation.

Randomisation has important advantages. The first is that random determination of links for packet propagation is expected to improve the balanced use of available bandwidth. The other is that randomisation reduces congestion and hence may minimise latency and packet loss.

5.2.4 Protocol applicability and assumptions

The main assumptions underlying the proposed protocols are: First, the underlying data link layer protocol ensures that each node is aware of the status of its links to neighbours. Thus, MAC provides immediate neighbour connectivity information regarding whether the node establishes connection or is disconnected. Second, the intermediate nodes have sufficient buffer size for each incoming link and infinite queues for each outgoing link. Furthermore, nodes can initiate different messages simultaneously to any subset of nodes and messages can traverse a given link in different directions. A node can transmit and receive at the same time but on a different channel.

Limited Flooding Protocol (LFP) is suitable for relatively highly mobile network environments with a random network characterised by relatively low communication traffic. The protocols should be applicable in areas including military communication networks, and vehicular speed applications such as mobile nodes on vehicles, ships, airplanes and the like. Moreover, the protocol has the following features:

- It has no dependency on a central entity for routing. Each node performs partial flooding when sending packets.
- It supports bi-directional links. But, these links may be composed of two unidirectional links.

- It functions reactively by performing limited packet propagation in the network. It does not use source routing.
- It uses sequence numbers but does not require the use of reliable or sequenced packet delivery.

5.3 Related Protocols

The advantage of pure flooding is that it avoids network delay by quickly delivering packets with minimal en-route computation. To reduce communication overhead, a Controlled Flooding protocol was proposed by Lesser in [Les90] for communication networks. In this protocol, a message is broadcast based on the basic flooding mechanism but not throughout the network. The extent of flooding traffic is further limited by assigning a weight (cost) to each link and a wealth to each message. The wealth assignment is made at the source node. A message is sent on a link only if its wealth is at least the link weight. Upon receiving a message, an intermediate node deducts the cost of the link from the message wealth.

The performance of the protocol is determined by the procedure used to assign the link costs. The weight assigned to each link is drawn from the exponential range. The problem of assigning a weight to each link to achieve optimal performance was investigated. A heuristic algorithm has been proposed for optimal weight assignment in heterogeneous networks. It has also been shown that for such a scheme to be optimal, the shortest path between every pair of nodes must be unique.

A similar routing protocol was proposed by Azar [Aza96] for high speed networks. Here, the communication link was assumed to be of high capacity and the aim was to minimise the processing overhead at the intermediate nodes. Various techniques of weight assignments were investigated to achieve optimal performance. However, it was shown that the weight assignment does not result in balanced use of network resources.

The functionality of our limited flooding protocol is similar to the controlled flooding proposed by Azar and Lesser. However, we use more adaptive mechanisms for restricting the scope of flooding in the network. The static weight assignment mechanisms proposed for fixed communication networks are not suitable for mobile ad hoc networks.

A reliable broadcast protocol has been proposed by Pagani in [Pag97] for networking environments where the rate of topology change is difficult to predict and requires routing algorithms that range between flooding and

the traditional routing protocols. The protocol is based on a multi-hop multicluster architecture proposed in [Ger95a]. This protocol switches to the pure flooding algorithm when the rate of mobility is high. Since the mobility rate is not uniform over all the hosts, the protocol adapts to the situations encountered at each node.

Chapter 6

Simulation Environment for Limited Flooding

In this chapter, we model the mobile ad hoc networks and determine the effect of various simulation parameters on the relative performance of the limited flooding protocol and pure flooding in the presence of node mobility. Some of these paramters are treated as variables while others are fixed in the simulation experiments.

6.1 An Overview of the Simulation Environment

6.1.1 The Network Model

We have modelled a mobile ad hoc network as an undirected graph with a finite set of nodes and links. Each node has a unique identifier and represents a mobile host capable of forwarding packets to its neighbours. The links are wireless communication paths between the mobile nodes. Each mobile node has a fixed communication area known as its transmission range. Two hosts within the communication range of each other are said to be neighbours.

An undirected link (j,k) connecting two nodes j and k is formed when the distance between j and k becomes at most the transmission range. When node j and k move out of the transmission range of each other, the link (j,k) is removed and the nodes cannot communicate directly but can do so via intermediate nodes. The number of neighbours of a mobile node is the degree of that node.

6.1.2 Assumptions

The main assumptions made in the simulation are:

- Nodes are relatively highly mobile.
- The processing time of data packets is negligible compared to the communication time.
- The transmitter and receiver have an infinite buffer. The packet is kept in the buffer until the transmitter is ready to send or the receiver is ready to receive.
- There is a separate up-link and down-link channel so that a node can receive and transmit at the same. However, a node can receive or transmit at most one packet at a time.

6.2 The Simulation Language

We have used PARallel Simulation Environment for Complex systems (PARSEC) for simulation of our routing protocols. PARSEC is a C-based simulation language developed by the Parallel Computing Laboratory at UCLA [Bag98] for sequential and parallel execution of discrete-event simulation models. PARSEC adopts the process interaction approach to discrete-event simulation. An object or a set of objects in the physical system is represented by a logical process. Interaction among physical processes is modelled by time-stamped message exchanges among the corresponding logical processes.

PARSEC is designed to cleanly separate the description of a simulation model from the underlying simulation protocol, sequential or parallel, used to execute it. It also provides powerful message-receiving constructs that result in shorter and more natural simulation programs. Several simulation languages and tools are currently available, commercially or on public domain, for network simulations. However, we chose PARSEC as the simulation language mainly due to the following reasons.

First, it offers good support for high-level simulations, since it has an easy and clean message-passing infrastructure. Second, at the time of decision-making, it was the only simulation language used for mobile wireless network simulation and freely available for academic and research purposes. Third, PARSEC and its predecessor Maisie have been widely used for network simulation including routing in mobile wireless networks. Several simulation libraries and tools based on PARSEC are currently under development by universities and commercial companies for mobile network simulations.

6.3 The Simulation Entities

The simulation environment built using PARSEC consists of the mobility entity, the channel entity, the node entity and communication entity as shown in figure 6.1. In PARSEC, the driver entity serves the same purpose as the main() function in the C programming language. In the simulation, the driver entity creates all other entities. It also collects various statistics information needed for performance evaluation.

PARSEC entities communicate with one another through message passing. For example, a message is sent by the communication model to the source node to initiate sending a packet to a randomly selected destination node. A source node can then send a message to the destination node upon receiving a send signal from the communication model. The destination sends an acknowledgement message to the source upon the receipt of a packet sent to it. Also a periodic Hello message is sent by every node to its neighbours to indicate the existence of connectivity. The messages used in the simulation can be broadly classified into three categories, namely data, acknowledgement and routing messages.

6.3.1 The mobility model

There is no standard mobility model for mobile ad hoc networks yet. Some authors of mobile ad hoc routing protocols have modelled mobile networks using fixed networks with unreliable links [Mur96, Per94, Par97a, Cor95]. They have simulated the performance of their protocols assuming a higher link failure rate. In [Zon97, San98], the random mobility model is used to model a mobile network. In this model, the current speed and direction of motion are independent of the previous values. Such a model may generate unrealistic motion behaviour due to randomness.

We simulate the mobile networks using a truly mobile environment. We use a modified Random Waypoint mobility model proposed by Johnson in [John96], which is an extension of a random walk. In this model, the movement of a mobile node is random with repeated pause and motion periods. It operates as follows:

- Each node moves in a straight line in a random direction at a certain fixed or randomly chosen speed for a randomly chosen length of time.
- After arriving at a new location a node waits for certain random *pause* time and start moving again.

6.3.2 The Mobility entity

The simulation is conducted with mobile nodes randomly distributed on a grid of 1 unit by 1 unit square. Two nodes A and B are said to be neighbours if the Euclidean distance between them in the grid is less than the transmission range.

The mobility entity provides two functions. First, it tracks the location of the mobile node. The location information is maintained in a location table. Second, it determines the speed at which a nodes moves, and the direction in which it moves at the end of every pause time. It uses the mobility model described above. All the nodes are stationary before the start of the simulation period. The nodes can then move in the coverage area as described below.

- Each node begins the simulation by choosing randomly between moving and pausing. If moving, it moves in a straight line. Otherwise, it remains stationary at the current location.
- When moving, the speed is chosen randomly from the speed range and direction. The node moves for a specified time chosen randomly from the mobility range.
- When pausing, the node remains stationary for a random period chosen from the pause range.

6.3.3 The Communication entity

In the simulation environment, mobile nodes communicate with each other in a random manner. The communication entity is responsible for initiating communication and handling the data traffic in the network. Every node chooses a destination at random and sends a random number of data packets to it. It then waits for a random amount of time before repeating this operation.

A number of parameters are used in the communication entity. These include:

- *Packets size*: Packets in one communication are either short or long.
- *Packet type*: Two types of packets are generated: high priority and low priority packets. Higher priority packets are sent on more links than lower priority packets.

- *Number of packets*: The number of packets sent to a destination is randomly determined from a uniformly distributed random variable with given minimum and maximum number of packets.
- *Wait time*: Drawn from an exponential range.
- *Data rates*: This is assumed fixed for the simulation and the same for all nodes.

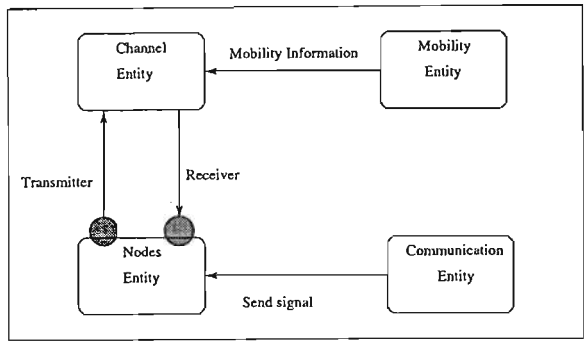

Figure 6.1: Simulation entities and their relationships

6.3.4 The Node entity

The mobile node in the ad hoc network is handled by the node entity which implements the proposed routing protocols. The transmitter and receiver entities are modelled in the node entity to simulate the transmitter and receiver sections of the node. The node depends on the communication entity to initiate sending data packets to a destination. It sends and receives data and acknowledgement packets.

A node also maintains neighbourhood information by sending a Hello message. The node model keeps track of statistics used for performance evaluation. These include the time spent in channel utilisation, number of data packets generated, number of data or acknowledgement packets relayed by the intermediate nodes, and node connectivity information.

6.3.5 The Channel entity

The channel entity is responsible for determining which nodes are able to communicate directly with each other. It models the wireless medium within which the nodes roam. To determine which nodes can communicate, the channel model obtains location information from the location information table.

6.3.6 Simulation parameters

Several parameters are used in the simulation experiments. Table 6.1 summarises these along with the default values.

Parameters	Default Value
Maximum simulation clock	1048576 ms
The area covered by simulation	1 unit X 1 unit
Length of hello messages	2317 bits
Length of the acknowledgment packet	1101 bits
Time after which acknowledgment timeouts	200 ms
Time when next hello message is generated	180 ms
Bandwidth used in the simulation	512 KB
Mobility range	50 to 100 s
Pause range	25 to 50 s
Speed range	0.2 to 0.5 unit/s
Packets range	0 to 10
Length of short packets	1049 to 2359 bits
Length of long packets	1049 to 13107 bits
Percentage small packets	50 %
Percentage large packets	50 %

Table 6.1: Simulation parameters

The two most significant parameters are network size and transmission range. Therefore, we treat these as variables in the simulation experiments.

Transmission range. The transmission range of the transceiver in an ad hoc network is smaller than the network coverage area. But since each node can act as a router, packets can be relayed along the neighbours. This parameter will be denoted by Tx.

Network size. We investigated the effect of increasing network size on the performance. This parameter will be denoted by N.

6.4 Discussion of the Results

This section presents a discussion of results of the simulation experiments. We have made several comparisons between Pure Flooding Protocol (PFP)

and all variations of limited flooding protocols. The simulation experiments are conducted in three scenarios as described below.

- *Scenario 1*: In this scenario, comparisons of each limited flooding protocol is made with pure flooding. The simulation results are presented in sections 6.4.2–6.4.4.
- *Scenario 2*: In this scenario, groups of limited flooding protocols are compared. Two sets of experiments are conducted. In the first set, the protocols based on random selection of communication paths are investigated. In the second set, the protocols based on deterministic selection of communication paths are investigated. The results are presented in sections 6.4.5 and 6.4.6.
- *Scenario 3*: In this scenario, comparisons of all limited flooding protocols are made with each other and with pure flooding. The results are presented in section 6.4.7.

Each performance metric is investigated separately as a function of transmission range and network size. The transmission range is varied between 10 % and 50 %. If transmission range is very high then virtually every node can hear every other nodes so the network is virtually complete (and thus uninteresting to us). The number of mobile nodes is varied between 5 and 20.

6.4.1 Performance metrics

Several parameters are identified for comparison of the limited flooding protocols with pure flooding.

1. *Mean Packet Relaying Load*: This is the average number of packets relayed per nodes in the network. This metric measures the mean traffic overhead that occurs at the intermediate nodes during packet forwarding. Thus, it indicates the extent to which a limited flooding protocol reduces the relaying overhead.
2. *Mean Channel Utilisation*: This is the average percentage channel utilisation by packet transmission. This indicates the reduction in channel utilisation made by a limited flooding protocol and thereby minimising bandwidth utilisation. The higher the utilisation, the greater the wireless channel utilisation overhead and the greater the congestion.

3. *Normalised Packet Delivery Ratio*: The average delivery ratio is:

$$MDR = \frac{\text{Total number of packets successfully received}}{\text{Total number of packets sent}}$$

The Normalised Packet Delivery Ratio (NPDR) for a given limited flooding protocol is:

$$NPDR = \frac{\text{Mean delivery ratio of this protocol}}{\text{Mean delivery ratio of pure flooding}}$$

This is a unit-free measure that indicates how close are the packet delivery capabilities of a limited flooding protocol and pure flooding.

6.4.2 Deterministic Limited Flooding

Simulation results of Deterministic Limited Flooding with Random paths (DLF-R) and Deterministic Limited Flooding with likely paths (DLF-L) are presented in this section. We have investigated the effects of transmission range and network size on the performance of the simulation metrics. Figures 6.2-6.4 present simulation results as a function of transmission range whereas figures 6.5–6.7 present the results obtained as a function of network size.

Mean number of packets relayed. Figure 6.2 shows that when the transmission range increases, the number of packets relayed also increases, but at a much lower rate than PFP. Figure 6.5 shows that the packet relaying overhead rises slightly with the number of nodes. But it is lower both in DLF-R and DLF-L compared to PFP. The DLF-R protocol relatively has the lowest packet relaying overhead.

Mean channel utilisation. Figure 6.3 shows that as transmission range increases, channel utilisation slightly increases but both limited flooding protocols are lower than PFP. Figure 6.6 shows that PFP has higher channel utilisation compared to both DLF-R and DLF-L protocols. The DLF-R and DLF-L protocols have similar performance but with DLF-L using slightly lower channel resources at all transmission ranges. The low channel utilisation shows that only few nodes can communicate due to smaller transmission ranges.

Normalised packet delivery ratio. By definition, this ratio for flooding is 100% in the simulation experiments. Figure 6.4 shows that the packet delivery increases with increase in transmission range with higher values observed at larger transmission range. The results indicate that DLF-R protocol has better overall performance compared to DLF-L and PFP. Figure 6.7 shows that DLF-L has slightly better packet delivery than DLF-R at smaller and larger network sizes.

Label prefix	Corresponding protocol
lf2*.dat	DLF-R protocol
lf3*.dat	DLF-L protocol
pf*.dat	PFP protocol

Table 6.2: Notations

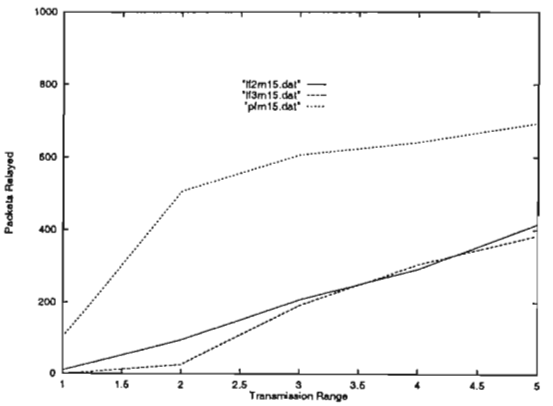

Figure 6.2: Packets relayed: N = 15

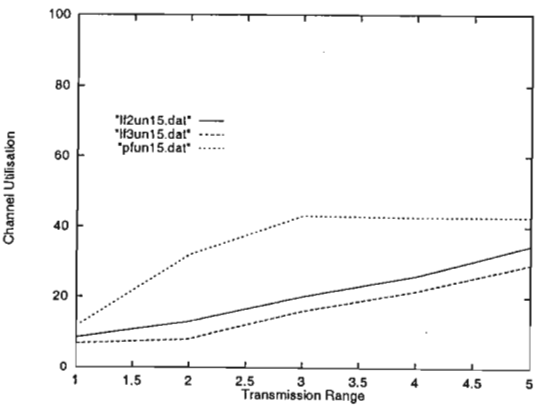

Figure 6.3: Channel utilisation: N = 15

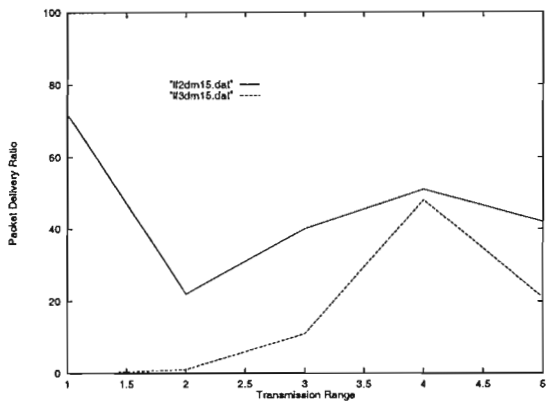

Figure 6.4: Normalised packet delivery ratio: N = 15

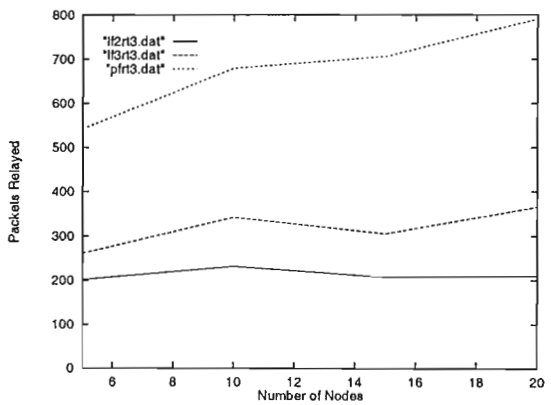

Figure 6.5: Packets relayed: Tx = 30% .

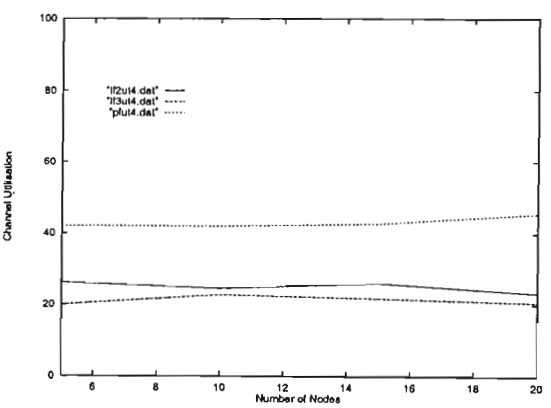

Figure 6.6: Channel utilisation: Tx = 30%.

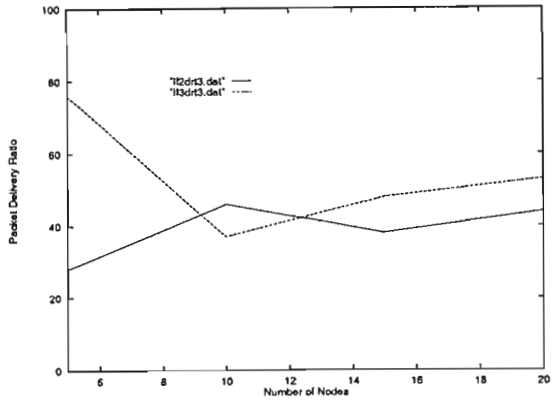

Figure 6.7: Normalised packet delivery ratio: Tx = 30%.

6.4.3 Prioritised Limited Flooding

In this section, the simulation results of Prioritised Limited Flooding with Random paths (PLF-R) and Prioritised Limited Flooding with Likely paths (PLF-L) are briefly described. Figures 6.8–6.10 present the simulation results as a function of transmission range whereas results as a function of network size are presented in figures 6.11–6.13.

Mean number of packets relayed. Figure 6.8 shows that limited flooding protocols have reduced relaying load with significant reduction made at a relatively low transmission range. Figure 6.11 shows that the relaying load is reduced in both variations of limited flooding. The PLF-R protocol has the lowest relaying load.

Normalised packet delivery ratio. Figure 6.10 shows that the packet delivery ratio is higher for PLF-R compared to PLF-L. In both protocols, it increases with increase in transmission range attaining a maximum value at higher value of transmission range. Figure 6.12 shows that PLF-R has a greater packet delivery than PLF-L. There is better delivery at smaller network size.

Mean channel utilisation. Figure 6.9 shows that with the increase in transmission range, channel utilisation increases slightly in PLF-R and PLF-L protocols, but both are lower than PFP. There is little difference between the limited flooding protocols. Figure 6.13 shows the channel utilisation increases slightly with increase in number of nodes. The results indicate that limited flooding protocols reduce the channel utilisation overhead compared to PFP. Also, PLF-L is better than a PLF-R as network size increases.

Label prefix	Corresponding protocol
lf5*.dat	PLF-R protocol
lf6*.dat	PLF-L protocol
pf*.dat	PFP protocol

Table 6.3: Notations

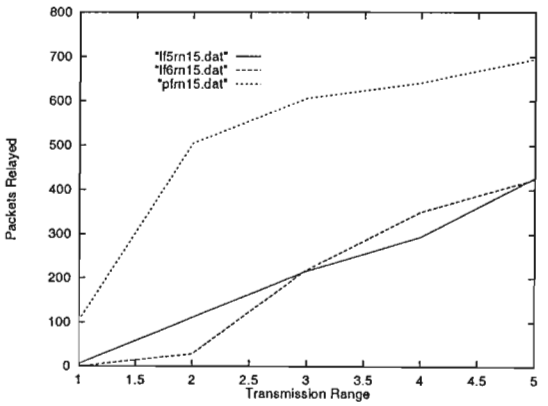

Figure 6.8: Packets relayed: N = 15

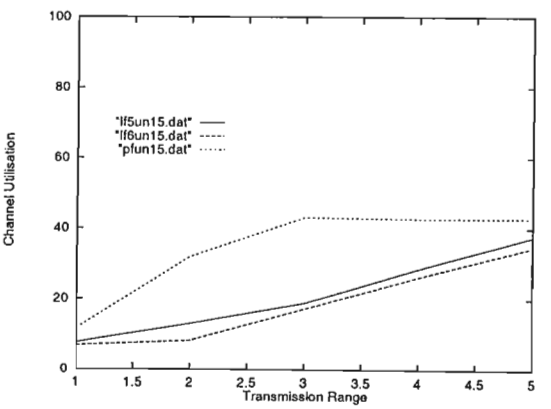

Figure 6.9: Channel utilisation: N = 15

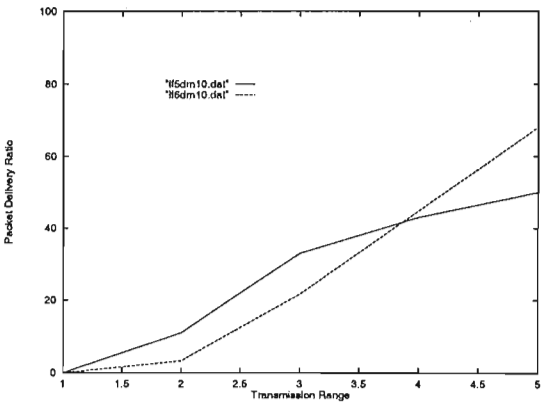

Figure 6.10: Normalised packet delivery ratio: N = 15

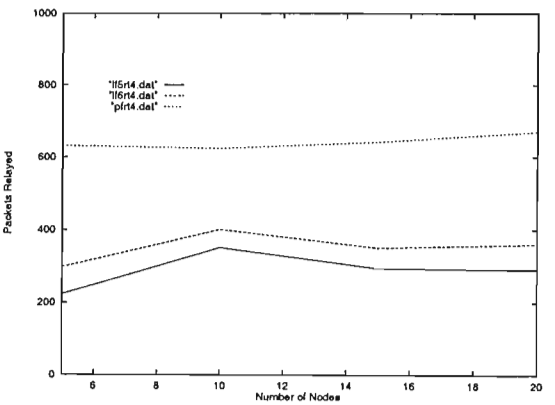

Figure 6.11: Packets relayed: Tx = 40%.

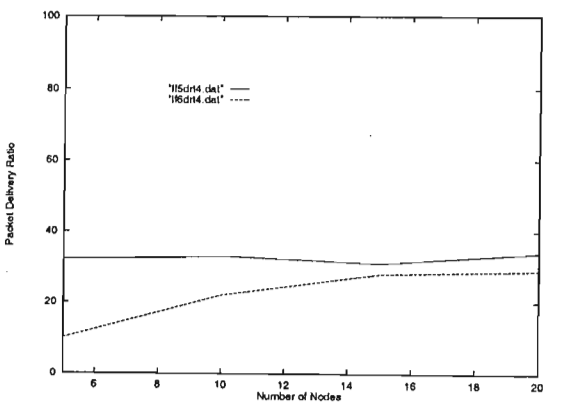

Figure 6.12: Normalised packet delivery ratio: Tx = 40%.

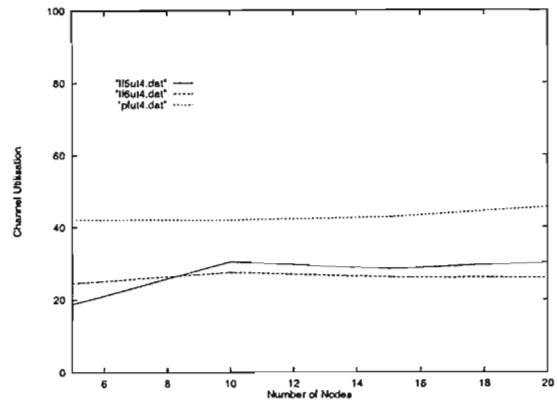

Figure 6.13: Channel utilisation: Tx = 40%.

6.4.4 Randomised Limited Flooding

Comparison of Randomised Limited Flooding with Random paths (RLF-R) and Randomised Limited Flooding with likely paths (RLF-L) is presented in this section. Figures 6.14–6.16 present the simulation results as a function of transmission range. The corresponding results as a function of number of nodes are presented in figures 6.17–6.19.

Mean number of packets relayed. Figure 6.14 shows the number of packets relayed is much higher in pure flooding than limited flooding protocols. The results also show that RLF-R greatly reduces relaying overhead compared to RLF-L. The difference between limited flooding protocols and PFP is higher with varying transmission range compared to varying number of nodes. Figure 6.17 shows that the relaying load is higher in PFP compared to limited flooding. The difference in the relaying load increases with increase in network size. RLF-R greatly reduces the relaying overhead compared to RLF-L.

Normalised packet delivery ratio. Figure 6.16 shows that packet delivery ratio increases with increase in transmission range. A better packet delivery is attained at higher transmission range in the RLF-L protocol. Figure 6.19 shows that the packed delivery ratio increases with increase in number of nodes until it reaches a maximum value and then declines. RLF-L has a greater packet delivery ratio than RLF-R.

Mean channel utilisation. Figure 6.15 shows the channel utilisation as a function of transmission range. Similar results are noted with varying transmission range, though RLF-R consumes slightly less resources than RLF-L and PFP. Figure 6.18 shows that channel utilisation increases with the increase in network size. Randomised limited flooding protocol consumes less

wireless channel resources than PFP, with RLF-R consuming relatively the lowest channel resources. Thus randomisation can help to balance distribution of channel usage.

Label prefix	**Corresponding protocol**
lf1*.dat	RLF-R protocol
lf4*.dat	RLF-L protocol
pf*.dat	PFP protocol

Table 6.4: Notations

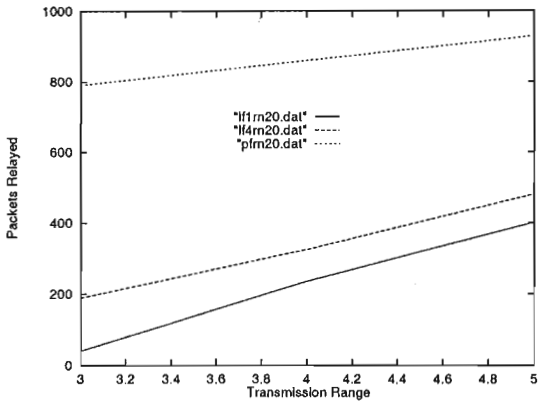

Figure 6.14: Packets relayed: N = 20

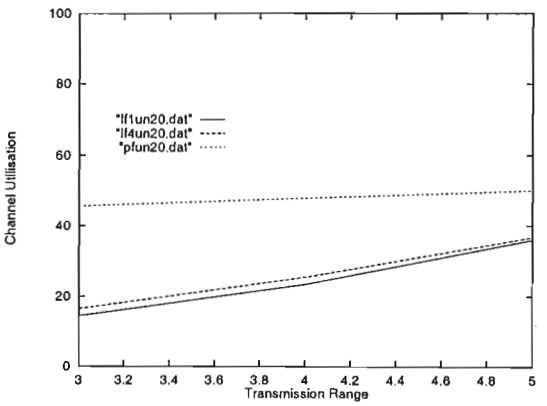

Figure 6.15: Channel utilisation: N =20

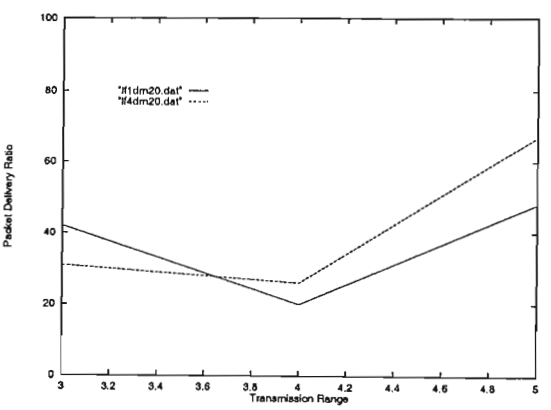

Figure 6.16: Normalised packet delivery ratio: N = 20

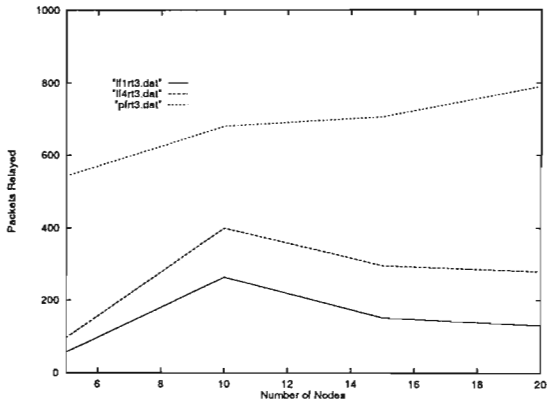

Figure 6.17: Packets relayed: Tx = 30%.

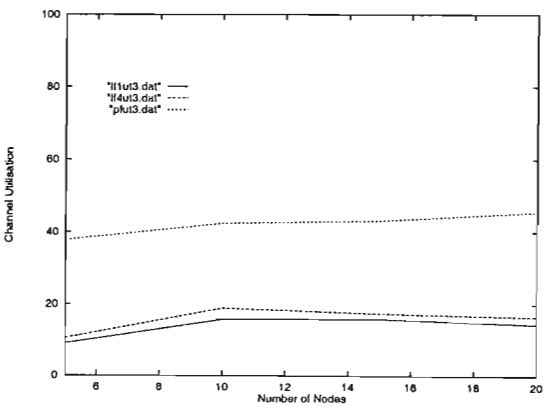

Figure 6.18: Channel utilisation: Tx = 30%.

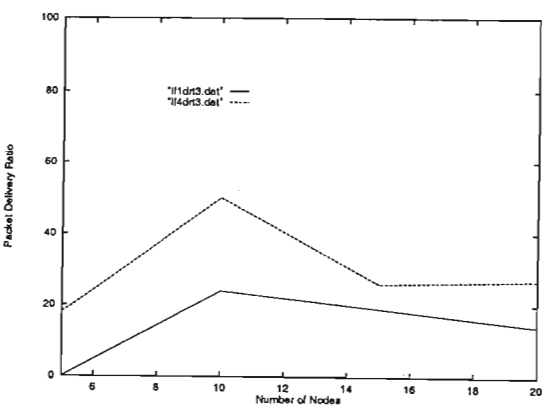

Figure 6.19: Normalised packet delivery ratio: Tx = 30%.

6.4.5 Comparisons of limited flooding: Randomised paths

In this section, we compare the three randomised limited flooding protocols namely, RLF-R, DLF-R and PLF-R. The simulation results as a function of network size are presented in figures 6.20-6.22 and in figures 6.23–6.25 the corresponding results as a function of transmission range are presented. In the simulations, transmission range of 40% units is used with varying number of nodes, whereas 10 mobile nodes are used with varying transmission range.

Mean number of packets relayed. Figure 6.20 shows that the relaying load is highest for pure flooding and increases with number of nodes. It decreases with the increase in the network size for limited flooding. This indicates that the limited flooding protocol is more scalable than pure flooding. Also, RLF-R relatively reduces relaying overhead compared to the other protocols. Figure 6.23 shows the mean number of packets relayed as a function of transmission range; all these limited flooding protocols reduce packet relaying overhead. Also, RLF-R greatly reduces packet relaying compared to other protocols.

Mean channel utilisation. Figure 6.21 shows that all limited flooding protocols consume less wireless channel resources than PFP, with RLF-R consuming the lowest. The wireless channel resource consumption rises with increase in number of nodes for some protocols. Figure 6.24 shows that with increase in transmission range, RLF-R consumes relatively the lowest channel resources whereas PFP is the highest.

Normalised packet delivery ratio. Figure 6.22 shows that packet delivery increases with increase in number of nodes until it reaches a maximum value for RLF-R and DLF-R. RLF-R has a better packet delivery than the other protocols. Figure 6.25 shows that in limited flooding, normalised packet delivery ratio increases with increase in transmission range. The higher normalised packet delivery ratio is achieved by PLF-R and RLF-R compared to DLF-R.

Label prefix	Corresponding protocol
lf1*.dat	RLF-R protocol
lf2*.dat	DLF-R protocol
lf5*.dat	PLF-R protocol
pf*.dat	PFP protocol

Table 6.5: Notations

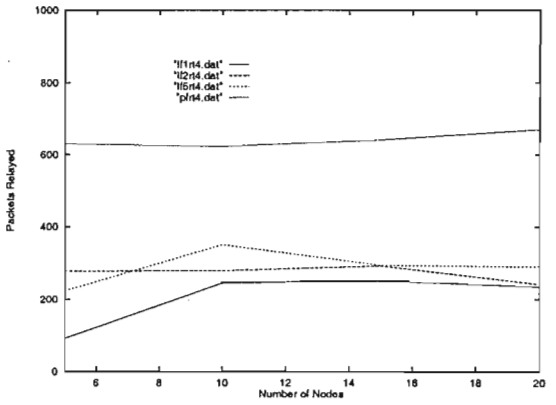

Figure 6.20: Packets relayed: Tx = 40%

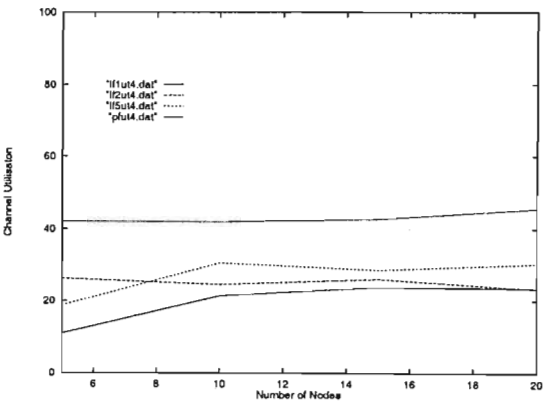

Figure 6.21: Channel utilisation: Tx = 40%

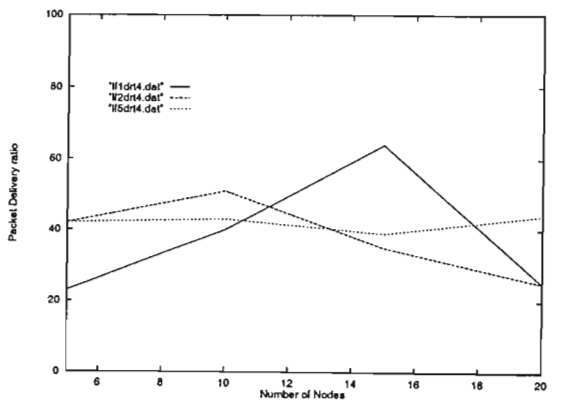

Figure 6.22: Normalised packet delivery ratio: Tx = 40%

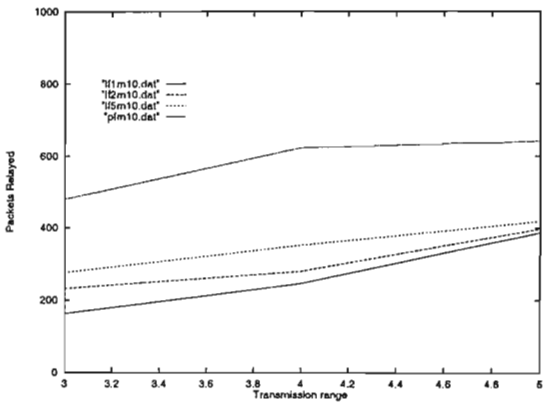

Figure 6.23: Packets relayed: N = 10

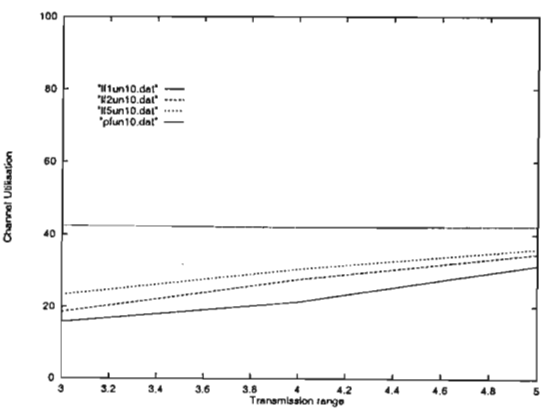

Figure 6.24: Channel utilisation: N = 10

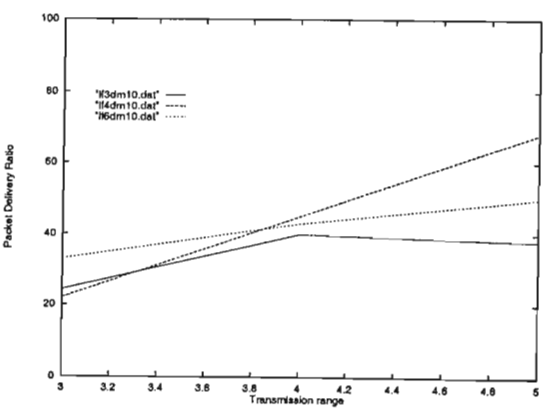

Figure 6.25: Normalised packet delivery ratio: N = 10

6.4.6 Comparisons of limited flooding: Likely paths

A comparison between RLF-L, DLF-L and PLF-L is presented in this section. We investigate the effect of network size and transmission range on the performance metrics. The simulation results as a function of network size are presented in figures 6.26–6.28 while figures 6.29–6.31 present the results obtained as a function of transmission range.

Mean number of packets relayed. Figure 6.26 shows that the relaying load at the intermediate node rises with increase in the number of nodes for pure flooding. The packet relaying load is low on the average in all limited flooding protocols investigated, in comparison with PFP. The packet relaying load decreases with increase in the network size for limited flooding protocols except DLF-L which shows a slight increase. The reason is that as network size increase, the load is shared among more nodes. The mean number of packets relayed as a function of transmission range is shown in figure 6.29. When the transmission range increases, the number of packets relayed also increases, but at a much lower rate than PFP. The results indicate that all limited flooding protocols have low relaying overhead, with DLF-L relatively having the lowest followed by PFL-L.

Mean channel utilisation. Figure 6.27 shows that PFP has higher channel utilisation than all the limited flooding protocols. RLF-L and PLF-L have the lowest channel utilisation than other protocols. Figure 6.30 shows that with increase in transmission range, channel utilisation has slightly increased but both lower than flooding, while DLF-L using slightly lower resources. The other limited flooding protocols produce similar results.

Normalised packet delivery ratio. Figure 6.28 shows the simulation results for normalised packet delivery ratio as a function of the number of nodes. The results indicate that DLF-L and RLF-L have slightly better packet delivery ratio. Figure 6.31 shows that the packet delivery ratio increases with increase in transmission range with higher values achieved at larger transmission range. The RLF-L protocol has better delivery ratio at higher transmission range than other protocols.

Label prefix	Corresponding protocol
lf3*.dat	DLF-L protocol
lf4*.dat	RLF-L protocol
lf6*.dat	PLF-L protocol
pf*.dat	PFP protocol

Table 6.6: Notations

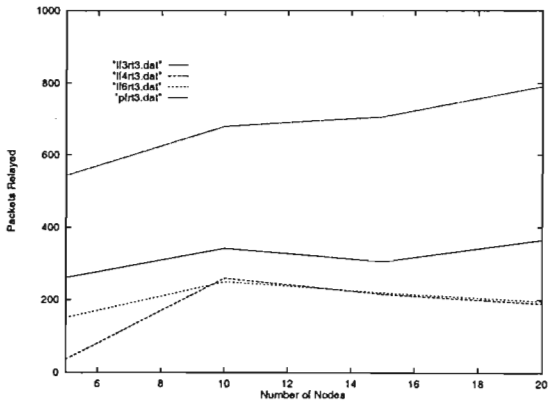

Figure 6.26: Packets relayed: Tx = 30%

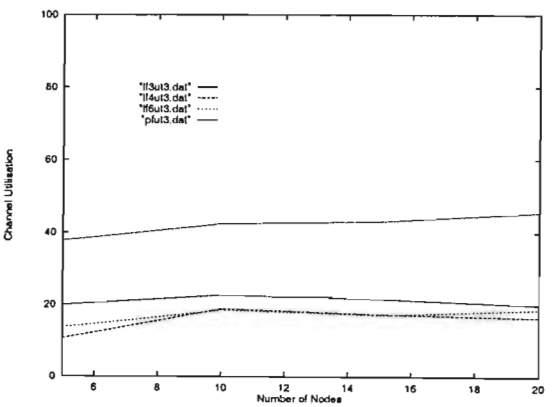

Figure 6.27: Channel utilisation: Tx = 30%

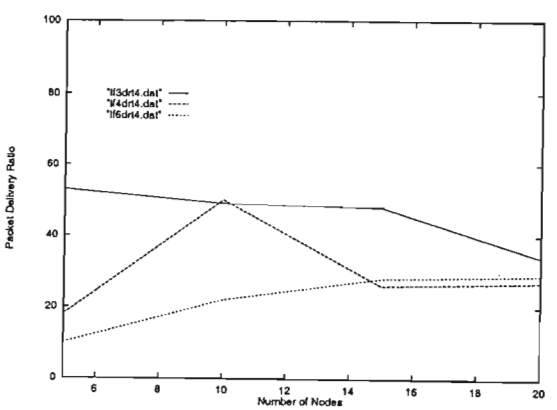

Figure 6.28: Normalised packet delivery ratio : Tx = 30%

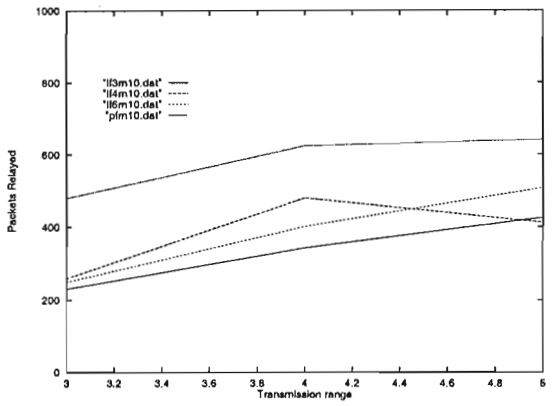

Figure 6.29: Packets relayed: N = 10

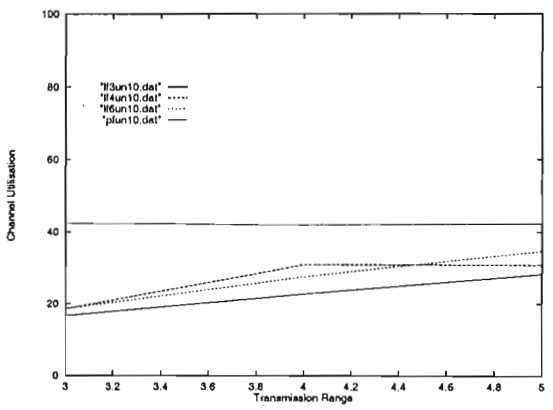

Figure 6.30: Channel utilisation: N = 10

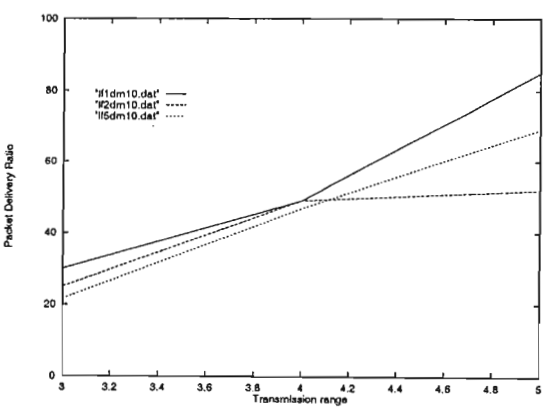

Figure 6.31: Normalised packet delivery ratio: N = 10

6.4.7 Comparisons of all protocols

In this section we discuss the simulation results for the comparisons made among all the protocols. Figures 6.32–6.34 present the results obtained as a function of network size while figures 6.35–6.37 present the results as a function of transmission range. Results presented in this section are based on transmission range of 40% with number of nodes varying between 5 and 20 nodes. Similarly for simulation conducted as a function of transmission range, we have used 10, 15 or 20 mobile nodes.

Mean number of packets relayed. Figure 6.32 shows that the relaying load is lower in all limited flooding protocols than PFP. The results also show that RLF-R and DLF-R have lower relaying overhead than other proposed protocols. Figure 6.35 shows that the number of packets relayed is much higher in PFP than limited flooding protocols. RLF-R and DLF-R greatly reduce relaying overhead compared to the other limited flooding protocols.

Mean channel utilisation. Figure 6.33 shows that mean channel utilisation increases with increase in transmission range for pure flooding and some limited flooding protocols. The decrease noticed for other protocols indicate that the protocols relativly use less wireless channels by sharing the network load among all the nodes. All limited flooding protocols consume less wireless channel resources than PFP. Among the simulated protocols, RLF-R and DLF-L relatively consume less channel resources. Figure 6.36 shows the mean channel utilisation as a function of transmission range. From the results obtained as a function of transmission range, similar results have been observed for all limited flooding protocols. But RLF-R and DLF-L slightly consume less channel resources.

Normalised Packet Delivery Ratio. Figure 6.34 shows that the packet delivery ratio increases with increase in number of nodes and then decreases for some protocols. The decreasing trend is also noted with increase in number of nodes. The results also show that RLF-R, RLF-L and DLF-L have higher packet delivery than the other protocols. The results as a function of transmission range shown in figure 6.37 have indicated that the packet delivery increases with increase in transmission range. A higher packet delivery is attained at larger transmission range. It can be seen from the figure that RLF-R, PLF-R and RLF-L perform better than the other protocols at larger transmission range.

Label prefix	Corresponding protocol
lf1*.dat	RLF-R protocol
lf4*.dat	RLF-L protocol
lf2*.dat	DLF-R protocol
lf3*.dat	DLF-L protocol
lf5*.dat	PLF-R protocol
lf6*.dat	PLF-L protocol
pf*.dat	PFP protocol

Table 6.7: Notations

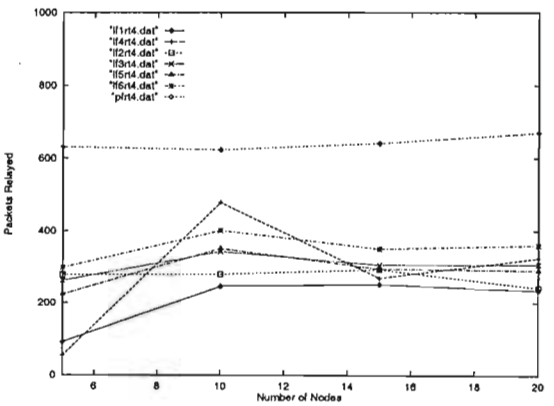

Figure 6.32: Packets relayed: Tx = 40%

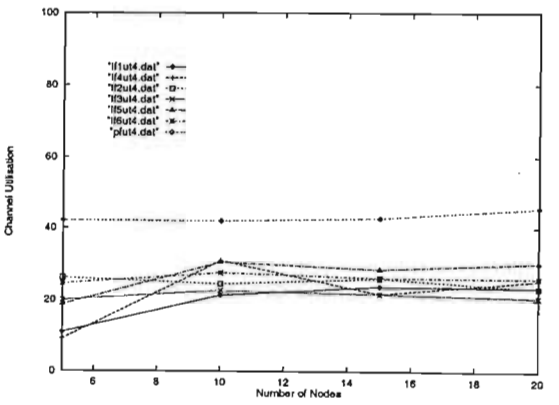

Figure 6.33: Channel utilisation: Tx = 40%

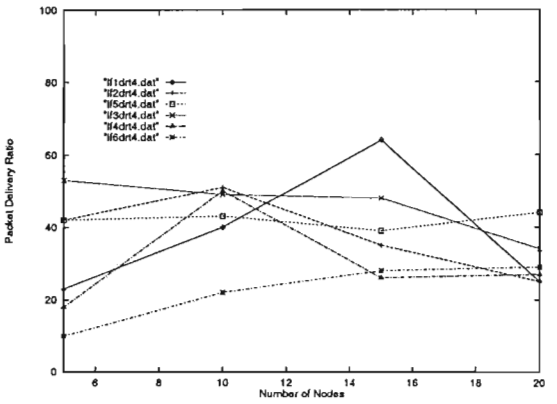

Figure 6.34: Packet Delivery ratio: Tx = 40%

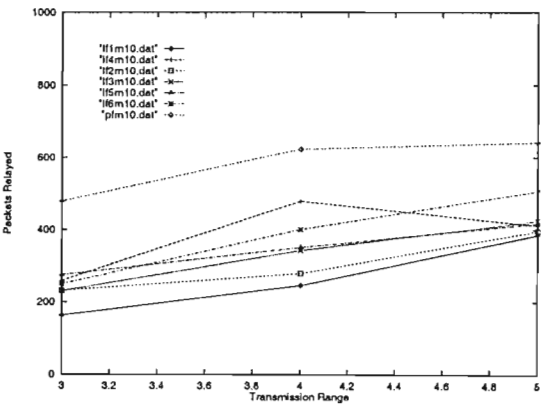

Figure 6.35: Packets relayed: N = 10

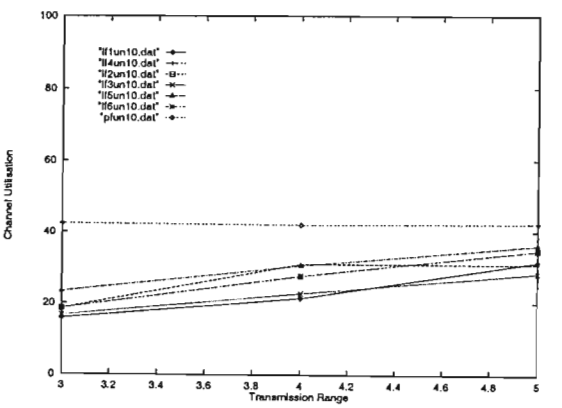

Figure 6.36: Channel utilisation: N = 10

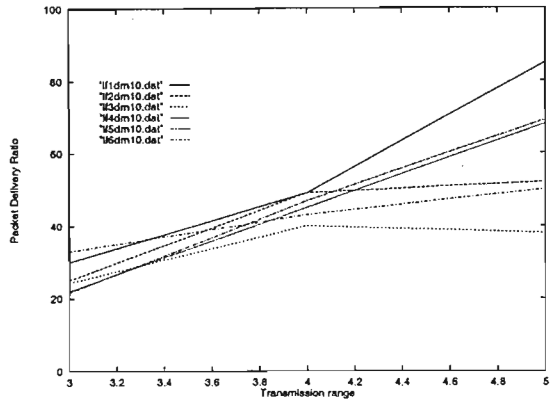

Figure 6.37: Normalised packet delivery ratio: N = 10

6.5 Summary and Conclusions

In this chapter, variations of limited flooding protocol and pure flooding were compared using discrete event simulation. The mean packet relayed, channel utilsation and packet delivery ratio were used as comparison metrics.

The various simulation results show that channel utilisation is low at lower transmission ranges. This is because fewer nodes can communicate. However, as transmission range increases, the number of nodes that can hear each other increases resulting in increased channel utilisation. Packet relaying overhead increases slightly with increase in transmission range but remains much lower than that for pure flooding. The overhead decreases slightly with increase in network size for most protocols, since the load is shared among all the nodes. Packet delivery also increases with increase in transmission range.

In general, the simulation results show that limited flooding uses less resources than pure flooding for a reasonable packet delivery performance and hence it is suitable for routing in mobile ad hoc networks. Within limited flooding, the protocols based on randomised selection of paths relatively outperform the likely paths. This occurs because randomised choice of communication paths results in a more balanced use of resources.

Chapter 7

Hybrid Cluster-based Hierarchical Routing

Mobile ad hoc networks for many applications are naturally clustered by geographical or physical boundaries [Hal96]. Most envisaged applications of ad hoc networks such as disaster recovery areas, people in meetings, students in classrooms, tactical military communication sites etc., can be considered as clusters. In this chapter we propose a routing protocol that exploits clustering.

7.1 Benefits of Hierarchical Clustering

Clustering provides several advantages in ad hoc networks including stabilising the network, improving functions such as routing and mobility management [Sha96] and MAC resource management [Ger95a]. Given the advantages of clustering, we use a hierarchical clustering architecture to improve routing performance. This is achieved through the use of hybrid routing mechanisms for inter-cluster and intra-cluster routing. Our proposal for the hierarchical routing mechanism is based on a clusterhead for coordinating the clusters.

As we saw in chapter 4, various routing protocols have been proposed for mobile ad hoc networks. Most of these protocols are fully distributed with a flat routing architecture where each node maintains a complete routing table to each possible destination. Although a few works use hierarchical routing, none of these protocols fully exploits the hierarchical clustering architecture with hybrid routing mechanisms. The most recently proposed hierarchical

routing protocol, known as Hierarchical State Routing [Iwa99], uses a multi-level clustering architecture to reduce routing overhead, but involves complex mechanisms for routing and mobility management.

Routing in a flat architecture faces scalability problems since overhead for flat routing protocols can grow faster than linearly as network size increases [Lau95]. A flat routing architecture has larger storage requirements of routing information, high computation overhead and high communication overhead.

A hierarchical architecture on the multicluster infrastructure reduces traffic by hiding information about the content of a cluster, and is relatively more stable in the mobile environment. In this architecture, there is one entry per destination for nearby nodes and one entry per set of destinations which are far away. Since less information is kept about nodes far away and complete information about closer nodes, routing table length can be significantly reduced.

However, there is a trade-off between a gain in table length and loss in path length. The optimal values for table length and cluster size in fixed networks have been determined in [Kle77]. But for mobile ad hoc networks, path optimality is of secondary consideration [Cor95, Par97b] compared to robustness and ensuring connectivity.

7.2 Description of a hybrid Hierarchical Cluster-based Routing (HCR)

Motivated by the advantages of clustering and hierarchical architecture, we propose a hybrid Hierarchical Cluster-based Routing protocol (HCR) for mobile ad hoc networks. HCR is a *hybrid* because intra-cluster and inter-cluster routing may be achieved using different protocols, reactive or proactive.

Our protocol is based on a hierarchical architecture with a two-level hierarchy. The *mobile nodes*, also known as *mobile stations*, form the lower level whereas *clusters* form the higher level. Clusters can be distinct or overlapping (see figure 7.1). Each cluster is coordinated by a clusterhead which can directly or indirectly communicate with every other node in the cluster. The clusterheads cannot directly communicate with each other; their communication is through one or more nodes or clusters. Some dynamic clustering algorithms are used to select the clusterhead among the set of nodes, but the clusterhead performs extra work. Among these, the clusterhead performs inter-cluster communication and facilitates local resource coordination. However, the cluster-based architecture differs from the cellular systems in that clusterheads have no special hardware components.

7.2.1 Definition of some terminology

Here, we define some terms used in HCR that do not appear in the standard MANET terminology [Per97].

- *Node ID*: A unique identifier of the mobile node.
- *Cluster*: A group of nodes that are close to each other.
- *Cluster ID:* A unique identifier of a cluster. The cluster ID is also an identifier of the clusterhead.
- *Clusterhead*: A unique leader of each cluster elected by a clustering algorithm during cluster formation or re-elected upon the original clusterhead failure. The clusterhead re-relection mechanism is discused in section 8.2.
- *Shared border node*: A node that is a member of more than one cluster. It can fully hear both clusters and exists in overlapping clusters.
- *Distributed border node*: A node that is a member of one cluster but can communicate with a node that is in another neighbouring cluster.
- *Ordinary node*: Any node that is neither a clusterhead nor a border node.

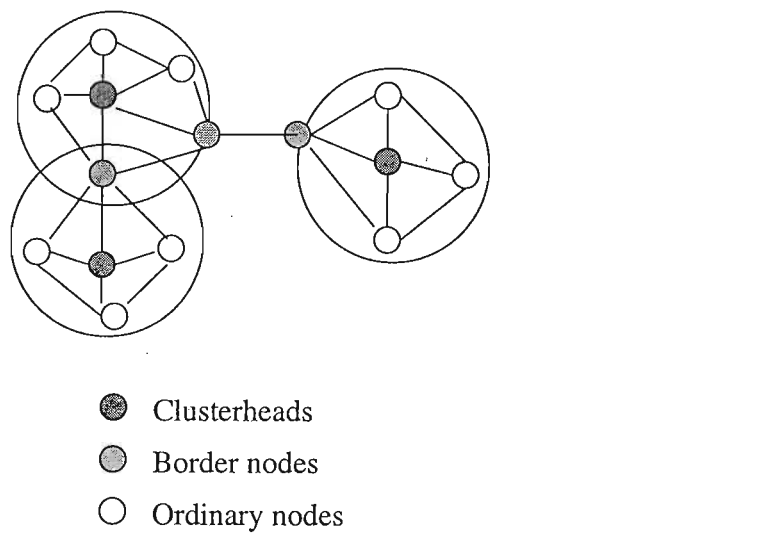

Figure 7.1: Network Architecture

7.2.2 Routing overview

The generic algorithm for cluster-based routing can be summarised as follows.

1. `Choose a clustering and cluster maintenance strategy.`
2. `Choose a routing strategy for intra-cluster routing.`
3. `Choose a routing strategy for inter-cluster routing.`

We discuss the first part in chapter 8. Our high-level routing rules for parts 2 and 3 are:

1. Routing within each cluster is flat and is performed in one of three methods: proactively maintained either at each node or at the clusterhead or reactively discovered on demand. The alternatives are discussed in section 7.3.
2. Routing to other clusters is performed reactively using route discovery. It involves first routing from source node to clusterhead or border node and second, routing from clusterhead or border node to the destination cluster. Within the destination cluster, routing is done as level 1 above. The alternatives are discussed in section 7.4.

7.2.3 Assumptions

The basic assumptions about the underlying link layer, physical layer as well as node and network conditions for the proposed routing protocol are summarised below.

- All hosts use a common wireless channel for communication.
- Communication links are bi-directional.
- The underlying data link layer protocol ensures distributed knowledge of the connectivity changes.
- Communication is more frequent within cluster and less frequent between clusters. Also, nodes do not move so fast as to make flooding the only appropriate routing protocol.
- The network is relatively large in size. All hosts wishing to communicate are willing to forward packets to other hosts within or between clusters.

- Every node has a unique ID and knows the identity of its neighbours. The network topology does not change during the execution of the clustering algorithm.

Frequent communication between nodes and low mobility rate is assumed for the protocols based on route computations whereas infrequent communication and relatively low mobility rate is assumed for protocols based on route discovery. The efficiency of the proactive or reactive protocol for intra-cluster and inter-cluster routing depends on the number of hops used for node-to-node or node-to-clusterhead communication. When one uses a small hop length, most routing is between clusters. On the other hand, when a hop length is large, most routing is carried out within clusters. Thus, as the number of hops along which a node and clusterhead communicate increases, the efficiency of intra-cluster routing may decrease while that of inter-cluster routing increases. But when the hop length becomes large, the cluster coverage will increase, decreasing clustering efficiency as a mechanism for partitioning a large network into small units for better routing management.

7.2.4 The main operations of the protocol

The main routing requirements in mobile ad hoc networks are to avoid the creation of loops, to avoid congestion and to enable fast convergence of the routes upon topology changes. The main components of the protocols are cluster formation, cluster management and packet routing. In routing, the operations common to all protocols are route generation, packet forwarding and route updates or maintenance. We briefly describe these mechanisms below.

Route generation. Mobile ad hoc routing algorithms must rapidly react either to compute and maintain current routes and topology tables using proactive protocols, or to discover up-to-date routes using reactive protocols. In the former, all pairs shortest paths are computed, whereas in the latter, route caches store necessary route information after learning routes through a route discovery mechanism.

Packet forwarding. Both route discovery packets and data packets are forwarded in the routing process. A packet is delivered to the destination node if the destination is a neighbour; otherwise it is forwarded to the next node. While a route discovery packet traverses through the network as a broadcast, a data packet follows an explicit path since routes are pre-determined and can be carried out based on source routing or on a hop-by-hop basis. Generally, packets may be dropped when the destination node

is in motion, a wireless link is not available or congestion occurs. The higher layer protocols are assumed to react to this problem.

Route updates. Changes in network topology are caused by node mobility or link failures. Changes may result in neighbourhood change, cluster membership alteration or network partition. Hence route updates are performed after a new route discovery or after a route computation.

Nodes can be classified by type (ordinary, border or clusterhead) or by roles (source, destination or intermediate). A route update depends on the type and role of nodes involved in the mobility. For example, movement of an ordinary node does not have the same impact as that of a border node or clusterhead: ordinary node mobility within a cluster affects a single cluster, that of the border node affects at least two clusters, whereas clusterhead mobility may affect other clusters. As clusterhead moves, all the information it maintains about other clusters may be lost. Also the impact of movement of the source node is not the same as that of an intermediate or destination node. The movement of source and destination nodes towards or away from each other may change the intermediate nodes previously used for communication and result in different route updates.

If a node moves from its initial location in the cluster but is not isolated from the cluster or the ad hoc network, it may supply the information in its cache to the neighbours or clusterhead which cache the route information. It also sends a route erase message to the initial clusterhead and other nodes with which it had been communicating. The update procedure may be complicated when multiple nodes (source and destination) are moving at the same time. Under this condition, routing algorithms such as the one we proposed in chapter 5 may be preferable.

We propose two techniques to reduce route update overhead. The first involves the maintenance of only affected nodes, cluster or a group of clusters instead of sending update information throughout the network. To this end, a three-level route update mechanism is proposed. *The local level route update* procedure maintains only affected nodes by mobility or link failure at a given time. For a route computation-based scheme, whenever there is a change in network topology, each node maintains not only the routes to the destination but also neighbours on the active routes. *The intermediate level route update* procedure maintains only affected clusters. This may involve a cluster and its neighbours, or clusters along the path from the source to destination. *Global level route update* is made during major topological changes such as network partitions (node or cluster isolation), new cluster formation or clusterhead election. A global level update is performed through global broadcast.

The second technique for reducing route update overhead is the update

policy itself. Since both periodic and continuous updates have high routing overhead, either on-demand or topology driven updates are employed.

Packet formats. The route discovery and reply packets contain relevant fields for gathering information along all the paths that the packet traverses. For route query within clusters, the query packet only propagates within a single cluster. The border nodes do not process such packets. The query packet consists of source and destination ID, sequence number and a route cache to accumulate information on link and node qualities. The reply packet consists of source and destination ID, sequence numbers, intermediate node IDs, and information on link and node qualities. In order to collect link and node status information, reply packets from the destination node are preferred.

For route query between clusters, query packets propagate in at least two clusters. The query packet consists of source and destination ID, sequence numbers and destination clusterhead. A route reply packet consists of IDs for source and destination nodes, intermediate nodes (border or ordinary), clusterheads, sequence numbers and information on node and link qualities. For the routing mechanism that involves route computation at each node or clusterhead, the forwarded packet contains the identifier of the source and destination nodes, the data packet and some control information.

7.3 Intra-cluster Routing Mechanisms

We have proposed a two-level routing mechanism for constructing and maintaining routes in mobile ad hoc networks. The intra-cluster routing uses one of three possibilities: route computation at nodes, route computation at clusterheads and routing on demand.

7.3.1 Route computation at nodes

In this method, each node participates in route computation and route maintenance procedures. A source node that wishes to send data uses the routes from its routing table. A modified link state algorithm or the recently proposed link vector algorithm [Beh97a, Gra95] can be adapted for route computation at each node (see section 4.1.3). For mobile ad hoc routing this routing algorithm has two advantages. First, a node maintains only a subset of the topology information unlike link state algorithms which maintain the entire topology map. Second, the use of diffusing computation for information dissemination, unlike flooding in the link state algorithm, reduces communication overhead.

The algorithm can be modified to make it more suitable to mobile ad hoc networks. For example, constructing routes only when a route is needed (not whenever topology changes) can reduce the overhead. Route maintenance is invoked when the node itself notices the change or is notified by other nodes. Node mobility may affect all routes that pass through it. Affected routes will be updated using the local or intermediate route maintenance procedure. Note that even if every node is in direct contact with the clusterhead, this approach is reasonable as it reduces load on clusterhead.

7.3.2 Route computation at clusterheads

In this routing mechanism, only the clusterheads compute and maintain the routes to nodes within their clusters. Each node keeps route(s) to the clusterhead. The advantage of route computation at a clusterhead is reduced computation and communication overhead. In order to further reduce communication overhead, the link layer protocol is assumed to use the clustering architecture for resource scheduling and channel access. When routes are computed at a clusterhead, a backup node can be used to reduce the risk of data loss.

The clusterhead performs route computation in a manner similar to the link state routing algorithm restricted to the clusterhead. The link state packet formation can be periodic or event-driven, the choice depends on the dynamics of the network topology. The clusterhead requests topology information periodically from all nodes in its cluster through a broadcast. Each node replies by appending its identity to the reply packet. Then, routing tables for all nodes within the cluster are computed. Once computed, the routes are stored at the clusterhead. These routing paths are obtained on demand from the clusterhead.

There are several reasons for choosing a link state type algorithm for route generation. These include: (a) low overhead (as only cluster knowledge is required); (b) compatibility with the Internet; (c) possibility of maintaining multiple routes to a destination; and (d) support of Quality Of Services routing. Moreover, we argue that node and link status information is much more useful for routing than distance characteristics in mobile ad hoc networks.

Topology change such as link down and link up or node up and node down may cause complete partition of the network or connection via a distributed border node. Route changes are made when the changes are detected by the clusterhead or an update message is received from another nodes or clusterheads. Suppose that a node is added or deleted from the cluster. The clusterhead updates the route as follows. First, addition or deletion of a node is detected by absence or presence of a beacon from it. Second, the

neighbour nodes notice the change and update their entries. Third, each neighbour node notifies changes to the clusterhead.

Consider an update scenario when a node moves and is isolated from a cluster. Route update is performed as follows: Changes can be heard by neighbouring nodes and each sends to the clusterhead via unicast. The hearing nodes delete the entry. The other nodes will learn the routes from the clusterhead or obtain route on demand. Data packets are forwarded following the explicit paths and there is no need for further route discovery unless the already discovered route becomes invalid.

7.3.3 Routing on demand

The third routing mechanism for intra-cluster routing is based on routing on demand similar to the routing protocol proposed by Johnson in [John96].

If a node has a route to the destination in its route cache, it uses this route. If the route is stale, a new route discovery is made. During the route discovery process, all nodes that hear can update their entries. Route discovery also occurs when the source node encounters failure of data transmission when a data packet is sent. A node initiating route discovery can carry on doing other tasks while the route discovery process is in progress.

Route discovery packets are forwarded to the next node until the desired target is reached. The route reply packet is a unicast sent directly to the source node following the explicit path indicated in the reply packet. Data packets are forwarded following the explicit path determined by the route query.

7.4 Inter-cluster Routing

Routing between clusters is the second stage of routing in the proposed routing protocols. It is concerned with packet routing from one cluster to another cluster. Clusterheads or designated border nodes are responsible for inter-cluster communications. This can be performed in one of three different ways: (a) making external route discovery when the route is needed; (b) maintaining complete external route information; or (c) sending data using explicit path or forwarding packets to other clusters with a single-hop knowledge.

We assume that communication within clusters is more frequent than communication between clusters. Hence, only a few source–destination pairs will be sending packets across clusters. In such environments, computing and keeping routes at each node or clusterhead may be inefficient. Therefore, in our proposal inter-cluster routing is carried out using demand-based routing.

A clusterhead communicates with other clusters through a designated border node or any border node along the path to the target destination cluster. To facilitate routing, the border node may store additional information such as list of border nodes in neighbouring clusters or list of all border nodes, etc. Alternatively, it may request this information from the clusterheads if this can be maintained at the clusterhead.

A node that wishes to send data to a destination that is not in its cluster, forwards the data to a clusterhead or border node. The clusterhead or border node then checks whether it has routes in its route cache. If it has a route, it forwards the packet to the destination clusterhead using this. If the destination node is not in the neighbour cluster, the packet is forwarded to the next cluster via the neighbour cluster. The process continues until the packet is finally delivered. If a clusterhead does not have a route to the required destination, it uses a route discovery procedure to get another route.

Depending on how intra-cluster routing is done, inter-cluster route discovery is performed in our proposal as follows:

1. If intra-cluster routing is based on route computation at each node, a route to the desired destination is first sought in its routing table. If not found, the node sends a unicast message to the clusterhead, which sends a route request broadcast message to the other clusterheads. A node can alternatively send a packet directly to the border node. The clusterhead maintains a list of the clusterheads and border nodes. The route reply packet is sent via the clusterhead back to the source node which adds the entry into its routing table.

2. If intra-cluster routing is based on route computation at clusterheads, a route to the desired destination is first searched in the clusterhead's routing table. If the destination is not found, the clusterhead initiates a route discovery process by sending a broadcast message to the other clusterheads via a designated border node. When the destination is found, the route reply packet is sent back to the clusterhead. The clusterhead records this route to its routing table. The clusterhead maintains a list of the clusterheads, intra-cluster routing table and designated border nodes and those in the adjacent clusters.

3. If intra-cluster routing is constructed on-demand, a route to the desired destination is first searched in its cluster. If the destination is not found, the node initiates an external route discovery process by sending a unicast message to the clusterhead. When the destination is found, the route reply packet is sent back and recorded. The clusterhead maintains a list of the clusterheads, border nodes and route cache. After the route

reply is obtained, route information is cached in the clusterhead for future use. Thus, route information about external clusters is updated at the clusterhead regardless of whether stage 1 routing information is maintained at the clusterhead, each node, or based on route discovery.

7.4.1 Packet forwarding

For routing packets across clusters, the following mechanisms can be used: routing through clusterheads; routing through the border nodes; or a combination of these. If routing is via a clusterhead, then the source node always sends its packet to the clusterhead. The clusterhead can choose any border node across which it forwards the packet.

If routing is via a border node, then the source node forwards the packet to the border node which relays the packet to the neighbouring cluster's border node. Packets are forwarded via border nodes but the border node may rely on the clusterhead for information on the destination cluster. When routing is via a clusterhead and gateway alternatively, the route request will always follow the route: Source, CH_1, B_1, CH_2, B_2, CH_3, B_3,...., B_k, CH_k, Destination where CH_i and B_i are clusterhead and border node for node i respectively. The clusterhead uses a designated border node for communication with the external clusterheads as shown in figure 7.2.

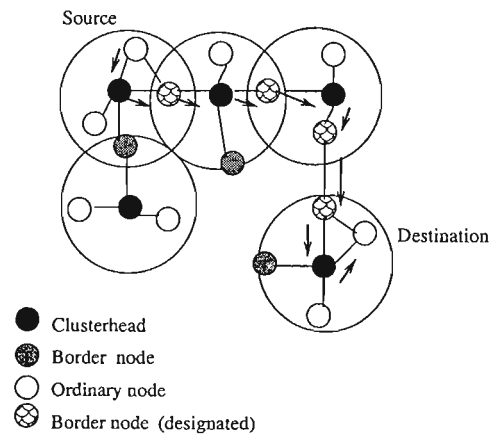

Figure 7.2: Clusterhead Border Node Routing

7.4.2 Multiple paths for inter-cluster routing

In most of the proposed mobile ad hoc routing protocols, the best route among the set of possible routes from source to destination is used. The definition of the best route depends on the routing metric used by the underlying

routing algorithm. Number of hops, mean queue delays, transmission times, bandwidth, mean network traffic or physical distance, etc., can be used as a routing metric. For example, if the routing metric is number of hop, a route with least number hops is the best route. In reactive routing protocols determination of such a route is protocol dependent. In some routing protocols the first arrived route is considered as the best route [Dub97, John96]. In [Dub97] the first arriving route among the most stable area of the network is selected as the best route.

There are several problems in using these routing protocols. First, the early-arrived route might not be the best route. Hence there is a need to select some optimal routes. Optimality here refers to routes that use additional metrics such as less congested route, minimum number of hops or high bandwidth, etc., instead of just the first received route. Although it is difficult to determine an optimal route in ad hoc networks, a routing objective function that uses aggregate metrics such as weighted shortest path metrics, interference metrics and network congestion metrics may be used to determine nearly optimal routes. Second, frequent route discovery attempts to construct new routes whenever the clusterhead has no routes to the desired destination may cause high communication overhead. Third, failure to stop stale route discovery packets once routes are obtained, as well as multiple reply packets, also cause additional overhead. The overhead due to such packets is high when there is a frequent route search through the entire network.

To avoid the first two problems, we propose that a clusterhead should keep multiple routes in the route cache among the received routes during route discovery. Based on criteria such as signal strength, node stability, etc., routes will be classified as primary or secondary routes. The secondary routes are used when the primary routes fail to route packets. By maintaining multiple routes between adjacent clusters, the effect of mobility on node or cluster connectivity can be reduced.

There is a trade-off between storage overhead due to keeping multiple paths, and communication overhead due to a frequent route discoveries. However, since communication overhead is more serious than storage overhead in mobile ad hoc networks [Das97a], keeping multiple paths at the clusterhead has more advantages than using a single-path routing.

The effect of multiple reply packet and stale route discovery packet might be reduced by restricting the scope of route request packets propagation in the network. One mechanism is broadcasting the route search packet starting from the cluster one-hop away, two-hops away, etc., and waiting for a reply within a predefined time interval. Thus route search continues to the remote clusters only if the search is unsuccessful in the nearby clusters.

Chapter 8

Simulation of Clustering Algorithms

Cluster management comprises cluster computation and cluster maintenance mechanisms.

8.1 Cluster Computation

Clustering forms interconnected groups of nodes that cover all mobile nodes in the network. Some goals for clustering are simplicity, stability, efficiency and distributed operation.

- *Simplicity*: The clustering algorithms should be easy to understand and check for correctness. They should avoid complex rules that cause nodes to spend much time in invoking the clustering algorithm. They should also guarantee that only one clusterhead is elected per cluster.
- *Stability*: The algorithm should achieve a stable clustering architecture. Reactions to topological changes due to mobility should not significantly affect the clusterhead changes and looping or oscillations must be avoided.
- *Efficiency*: The algorithm should be efficient in terms of communication overhead and computational complexity.
- *Distributed operation*: The algorithm should operate asynchronously in a distributed manner and support mobility.

An optimal clustering architecture depends on various factors including network topology, network traffic, routing algorithm and geographical location of the nodes [Ame88]. If the network topology is stable, centralised clustering algorithms can be used to achieve an optimal clustering topology. On the other hand, if the network topology is highly unstable due to mobility or unreliable wireless links, no clustering algorithm achieves optimality. Our aim is to obtain a relatively "good" clustering topology that meets some of the clustering design goals.

One possibility for clustering is that each node is a neighbour of the clusterhead and at most two hops from the other members of the cluster. The advantage is effective communication within each cluster and low routing overhead for intra-cluster routing. Since the number of nodes per cluster depends on radio coverage and network configuration, a two-hop clustering architecture may be sufficient with relatively large coverage area.

Another possibility is, for example, that each node is at most two hops from the clusterhead and at most four hops from other members of the cluster. Under this condition, intermediate nodes are used for communication between a node and clusterhead. Compared to a two-hop clustering architecture, there could be a high communication delay. Also, a mechanism is required for reliably broadcasting a Hello packet to indicate the presence or absence of a link as well as to facilitate clusterhead election. This makes the information for cluster formation and the execution of dynamic cluster algorithms more complex. Indeed, as the number of hops between a node and clusterhead increases, the benefit of reduced routing overhead decreases and clustering complexity increases.

We choose a two-hop clustering architecture where at most two hops exist between ordinary nodes in a cluster and one hop between a node and its clusterhead.

Operation

In addition to information obtained from routing tables or route caches, some lookup tables are maintained to facilitate clustering. In the *neighbour table*, each entry stores the ID of the neighbour node and its role (clusterhead, border node or ordinary node). It also maintains, node mobility frequency, and signal strength of the node. In the *inter-cluster adjacency table* is information about neighbouring clusters which includes, IDs of neighbouring clusterhead, border node and the corresponding information about stability, relaying load and signal strength.

Initially, mobile nodes are in non-clustered status and asynchronously communicate while in the transmission range of each other. Figure 8.1 shows

one possible topology before clustering. All nodes will be in non-clustered status and can then become a clusterhead, border node or ordinary node. Each node broadcasts its neighbour table information periodically in a Hello message. A node then collects complete neighbour topology information from the neighbour table for cluster formation. After clustering, each node will be grouped into a cluster with a clusterhead.

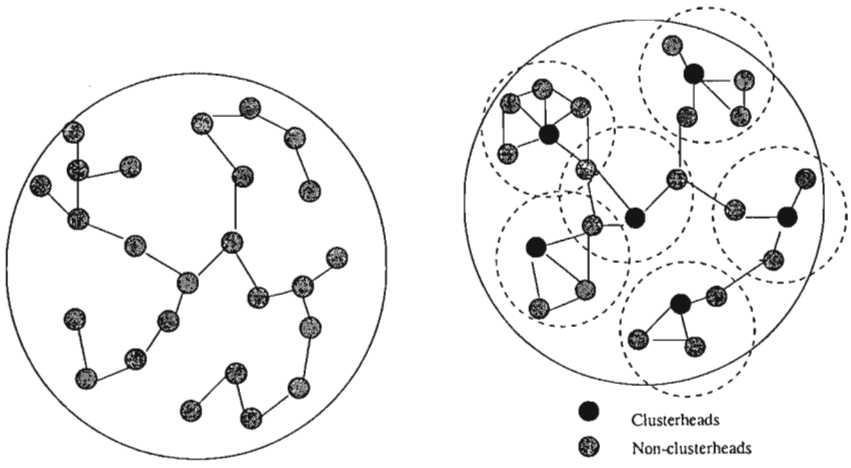

Figure 8.1: Topology Before and After Clustering

8.2 Proposals

Based on the distributed clustering approach proposed in [Bak84], two clustering algorithms were described in [Ger95b] for mobile multicluster network architectures. The first is the lowest ID clustering algorithm where the lowest ID node is elected as a clusterhead. The second is the highest connectivity clustering algorithm, where a node with highest connectivity is selected as a clusterhead. The performance of the algorithms were compared in a discrete event simulation. The lowest ID based clustering algorithm was found to be more stable in a highly mobile environment.

All nodes in ad hoc networks may not have the same rate of mobility and signal strength. In fact, some routing algorithms which select routes through the most stable areas of the network have been proposed in [Dub97, Toh96]. Our proposals for clustering includes some stability conditions such as electing the clusterhead not from all nodes but only from relatively stable nodes and better signal strength. A clusterhead will be randomly selected from a subset of nodes based on one of the following criteria: *sufficiently low mobility*, or *sufficiently high signal quality*. Toh has proposed choosing only nodes with *sufficiently high relaying load* as an intermediate node [Toh96].

Below we will discuss three clustering algorithms, namely Lowest ID, Node Mobility-based and Signal Strength-based Clustering algorithms.

Lowest ID Clustering algorithm(LID). This was introduced by Gerla [Ger95a]. In Lowest ID clustering, every node has an Integer ID and a node with the lowest ID is selected between any two nodes that can communicate. Clusterhead election by node ID can be performed as follows.

1. Reset neighbourhood and clusterhead tables
2. Send HELLO message to all neighbours and get reply
3. Run the clusterhead election algorithm as follows:

```
IAMCH[id] = true;  // Declare that I am a Clusterhead
   for(i=1;i<id;i++)
     if ((IAMCH[i])&&(neighbour[i])) IAMCH[id] = false;
```

4. Broadcast clustering update information
5. Repeat

Node Mobility-based Clustering algorithm (NMC). In this clustering algorithm, we use information on node mobility for clustering formation. Each node broadcasts the list of nodes (with ID and mobility frequency) that it can hear from. Mobility frequency shows the number of times a node is disconnected from its neighbour. A node will be considered stable if its movement does not exceed a mobility threshold. The mobility threshold is the mean node mobility determined through simulation. A node can be elected as a clusterhead only if the mobility is low and no member of its neighbourhood is a clusterhead. In case of a tie the one with the lowest ID may be chosen.

Signal Strength-based Clustering algorithm (SSC). In signal based clustering, signal strength of the link is used for clustering. A node can be a clusterhead only if the signal is strong and none of its neighbours is a clusterhead. A node has a highest signal quality if it doesn't experience co-channel interference, multipath fading or physical obstacles that cause link failures. Each node contains a field that records channel quality for each link. A given node will qualify for clusterhead if it is a lowest ID and has signal strength above a certain threshold value.

Cluster maintenance

A cluster topology change can occur due to node mobility, new node addition or existing node deletion. A new node joins the cluster if it is within a distance of 1-hop from any of the existing clusterheads. Otherwise, it forms a new cluster for which it acts as a clusterhead. When an ordinary node is deleted from the cluster, the neighbours of the deleted nodes update their neighbourhood information. If the deleted node is a clusterhead, a new clusterhead is elected as described below.

Clusterhead re-election. A clusterhead may fail to function or its replacement may be needed due to mobility. Also, it may be completely isolated from its cluster but within reach of communication with other nodes in the network. The clusterhead failure or mobility is detected by members of the cluster through neighbourhood information exchange. Under such circumstances, a new clusterhead is elected to replace the old one or to coordinate a newly formed cluster. The clusterhead re-election process is carried out as follows. A node that detects the absence of the clusterhead assumes responsibility as the clusterhead and initiates the re-election process by sending this information to its neighbours. Based on the neighbour table information, every node runs the clustering algorithm and determines its status. Then a node that meets the election criteria becomes the clusterhead and conveys the result to its neighbours. It then reconstructs the necessary information and continues its role as a clusterhead.

Improving clustering performance. We propose several techniques to improve clustering efficiency and stability. The first is a **preventive** mechanism. A clusterhead is chosen among the most stable nodes in the population of nodes. The second is a **risk minimisation** mechanism where a designated backup node may be used during clusterhead failure. Either the information is maintained at both clusterhead and backup node or such information will be forwarded by the clusterhead upon request. Finally, one can switch to **flooding** while the clusterhead is inoperable. To reduce the flooding overhead, limited flooding protocols proposed in chapter 5 can be used.

The size of clusters may be used to improve clustering performance. So if the optimal value of this parameter is known, an appropriate cluster adjustment can be made. A cluster splits if the number of nodes in the cluster exceeds the split threshold, merges if the number of nodes becomes less than merge threshold. The cluster split and cluster merge thresholds are fixed parameters that are computed based on the cluster size. The larger the cluster size, the greater the value of these thresholds will be.

The merge threshold is chosen in such a manner that when there is a merge the resulting cluster size is as close to the target cluster size as possible.

The cluster merge and cluster split algorithms are run by the clusterhead. The clusterhead finds the most suitable neighbouring cluster with which a given cluster can merge. It runs the algorithm only if after merge, the size of the new cluster is close to the preferred cluster size and does not exceed the cluster split threshold.A stable clustering algorithm is needed for efficient operation of this mechanism. The split size and merge size are periodically checked to invoke splitting or merging. Cluster splitting and merging are possible optimisation mechanisms and are not considered in our clustering algorithms.

8.3 Simulation

In this section, we model and simulate mobile ad hoc networks using a hierarchical clustering architecture. Although the PARSEC simulation language is used, the simulation models are different from those in limited flooding since they also maintain clustering information.

The Node entity. The node entity simulates the mobile node. Each node maintains two categories of information. First, it maintains information for packet routing. This includes information about link state packets, link state database and link state routing table. Second, it maintains information for cluster formation. This includes information about node mobility and signal strength.

At the beginning of the simulation, nodes are randomly placed inside the simulation area. The simulation is run for each of a number of different moving probabilities and network size combinations. The number of nodes, moving probability and the radio transmission ranges are specified at the beginning of the simulation. Each node broadcasts information such as up-to-date routes, neighbourhood list and ID of its clusterhead. Each node also receives information needed to initiate or end the node operations, collect statistics and initiate packet generation. Upon receiving the information, each node updates its routing and clustering information, forms a link state packet and elects its clusterhead as needed.

The Coordinator entity. The coordinator entity handles the coordinating functions of the mobile node and channel access. That is, determining the initial location of a node, determining node reachability, recording the data output packet to be sent and delivering data input packets to nodes in hearing range.

The Synchronizer entity. The synchronizer model synchronizes the operation of the simulation models. This includes sending a signal to initiate or end receiving a data packet and to start or end the data slot time for each node in the simulation.

The Data traffic entity. The data traffic entity is used to model data traffic in the network. It generates packets used in the simulation. The number, size and type of packets are determined in this model. The packets are generated and send to a random destination according to the routing algorithm.

The Mobility entity. A random mobility pattern is considered. Each node moves randomly by one unit in each direction with a pre-determined moving probability. A mechanism is provided to enable various forms of mobility pattern and moving probabilities for the simulation.

The Channel entity. This entity provides options for modeling the wireless channel. The current simulation results are based on a free space propagation model [Rap95], where the received signal strength only depends on its distance to the transmitter.

8.3.1 Performance metrics

Comparisons are made as a function of two parameters: *moving probability* and *network size*. The metrics used in performance evaluation are:

- *Mean number of clusters*: Shows the mean number of clusters in the network.
- *Mean clusterhead changes*: Shows the mean number of clusterhead changes. It measures the degree of stability of clusterheads.
- *Mean cluster membership changes*: Shows the mean number of node switches between clusters. It measures clustering algorithm stability.

8.4 Discussion of the Simulation Results

In this section, we describe the simulation results of the clustering architecture. We first discuss the effect of moving probability and network size on each clustering metric for each algorithm. Then, we compare each clustering algorithm with respect to these metrics. The number of mobile nodes used in the simulation are 5, 10 and 20 whereas the moving probability levels of 10%, 20% and 50% are considered.

8.4.1 Lowest ID Clustering algorithm(LID)

8.3–8.6 while those based on number of nodes are in figures 8.7–8.10.

Effect of moving probability. Figure 8.2 shows that number of clusters, clusterhead changes and cluster membership changes increase with increase in moving probability. The number of clusters increases faster than both clusterhead changes and cluster membership changes. Figure 8.3 shows that the number of clusters increases with increase in moving probability at all network sizes but declines slightly at higher moving probability for N = 20. The larger network size results in more clusters. Figure 8.4 shows that the number of clusterhead changes increase with moving probability for N = 20. But the change is relatively higher at larger network size. Figure 8.5 shows that cluster membership changes increases slightly with moving probability showing little difference for different network sizes.

Effect of network size. Figure 8.6 shows that the clustering metrics for LID clustering algorithm at moving probability of 20%. The results indicate that all metrics increase with increase in number of nodes. Figure 8.7 shows the number of clusters formed as a function of number of nodes. The number of clusters increases with increase in number of nodes at all moving probability levels but with slight increase for P = 10% at larger network size. The highest number of clusters is obtained at higher moving probability (P = 50%). This implies that the larger the moving probability, the higher will the cluster size be due to the possibility of cluster partitioning which might result in the formation new clusters. Thus, for a network with high mobility, the LID algorithm is not a stable. Figure 8.8 shows that the number of clusterhead changes increases with network size. But the clusterhead change is relatively greater at higher moving probability. Figure 8.9 shows that cluster membership changes increases slightly with network size. This indicates that as the network size increases, the number nodes that move within clusters increases. The cluster membership changes is not high because nodes can move within clusters without switching to other clusters.

The results indicate that the number of clusters is more affected by moving probability than clusterhead and cluster membership changes. The latter implies stability of the clustering architecture.

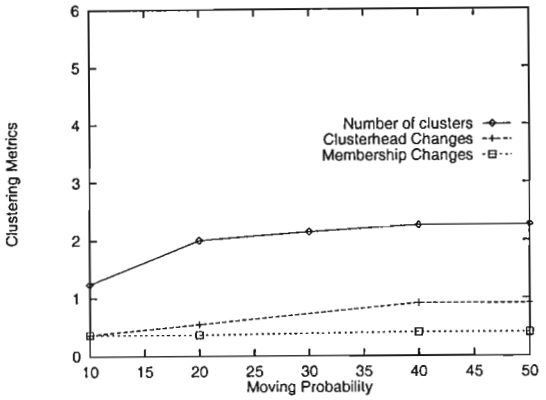

Figure 8.2: Clustering Metrics: N = 10

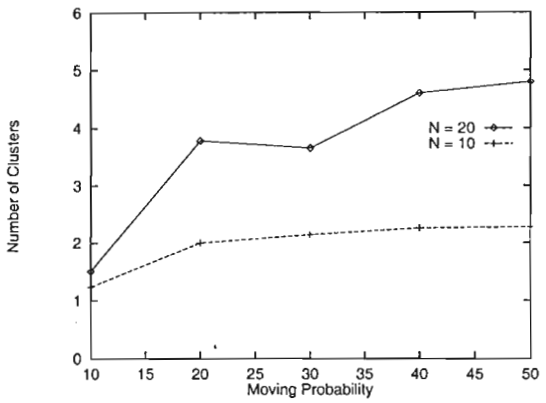

Figure 8.3: Number of Clusters: N = 10, 20

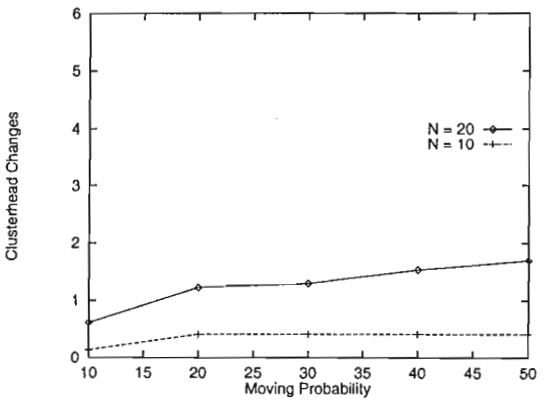

Figure 8.4: Clusterhead Changes: N = 10, 20

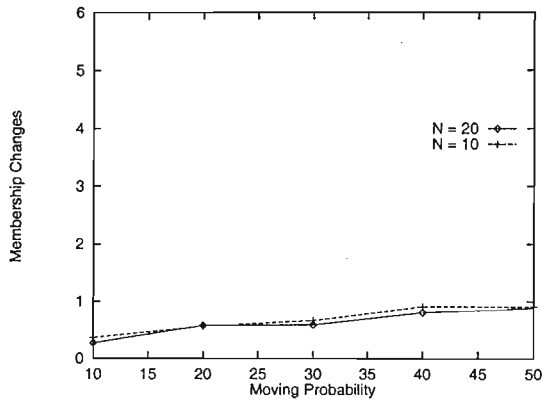

Figure 8.5: Cluster Membership Changes: N = 10, 20

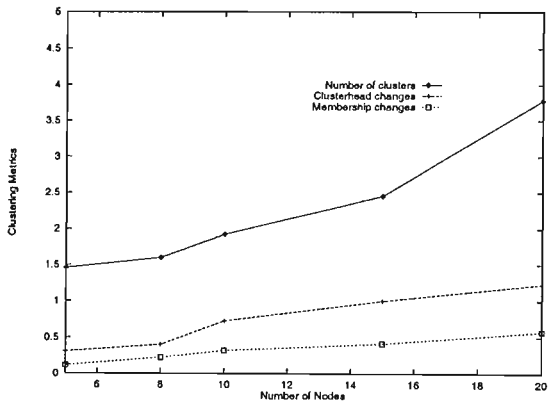

Figure 8.6: Clustering Metrics P = 20

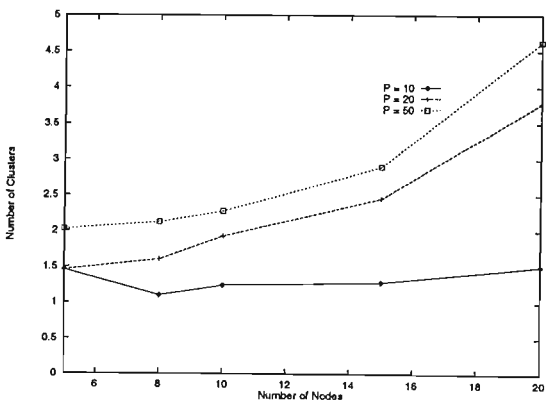

Figure 8.7: Number of Clusters: P = 10, 20, 50

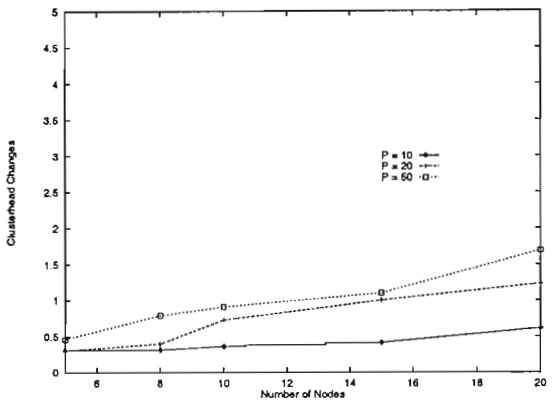

Figure 8.8: Clusterhead Changes: P = 10, 20, 50

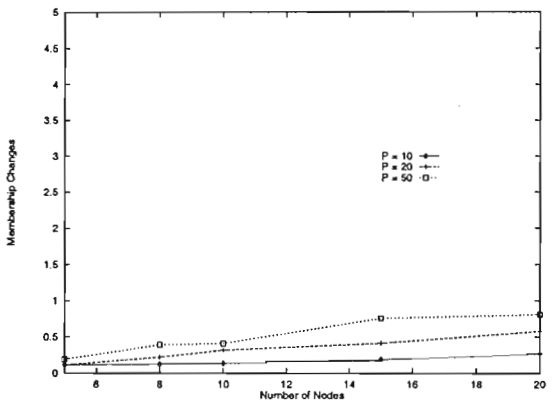

Figure 8.9: Cluster Membership Changes: P = 10, 20, 50

8.4.2 Node Mobility-based Clustering algorithm (NMC)

Effect of moving probability. Figure 8.10 shows that all clustering metrics increase with increase in moving probability. Figure 8.11 shows the number of clusters increases with increase in moving probability at all network sizes. The largest number of clusters is obtained at larger network size of N = 20 in the simulation. Figure 8.12 shows that clusterhead changes increase with moving probability. But the change is relatively higher at larger network size. Figure 8.13 shows that cluster membership change increases slightly with moving probability at N = 5 and 10. Little difference is observed at various network sizes and moving probabilities.

Effect of network size. Figures 8.14 and 8.15 show that the number of clusters increases with increase in network size at all the simulated moving probability. Figure 8.16 shows that the number of clusterhead changes increase with network size. But the number of changes is relatively greater at higher moving probability. Figure 8.17 shows that cluster membership change increases slightly with network size.

The simulation results based on mobility-based clustering algorithm indicate that moving probability and network size have relatively reduced the number of clusterhead changes and cluster membership changes. This makes NMC to be a good candidate clustering algorithm for mobile ad hoc networks.

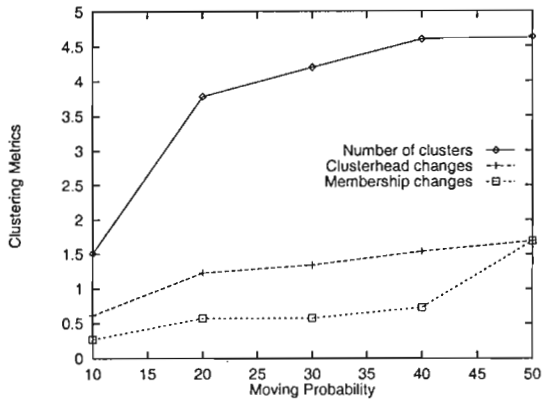

Figure 8.10: Clustering Metrics: N = 20

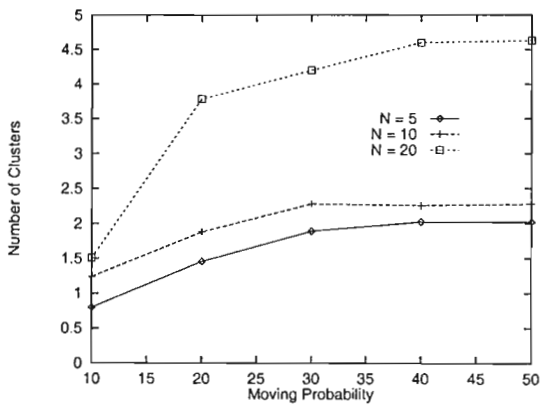

Figure 8.11: Number of Clusters: N = 5, 10, 20

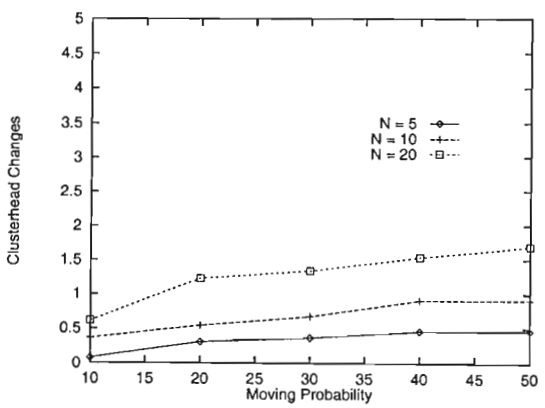

Figure 8.12: Clusterhead Changes: N = 5, 10, 20

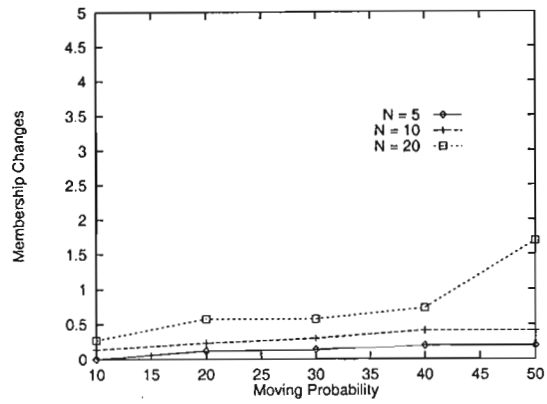

Figure 8.13: Cluster Membership Changes: N =5, 10, 20

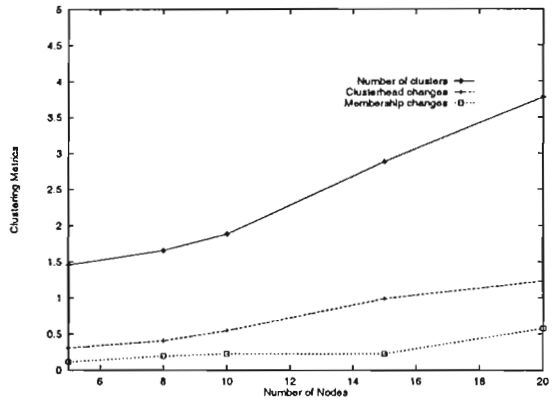

Figure 8.14: Clustering Metrics P = 20

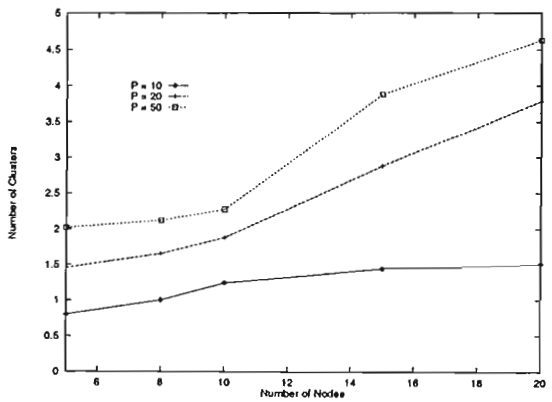

Figure 8.15: Number of Clusters: P = 10, 20, 50

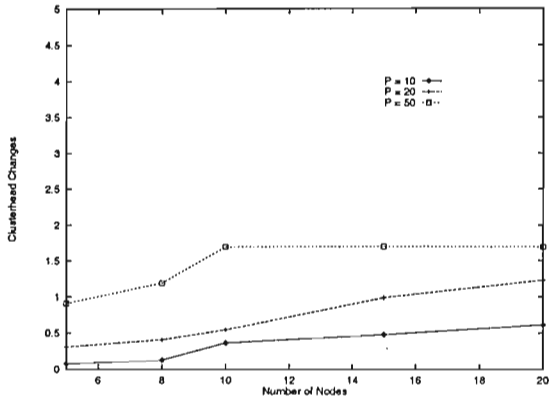

Figure 8.16: Clusterhead Changes: P = 10, 20, 50

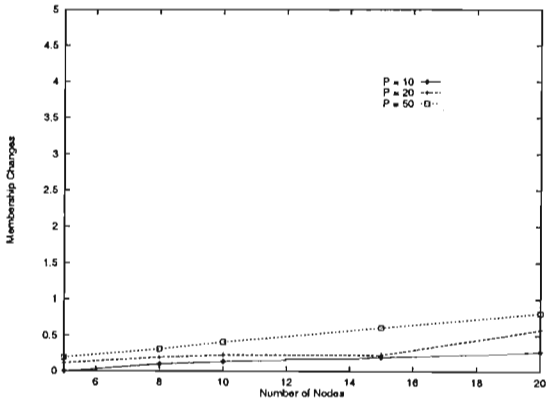

Figure 8.17: Cluster Membership Changes: P = 10, 20, 50

8.4.3 Signal Strength-based Clustering algorithm (SSC)

Effect of moving probability. Figure 8.18 shows that the number of clusters is high and increases slightly at higher moving probability. The number of clusters formed significantly exceeds both the clusterhead and cluster membership changes. A constant trend for the number of clusterhead changes and cluster membership changes indicate that there is less mobility between clusters. Figure 8.19 shows that the number of clusters increases with increase in moving probability for N = 10 and only at higher moving probability for N =5. This shows that number of clusters is more affected by moving probability at larger network sizes than at lower network sizes. For lower network sizes, larger number of clusters occur only if mobility is high.

Figure 8.20 shows that the number of clusterhead changes increases at a higher moving probability. The number of clusterhead changes is relatively higher at larger network size. Similarly, figure 8.21 shows that cluster membership changes increase slightly with moving probability. Little difference is observable as moving probability increases.

Effect of network size. Figure 8.22 shows that all metrics increase with increase in network size up to N = 10. There is a slight increase in clustering metrics when the network size exceeds 15 indicating that the clustering algorithm becomes relatively more stable. While the increase in the number of clusters is high as network size rises, the number of clusterhead changes and the cluster membership changes are relatively low.

Number of clusters as a function of network size is shown in figure 8.23. The number of clusters increases with increase in network size at all moving probability levels compared. This indicates that the higher the moving probability the greater the number of clusters will be. The increase in number of clusters is relatively low at larger network size. This indicates that nodes may move within their clusters without forming new clusters.

Figure 8.24 shows that clusterhead changes slightly increase with the number of nodes. But the number of clusterhead changes is relatively greater at higher moving probability levels. Figure 8.25 shows cluster membership changes. It shows a constant trend but with a slight increase with network size at higher moving probabilities. Both simulation experiments demonstrate that number of clusters is more affected by moving probability and network size. While higher mobility results in more clusters due to cluster partition, increase in the network size also results in increase in the number of hops through which nodes can communicate which again invokes the formation of new clusters. The number of clusterhead changes and number of nodes that switch between clusters are relatively low.

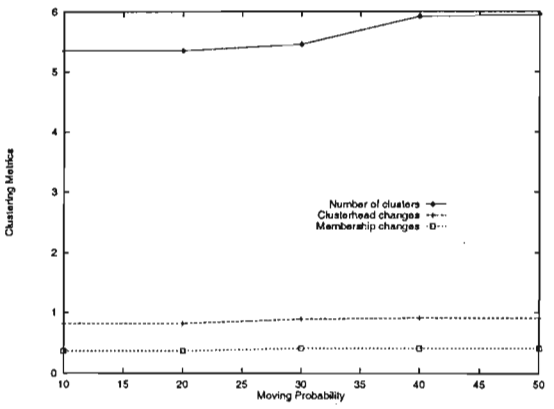

Figure 8.18: Clustering Metrics: N = 10

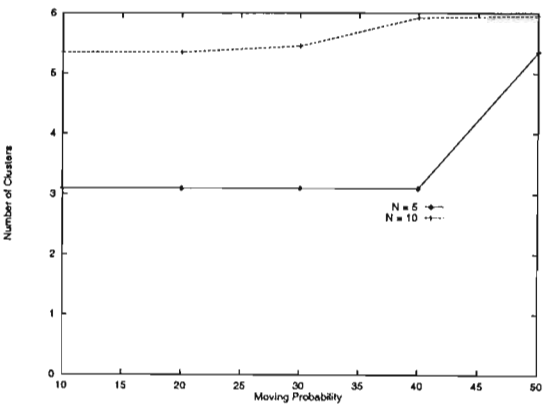

Figure 8.19: Number of Clusters: N = 5, 10

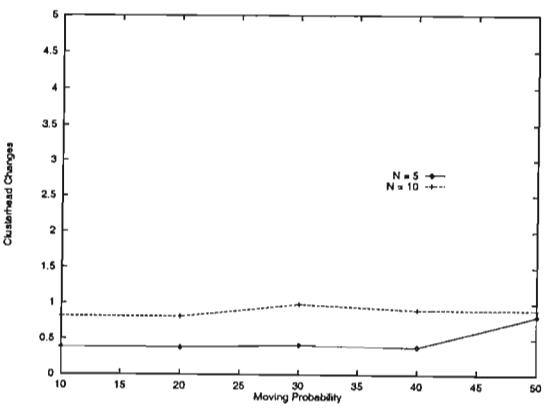

Figure 8.20: Clusterhead Changes: N = 5, 10

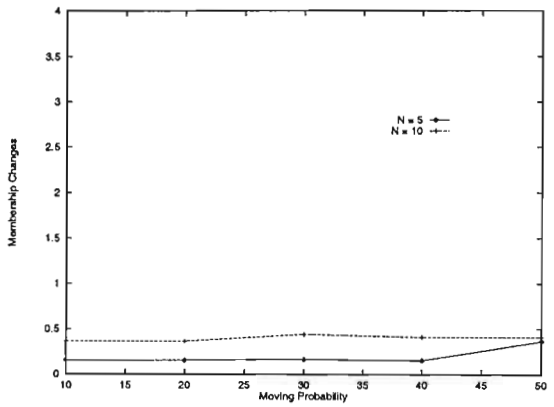

Figure 8.21: Cluster Membership Changes: N = 5, 10

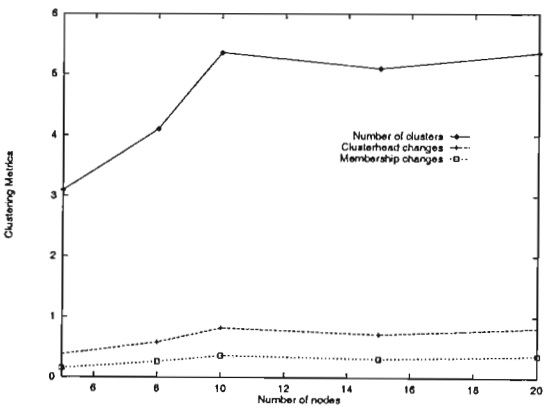

Figure 8.22: Clustering Metrics: P = 10

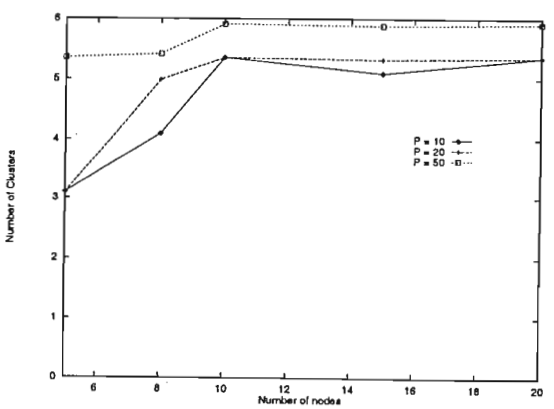

Figure 8.23: Number of Clusters: P = 10, 20, 50

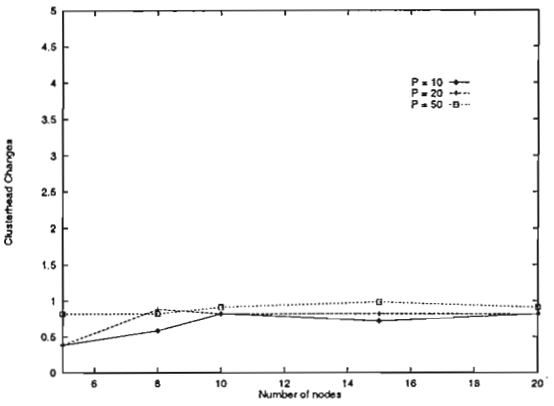

Figure 8.24: Clusterhead Changes: P = 10, 20, 50

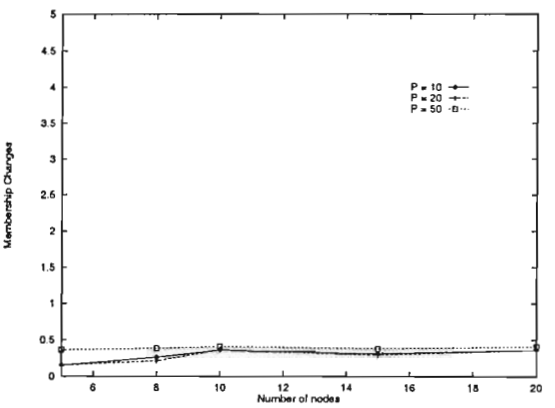

Figure 8.25: Cluster Membership Changes: P = 10, 20, 50

8.4.4 Comparison between the clustering algorithms

As before, number of clusters, clusterhead changes and cluster membership changes are used as comparison metrics. Figures 8.26–8.28 present the values obtained as a function of moving probability whereas figures 8.29–8.31 are the results obtained as a function of number of nodes.

Effect of moving probability. Figure 8.26 shows that the number of clusters increases with increase in moving probability for all algorithms. The largest number of clusters is formed with SSC whereas LID and NMC have resulted relatively in fewer clusters. NMC has slightly fewer clusters than the LID algorithm at some moving probability levels. Figure 8.27 shows the clusterhead change is relatively higher in SSC. The figure shows that at low to moderate moving probability, the clustering architecture is stable for the LID and NMC algorithms. Figure 8.28 shows that cluster membership changes is low in NMC at relatively low moving probability levels.

Effect of network size. Figure 8.29 shows that the number of clusters formed in each clustering algorithm as a function of number of nodes. It can be noted that the number of clusters increases with increase in number of nodes. In all the comparisons, SSC resulted in larger cluster size with NMC and LID giving similar results. Figure 8.30 and 8.31 show the results for number of clusterhead and cluster membership changes respectively as a function of network size. The number of clusterhead changes increases slightly with number of nodes, NMC resulting in less clusterhead changes at smaller network sizes than SSC. The number of cluster membership changes are similar but with NMC having slightly fewer cluster membership changes at smaller network size.

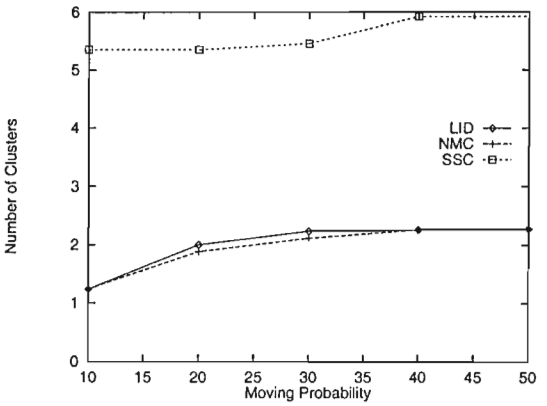

Figure 8.26: Number of clusters: N = 10

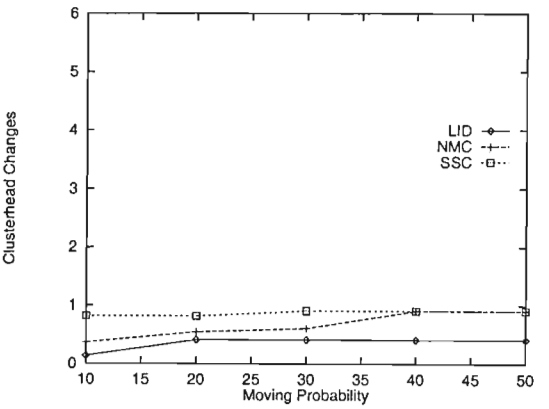

Figure 8.27: Clusterhead Changes: N = 10

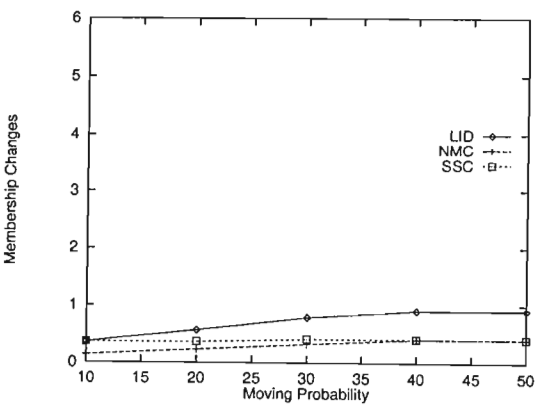

Figure 8.28: Cluster Membership Changes: N = 10

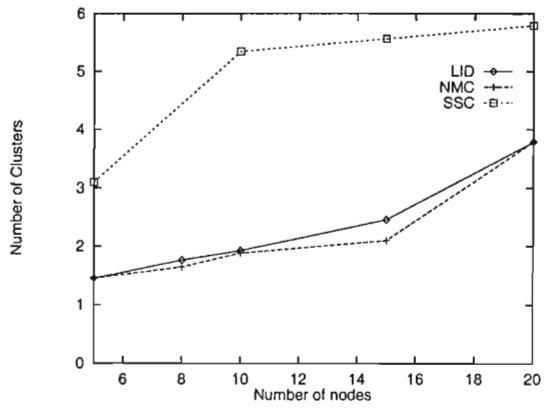

Figure 8.29: Number of Clusters: P = 20

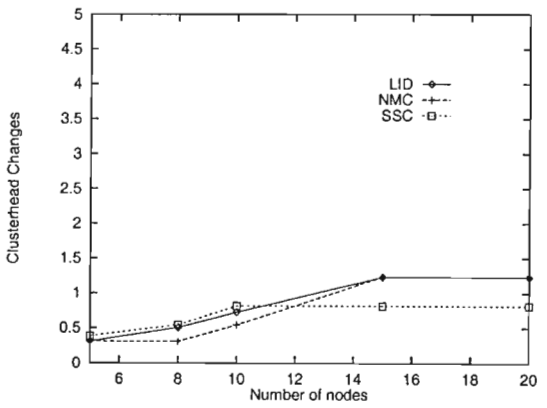

Figure 8.30: Clusterhead Changes: P = 20

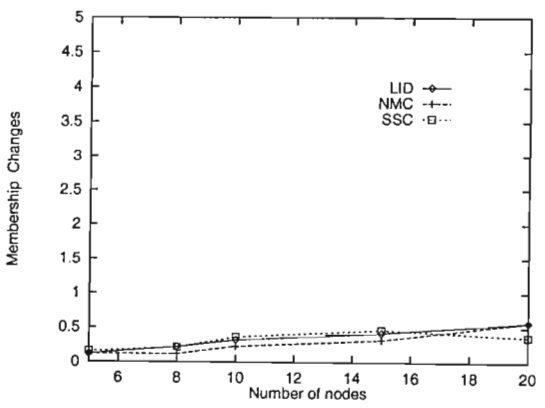

Figure 8.31: Cluster Membership Changes: P = 20

8.5 Summary and Conclusions

In this chapter, the simulation results of Lowest ID (LID), Node Mobility-based Clustering (NMC) and Signal Strength-based Clustering (SSC) algorithms were presented. Characteristics and performance of these clustering algorithms were compared as a function of moving probability and network size.

In general, higher moving probability results in more clusters, since formation of new clusters occurs as nodes move around. Of course, more nodes also result in more clusters. The number of clusterhead changes and cluster membership changes are low relative to number of clusters. Part of the reason is that a node can move just within its cluster or a cluster may move together without affecting the network topology.

Comparisons of the three clustering algorithms showed that the NMC clustering algorithm has a similar performance to the LID and SSC clustering algorithms in some cases, but in most cases the number of clusterhead and cluster membership changes is lower in NMC, especially for a relatively low mobility environment. Hence, the NMC algorithm seems attractive and can be used for cluster formation to help routing in ad hoc network applications that require low mobility.

Chapter 9

Conclusions and Future Work

In this chapter, we present a summary of the results of both limited flooding and the hierarchical clustering architecture. Then possible future research directions are described.

9.1 Summary and Conclusions

9.1.1 Limited flooding protocols

Chapter 5 introduced the limited flooding protocol. Limited flooding uses the same principles as flooding but intermediate nodes only pass on packets to some of their neighbours. Several variations of the limited flooding protocol based on deterministic, random and priority schemes were explored. The significance of this protocol is that it can be used in networks with unpredictable topological changes and relatively high mobility since it does not require maintaining routing tables.

In chapter 6, simulation results for the variations of the limited flooding protocols and pure flooding were presented. The performance metrics used included relaying load, channel utilisation and packet delivery ratio. Each metric was investigated as a function of two parameters, network size and transmission range.

The simulation results indicate:

- all limited flooding protocols have better performance than pure flooding for all metrics considered.

- while in general limited flooding protocols have similar performance, protocols based on random selection of paths are generally better than those based on likely paths.
- for protocols based on random selection of paths, RLF-R has better performance than both DLF-R and PLF-R for most metrics used in performance evaluation.
- for protocols based on likely paths, DLF-L has better packet relaying load and channel utilisation.
- packet delivery ratio increases with rise in transmission range.
- with increase in transmission range, the effect of mobility is reduced.

Thus limited flooding is a good candidate for broadcast routing in dynamic networks.

9.1.2 Hierarchical clustering architecture

In chapter 7 the Hierarchical Routing Protocol family was introduced. Here the routing within clusters and between clusters are viewed separately. Three possibilities for intra-cluster routing were proposed and it was argued that on-demand routing is suitable for inter-cluster routing. The purpose of hierarchical clustering is reduction of storage requirements, communication overhead and computational complexity at each mobile node.

In chapter 8 several clustering algorithms are discussed and then evaluated using simulation. The performance metrics used were the mean number of clusters, clusterhead changes and cluster membership changes. Each performance metric was evaluated as a function of network size and moving probability.

Simulation results show:

- the numbers of clusters, clusterhead changes and cluster membership changes mostly increase with increase in moving probability and network size.
- at relatively low mobility, all three clustering algorithms are stable.
- the mobility-based clustering algorithm has better performance in stablising the clusterhead.

9.2 Future Work

Further research can be carried out in many directions. Some possibilities are:

- *Further evaluation of the protocols.* They can be evaluated using additional performance metrics and more varied models. For example, performance metrics such as mean end-to-end delay and mean throughput can be used to evaluate each protocol at varied transmission range and network size. Also, the mobility model can be enhanced to include group mobility with mobile nodes that move with a common mission in some mobile ad hoc network applications.

- *Extension of limited flooding.* One possibility is to design additional mechanisms for further restricting packet propagation, or to employ it in the cluster based routing architecture.

- *Further exploration of hierarchical cluster based routing.* First, all routing schemes can be investigated with new mobility-based clustering algorithms that make use of dynamic location information. For example, mobility information may be tracked through a GPS and used to assist routing. Also, performance can be improved by introducing some mechanisms for optimizing the cluster size. This may involve defining the threshold value and designing algorithms for splitting the cluster when the number of nodes in the cluster exceeds this threshold and merging when number of nodes is less than this threshold. Also, protocol scalability can be further investigated by conducting the simulation experiments for larger network size.

- *Adaption of the protocols to support multicasting.* Limited flooding protocols can be adapted by restricting the broadcast message to selected multicast group instead of the entire nodes in the network.

- *Use of the clustering architecture in mobility management.* Using clusterhead in a stable clustering algorithm, the hierarchical clustering architecture can be used to track location information in mobile ad hoc networks.

Chapter 10

Bibliography

Bibliography

[Ame88] F. Amer, Y-N Lien, A. Ghieth: *Performance Evaluation of A Hierarchical Hybrid Adaptive Routing Algorithm for Large Scale Computer Communication Networks.* In proceedings of Computer Networking Symposium, pp. 266–275, April 1988.

[Aza96] Y. Azar, J. Naor, R. Rom: *Routing Strategies in Fast Networks.* IEEE Transactions on Computers, 45(2):165–173, 1996.

[Bag95] R. Bagrodia, W. Chu, L. Klenrock, G. Popek: *Vision, Issues and architecture for nomadic compting.* IEEE Personal Communications, pp. 14–27, December 1995.

[Bag98] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, H. Song: *PARSEC: A Parallel Simulation Environment for Complex systems.* IEEE Computer, Vol. 31(10):77–85, October 1998.

[Bah94] V. Bharghavan, A. Demers, S. Shenker and L. Zhang: *MACAW: A Media Access Protocol for Wireless LANs.* In Proceedings of ACM SIGCOMM'94, pp. 212–224, 1994.

[Bak95] A. bakre and B.R. Badrinath: *I-TCP: Indirect TCP for mobile hosts.* In proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS), PP. 136–143, June 1995.

[Bak84] D.J. Baker, A. Ephremides, J.A. Flynn: *The design and simulation of a mobile radio network with distributed control.* IEEE Journal of Selected Areas in Communications, 2(1):226–237, 1987

[Bak96] B.S. Bakshi, P. Krishna, D.K. Pradhan and N.H. Vaidya: *Providing Seamless Communication in Mobile Wireless Networks.* Technical Report TR–96–015, Texas A & M University, May 1996.

[Bal93] T. Ballardie, P. Francis, J. Crowcroft: *Core based Trees (CBT)-An architecture for Scalable Inter-Domain Multicast Routing.* In proceed-

ings of ACM SIGCOMM'93, San Francisco, CA, pp. 85–95, October 1993.

[Bar93] A. Bar-Noy and I. Kessler: *Tracking Mobile Users in Uireless Communication Networks.* IEEE Transaction on Information Theory, 39(6):1877-1886, November 1993.

[Bar95] A. Bar-Noy and I. Kessler, M. Sidi: *Mobile users: To update or not to Update?* Wireless Networks 1(1995):175–186, July 1995.

[Beh97a] J. Behrens: *Distributed routing for Very Large Networks Based on Link Vectors.* PhD Thesis, University of California, Santa Cruz, May 1997.

[Beh97b] J. Behrens, J.J. Garcia–Luna–Aceves: *Fast Dissemination of Link States Using Bounded Sequence Numbers with no Periodic Updates or Age Fields.* University of California, Santa Cruz, 1997.

[Ber92] D. Bertsekas, R.Gallager: *Data Networks.* Second edition, Prentice Hall, 1992.

[Bha99] A. Bhattacharya, S. K. Das: *LeZi–Update: An Information-Theoretic Approach to Track Mobile Users in PCS Networks.* In proceedings of the fifth ACM/IEEE Internationl Conference on Mobile Computing and Networking (MOBICOM'99), Seattle, Washington, pp. 1–12, August 1999.

[Bla96] U. Black *Mobile Wireless Networks*, First edition, Prentice Hall 1996.

[Bom98] E. Bommaiah, M. Liu, A. McAuley, and R. Talpade: *AMRoute: Ad hoc multicast routing protocol.* Internet protocol, work in progress, 1998.

[Bos92] L. Bosack: *Method and apparatus for routing communications among computer networks.* U.S. Patents assigned to Cisco systems, Inc., Menlo Park, CA, February 1992.

[CDP95] *CDPD system specification &iimplementation Guide*, January 1995.

[Che98] T.-W. Chen and M. Gerla: *Global State Routing: A New Routing Scheme for Ad-hoc Wireless Networks.* In proceedings of IEEE ICC'98, 5 pages, June 1998.

[Chi97] C.-C. Chiang: *Routing in Clustered Multi-hop, Mobile Wireless Networks with Fading Channel.* In proceedings of IEEE SICON'97, pp.197–211, April 1997.

[Com95] D. E. Commer: *Internetworking with TCP/IP*, vol. 1, third edition, Prentice Hall, 1995.

[Cor95] M.S. Corson, A. Ephremides: *A highly adaptive distributed routing algorithm for mobile wireless networks.* ACM/Baltzer wireless Networks Journal, 1(1):61–81, 1995.

[Cox92] D. Cox: *Wireless network access for personal communications.* IEEE communications, pp. 96–115, December 1992.

[Das97a] B. Das and V. Bharghavan: *Routing in Ad hoc Networks using Minimum Connected Dominating Set (MCDS).* In proceedings of 1997 IEEE International Conference on Communications (ICC'97), 1997.

[Das97b] B. Das, R. Sivakumar and V. Bharghavan: *Routing in Ad hoc Networks using Spine.* IEEE International Conference on Computers and Communications Networks '97, 1997.

[Day97] R.A. Dayem: *Mobile Data and Wireless LAN Technologies.* First edition, Prentice Hall, 1997.

[Dee90] S.E. Deering, D.R. Cheriton: *Multicast routing in Datagram Internetworks and Extended LANS.* ACM Transactions on Computer Systems, 8(2):85–110, May 1990.

[Dee96] S.E. Deering, D.L. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, L. Wei: *The PIM Architecture for Wide Area Multicast Routing.* IEEE/ACM Transactions on Networking, 4(2):153–162, April 1996.

[Den98] M.K. Denko and W. Goddard: *Routing Algorithms in Mobile Ad hoc Networks using Clustering.* In proceedings of the 13th MSc/PhD Annual Conference in Computer Science, University of Stellenbosch, South Africa, pp. 6-18, July 1998.

[Den99] M.K. Denko and W. Goddard: *Limited Flooding in Mobile Ad hoc Networks.* In proceedings of the 14th MSc/PhD Annual Conference in Computer Science, Golden Gate, South Africa, pp. 21-24, June 1999.

[Den01] M.K. Denko and W. Goddard: *Clustering in Mobile Ad hoc networks.* In proceedings of the 5th International Conference on Communication Systems (AFRICOM 2001), Breakwater lodge, Cape Town, South Africa, May 2001.

[Dub97] R. Dube: *Signal Stability based adaptive routing for Ad Hoc Mobile networks.* IEEE Personal Communications, pp. 36–4 February 19975.

[Est92] D. Estrin, Y. Rekhter, and S. Hotz: *Scalable inter-domain routing architecture.* Computer Communications Review, 22(4):40–52, October 1992.

[Est93] D. Estrin, M. Steenstrup and G. Tsudik: *A Protocol for route establishment and packet forwarding across multipoint internets.* IEEE/ACM Transactions on Networking, 1(1):56-70, February 1993.

[Ful95] C.L Fullmer and J.J. Gracia–Luna–Aceves: *Floor Acquisition Multiple Access (FAMA) for packet Radio networks.* In proceedings of ACM SIGCOMM95, 1995.

[Gra94] J.J. Gracia–Luna–Aceves and W.T. Zaumen. *Area–based, loop–free internet routing.* In proceedings of IEEE INFOCOM'94, Toronto, Canada, pp. 1000–1008, June 1994

[Gra95] J.J. Gracia–Luna–Aceves and J. Behrens: *Distributed, Scalable routing Based on Vectors of Link States.* IEEE Journal of Selected Areas in Communications, 13(8):1383–1395, October 1995.

[Ger95a] M. Gerla and J. T. Tsai: *Multicluster, mobile, multimedia Radio Networks.* Journal of Wireless Networks, 1(3):255–265, October 1995.

[Ger95b] M. Gerla, J.T. Tsai, N. Bambos and S.C. Chen: *A distributed, Mobile Wireless Infrastructure for Multimedia Applications.* Fifth WINLAB Workshop on Third Generation Wireless Networks, New Jersey, April 1995.

[Gru91] J.L. Grubb, *The Traveller's Dream Come True.*, IEEE Communications, 29(11):48–51, 1991.

[Haas97] Z.J. Haas, *A new routing protocol for reconfigurable Networks.* IEEE ICUPC'97, pp. 562-566, October 1997.

[Haa98] Z. J. Haas and M.R. Pearlman: *The Performance of Query Control Schemes for the Zone Routing Protocol.* ACM SIGCOMM'98, pp. 167–177, September 1998.

[Hal96] D.A. Hall: *Tactical Internet System Architecture for Task Force XXI.* In proceedings of the Tactical Communications Conference, Fort, Wayne, Indiana, May 1996.

[Hed88] C. Hedrick: *Routing Information Protocol.* RFC 1058, June 1988.

[Hui95] C. Huitema: *Routing in the Internet.* Prentice Hall, 1995.

[IEE97] IEEE Computer Society lANA MAN standards Committee: *Wireless LAN Medium Access Protocol (MAC) and Physical Layer (PHY) Specification.* IEEE Std 802.11-1997. IEEE, New York, 1997.

[Imi94] T. Imielinski and B.R. Badrinath: *Challenges in data management,* Communications of the ACM, 37(10):19-27, October 1994.

[Ioa91] J. Ioannidis: *IP based Protocols for Mobile Internetworking.* In Proceedings of ACM SIGCOMM, 1991.

[Ioa93] J. Ioannidis: *Protocol for Mobile Internetworking.* PhD Thesis, Columbia University, 1993.

[ISO89] International Standards Organisation: *Inter-Domain IS-IS Protocol.* ISO/IEC JTC/SC6 WG2 N323, September 1989.

[Iwa99] A. Iwata, C.-C. Chiang, G. Pei, M. Gerla, and T.-W. Chen: *Scalable Routing Strategies for Ad Hoc Wireless Networks.* IEEE Journal on Selected Areas in Communications, Special Issue on Ad-Hoc Networks, pp. 1369–1379, August 1999.

[Jin98] D. Jing and Z. Haas: *Dual Busy Tone Multiple Access (DBTMA): A new Medium Access Control for packet Radio Networks, submitted for publication, 1998.*

[John95] D. B. Johnson, *Scalable support for transparent mobile host internetworking,* Wireless Networks 1(1):311–321, 1995.

[John96] D.B. Johnson and D.A. Maltz: *Dynamic Source Routing in mobile ad hoc networks,* in Mobile Computing, edited by T. Imielinski and H. Korth, Kluwer, pp. 153–181, 1996.

[Jubin87] Z. Jubin and J.D. Tornow: *The DARPA Packet Radio Network protocols.* In proceedings of the IEEE, 75(1):21–32, January 1987.

[Kah78] R.S. Kahn, J. Gronemeyer, J. Burchfiel and R. Kunzelman: *Advances in Packet Radio technology.* In proceedings of IEEE, 66(11):1468–1496, November 1978.

[Kar90] P. Karn: *MACA: A new Channel Access Method for Packet Radio.* In Proceedings of ARRL/CRRL Amateur Radio 9th Computer Networking Conference, PP. 134–140, September 1990.

[Kle77] L. Kleinrock and F. Kamoun: *Hierarchical Routing for Large Networks: Performance Evaluation and Optimization,* Computer Networks, 1(1977): 155–174, 1977.

[Kra98] R. Kravets, P. Krishnan: *Power management techniques for mobile communication.* In proceedings of the fourth ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'98), Dallas, Texas, pp. 157–16, October 1998.

[Kri94] P. Krishna, N.H. Vaidya and D.K. Pradhan: *Location Management in Distributed Mobile Environments.* In proceedings of the Third Intl. Conference on Parallel and Distributed Information Systems, pp. 81–89, September 1994.

[Kri96] P. Krishna, M. Chatterjee, N.H. Vaidya and D.K. Pradhan: *A cluster based Approach for routing in Ad hoc networks.* In proceedings of Second USENIX Symposium on mobile and Location Independent Computing, pp. 1–10, January 1996.

[lau95] G.S. Lauer: *Packet Radio Routing.* In Routing in Communication Networks, edited by M. Steenstrup, Prentice Hall, pp. 351–396, 1995.

[Lee99] S.-J. Lee, M. Gerla, C.-C Chiang: *On demand Multicast routing Protocol for ad hoc networks.* In proceedings of IEEE WCNC'99, New Orleans, LA, pp. 1298–1304, September 1999.

[Les90] O. Lesser, R. Rom: *Routing by controlled flooding in communication networks.* In proceedings of IEEE (INFOCOM'90), San Francisco, California, pp. 910–917, June 1990.

[Liu95] G. Liu, G. Maquire, Jr: *A predictive mobility management algorithm for wireless mobile computing and communications.* In proceedings of IEEE International Conference on Universal Communications (ICUPC'95), Tokyo, Japan, November 1995

[Lin97] C.R. Lin and M. Gerla: *Asynchronous Multimedia Multi-hop Wireless Networks.* In proceedings of IEEE (INFOCOM'97), April 1997.

[Lor96] J.R. Lorch and A.J. Smith: *Reducing processor power consumption by improving time management in a single user Operating System.* In proceedings of the second ACM International Conference on Mobile Computing and Networking (MOBICOM'96), 1996.

[Lou91] K. Lougheed and Y. Rekhter. *Border Gateway Protocol 3 (BGP-3).* RFC 1267, SRI International, Menlo Park, CA, October 1991.

[Maa97] M. H. Maass: *Location Aware mobile applications based on directory services.* In the proceedings of the third Annual ACM/IEEE Intenational Conference on Mobile Computing and Networking (MOBICOM'97), Budapest, Hungary, pp. 23–33, September 1997.

[Mac98] Joseph P. Macker, M.S. Corson: *Mobile Ad hoc networking and the IETF.* Mobile computing and communication review, 2(4):9–12, October 1998.

[Mac00] Joseph P. Macker, M.S. Corson: *Mobile Ad hoc networking and the IETF.* Mobile computing and communication review, 4(1):13–16, January 2000.

[Mad95] U. Madhow, M.L. Hoing and K. Steiglitz: *Optimisation of wireless resources for personal communications mobility tracking.* IEEE/ACM Transaction Networking 3(6):698–706, 1995.

[MAN97] *Internet Engineering Task Force. MANET Working Group Charter* , http://www.ietf.org/html.charters/manet-charter.html, 1997.

[Mcc96] P. McConnell: *comparison of CDPD, ARDIS and RAM Wireless Data Communication systems.* Sirra Wireless, Inc., white paper, 1996.

[McQ74] J. M. McQuillan: *Design Considerations for Routing Algorithms in Computer Networks.* In proceedings of Seventh Hwaii International Conference Syst. Sci., pp. 22-24, January 1974.

[Mer95] A. Mercjant and B. Sengupta: *Assignment of cells to switches in PCS networks.* IEEE/ACM Transaction in Networking 3(5): 521–526, 1995.

[Moh94] S. Mohan and R. Jain. *Two user Location Strategies for personal Communication Services.* IEEE Personal Communications, 1(1):42–50, 1994.

[Moy94a] J. Moy: *Open Shortest Path First.* RFC 1583, March 1994.

[Moy94b] J. Moy: *Multicast Routing Extension for OSPF.* Communication of the ACM, vol.37, No. 8, August 1994.

[Mur96] S. Murthy and J.J. Garcia–Luna–Aceves: *An Efficient Routing Protocol for Wireless Networks.* ACM Mobile Networks and Applications, Special Issue on Routing in Mobile Communication Networks, 1(1):183–197, October 1996.

[Nar97] B. Narendran, J. Sienicki, S. Yajnik and P. Agrawal: *Evaluation of and adaptive power and error control algorithms for wireless systems.* In proceedings of IEEE International Conference on Communications (ICC'97), 1997.

[Ng99] M.J.-Ng. and I.-T. Lu: *A Peer-to-Peer zone-based two-level link state routing for mobile Ad Hoc Networks.* IEEE Journal of Selected Areas in Communications, Special Issue on Ad-Hoc Networks, pp. 1415-25, August 1999.

[Pag97] E. Pagani, G.P. Rossi: *Reliable broadcast in mobile multi-hop packet networks.* , Proceedings of the third annual ACM/IEEE International Conference on mobile computing and networking (MOBICOM'97), pp. 34–42, 1997.

[Par97a] V.D. Park and M.S. Corson: *A highly adaptive distributed routing algorithm for mobile wireless networks.* In proceedings of IEEE (INFOCOM'97), Kobe, Japan, pp. 1405–1413, April 1997.

[Par97b] V. Park and M.S. Corson: *Temporally-Ordered Routing Algorithm (TORA97).* Internet Draft, Internet Engineering Task Force, work in progress, December 1997.

[Per94] C.E. Perkins and P. Bhagwat: *Highly Dynamic Destination Sequenced Distance Vector Routing (DSDV) for mobile computers.* In proceedings of ACM SIGCOMM, pp. 234–244, 1994.

[Per96] C.E. Perkins, ed., IETF: *IP Mobility Support.* IETF RFC 2002, October 1996.

[Per97] C. E. Perkins, *Mobile Ad hoc Networking Terminology.*, IETF Draft, Work in Progress, 1997.

[Per99] C.E. Perkins, Elizabeth M. Royer, Samir R. Das: *Ad Hoc On-demand Distance Vector Routing.* In proceedings of the second IEEE workshop on Mobile Computing Systems and Applications, pp. 90–100, February 1999.

[Pos81] J. Postel: *Internet Protocol.*, Internet RFC 791, 1981.

[Rap95] .S. Rappaport: *Wireless Communications: Principles and Practices.* Prentice Hall, October 1995.

[Ros96] C. Rose: *Minimizing the average cost of paging and registration: a timer based method.*, Wireless Networks, 2(2):109–116, June 1996.

[San98] M. Sanchez: *Mobility models..* http://www.disca.upv.es/misan/mobmodel.htm, 1998.

[Sat93] M. Satyanarayanan: *Accessing information on demand any time.* IEEE Personal Communications, February 1996.

[Sch87] N. Schacham and J. Westcott: *Future directions in packet radio architectures and protocols.* In proceedings of the IEEE, 75(1):83–99, January 1987.

[Seg97] A. Segall, P. Bhagwat, A. Krishna: *QoS Routing using Alternative paths.* 1997

[Sha96] J. Sharony: *A mobile Radio Network Architecture with Dynamically Changing Topology using Virtual Subnets.* IN Journal of ACM/Baltzer Mobile Networks and Applications, 1(1):75–86, 1996.

[Su98] C-J. Su, L. Tassiulas: *Joint Broadcast Scheduling and user's cache management for efficient information delivery.* In proceedings of the fourth ACM/IEEE Internation Conference on Mobile Computing and Networking (MOBICOM'98), Dallas, Texas, pp. 33-42, October 1998.

[Tab93] H. Xie,S. Tabbane and D.J. Goodman: *Dynamic location area management and performance analysis.* In proceedings of 43rd IEEE Vehicular Technologies Conference, pp. 536–539, 1993.

[Tan96] A.S. Tanenbum: *Computer Networks.* Third edition, Prentice Hall, 1996

[Tho] S.A. Thomaseroka: *IPng and the TCP/IP protocols: Implementing the next generation Interne.* John Wiley &Sons, 1996.

[Toh96] C.-H. Toh: *A novel distributed routing protocol to support ad-hoc mobile computing.* In proceedings of 15th IEEE Annual International Phoenix Conference on Computer Communications ,pp. 480–486, 1996.

[Top89] D.M. Topkis: *Performance Analysis of information dissemination by flooding.* IEEE Journal of Selected Areas in Communication, vol. 7, no. 3, April 1989.

[Tsa89] W.T. Tsai, C.V. Ramamoorthy, W.K. Tsai and O. Nishiguchi: *An adaptive Hierarchical Routing Protocol.* IEEE Transactions in communications, 38(8):1059-1075, 1989.

[Val99] A.G. Valko: *Cellular IP: A new approach to Internet Host Mobility.* ACM Computer Communications Review, vol. 29, no. 1, January 1999.

[Wad93] H. Wada, T. Ohnishi and B. Marsh: *Mobile Computing Environment based on Internet Packet Forwarding.* Winter USENIX Sysmposium, 1993.

[Wei91] M. Weiser: *The Computer for the 21st Century.* Scientific America, pp. 94–104, September 1991.

[Won95] P. Wong and D. Britland: *Mobile Data Communications Systtems.* Artech House, 1995.

[Wu98] C.W. WU, Y.C. Tay and C-K. Toh: *Ad hoc Multicast Routing protocol utilizing Increasing id-numberS (AMRIS).* IETF-Internet draft, November, 1998.

[Wyc95] J. Wyckoff, D. Doering, B. Hume: *NetWare Link Services Protocol: An Advanced Theory of Operations.* Novell AppNotes, November 1995.

[Zau95] W.T. Zaumen and J.J. Gracia–Luna–Aceves: *Shortest multipath routing using diffusing computations.* Technical report, SRI International, 1995.

[Zon97] M.M. Zonoozi and P. Dassanayake: *User mobility modeling and characterisation of mobility patterns.* IEEE Journal of Selected Areas in Communications, 15(7):1239–1252, September 1997.

Appendix A

Abbreviations and Terminology

- AIRLAN: Wireless network interface card of Solectek, Inc.
- ARLAN: Wireless bridge/network interface card of AIRONET Wireless Communications, Inc.
- ARDIS: Advanced Radio Data Integrated System.
- ATM: Asynchronous Transfer Mode.
- BGP: Border Gateway Protocol.
- CDMA: Code Division Multiple Access.
- CDPD: Cellular Digital Packet Data.
- DNS: Domain Name System.
- EIGRP: Extended Interior Gateway Routing Protocol.
- FDDI: Fiber Distributed Data Interface.
- GPS: Global Positioning System
- IDPR: Inter-Domain Policy Routing.
- IEEE: Institute for Electrical and Electronic Engineers.
- IETF: Internet Engineering Task Force.
- IGRP: Interior Gateway Routing Protocol.
- IP: Internet Protocol.

- IPX: Internet Protocol Exchange.
- ISO: International Standards Organisation.
- ISO/OSI: Open System Interconnection reference model.
- IS-IS: Intermediate System to Intermediate System.
- MAC: Medium Access Control Protocol
- Mobile IP: Mobile Internet Protocol.
- Mobile IPX: Mobile IP protocol based on IPX.
- NLSP: Netware Link Services Protocol.
- OSI: Open System Interconnection.
- OSPF: Open Shortest Path First.
- PDA: Personal Digital Assistant.
- PCN: Personal Communication Network.
- RangeLAN2: Wireless network adapter of Proxim, Inc.
- RIP: Routing Information Protocol.
- RMD: RAM Mobile Data service.
- TCP: Transmission Control Protocol.
- TCP/IP: Internet protocol stack.
- WaveLAN: Wireless network adapter of AT&T.

Appendix B

Limited Flooding Algorithms Source Code

```
/*************************************************************************/
/*                                                                       */
/* Mobile Ad hoc Network Simulation Program: includes all modules        */
/*                                                                       */
/* Author: Mieso Denko                                                   */
/*                                                                       */
/* Last Moddified:  August 23, 2000                                      */
/*                                                                       */
/*                                                                       */
/ *************************************************************************/
#include "messagess1.h"
#define NumberOfNodes
#define PI 3.141
#define DEBUG
#define ARRIVAL        0.4
#define DATA_RATE       512
#define Real_to_sim    DATA_RATE*1024.0/1000
extern double drand48();
extern double srand48();

// THE PROTOCOLS

#define LMFLOOD1    1
#define LMFLOOD2    0
#define LMFLOOD3    0
#define LMFLOOD4    0
#define LMFLOOD5    0
#define LMFLOOD6    0
#define LMFLOOD0    0

// SIMULATION ENTITIES

entity MOBILITY();
entity MOBILITY_TRACK();
entity Link();
entity Node(int,ename);
entity TRANSMITTER(ename,ename);
entity RECEIVER(ename,ename);

// DEFAULT VALUES FOR PARAMETERS AND CONSTANTS

#define RANGE           0.4
#define Area_Size       1.0
#define mins_pkt        2.000
#define maxs_pkt        4.500
#define minl_pkt        2.000
#define maxl_pkt        5.000
#define ack_timeout     200
#define MAX_SIM_TIME    1048576
#define max_sim_time    2000
#define hello_length    2.442
#define ack_length      2.100
#define VEL_MIN         0.200
#define VEL_MAX         0.500
```



```
#define MIN_PAUSE       25.0
#define MAX_PAUSE       50.0
#define MIN_MOBILITY    50.0
#define MAX_MOBILITY    100
#define MIN_NUM_PKT     1
#define MAX_NUM_PKT     10
#define ne_exp_time     500
#define hello_period    180
#define LFP_FLG  1
int HELLO_LENGTH  =  hello_length*Real_to_sim;
int ACK_LENGTH    =  ack_length*Real_to_sim;
int MINS_PKT      =  mins_pkt*Real_to_sim;
int MAXS_PKT      =  maxs_pkt*Real_to_sim;
int MINL_PKT      =  minl_pkt*Real_to_sim;
int MAXL_PKT      =  maxl_pkt*Real_to_sim;
int ACK_TIMEOUT   =  ack_timeout*Real_to_sim;
int NE_EXP_TIME   = ne_exp_time*Real_to_sim;
int HELLO_PERIOD  = hello_period*Real_to_sim;
double tx_usage_data[NumberOfNodes];
double tx_usage_rt[NumberOfNodes];
double rx_usage_data[NumberOfNodes];
double rx_usage_rt[NumberOfNodes];
double avg_mobility = 0.0,double avg_comm = 0.0;
double avg_data = 0.0,avg_rt = 0.0;
int delay [10],new_time[10],old_time[10];
int avg_num_hellos = 0, avg_data_timeout = 0;
int avg_data_success = 0,  Avg_route_success= 0;
int avg_data_sent = 0, avg_data_received = 0;
int avg_ack_sent 0,avg_ack_received = 0;

// SOME IMPORTANT FUNCTIONS

int expon(int mean)
  {
   return (int) ((-log((double)(double)(1.0-drand48())))* ((double)mean));
  }
int uniform_random (int low, int high)
 {
  return low + rand()%(high-low+1);
 }
double NODE_DISTANCE(int A, int B)
 {
  double xa, ya, xb, yb ;
  double answer;
  int na, nb;
   if (A == B) {  return 0.0; }
     xa = loc_data[A].LocX;
     ya = loc_data[A].LocY;
     xb = loc_data[B].LocX;
     yb = loc_data[B].LocY;
     answer = sqrt((xb-xa)*(xb-xa) + (yb-ya)*(yb-ya));
     return answer;
 }
int NODE_NUMBER(ename A)
 {
```

```
  int i;
  for (i=0;i<NumberOfNodes;i++) {
    if ename_cmp(A,loc_data[i].NodeName) return loc_data[i].NodeNumber; }
  return -1;
 }
ename NODE_NAME(int i)

 {
  return loc_data[i].NodeName;
 }

int NODE_DEGREE()
{
 int degrees =0;
 int i,from;

       for (i=0;i<NumberOfNodes;i++)  {
        if (( from!=i) &&(NODE_DISTANCE(from,i)) <=RANGE)
              degrees++;
        }
          return  degrees;
       }

entity MOBILITY()

 {
 int i;
 double x,y;
 ename mob_con;
 char L;
     mob_con = new MOBILITY_TRACK();

 for(;;)
  {
          hold(200);
  {

 }
    for(i=0;i<NumberOfNodes;i++)
  {
      if(loc_data[i].Moving)
  {
        switch (loc_data[i].Motion_Type)
  {
 case 'L':
         x = loc_data[i].LocX + 200*loc_data[i].Mob_coeff[0];
           if (x >= Area_Size)
           {
             x = Area_Size;
             loc_data[i].Mob_coeff[0] = -1 * loc_data[i].Mob_coeff[0];
           }
           if (x <= 0.0)
            {
             x = 0.0;
             loc_data[i].Mob_coeff[0] = -1 * loc_data[i].Mob_coeff[0];
```



```
           }
           loc_data[i].LocX = x;
           y = loc_data[i].LocY+ 200*loc_data[i].Mob_coeff[1];
        if (y >= Area_Size)
           {
             y = Area_Size;
             loc_data[i].Mob_coeff[1] = -1 * loc_data[i].Mob_coeff[1];
            }
           if (y <= 0.0)
             {
             y = 0.0;
             loc_data[i].Mob_coeff[1] = -1 * loc_data[i].Mob_coeff[1];
             }
         loc_data[i].LocY = y;
         break;
         default:
       printf("# Not valid motion profile\n");
       }
     }
    }
  }
 }

void setMotionParams(int i)
  {
  int vel_variation = (VEL_MAX - VEL_MIN)*Real_to_sim;
  double min_vel = VEL_MIN*Real_to_sim/1000;
  double vel;
  int max_slope = 90;
  int min_slope  = 0;
  double slope,scaling;
  double hor,ver;
  vel = VEL_MIN + (VEL_MAX-VEL_MIN)*drand48();
  loc_data[i].vel = vel;
  slope = (PI/2)*drand48();
  hor = vel/sqrt(1+tan(slope)*tan(slope));
  ver = tan(slope)*hor;
  scaling= 1000*Real_to_sim;
  hor = hor/scaling;
  ver = ver/scaling;
  if((int)(2.0*drand48()))
    hor = -1 * hor;
  if((int)(2.0*drand48()))
    ver = -1 * ver;
    loc_data[i].Mob_coeff[0] = hor;
    loc_data[i].Mob_coeff[1] = ver;
    }
entity MOBILITY_TRACK()
 {
  int i,now, MOBILITY_TIME, current_event_time;
  int next_event[NumberOfNodes];

  for(i=0;i<NumberOfNodes;i++)
    {
      loc_data[i].Moving = (int)(2.0*drand48());
```

```
     setMotionParams(i);
    if (loc_data[i].Moving)
     {
     next_event[i] = Real_to_sim*(MIN_MOBILITY + (MAX_MOBILITY -
           MIN_MOBILITY)* 1.0*drand48());
  if(next_event[i] > max_sim_time*Real_to_sim)
      {
         avg_mobility += loc_data[i].vel*max_sim_time*Real_to_sim;
      }
    else
      {
      avg_mobility += loc_data[i].vel*next_event[i];
      }
     }
 else
   {
      next_event[i] = Real_to_sim*(MIN_PAUSE + (MAX_PAUSE -
       MIN_PAUSE)* 1.0*drand48());
   }
}
for (;;)
  {
  now = simclock();
   for(i=0;i<NumberOfNodes;i++)
    {
      if(now >= next_event[i])
      {
        if(loc_data[i].Moving)
       {
        loc_data[i].Moving = 0;
        next_event[i] += Real_to_sim*(MIN_PAUSE + (MAX_PAUSE -
           MIN_PAUSE)* 1.0*drand48());
       }
    else
     {
         setMotionParams(i);
         loc_data[i].Moving = 1;
         MOBILITY_TIME = Real_to_sim*(MIN_MOBILITY + (MAX_MOBILITY -
             MIN_MOBILITY)* 1.0*drand48());
        current_event_time = next_event[i];
        next_event[i] += MOBILITY_TIME;
        if(next_event[i] > max_sim_time*Real_to_sim)
        {
         avg_mobility += loc_data[i].vel*(max_sim_time*Real_to_sim -
               current_event_time);
        }
       else
          {
          avg_mobility += loc_data[i].vel*MOBILITY_TIME;
          }
      }
     }
     }
    hold (1);
 }
```



```
 }

// SOME FUNCTIONS

void PacketType(int i)
   {
     int priority;
     if ((int)(2.0*drand48()))
         ninfo[i].ptype   =1;
     else
         ninfo[i].ptype   = 0;
   }
void randomd(int i)

   {
     int dest;
     int Max =NumberOfNodes-1;
     dest = i;
     for(;dest == i;)
     dest =  (int)(NumberOfNodes*1.0*drand48());
     ninfo[i].Destination = dest;
   }
void randomp(int i)
  {
   if ((int)(2.0*drand48()))

   ninfo[i].pkt_length = MINS_PKT + (MAXS_PKT - MINS_PKT)* 1.0*drand48();
   else
    ninfo[i].pkt_length = MINL_PKT + (MAXL_PKT - MINL_PKT)* 1.0*drand48();
 }

// ROUTING RELATED FUNCTIONS

void del_neighbor_list(struct neighbor_list  *list)
  {
  struct neighbor_list *current, *next;

  current = list;
  while(current) {
 next = current->next_neighbor;
    free(current);
    current = next;
  }
 }

struct rmpr_rtentry* rmpr_rtlookup(struct routing_table *rt,int ipaddr)
  {
  struct routing_table *lptr;

  lptr = rt;
  while(lptr != NULL) {
    if (lptr->rtentry->dest == ipaddr) return lptr->rtentry;
    lptr = lptr->next;
  }
  return NULL;
```

```
  }
struct routing_table* rmpr_rtaddroute(struct routing_table *rt,int
 dest, int ptype, int metric, unsigned long sequence, unsigned int
 flags) {
  struct rmpr_rtentry *route;
  struct routing_table *table_entry, *lptr;
  if((route = rmpr_rtlookup(rt,dest)) != NULL) {
    if((sequence>route->sequence)||((sequence==route->sequence)&&
        (metric<route->metric))) {
      route->dest = dest;
      route->ptype = ptype;
      route->metric = metric;
      route->sequence = sequence;
      route->flags = flags;
    }
  }
  else {
    if((route = malloc(sizeof(struct rmpr_rtentry))) == NULL) {
      printf("#Out of memory\n");
      exit(-1);
    }
    if((table_entry = malloc(sizeof(struct routing_table))) == NULL) {
      printf("#Out of memory\n");
      exit(-1);
    }
 if (rt == NULL){
      rt = table_entry;
      lptr = rt;
    }
    else {
      lptr = rt;
      while(lptr->next != NULL) lptr = lptr->next;
      lptr->next = table_entry;
      lptr = lptr->next;
    }

    lptr->rtentry = route;
    lptr->next = NULL;
    lptr->rtentry->dest = dest;
    lptr->rtentry->ptype = ptype;
    lptr->rtentry->metric = metric;
    lptr->rtentry->sequence = sequence;
    lptr->rtentry->flags = flags;
    lptr->rtentry->source_list = NULL;
  }
  return rt;
 }

struct routing_table* rmpr_rtdelroute(struct routing_table *rt,int dst) {
  struct routing_table *tmp_rt, *rt_prev, *lptr;

  lptr = rt;
  if(!lptr) return NULL;
  if(lptr->rtentry->dest ==dst) {
    tmp_rt = lptr->next;
```



```
    del_neighbor_list(lptr->rtentry->source_list);
    if(lptr->rtentry) free(lptr->rtentry);
    if(lptr) free(lptr);
    rt = tmp_rt;
    return rt;
  }
  while(lptr->next) {
    if(lptr->next->rtentry->dest == dst) {
      tmp_rt = lptr->next;
      lptr->next = lptr->next->next;
      del_neighbor_list(tmp_rt->rtentry->source_list);
      if(tmp_rt->rtentry) free(tmp_rt->rtentry);
      if(tmp_rt) free(tmp_rt);
  return rt;
    }
    rt_prev = lptr;
    lptr = lptr->next;
  }
  if(lptr->rtentry->dest == dst) {
    rt_prev->next = NULL;
    del_neighbor_list(lptr->rtentry->source_list);
    if(lptr->rtentry) free(lptr->rtentry);
    if(lptr) free(lptr);
  }
  return rt;
 }

void rmpr_rtaddsrc(  struct routing_table *rt,int dst,int ipaddr)
 {
  struct rmpr_rtentry *route;
  struct neighbor_list *list_item, *last_item;

  if((route = rmpr_rtlookup(rt,dst)) != NULL) {
    if((list_item = malloc(sizeof(struct neighbor_list))) == NULL) {
      printf("#Out of memory\n");
      exit(-1);
    }
    list_item->neighbor = ipaddr;
    list_item->next_neighbor = NULL;
    if (route->source_list == NULL) {
      route->source_list = list_item;
    }
    else {
      last_item = route->source_list;
      while(last_item->next_neighbor) {
        last_item = last_item->next_neighbor;
   }
      last_item->next_neighbor = list_item;
    }
  }
 }

struct neighbor_list* Lookup_ne( struct neighbor_list *neighbors,int ipaddr)
  {
  struct neighbor_list *lptr;
```

```
  lptr = neighbors;
  while(lptr != NULL) {
    if (lptr->neighbor == ipaddr) return lptr;
    lptr = lptr->next_neighbor;
  }
  return NULL;
}

struct neighbor_list* Add_ne( struct neighbor_list *neighbors,int ipaddr,
          unsigned long seq_no) {
  struct neighbor_list *ne, *lptr;
  if((ipaddr < 0) && (ipaddr >= NumberOfNodes)) {
      exit(-1);
  }
  if((ne = Lookup_ne(neighbors,ipaddr)) != NULL) {
    ne->neighbor = ipaddr;
    ne->sequence_no = seq_no;
  }
  else {
    if ((ne = malloc(sizeof(struct neighbor_list))) == NULL) {
      printf("#Out of memory\n");
      exit(-1);
    }
        if (neighbors == NULL) {
          neighbors = ne;
          lptr = neighbors;
         }
            else {
              lptr = neighbors;
              while(lptr->next_neighbor != NULL)
              lptr = lptr->next_neighbor;
              lptr->next_neighbor = ne;
              lptr = lptr->next_neighbor;
              }
                lptr->neighbor = ipaddr;
                lptr->sequence_no = seq_no;
                lptr->next_neighbor = NULL;
  }
  return neighbors;
}
struct neighbor_list* Del_ne( struct neighbor_list  *neighbors,int ne) {
  struct neighbor_list  *tmp_ne, *ne_prev, *lptr;
  lptr = neighbors;
  if(!lptr) return NULL;
  if(lptr->neighbor == ne) {
    tmp_ne= lptr->next_neighbor;
    if(lptr) free(lptr);
    neighbors = tmp_ne;
    return neighbors;
  }
  while(lptr->next_neighbor) {
    if(lptr->next_neighbor->neighbor == ne) {
      tmp_ne = lptr->next_neighbor;
      lptr->next_neighbor = lptr->next_neighbor->next_neighbor;
```



```
      if(tmp_ne) free(tmp_ne);
      return neighbors;
    }
    ne_prev = lptr;
    lptr= lptr->next_neighbor;
  }
  if(lptr->neighbor == ne) {
    ne_prev->next_neighbor = NULL;
    if(lptr) free(lptr);
  }
  return neighbors;
}
struct data_fwl* rmpr_dflookup(struct data_fwl *fwl, int ipaddr,
int daddr,unsigned long seq_no)
{
  struct data_fwl *local_fwl;
  local_fwl = fwl;
  while(local_fwl != NULL) {
      if ((local_fwl->src == ipaddr) &&(local_fwl->sequence_no == seq_no) && (local_fwl->dst==daddr

      local_fwl = local_fwl->next;
  }
  return NULL;
}
struct ack_fwl* rmpr_aflookup(struct ack_fwl *fwla, int ipaddr, int daddr,unsigned long seq_no)
{
  struct ack_fwl *local_fwla;
  local_fwla = fwla;
  while(local_fwla != NULL) {
      if ((local_fwla->src == ipaddr) &&(local_fwla->sequence_no == seq_no)
           && (local_fwla->dst==daddr)) return local_fwla;
      local_fwla = local_fwla->next;
  }
  return NULL;
}
 struct data_fwl* rmpr_fwladd(struct data_fwl *fwl,int src,int dst,
      unsigned long seq_no, int metric)
{
  struct data_fwl *existing_fwl, *new_fwl, *lptr;
if((existing_fwl = rmpr_dflookup(fwl,src,dst,seq_no)) != NULL) {
      if(existing_fwl->hops_to_source > metric) {
      existing_fwl->src = src;
      existing_fwl->dst = dst;
      existing_fwl->sequence_no = seq_no;
      existing_fwl->hops_to_source = metric;

      return fwl;
       }
          }
 else {
      if((new_fwl = malloc(sizeof(struct data_fwl))) == NULL) {
       printf("#Out of memory\n");
       exit(-1);
     }
      if (fwl== NULL){
```

```
      fwl = new_fwl;
      lptr = fwl;
     }
             else {

              lptr = fwl;
              while(lptr->next != NULL) lptr = lptr->next;
              lptr->next = new_fwl;
               lptr= lptr->next;
                }
    lptr->src = src;
    lptr->dst = dst;
    lptr->sequence_no = seq_no;
    lptr->hops_to_source = metric;
     return fwl;
 }
}
struct ack_fwl* rmpr_afwladd(struct ack_fwl *fwla,int src,int dst,
       unsigned long seq_no, int metric)

{
 struct ack_fwl *existing_fwla, *new_fwla, *lptr;

if((existing_fwla = rmpr_aflookup(fwla,src,dst,seq_no)) != NULL) {
     if(existing_fwla->hops_to_source > metric) {
     existing_fwla->src = src;
     existing_fwla->dst = dst;
     existing_fwla->sequence_no = seq_no;
     existing_fwla->hops_to_source = metric;

     return fwla;
      }
         }

else {

  if((new_fwla = malloc(sizeof(struct ack_fwl))) == NULL) {
      printf("#Out of memory\n");
      exit(-1);
    }
    if (fwla== NULL){
      fwla = new_fwla;
      lptr = fwla;
     }
             else {

              lptr = fwla;
              while(lptr->next != NULL) lptr = lptr->next;
              lptr->next = new_fwla;
               lptr= lptr->next;
                }
    lptr->src = src;
    lptr->dst = dst;
    lptr->sequence_no = seq_no;
    lptr->hops_to_source = metric;
```



```
      return fwla;
  }
 }
void display_neighbors(struct neighbor_list *neighbors, int
  node_num,int current_time) {
  struct neighbor_list *lptr;
  lptr = neighbors;
 }

void display_data_flist(struct data_fwl *fwl, int node_num, int current_time)

 {
  struct data_fwl *lptr;
  lptr = fwl;
  while(lptr) {
  lptr= lptr->next;
  }
 }

void display_ack_flist(struct ack_fwl *fwla, int node_num, int
  current_time)
 {
  struct ack_fwl *lptr;
  lptr = fwla;
 }
void display_routes(struct routing_table *rt, int node_num, int
  current_time) {
  struct routing_table *lrt;
  struct rmpr_rtentry *lroute;
  struct neighbor_list *src_list;
  lrt = rt;
   while(lrt) {
    if((lroute = lrt->rtentry)!=NULL) {
      src_list = lroute->source_list;
    }
    lrt = lrt->next;
  }
 }

// THE COMMUNICATOR ENTITY

entity COMMUNICATOR()
 {

  int i,dest, now;
  int next_packet[NumberOfNodes];
  int next_event[NumberOfNodes];
   hold(1);
  for (i=0;i<NumberOfNodes;i++) {
      randomd(i);
      ninfo[i].Sending = (int)(2.0*drand48());
      if (ninfo[i].Sending) {

        next_event[i] = uniform_random(MIN_NUM_PKT,MAX_NUM_PKT)*Real_to_sim;
```

```
        randomp(i);
        PacketType(i);

     send SEND_MSG{ninfo[i].Destination,ninfo[i].ptype,ninfo[i].pkt_length}
          to NODE_NAME(i) ;

     next_packet[i] = ninfo[i].pkt_length;
       }
        else {
 next_event[i] = expon((int) ((float)1/((float)ARRIVAL)))*Real_to_sim;
        }
 }
 now = simclock();
 while(now < Real_to_sim*max_sim_time) {
   now = simclock();

   for (i=0;i<NumberOfNodes;i++) {

        if (now >=next_event[i]) {

             if(ninfo[i].Sending) {
              ninfo[i].Sending = 0;
  next_event[i]+= expon((int) ((float)1/((float)ARRIVAL)))*Real_to_sim;

     }


           else {
            randomd(i);
            randomp(i);
            PacketType(i);
            ninfo[i].Sending = 1;

     send SEND_MSG{ninfo[i].Destination,ninfo[i].ptype,ninfo
         [i].pkt_length} to  NODE_NAME(i) ;
  next_packet[i] = now + ninfo[i].pkt_length;
  next_event[i] += uniform_random(MIN_NUM_PKT,MAX_NUM_PKT)* Real_to_sim;
            }
        }
   if((now >= next_packet[i])&& ninfo[i].Sending) {
     randomp(i);
     PacketType(i);
     send SEND_MSG{ninfo[i].Destination,ninfo[i].ptype,ninfo[i].pkt_length}
         to  NODE_NAME(i) ;
  }
   }
     hold(4);
}
}

entity TRANSMITTER (ename Parent, ename Link)
 {
  int i, src, dest, from, ptype, length, hops, prev_hop,sqno;
  unsigned long seq_no,seq_nop,seq_noa;
  struct message_neighbors hello_ne[NumberOfNodes];
```



```
 int ParentID;
 int tx_ready_time_data;
 int tx_ready_time_rtng;
 int now;
 ParentID = NODE_NUMBER(Parent);
 tx_ready_time_data = 0;
 tx_ready_time_rtng = 0;
 tx_usage_data[ParentID] = 0.0;
 tx_usage_rt[ParentID] = 0.0;
 for(;;) {
          now = simclock();
            {
  receive(HELLO_MSG h)  when (now >= tx_ready_time_rtng)
            {
              src = h.src;
              seq_no = h.sequence_no;
              i =0;
                while(h.hello_ne[i].neighbor != -1)
                   {
                     hello_ne[i].neighbor = h.hello_ne[i].neighbor;
                     hello_ne[i].sequence_no = h.hello_ne[i].sequence_no;
                       i++;
                        }
                     hello_ne[i].neighbor = -1;
       send HELLO_MSG{src,seq_no,hello_ne} to Link;
       tx_ready_time_rtng = now + HELLO_LENGTH;
       tx_usage_rt[ParentID] += HELLO_LENGTH;
       avg_num_hellos++;
    }
or receive(DATA_PKT datap)  when (now >= tx_ready_time_data) {

     dest = datap.dest;
      src = datap.src;
      from = datap.from;
      sqno = datap.sequence_no;
      ptype = datap.ptype;
      length = datap.length;
       send DATA_PKT{dest,src,from,sqno,ptype,length} to  Link;
      tx_ready_time_data = now + length;
      tx_usage_data[ParentID] += length;
    }
or receive( DATA_ACK dataa) when (now >= tx_ready_time_data)  {
      dest = dataa.dest;
      src = dataa.src;
      from = dataa.from;
      sqno = dataa.sequence_no;
      ptype = dataa.ptype;
      send DATA_ACK{dest,src,from,sqno,ptype} to  Link;
      tx_ready_time_data = now + ACK_LENGTH;
      tx_usage_data[ParentID] += ACK_LENGTH;
    }
or timeout after (5) {
}
  }
}
```

```
}

entity RECEIVER (ename Parent, ename Link)
 {
  int i, src, dest, from, ptype, length, hops, prev_hop;
  unsigned long seq_no,seq_nop,seq_noa,sqno;
  struct message_neighbors hello_ne[NumberOfNodes];
  int ParentID;
  int rx_ready_time_data;
  int rx_ready_time_rtng;
  int now;
  int count;
  ParentID = NODE_NUMBER(Parent);
  rx_ready_time_data = 0;
  rx_ready_time_rtng = 0;
  rx_usage_data[ParentID] = 0.0;
  rx_usage_rt[ParentID] = 0.0;
  count =0;
  for(;;) {
  now = simclock();
      {
    receive(HELLO_MSG he)  when (now >= rx_ready_time_rtng)  {
        src = he.src;
        seq_no = he.sequence_no;

              i=0;
                 while(he.hello_ne[i].neighbor != -1) {
                 hello_ne[i].neighbor = he.hello_ne[i].neighbor;
                 hello_ne[i].sequence_no = he.hello_ne[i].sequence_no;
                 i++;
              }

     hello_ne[i].neighbor = -1;
     send HELLO_MSG{src,seq_no,hello_ne} to  Parent after HELLO_LENGTH;
     rx_ready_time_rtng = now + HELLO_LENGTH;
     rx_usage_rt[ParentID] += HELLO_LENGTH;
     avg_num_hellos++;
      }
      or receive(DATA_PKT datap) when (now >= rx_ready_time_data)

       {
        dest = datap.dest;
        src = datap.src;
        from = datap.from;
        sqno = datap.sequence_no;
        ptype = datap.ptype;
        length = datap.length;
        send DATA_PKT{dest,src,from,sqno,ptype,length} to  Parent after length;
        rx_ready_time_data = now + length;
        rx_usage_data[ParentID] += length;
       }
      or receive(DATA_ACK dataa) when (now = rx_ready_time_data) {

        dest =dataa.dest;
        src = dataa.src;
```



```
        from = dataa.from;
        sqno = dataa.sequence_no;
        ptype = dataa.ptype;
       send DATA_ACK{dest,src,from,sqno,ptype} to   Parent after ACK_LENGTH;
        rx_ready_time_data = now + ACK_LENGTH;
        rx_usage_data[ParentID] += ACK_LENGTH;
       }
        or timeout after(20)
    }
  }
 }

// THE NODE ENTITY

entity Node (int ID, ename Link)
 {
  ename Node_Tx, Node_Rx;
  int src, dst, relayed,prev_hop, metric, from, ptype;
  unsigned long seq_no,sqno;
  struct message_neighbors hello_ne[NumberOfNodes];
  struct neighbor_list *lptr;
  struct rreq_list *pend;
  struct routing_table *myroute;
  struct rmpr_rtentry *routeen;
  int modified;
  int next_hello_time;
  unsigned long my_seq_no = 0;
  unsigned long seq_nop = 0;
  unsigned long seq_noa = 0;
  struct neighbor_list *neighbors = NULL;
  struct routing_table *rt = NULL;
  struct data_fwl *fwl = NULL;
  struct data_fwl *dfl = NULL;
  struct data_fwl *new_fwl;
  struct ack_fwl *fwla= NULL;
  struct ack_fwl *dfla =NULL;
  struct ack_fwl *new_fwla;
  struct rmpr_rtentry *route = NULL;
  struct neighbor_list *rmpr_neighbor = NULL;
  int ne_exp_timer[NumberOfNodes];
  int rt_exp_timer[NumberOfNodes];
  struct rr_et *rr_exp_timer = NULL;
  int expected_ack_time[NumberOfNodes];
  int send_ack_time[NumberOfNodes];
  int i,now;
  int pkt_length;
  next_hello_time = HELLO_PERIOD*1.0*drand48();
  for (i=0;i<NumberOfNodes;i++) {
    expected_ack_time[i] = -1;
    ne_exp_timer[i] = -1;
   rt_exp_timer[i] = -1;
    send_ack_time[i] = -1;
  }
  Node_Tx = new TRANSMITTER(self,Link);
  loc_data[ID].tx_name = Node_Tx;
```

```
 Node_Rx = new RECEIVER(self,Link);
 loc_data[ID].rx_name = Node_Rx;

for(;;) {
   now = simclock();
     receive(HELLO_MSG he) {
             src = he.src;
             seq_no = he.sequence_no;
               i = 0;
         while(he.hello_ne[i].neighbor != -1) {
               hello_ne[i].neighbor = he.hello_ne[i].neighbor;
               hello_ne[i].sequence_no = he.hello_ne[i].sequence_no;

               i++;
             }
            hello_ne[i].neighbor = -1;
       if((route = rmpr_rtlookup(rt,src)) != NULL)
       {
        rmpr_rtaddroute(rt,src, src, 1, seq_no, 0);
        rt_exp_timer[src]);
       }
       else {
         rt = rmpr_rtaddroute(rt,src, src, 1, seq_no, 0);
           }
       rmpr_neighbor = Lookup_ne(neighbors,src);
       if(rmpr_neighbor == NULL) {
         neighbors = Add_ne(neighbors,src,seq_no);
          ne_exp_timer[src] = now + NE_EXP_TIME;
         }
       else {
         neighbors = Add_ne(neighbors,src,seq_no);
         ne_exp_timer[src] = now + NE_EXP_TIME;
         i=0;
         while(hello_ne[i].neighbor != -1) {
           if(hello_ne[i].neighbor != ID) {
             if((route = rmpr_rtlookup(rt,hello_ne[i].neighbor)) == NULL) {
             rt = rmpr_rtaddroute(rt,hello_ne[i].neighbor, src, 2,
                 hello_ne[i].sequence_no, 0);
             }
             else {
               if((hello_ne[i].sequence_no >route->sequence)||((
                     hello_ne[i].sequence_no ==route->sequence) &&
                     (route->metric > 2))) {
                rmpr_rtaddroute(rt,hello_ne[i].neighbor, src, 2,
                   hello_ne[i].sequence_no, 0);
               }
             }
           }
           i++;
         }
       }
     }
or receive(SEND_MSG sendd) {
       pkt_length = sendd.length;
       dst        = sendd.destination;
```



```
       ptype      = sendd.ptype;
       src  = ID;
       from = ID;
  send DATA_PKT{dst,src,from,seq_nop,ptype,pkt_length} to Node_Tx;
  avg_data_sent++;
  seq_nop++;
  old_time[src]= simclock();
  metric++;
expected_ack_time[dst] = now + metric*(pkt_length+ACK_LENGTH) + ACK_TIMEOUT;
  }
      or receive(DATA_ACK dataa) {
       dst = dataa.dest;
       src = dataa.src;
       from =dataa.from;
       sqno =dataa.sequence_no;
       ptype = dataa.ptype;
if ( ID != src )  {
if((ID == dst) && (dfla = rmpr_aflookup(fwla,src,dst,sqno)) == NULL) {
      new_time[src]= now;
      delay[src] = new_time[src]-old_time[src];
      avg_delay = avg_delay+ delay[src];
      avg_ack_received++;
      fwla = rmpr_afwladd(fwla,src,dst,sqno,metric);
      new_fwla = rmpr_aflookup(fwla,src,dst,sqno);
            if (now <=expected_ack_time[src]) {
               expected_ack_time[src]= -1;
               avg_data_success++;
               }
        }

       else {

         if((dfla = rmpr_aflookup(fwla,src,dst,sqno)) != NULL) {

           }
               else {

                  fwla = rmpr_afwladd(fwla,src,dst,sqno,metric);
                  new_fwla = rmpr_aflookup(fwla,src,dst,sqno);
                  from = ID;
            send DATA_ACK{dst,src,from,seq_noa,ptype} to  Node_Tx;
                   metric++;
                   relayeda++;
             }
           }
       if(Lookup_ne(neighbors,from) != NULL) {
       ne_exp_timer[from] = now + NE_EXP_TIME;
        }
       }
      }
or receive(DATA_PKT datap) {
       dst = datap.dest;
       src = datap.src;
       from =datap.from;
       sqno = datap.sequence_no;
```

```
      ptype = datap.ptype;
      pkt_length = datap.length;
if ( ID != src ) {
     idnesrc++;
      if((ID == dst)  && (dfl = rmpr_dflookup(fwl,src,dst,sqno)) == NULL) {
         idedst++;
         avg_data_received++;
  send DATA_ACK{src,ID,ID,seq_noa,ptype} to Node_Tx  after pkt_length;
              avg_ack_sent++;
             seq_noa++;
          fwl = rmpr_fwladd(fwl,src,dst,sqno,metric);

          new_fwl = rmpr_dflookup(fwl,src,dst,sqno);
             }
 else {
     idnedst++;
           if((dfl = rmpr_dflookup(fwl,src,dst,sqno)) != NULL) {
                 }
            else {

              fwl = rmpr_fwladd(fwl,src,dst,sqno,metric);
              new_fwl = rmpr_dflookup(fwl,src,dst,seq_no);
              from = ID;
            send DATA_PKT{dst,src,from,seq_nop,ptype,pkt_length} to Node_Tx;
            relayedd++;
            metric++;
        }
       }
   }
  else
    if (ID==src)  {
    idsrc++;
 }

 }
 or timeout after (5)  {
      if(now > next_hello_time) {
      i=0;
      lptr = neighbors;
      while(lptr != NULL) {
      hello_ne[i].neighbor = lptr->neighbor;
      hello_ne[i].sequence_no = lptr->sequence_no;
      i++;
      lptr = lptr->next_neighbor;
         }
      hello_ne[i].neighbor = -1;
      send HELLO_MSG{ID,my_seq_no,hello_ne} to loc_data[ID].tx_name;
      my_seq_no++;
      next_hello_time += HELLO_PERIOD;
      next_hello_time +=next_hello_time;
 }
    for(i=0;i<NumberOfNodes;i++) {

 if (now > ne_exp_timer[i] && ne_exp_timer[i] != -1) {
 neighbors = Del_ne(neighbors,i);
```



```
     ne_exp_timer[i] = -1;
        }
      }
    for(i=0;i<NumberOfNodes;i++) {
         if (i != ID) {
  if(now > expected_ack_time[i]  && expected_ack_time[i] != -1)  {
              expected_ack_time[i] = -1;
          }
        }
      }
    }
 }
}

// THE CHANNEL ENTITY

entity Link()
{
  int i, now, dest, src, from, ptype, length, hops, prev_hop,deg;
  unsigned long seq_nop,seq_noa,seq_no,sqno;
  int Ndegree,Num_of_Links,NID;
  int Degree =0;
  int ifrom,start =0;
  double ratio =0.0;
  struct message_neighbors hello_ne[NumberOfNodes];
   int  k =0;
 for(;;) {
    now = simclock();
    receive (HELLO_MSG he)
      {
        src = he.src;
        seq_no = he.sequence_no;
          i=0;
          while(he.hello_ne[i].neighbor != -1) {
          hello_ne[i].neighbor = he.hello_ne[i].neighbor;
          hello_ne[i].sequence_no = he.hello_ne[i].sequence_no;
          i++;
        }
        hello_ne[i].neighbor = -1;
         for(i=0;i<NumberOfNodes;i++) {

         if((i != src) && (NODE_DISTANCE(src,i)<=RANGE)) {
          send HELLO_MSG{src,seq_no,hello_ne} to  loc_data[i].rx_name;
          }
        }
      }

 or receive(DATA_ACK dataa) {
        dest = dataa.dest;
        src  = dataa.src;
        from = dataa.from;
         sqno =dataa.sequence_no;
        ptype = dataa.ptype;
  for(i=0;i<NumberOfNodes;i++)
     {
```

```
if((i != from) && (NODE_DISTANCE(from,i)<=RANGE) &&  ptype ==1 ) {

          send DATA_ACK{dest,src,from,sqno, ptype} to loc_data[i].rx_name;
      }
        else
           {
            if ((NODE_DISTANCE(from,i)) > RANGE)
           avg_out_of_rangea++;
            }
         }
     }
or receive(DATA_PKT datap) {
      dest   = datap.dest;
      src    = datap.src;
      from   = datap.from;
      sqno   = datap.sequence_no;
      ptype  = datap.ptype;
      length = datap.length;
      Ndegree = NODE_DEGREE();
 total_degree= total_degree + Ndegree;
 sim_counter = sim_counter + 1;
 ratio=  (double)Ndegree/(double)NumberOfNodes;
  connectivity = connectivity + ratio;
    if (LMFLOOD1 || LMFLOOD4)
         {
            Num_of_Links = rand ()%Ndegree;
         }
    if (LMFLOOD2 || LMFLOOD3)
         {
         Num_of_Links = ((Ndegree*3)/4)+ 1;
         }
    if (LMFLOOD5 || LMFLOOD6)
          {
            if (ptype == 1)
             {
                  Num_of_Links = Ndegree;
                 }
              else  {
                    Num_of_Links = (Ndegree/2);
                   }
          }
   if (LMFLOOD1 || LMFLOOD2||LMFLOOD5)
    {
      i=0;
      while ((i < Num_of_Links) && ( Num_of_Links !=0)) {
      NID = rand()%Num_of_Links;
      if ((NID !=from) && (NODE_DISTANCE(from,NID)) <= RANGE) {
      send DATA_PKT{dest,src,from,sqno,ptype,length} to l
          oc_data[NID].rx_name;
      avg_within_ranged++;
        }
     else  {
        if ((NODE_DISTANCE(from,NID)) > RANGE)
        }
```



```
          if ( i==from)
             {
                 ifrom++;
              }
          i++;
     }
     }
 if (LMFLOOD3 || LMFLOOD4||LMFLOOD6)
   {
       i=0;
        while ((i < Num_of_Links) && ( Num_of_Links !=0)) {
        if ((i !=from) && (NODE_DISTANCE(from,i)) <= RANGE) {
        send DATA_PKT{dest,src,from,sqno,ptype,length} to loc_data[i].rx_name;
         avg_within_ranged++;
          }
       else  {
          if ((NODE_DISTANCE(from,i)) > RANGE)
                          avg_out_of_ranged++;
         }
           if ( i==from)
              {
                  ifrom++;
               }
    i++;
     }
   }
 if (LMFLOOD0)
  {
     for(i=0;i<NumberOfNodes;i++)  {
        if ((i !=from) && (NODE_DISTANCE(from,i)) <= RANGE) {
        send DATA_PKT{dest,src,from,sqno,ptype,length} to loc_data[i].rx_name;
         avg_within_ranged++;

          }
       else {
          if ((NODE_DISTANCE(from,i)) > RANGE)
                          avg_out_of_ranged++;
         }
           if ( i==from)
              {
                  ifrom++;
               }
        }
    }
  }
 or timeout after (50)
        {
    }
  }
 }

// THE DRIVER ENTITY

entity driver()
```

```
{
ename Parent, Link, tra, rec,cmo, channel, dc, MOBC, COMM, mob_con;
ename Node_Rx,Node_Tx,CID;
int steps, now, i, j,src,ID,num,NUMNODE;
FILE *output;
double x,y,a,b,seq_no;
setmaxclock(MAX_SIM_TIME);
channel = new Link();
x = drand48();
y = drand48();
for( i=0;i<NumberOfNodes;i++) {
  NetNode[i] = new Node(i,channel);
  loc_data[i].NodeName = NetNode[i];
  loc_data[i].NodeNumber = i;
  loc_data[i].LocX = x;
  loc_data[i].LocY = y;
  loc_data[i].Motion_Type = 'L';

     x += 0.1;
 if   (x >  1.0)
      {
       x =  0.1;
       y += 0.1;
      }
   }
      MOBC  = new MOBILITY();
      COMM = new COMMUNICATOR();
   }
}
  hold(MAX_SIM_TIME-simclock());

// COLLECT SIMULATION RESULTS

finalize{

  output = fopen ("result.dat", "w");

  avg_mobility = avg_mobility/(Real_to_sim*max_sim_time*NumberOfNodes);
  for(i=0;i<NumberOfNodes;i++)
  {
    tx_usage_data[i] = 100.0*tx_usage_data[i]/(max_sim_time*Real_to_sim);
    tx_usage_rt[i]   = 100.0*tx_usage_rt[i]/(max_sim_time*Real_to_sim);
    rx_usage_data[i] = 100.0*rx_usage_data[i]/(max_sim_time*Real_to_sim);
    rx_usage_rt[i]   = 100.0*rx_usage_rt[i]/(max_sim_time*Real_to_sim);
  }
  for(i=0;i<NumberOfNodes;i++) {
    avg_data += tx_usage_data[i];
    avg_data += rx_usage_data[i];
    avg_rt   +=tx_usage_rt[i];
    avg_rt   += rx_usage_rt[i];
  }
 avg_data = avg_data/(2*NumberOfNodes);
 avg_comm = (avg_data + avg_rt)/2;
 for (i=0;i<NumberOfNodes;i++)
  {
```



```
   fprintf(output,"[Node %d]: Transmitted Data for %.2f
              %%\t of the time\n", i, tx_usage_data[i]);
   fprintf(output,"[Node %d]: Transmitted Rtng for  %.2f
              %%\t of the time\n", i, tx_usage_rt[i]);
   fprintf(output,"[Node %d]: Received    Data for %.2f  %%\t of the
               time\n", i, rx_usage_data[i]);
   fprintf(output,"[Node %d]: Received Rtng for  %.2f  %%\t of the
               time\n", i, rx_usage_rt[i]);
    }
 fprintf(output,"\t %.2f %%\t time spent on data\n", avg_data);
 fprintf(output,"\t  %.2f %%\t time spent on Rtng\n", avg_rt);
 fprintf(output,"\t %.2f %%\t  time spent on Data and Routing \n",
               avg_comm);
 fprintf(output,"\t %.2f %%\t  time spent IDLE\n", 100.0 -avg_comm);
 fprintf(output,"\t%.2f\tAvg # of timeouts on ack \n",
              avg_data_timeout*1.0/NumberOfNodes);
 fprintf(output,"\n\t%.2f\tAvg # of data sent\n",
               avg_data_sent*1.0/NumberOfNodes);
 fprintf(output,"\t%.2f\tAvg # of data packets rcvd\n",
                avg_data_received*1.0/NumberOfNodes);
 fprintf(output,"\t%.2f\tAvg # of ack  packets sent\n",
                avg_ack_sent*1.0/NumberOfNodes);
 fprintf(output,"\t%.2f\tAvg # of ack  packets rcvd\n",
                avg_ack_received*1.0/NumberOfNodes);
 fprintf(output,"\t%.2f\tAvg # of comm successes\n",
               avg_data_success*1.0/NumberOfNodes);
 fprintf(output,"\n#Average Network mobility: %2.2f
               m/s\n",avg_mobility);
 fprintf(output,"#Number of Nodes %d\n",NumberOfNodes);
 fprintf(output,"#Transmission Range %f\n",RANGE);
 fprintf(output,"\n#Avg Connectivity is: %2.2f links\n",
               connectivity/sim_counter);
 fprintf(output,"\n# Mean Nodal Degree is: %2.2f nodes\n",
               total_degree*1.0/(NumberOfNodes*MAX_SIM_TIME));
 fprintf(output,"\n\t%.2f\tAvg # data packets delivered
               \n",(avg_data_received*1.0/avg_data_sent)*100);
 fprintf(output,"\n\t%.2f\tAvg # ack packets delivered \n",
               (avg_ack_received*1.0/avg_ack_sent)*100);
 fprintf(output,"\t%.2f\tAvg# relayed (DATA)\n",
               relayedd*1.0/NumberOfNodes);
 fprintf(output,"\t%.2f\tAvg# relayed
               (ACK)\n", relayeda*1.0/NumberOfNodes);
 fclose(output);
  }
 }
```

Appendix C

Clustering Algorithms Source Code

```
/*************************************************************************/
/*                                                                       */
/* Hierarchical Clustering Simulating program: includes all modules      */
/* Last Modified:  September 23, 2000                                    */
/*                                                                       */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <string.h>
#include <malloc.h>
#include "header.h"
#define DEBUG 2
#define PKT_DEBUG  1
#define DATA_SLOTS_DEBUG 1
#define CALL_DEBUG  1
#define END_QUEUE_DEBUG 0
#define ROUTING_PRO1 1
#define CONG_CTRL 1
#define NO_ROUTE_COUNT 0
#define FIFO 0
#define CHANNEL_MODEL 0
#define CHANNEL_MODE2 0
#define BOTH_MODELS 1
#define LINK_STATE 1
#define CLUST_ALGOR1  1
#define CLUST_ALGOR2  0
#define CLUST_ALGOR3  0
#define CLUST_ALGOR4  0

// SIMULATION ENTITIES

entity synchronizer();
entity coordinator();
entity nodes(int,int,int);
entity channel (ename);
entity traffic_gen(int);

// DATA  STRUCTURES

struct SIGNAL_STRENGTH
{
 int id;
 int my_sig;

};
struct MOBILITY_STATUS
{
 int id;
 int my_mob;
};
```



```
struct POSITION {
 int x;
 int y;
};

struct ROUTING_LIST
{
 int next_addr[N];
 int hop[N];
 int head[N];
};

struct SIGNAL_STRENGTH  signals[N][N];
struct MOBILITY_STATUS  mobi[N][N];
struct POSITION position[N];
struct POSITION old_position[N];

// MESSAGES

message idmsg{ename id;};
message idmsg1{ename id1[N];};
message idmsg2{ename id2;};
message idmsg7{ename id7;};
message idmsg8{ename id8s;};
message idmsg3{ename id3[N];};
message idmsg4{ename id4[N];};
message idmsg5{ename id5[N];};
message step_msg{int step;};
message linkstate_pkt{int id; int nu; int no;} l_state;
message  LINKSTATE_PKT
  {
  int source_id;
  int dest_id;
  int source_type;
  int scope;
  unsigned long  sequence_num;
  int update_type;
  int cost_metric;
  int time_gen;
  };

message  ROUTE_REQ
 {
 int source_id;
 int dest_id;
 int source_type;
 int scope;
 unsigned long  sequence_num;
 int prev_node;
 int update_type;
 int cost_metric;
 int time_gen;
 };

message  ROUTE_REP
```

```
{
int source_id;
int dest_id;
int source_type;
int prev_node;
int scope;
unsigned long  sequence_num;
int update_type;
int cost_metric;
int time_gen;
};

message linkstate{int id; int nu; int no;} links;
message position_msg{int id; int x; int y;} p;
message my_signal {int id; struct SIGNAL_STRENGTH sig[N][N];} signal_stren;
message my_stabil {int id; struct MOBILITY_STATUS mob[N][N];}mobi_rate;
message ctrl_ph{int id; int Iamch; int neighbor[N]; struct ROUTING_LIST
     r_list; struct DATA_SLOTS d_slots[M+1];};
message ctrl_ph2{int id; int Iamch; int neighbor[N];struct ROUTING_LIST
     r_list; };
message data_out{int slot_no; struct ACTIVE_Q pkt_in;}datumo;
message data_slot_end{};
message SIR_ready{};
message ctrl_turn_msg{};
message ctrl_end_msg{};
message move{};
message data_in{int slot_no; struct ACTIVE_Q pkt_in;}datum;
message the_data_slot{int no;} the_d_slot;
message call_req{struct PKT first_pkt;}c_req;
message call_trfc_gen{struct PKT pkt;}c_trfc;
message radio_signal{};

// DEFAULT VALUES FOR PARAMETERS and CONSTANTS

 int signal[N];
 int mobil[N];
 int cong_flag[N][N];
 int I_am_clusterhead[N];
 int old_I_am_clusterhead[N];
 int I_am_gw[N];
 int old_I_am_gw[N];
 int ch_changes=0, my_ch_changes=0;
 int changes =0;
 int mob_count[100];
 long FRAME_TIME;
 int source_id;
 int dest_id;
 int INFINITE_HOP;
 int P_suc_1st_try;
 extern void draw_route();
 int useful_link(int n1, int n2);
 int res_link(int n1, int n2);
 extern long lrand48();
 extern double drand48();
 extern long  srand48();
```



```
 extern int rand();
 extern double pow(),sqrt(),log(), log10();
 extern void draw_position();
 extern void clearwin();
 extern void redraw();
 int Done;
 extern int atoi();
 extern long atol();
 void cal_statistics();
 void dtostr();
 char stat_text[40][80],old_stat_text[40][80];
 extern double power_measure();
 extern void impulse_produce();
 char temp_string[80];
 int  temp1;
 ename e_traffic_gen[N],e_no_route_checker;
 int frame=0;
 int frame_counter;
 struct QUEUE
  {
   int pkt_num;
   int hops_traversed;
   struct QUEUE *next;
  };
 struct QUEUE *queue[N];
 extern draw_pkt();
 void forward_pkt();
 double exp_pick();
 int CALL_ON[N][N];
 int pkt_gen_PRMA=0, pkt_got_PRMA=0,pkt_dropped_PRMA=0,out_of_seq=0,
      no_of_loops=0;
 struct PKT_STATUS pkt_status[N][N];
 double mean_delay=0.0, mean_square_delay=0.0, var_delay=0.0,
   avg_hops_traversed=0.0;
 double mean_no_of_clusters=0.0;
 int last_frame_res_link[N][N];
 int res_link();
 int MOVING_PATTERN[N];
 struct POSITION traj[N];
 FILE *file1, *file2;
 long seed;
 int seed2;

entity driver () {

  ename node[10], coord, sync,coo[10],syn[10];
  ename channl;
  ename name[10];
  ename coid;
  int i,j,k;
  clocktype time;
  setmaxclock(1000);

  FRAME_TIME=(long)N*CTRL_SLOT_TIME+num_data_slots*DATA_SLOT_TIME;
  INFINITE_HOP=N;
```

```
for(i=0;i<N;i++)
 {
    MOVING_PATTERN[i]=RAND_MOVE;
    do
    {
   traj[i].x=1-(int)(lrand48()%3);
   traj[i].y=1-(int)(lrand48()%3);
    }
    while ((traj[i].x==0)&&(traj[i].y==0));
 }
 for(i=0;i<N;i++)
    queue[i]=NULL;
 for(i=0;i<N;i++)
    for(j=0;j<N;j++)
    {
      CALL_ON[i][j]=0;
      cong_flag[i][j]=0;
      pkt_status[i][j].received=0;
      pkt_status[i][j].repeat_received=0;
      pkt_status[i][j].dropped=0;
      pkt_status[i][j].repeat_dropped=0;
      pkt_status[i][j].no_route_pkts=0;
      pkt_status[i][j].loops=0;
      pkt_status[i][j].no_route_frames=0;
      pkt_status[i][j].last_pkt_heard=0;
      pkt_status[i][j].out_of_seq=0;
      pkt_status[i][j].idle=0;
      for(k=0;k<=DELAY_REQ;k++)
      {
    pkt_status[i][j].got[k]=0;
    pkt_status[i][j].lost[k]=0;
      }
    }
   for (i = 0; i<N; i++)
     {
       node[i]  = new nodes(i, (int)lrand48()%max_x, (int)lrand48()%max_y);
       coord = new coordinator();
       sync  = new synchronizer();
             e_traffic_gen[i] = new traffic_gen(i);
               if((CHANNEL_MODEL)||(BOTH_MODELS))
               {
                  channl = new channel(coord);
                  send  idmsg{channl} to coord;
               }
     }
        for (i=0; i < N; i++) {
             send idmsg{coord} to node[i];
             send idmsg2{coord} to sync;
             send idmsg7{sync}  to node[i];
             send idmsg3{node} to coord;
             send idmsg4{node} to sync;
             send idmsg5{node} to e_traffic_gen[i];}
    }
double estimate ( double nsx, double nsy, double ndx, double ndy, double
```



```
        osx, double osy,double  odx, double ody)
   {
      double source_change, end_change, total_change ;
      double x_change, y_change ;
        x_change = osx - nsx ;
        y_change = osy - nsy ;
        source_change = sqrt ( x_change * x_change + y_change * y_change ) ;
         x_change = odx - ndx ;
         y_change = ody - ndy ;
           end_change = sqrt ( x_change * x_change + y_change * y_change ) ;
           total_change = source_change + end_change ;
          return ( total_change );
   }
   double path_loss ( lambda, length )
   double lambda, length ;
    {
     double loss, power_loss ;
      if ( length  <= 0.000000001 )
        loss = 1 ;
      else
       loss = lambda / ( 4 * PI * length ) ;
   power_loss = loss * loss ;
   return (power_loss) ;
    }

double path_length ( begin, end )
       struct point *begin, *end ;
       double length, length_x, length_y ;

       length_x = end->x_location - begin->x_location ;
       length_y = end->y_location - begin->y_location ;

       length = sqrt ( length_x * length_x + length_y * length_y ) ;

       return ( length ) ;
 }

double bkgnoise()

 {
  double rx_noisefig, rx_temp, rx_bw ;
  double bkg_temp, bkg_noise, amb_noise ;
  double noise ;
  rx_noisefig = 1.2 ;
  rx_temp = (rx_noisefig - 1.0) * 290.0 ;
  bkg_temp = BKG_TEMP ;
  rx_bw = 10000000.0 ;
  bkg_noise = (rx_temp + bkg_temp) * rx_bw * BOLTZMANN ;
  amb_noise = rx_bw * AMB_NOISE_LEVEL ;
  noise = bkg_noise + amb_noise ;
 return(noise) ;
 }

double exp_pick(double para)
   {
```

```
   return((double)(1.0/para*log((double)1.0/(1.0-drand48()))));}
void dtostr(d,string)
  double d;
  char string[];
  {
     int j,jj,jjj;
     int int_d;
     double frac_d;
     int_d=(int)d;
     frac_d=d-(double)int_d;
      j=0;
      do {
      string[j++]=int_d%10+'0';
      }
      while ((int_d/=10)>0);
      string[j]='\0';
      for(j=0,jj=strlen(string)-1;j<jj;j++,jj--)

       {
        jjj=string[j];
        string[j]=string[jj];
        string[jj]=jjj;
       }

   j=strlen(string);
   string[j++]='.';
   string[j++]=(int)(frac_d*10)%10+'0';
   string[j++]=(int)(frac_d*100)%10+'0';
   string[j++]=(int)(frac_d*1000)%10+'0';
   string[j++]=(int)(frac_d*10000)%10+'0';
   string[j]='\0';
 }

int res_link(int n1,int n2)

 {
   if( (last_frame_res_link[n1][n2]!=0)&&(frame==
       last_frame_res_link[n1][n2]+1))
       return(1);
   else
      return(0);
 }
int useful_link(int n1, int n2)

 {
 int k;

   for(k=1;k<N;k++)

  if (I_am_clusterhead[k]&&(sqrt(pow((double)(position[n1].x-position[k].x),
       2.0)+pow((double)(position[n1].y-position[k].y),2.0) <= COMM_RANGE)&&
       (sqrt( pow((double)(position[k].x-position[n2].x),2.0)+ pow((double)
       (position[k].y-position[n2].y),2.0)))<=COMM_RANGE))
  { return(1);
  }
```



```
}
   void forward_pkt()
 {

   int i;
   int pkt_num;
   static pkt_num_last_got=0;
   struct QUEUE *temp1, *temp2;
   int just_forwarded[N];
   int pkt_drawn;
   int temp_hops_traversed;
   double temp_delay, temp_no_of_clusters;
   pkt_drawn=0;
   for(i=0;i<N;i++)  { just_forwarded[i]=0; redraw_pkt[i].pkt_num=0; }
   temp1=(struct QUEUE *)malloc(sizeof(struct QUEUE));
   temp1->pkt_num=frame;
   temp1->hops_traversed=0;
   temp1->next=NULL;
   if(queue[source_id]!=NULL)
   {
      temp2=queue[source_id];
      while(temp2->next!=NULL) temp2=temp2->next;
      temp2->next=temp1;
   }
   else
      queue[source_id]=temp1;
       pkt_got_PRMA++;
   pkt_status[datum.pkt_in.pkt.source][datum.pkt_in.pkt.dest].received++;
   for(i=0;i<N;i++)
   {
      if( ((!just_forwarded[i])&&(queue[i]!=NULL))||
      ((just_forwarded[i])&&(queue[i]!=NULL)&&(queue[i]->next!=NULL)))
      {
     temp1=queue[i];
     pkt_num=queue[i]->pkt_num;
     temp_hops_traversed=queue[i]->hops_traversed;
     queue[i]=queue[i]->next;
     free(temp1);

         if( (routing_table[i].hop[dest_id]!=INFINITE_HOP)&&
         (routing_table[i].next_addr[dest_id]!=source_id)&&
         (useful_link(i,routing_table[i].next_addr[dest_id]))&&
         ((res_link(i,routing_table[i].next_addr[dest_id]))||(
          rand()%100<P_suc_1st_try)) )
     {
        last_frame_res_link[i][routing_table[i].next_addr[dest_id]]=frame;
        pkt_drawn++;
        redraw_pkt[pkt_drawn].pkt_num=pkt_num;
        redraw_pkt[pkt_drawn].node1=i;
        redraw_pkt[pkt_drawn].node2=routing_table[i].next_addr[dest_id];
        if(routing_table[i].next_addr[dest_id]!=dest_id)
        {
           temp1=(struct QUEUE *)malloc(sizeof(struct QUEUE));
           temp1->pkt_num=pkt_num;
           temp1->hops_traversed=temp_hops_traversed+1;
```

```
        temp1->next=NULL;
        if(queue[routing_table[i].next_addr[dest_id]]!=NULL)
        {
               temp2=queue[routing_table[i].next_addr[dest_id]];
               while(temp2->next!=NULL) temp2=temp2->next;
               temp2->next=temp1;
             }
      else
         queue[routing_table[i].next_addr[dest_id]]=temp1;
         just_forwarded[routing_table[i].next_addr[dest_id]]=1;
      }
      else
      {
       pkt_got_PRMA++;
        pkt_status[datum.pkt_in.pkt.source]
        [datum.pkt_in.pkt.dest].received++;
       avg_hops_traversed*(double)(pkt_got_PRMA-1))/(double)pkt_got_PRMA;
      }
   }
   else
          {
      pkt_dropped_PRMA++;
        }
    }
      pkt_got_PRMA++;
         temp_delay=(double)(frame-pkt_num+1);
         mean_delay=(double)temp_delay/(double)pkt_got_PRMA
        +(double)mean_delay/(double)pkt_got_PRMA*(double)
          (pkt_got_PRMA-1.);
        mean_square_delay=(double)(pow((double)temp_delay,2.0)+
        mean_square_delay*(pkt_got_PRMA-1.))/(double)pkt_got_PRMA;
             var_delay=mean_square_delay-pow(mean_delay,2.0);
         avg_hops_traversed=(double)((double)(temp_hops_traversed+1)+
        avg_hops_traversed*(double)(pkt_got_PRMA-1))/(double)pkt_got_PRMA;
         if(DEBUG>=1)
             if(pkt_num<pkt_num_last_got) out_of_seq++;
        pkt_num_last_got=pkt_num;
  }
    temp_no_of_clusters=0.0;
    for(i=0;i<N;i++)
  if(I_am_clusterhead[i])
    {
  temp_no_of_clusters=temp_no_of_clusters+1.0;
  mean_no_of_clusters=((double)temp_no_of_clusters+
     mean_no_of_clusters*((double)frame-1.))/(double)frame;
     }
   }
 }
void cal_statistics()
 {
 static int old_pkt_gen_PRMA=0,old_pkt_got_PRMA=0,old_pkt_dropped_PRMA=
    0,old_out_of_seq=0,old_no_of_loops=0;
 static double old_mean_delay=0.0,old_mean_square_delay=0.0,
    old_var_delay=0.0,old_avg_hops_traversed=0.0,old_mean_no_of_clusters=0.0;
 static int old_ch_changes=0, old_my_ch_changes=0;
```



```
 char text[80];
 int i,j;
 int temp_pkt_dup;
     for(i=0;i<40;i++)
       for(j=0;j<80;j++)
         old_stat_text[i][j]=stat_text[i][j];
         for(i=0;i<40;i++)
          stat_text[i][0]='\0';
          strcat(&stat_text[1][0],"offered load at this frame = ");
          itostr(pkt_gen_PRMA-old_pkt_gen_PRMA, &text[0]);
          strcat(&stat_text[1][0],&text[0]);
          itostr(pkt_got_PRMA-old_pkt_got_PRMA, &text[0]);
          strcat(&stat_text[2][0],&text[0]);
          strcat(&stat_text[3][0],&text[0]);
          temp_pkt_dup=0;
          for(i=0;i<N;i++)
            for(j=0;j<N;j++)
            temp_pkt_dup=temp_pkt_dup+pkt_status[i][j].repeat_received;
            strcat(&stat_text[4][0],&text[0]);
            strcat(&stat_text[5][0],&text[0]);
            temp_pkt_dup=0;
            for(i=0;i<N;i++)
              for(j=0;j<N;j++)
              temp_pkt_dup=temp_pkt_dup+pkt_status[i][j].repeat_received;
              itostr(pkt_got_PRMA-temp_pkt_dup, &text[0]);
              strcat(&stat_text[5][0],&text[0]);
              for(i=0;i<N;i++)
                for(j=0;j<N;j++)
                temp_pkt_dup=temp_pkt_dup+pkt_status[i][j].repeat_dropped;
       }

entity channel (ename coord) stacksize (20000)

{
    ename  coord1;
    double distance ;
    double signal_power, inter_power, noise, wamis_sir ;
    double loss, lambda ;
    double free_space ;
    double shadowing, change ;
    struct point head, foot ;
    int match_code ;
    int conflict, initial, start, small_scale ;
    int i, j, k ;
    message SIR_ready sready;
    correlat_value = 0.0 ;
    start_simulate = 0 ;
    for ( i = 0 ; i < N ; i++ )
       for ( j = 1 ; j < N ; j ++ )
        {
     link[i][j].sx_location = 0.0 ;
     link[i][j].sy_location = 0.0 ;
     link[i][j].dx_location = 0.0 ;
     link[i][j].dy_location = 0.0 ;
     for ( k = 0 ; k < Max_duration ; k++ )
```

```
    {
       link[i][j].power[k].area_mean = 0.0 ;
       link[i][j].power[k].power_expect = 0.0 ;
       link[i][j].power[k].power_real   = 0.0 ;
       link[i][j].power[k].delay = 0.0 ;
       link[i][j].power[k].phase = 0.0 ;
       link[i][j].power[k].signal = 0 ;
       link[i][j].set_already = 0;
    }
 }
for ( ; ; )
    receive ( radio_signal rads )
    {
    if ( start_simulate == 0 )  initial = 0 ;
    start_simulate = 1 ;
    for ( i = 0; ( i <N ) ; i++ )
        if( trcver[i].status == 'r' )
    {
        signal_power = 0.0 ;
        lambda = C / TX_freq ;
        noise  = 0.0 ;
        match_code = 0 ;
        conflict = 0 ;
        foot.x_location = trcver[i].x_location ;
        foot.y_location = trcver[i].y_location ;
        for ( j = 1 ; ( ( j <N) && ( conflict == 0) ) ;
        j++ )
         if ( trcver[j].status == 't' )
        {
            head.x_location = trcver[j].x_location ;
            head.y_location = trcver[j].y_location ;
            distance = path_length ( &head, &foot ) ;
            loss = path_loss (  lambda, distance ) ;
            if ( trcver[i].code == trcver[j].code )
            {
                if ( match_code == 0 )
                {
                     match_code = 1;
                     change = estimate(
                     trcver[j].x_location, trcver[j].y_location,
                     trcver[i].x_location, trcver[i].y_location,
                     link[j][i].sx_location,link[j][i].sy_location,
                     link[j][i].dx_location,link[j][i].dy_location);
                     trcver[i].trans_num = j ;
                     if ( change > Small_Scale )
                        small_scale = 0 ;
                     else
                        small_scale = 1 ;

                     if ( link[j][i].set_already == 0 )
                        start = 1 ;
                     else
                        start = 0 ;
                     if ( change == 0.0 )
                     {
```



```
                          shadowing = link[j][i].shadowing ;
                          }
                          else
                          {

                          signal_power = shadowing;
                          link[j][i].set_already = 1 ;
                          link[j][i].distance = distance ;
                          link[j][i].shadowing = shadowing ;}
                     }
                     else
                     {
                          conflict = 1 ;
                          signal_power = 0 ;
                     }
                }
                else
                {
                inter_power = Initial_Power * loss ;
                inter_power = inter_power / Processing_Gain ;
                noise = noise + inter_power ;
                }
                link[j][i].sx_location = head.x_location ;
                link[j][i].sy_location = head.y_location ;
                link[j][i].dx_location = foot.x_location ;
                link[j][i].dy_location = foot.y_location ;
            }
            inter_power = noise + bkgnoise() ;
            wamis_sir = signal_power / inter_power ;
            trcver[i].sir_value = signal_power /  inter_power;
            free_space = free_space / inter_power ;
            if(trcver[i].sir_value>0.0)
               trcver[i].sir_value = 10.0 * log10 ( trcver[i].sir_value ) ;
            else
        {
           trcver[i].sir_value = -100000.;
            }
           }
           initial = 1 ;
           send  sready to coord;
          }
}

entity synchronizer() stacksize (20000)

{
 int i,j;
 ename node[N];
 ename step_id, coid,s_id;
 ename  id;
 message testmes2 {ename ids2[10];};
 int step_y;
 int cycounter,nidcount2;
    receive(idmsg2 idms)
    coid = idms.id2;
```

```
   cycounter = cycounter+1;
   receive(idmsg4 id)
  for (i=0;i<N;i++)  { node[i] = id.id4[i];
     nidcount2 = nidcount2 + 1;
     }
for (;;)
{
   frame++;
   for(i=0; i<N;i++) {
        hold(5);
      send ctrl_turn_msg {} to node[i];
      hold(1);          }
  if(PKT_DEBUG)
     for (j=0;j<N;j++)
     send ctrl_end_msg{} to node[j];
    for (j=1;j<num_data_slots;j++)
  {
      hold[1];
     if (PKT_DEBUG)
     for (i=0;i<N;i++)                          {
     send the_data_slot{j} to node[i];
     hold(1);
     send data_slot_end{} to coid;
     hold(1);
 }
    }
   if((1)||(CHANNEL_MODEL) )
     {
     if (frame <100)
     {
     file1=fopen("frame.file","w");
     fprintf(file1,"frame=%d, ch_changes=%lf, my_ch_changes=%lf\n",
     frame,(double)ch_changes/frame, (double)my_ch_changes/frame);
     fprintf(file1,"***********\n");
     fclose(file1);
     }
  }
  for(j=0;j<N;j++)
     for(j=0;j<=num_data_slots;j++)
      {
       hold(ACQ_TIME);
       if(PKT_DEBUG)
         for(i=0;i<N;i++)
          send the_data_slot{j} to node[i];
          send data_slot_end{} to coid;
        }
    }
  }
 }

entity coordinator() stacksize (60000)


double dist;
int i, j, jj, pos[10][2];
```



```
ename node[N],syid[10];
ename var1[10],var2[10],c_id;
int ccount = 0;
int mobi_st,Signal_st;
message move mo;
message ctrl_ph ctrl;
int no_pkt_process=0,busy_i;
int slot_no, pkt_in_range[N], pkt_heard;
struct ACTIVE_Q pkt[N];
ename channl,cyid[10];
int id, nidcount2,nid;
receive(idmsg3 idms)
   for ( i =0; i<N;i++) {
    node[i] = idms.id3[i];
if((CHANNEL_MODEL)||(BOTH_MODELS))
       {receive(idmsg idms)
        channl=idms.id;
       }
 for (;;)
 receive (position_msg posi)
     {
      p=posi;
      pos[p.id][0] = p.x;
      pos[p.id][1] = p.y;
      nid          =p.id;
      ccount = ccount+1;
      hold(1);
      for(i=0; i<N;i++) {
          hold(2);

       send ctrl_turn_msg {} to node[i];

       hold(1);          }

for(i=0;i<N;i++) {

   dist[i] =  (sqrt(pow((double)(pos[ctrl.id][0]-pos[i][0]),2.0) +
         pow((double)(pos[ctrl.id][1]-pos[i][1]),2.0)));
   signals[id][i].my_sig = dist[i];
         if (dist < COMM_RANGE*3/4)
           signals[id][i].my_sig =1;
     else
          signals[id][i].my_sig = 0;
send  my_signal {id, signals}  to node [i];
                       }

 for(i=0; i<N;i++)
  {

   if(lrand48()%100<Moving_prob)

        { send  mo to node[i];

         }
  }
```

```
    or receive (my_stabil stab)
       {
        mobi_rate   =stab;
        mobi[id][mobi_rate.id].id= mobi_rate.id;
        mobi[id][mobi_rate.id].my_mob= mobi_rate.mob[id]
             [mobi_rate.id].my_mob;
         for (i=0;i<N;i++)
            {
        send  mobi_rate to node[i]  after 5;

          }
       }
or  receive (ctrl_ph cntr)
  {
      ctrl = cntr;
for(i=0;i<N;i++)
    {
     if (i >3 )
         { Signal_st = 5;

           }
     if ( i == 1)
        { mobi_st = 1;}
 send  linkstate {i, Signal_st, mobi_st} to node[i];}
 send radio_signal{} to channl;
 I_am_clusterhead[ctrl.id]=ctrl.Iamch;

  if(CHANNEL_MODEL)
     {

   for(i=0;i<N;i++)
     { trcver[i].status='r'; trcver[i].code=1; }
        trcver[ctrl.id].status='t';
      send radio_signal{} to channl;
      receive(SIR_ready sir)
      {
       for(i=0;i<N;i++)
         if((ctrl.id<i)&&(trcver[i].sir_value>=54.-20.*log10((double)
             COMM_RANGE/250.))&&(Neighbors[i][ctrl.id]==1))
           {
    send ctrl to node[i];
       }
    else if((ctrl.id<i)&&(trcver[i].sir_value>=54.-20*log10((double)
            COMM_RANGE/250.)+6.4)&&(Neighbors[i][ctrl.id]!=1))
   { send ctrl to node[i];
    }
        else if((ctrl.id>i)&&(Neighbors[ctrl.id][i]==1))
    send ctrl to node[i];
}
  }
 else
  {
    for (i=0; i<N; i++)
      if (i != ctrl.id)
```



```
          {
           if (sqrt(pow((double)(pos[ctrl.id][0]-pos[i][0]),2.0) +
                 pow((double)(pos[ctrl.id][1]-pos[i][1]),2.0))
                 <= (double) COMM_RANGE)
           send ctrl to node[i];
        }
     }
    }
or   receive(data_out datao)
  {
      datumo=datao;
      slot_no=datumo.slot_no;
      if (no_pkt_process >10) {no_pkt_process = 10; break;}
         no_pkt_process++;
   pkt[no_pkt_process].pkt.type=datumo.pkt_in.pkt.type;
   pkt[no_pkt_process].pkt.source=datumo.pkt_in.pkt.source;
   pkt[no_pkt_process].pkt.dest=datumo.pkt_in.pkt.dest;
   pkt[no_pkt_process].pkt.pkt_num=datumo.pkt_in.pkt.pkt_num;
   pkt[no_pkt_process].pkt.time_gen=datumo.pkt_in.pkt.time_gen;
   pkt[no_pkt_process].inter_source=datumo.pkt_in.inter_source;
   pkt[no_pkt_process].inter_dest=datumo.pkt_in.inter_dest;
   pkt[no_pkt_process].code=datumo.pkt_in.code;
   or receive (data_slot_end datae)
    {
       if((no_pkt_process!=0)&& (no_pkt_process <=
          datumo.pkt_in.pkt.pkt_num;))
   {
     for(i=0;i<N;i++)
     {
          for (j=0; j< 4;j++)
           send data_in{slot_no,pkt[i]} to node[i];
      busy_i=0;
      for(j=1;j<=no_pkt_process;j++)
      if(pkt[j].inter_source==i) {busy_i=0; break;}
       if (busy_i!=0) continue;
        for(j=1;j<=no_pkt_process;j++)
          if((!CHANNEL_MODEL)&&
            (sqrt(pow((double)(pos[i][0]-pos[pkt[j].inter_source][0]),2.0)+
            pow((double)(pos[i][1]-pos[pkt[j].inter_source][1]),2.0))
            <=(double)COMM_RANGE)) {
            pkt_in_range[j]=1;
                 }
          else if((CHANNEL_MODEL)&&(Neighbors[i][pkt[j].inter_source]==1))
          pkt_in_range[j]=1;
          else  {
          pkt_in_range[j]=0; }

      for(j=1;j<=no_pkt_process;j++)
       if((pkt_in_range[j])&&(pkt[j].pkt.pkt_num>0))
       {

        pkt_heard=1;
        for(jj=1;jj<=no_pkt_process;jj++)

       if((j!=jj)&&(pkt_in_range[jj])&&(pkt[j].code==pkt[jj].code)&&
```

```
            (pkt[jj].pkt.pkt_num>0))
           { pkt_heard++; break;}
          if(pkt_heard ==1)
           send data_in{slot_no,pkt[j]} to node[i];

         }
         else if((pkt_in_range[j])&&(pkt[j].pkt.pkt_num<=0))
         {
          pkt_heard=1;
          for(jj=1;jj<=no_pkt_process;jj++)
            if((j!=jj)&&(pkt_in_range[jj])&&(pkt[j].code==pkt[jj].code))
           { pkt_heard++; break;}
          if(pkt_heard==1)
           send data_in{slot_no,pkt[j]} to node[i];
         }

       }
     }
     no_pkt_process=0;
      }

entity traffic_gen(int id)

{

 ename node [N];
 struct PKT first_pkt,pkt;
 int temp_dest;
 int accept;
 int pkt_num;
 double call_service_time;
 double call_int_arrvl_time;
 int temp_num_pkts = 5;
 int i,j;

 receive(idmsg5 id)
  for (i=0;i<N;i++) {
  node[i] = id.id5[i];
  }
 while(1)
  {
          accept=0;
          pkt_num=0;
  while(accept==0)

{
 call_int_arrvl_time=(double)exp_pick((double)1.0/(double)
          MEAN_CALL_INT_ARRVL_TIME);
 hold(1);
 while(temp_dest = (int)(lrand48()%N+1)==id);
   {
 call_service_time=(double)exp_pick((double)1.0/(double)
          MEAN_CALL_SERVICE_TIME);}
       if(PR_ALOHA)     {    accept=1;break; }
```



```
                temp_dest    = (int)(lrand48()%N+1);
                first_pkt.type='r';
                first_pkt.source=id;
                first_pkt.dest=temp_dest;
                first_pkt.pkt_num=-1;
                first_pkt.time_gen=frame;
          send  call_req{first_pkt} to node [id];

                for(i=0;i<=NUM_CALL_SET_UP_PKTS-6;i++)
                  {

                   hold(1);

                   if (CALL_ON[id][temp_dest]>0)
                     { accept=1; break;
                     }
                     first_pkt.pkt_num=pkt_num;
                     first_pkt.time_gen=frame;

              temp_dest    = (int)(lrand48()%N+1);

               send call_trfc_gen{first_pkt} to node[id];

                  }

for(i=0;i<temp_num_pkts;i++)
{
  if((CONG_CTRL)&&(!PR_ALOHA))

   {

  j=0;
  while(cong_flag[id][temp_dest]>0)

    {

    pkt.type='r'; pkt.source=id; pkt.dest=temp_dest;
    pkt.time_gen=frame;

    if(j%2==0)
          { pkt.pkt_num=0; pkt_status[id][temp_dest].idle++;

             }

        else if(i<temp_num_pkts)

          {pkt_num++; i++; pkt.pkt_num=pkt_num; pkt_gen_PRMA++;
             }
     else
          {
           break;}

       j++;

        send call_trfc_gen{pkt} to node [id];
```

```
      }
    }
  }
    pkt_num++;
    pkt.type='r';
    pkt.source=id;
    pkt.dest=temp_dest;
    pkt.pkt_num=pkt_num;
    pkt.time_gen=frame;
    pkt_gen_PRMA++;

          if((PR_ALOHA)&&(pkt_num==1))
                {
                 send call_req{pkt}  to node [id];

                }
         else
             {
                 send call_trfc_gen{pkt} to node [id];
              }

   }
     CALL_ON[id][temp_dest]=0;

      pkt_status[id][temp_dest].last_pkt_heard=0;

      for(i=0;i<DELAY_REQ;i++)
        {
           pkt_status[id][temp_dest].got[i]=0;
           pkt_status[id][temp_dest].lost[i]=0;
        }
  }
}

tity nodes(int id, int x, int y) stacksize(60000)


 int i,j,ii;
 int i1,i2,i3;
 int j1,j2,j3;
 int neighbor[N];
 int nidcount,move_count = 0;
 int nidcount2 = 0;
 double dist;
 int nid;
 int I_am_ch[N];
 clocktype updated_time[N];
 struct ROUTING_LIST routing_list;
 struct ROUTING_LIST other_routing_lists[N];
 struct DATA_SLOTS data_slots[N][M+1];
 int my_ch,my_2nd_ch,my_3rd_ch,temp_ch;
 int temp_addr;
 int try[M+1]; /
 struct ACTIVE_Q active_q[M+1];
 struct PRMA_Q *temp1_PRMA_q,*temp2_PRMA_q,*temp3_PRMA_q;
```



```
 struct PRMA_Q *first_PRMA_q,*PRMA_q;
 int my_cong_flag[N][N];
 int q_len;
 int my_ch_old, I_am_ch_old;
 int temp_head;
 ename coid, s_id,channl;
 message ctrl_ph2  ctrl2;
 message ctrl_ph  ctrl;
 ename node[10],coord,sync;
 ename c_id,syid[10];
 ename var3[10],var4[10];
 ename nod;
 int  pos[10][2];
 int runner;

 for(i=0;i<N;i++)
    {
       updated_time[i]=simclock();

       I_am_ch[i]=0;
       neighbor[i]=0;
       Neighbors[id][i]=0;
}

 for(i=0;i<N;i++)
   for(j=0;j<N;j++)
     my_cong_flag[i][j]=0;
 first_PRMA_q=NULL; PRMA_q=NULL;
 for(i=0;i<N;i++)
    for(j=1;j<=num_data_slots;j++)
    {
   data_slots[i][j].status='i';
   data_slots[i][j].code = i;
    }

if(ROUTING_PRO1)
 {

    for(i=0;i<N;i++)
  if(i!=id)
    for(j=0;j<N;j++)
    {
      other_routing_lists[i].next_addr[j]=0;
      other_routing_lists[i].hop[j]=INFINITE_HOP;
     }
    }
       for(i=0;i<N;i++)
            {
             routing_list.next_addr[i]=0;
             routing_list.hop[i]=INFINITE_HOP;
             }
 routing_list.next_addr[id]=id;
 routing_list.hop[id]=0;
 receive(idmsg idms)
```

```
 for (i=0;i<N;i++)
 {
coid = idms.id;
  }
receive(idmsg7 idms)

 for (i=0;i<N;i++)
 {
sync = idms.id7;

 }
   send position_msg{id, x, y} to coid;
 for(;;)
 {
     receive (ctrl_turn_msg ctmsg )
                 {
      for(i=0;i<id;i++)

            if( (neighbor[i]==1)&& (
            simclock()-updated_time[i] >
            (clocktype)(/*N+*/id-1)*CTRL_SLOT_TIME))

            {
           neighbor[i]=0; Neighbors[id][i]=0;
       routing_list.next_addr[i]=0;
       routing_list.hop[i]=INFINITE_HOP;
       for(j=0;j<N;j++)

          if(routing_list.next_addr[j]==i)
          {
             routing_list.next_addr[j]=0;
             routing_list.hop[j]=INFINITE_HOP;
                 }
           }

          if(ROUTING_PRO1)
     {
        for(i=0;i<N;i++) {

      if((routing_list.hop[i] !=INFINITE_HOP)&&(i!=id))
                for(j=0;j<N;j++)
                  routing_list.hop[i]=other_routing_lists[j].hop[i]+1;
          routing_list.next_addr[i]=j;

             if((j!=i)&&(j!=id)&&(routing_list.hop[j]==1)&&
       (other_routing_lists[j].next_addr[i]!=id)&&
       (other_routing_lists[j].hop[i]<routing_list.hop[i]-1))
                 {
          routing_list.hop[i]=other_routing_lists[j].hop[i]+1;
          routing_list.next_addr[i]=j;
              }
            }
          }
 if (CLUST_ALGOR1)
 {
```



```
      I_am_ch[id] = 1;
          for(i=0;i<id;i++)
          if ((I_am_ch[i]==1)&&(neighbor[i]==1))
               { I_am_ch[id] = 0; break; }
     }
 if (CLUST_ALGOR2)
   {
     I_am_ch[id] = 1;
     signal[id]= signals[id][id].my_sig;

      for(i=0;i<N;i++)   {

      signal[i]= signals[id][i].my_sig;
      runner = runner +1;

if ((I_am_ch[i]==1)&&(neighbor[i]==1) && (signal[i] =1) || (signal[i]=0))
    {

    I_am_ch[id] = 0;

    break;
          }
                      }

     }
 if (CLUST_ALGOR3)

{
      I_am_ch[id] = 1;  mobil[id]= mobi[id][id].my_mob;
      for(i=0;i<N;i++)
       mobil[i]= mobi[id][i].my_mob;

      if ((I_am_ch[i]==1)&&(neighbor[i]==1) && (mobil[i]==0)|| (mobil[i]==1))
           {

           I_am_ch[id] = 0;
           break; }
     }

 if (CLUST_ALGOR4)
{
      I_am_ch[id] = 1;  mobil[id]= mobi[id][id].my_mob;
      for(i=0;i<N;i++)
      if ((I_am_ch[i]==1)&&(neighbor[i]==1) && ((int)(2.0*drand48())))

           {
           I_am_ch[id] = 0;
           break; }
   }
     if((I_am_ch[id]!=I_am_ch_old)&&(frame>1))
           {

             ch_changes++;
             changes = ch_changes++;
     }
```

```
      I_am_ch_old=I_am_ch[id];

    for(j=1;j<=num_data_slots;j++)
     if((data_slots[id][j].status!='i')&&
    (data_slots[id][j].inter_source!=id)&&
    (data_slots[id][j].inter_dest!=id))
     data_slots[id][j].status='i';
     if(I_am_ch[id])
   {
     my_ch=id; my_2nd_ch=0; my_3rd_ch=0; I_am_gw[id]=0;
     if((my_ch!=my_ch_old)&&(frame>1))
     my_ch_changes++;
     my_ch_old=my_ch;

   for(j=1;j<=num_data_slots;j++) data_slots[id][j].code=id;
   for(i=0;i<N;i++)
    if(neighbor[i]==1)
     for(j=1;j<=num_data_slots;j++)
  if((data_slots[i][j].status!='i')&&
     (data_slots[id][j].status=='i')&&
     (data_slots[i][j].inter_source!=id)&&
     (data_slots[i][j].inter_dest!=id)&&
     (data_slots[i][j].code==id))
          {
   if(data_slots[id][j].status!='r')
    data_slots[id][j].status=data_slots[i][j].status;
   data_slots[id][j].code=id;
   data_slots[id][j].source=data_slots[i][j].source;
   data_slots[id][j].inter_source=data_slots[i][j].inter_source;
   data_slots[id][j].inter_dest=data_slots[i][j].inter_dest;
   data_slots[id][j].dest=data_slots[i][j].dest;
   }
}
     else
 {
  for(i=0;i<N;i++)
   if((neighbor[i]==1)&&(I_am_ch[i])) {my_ch=i; break;}

   if((my_ch!=my_ch_old)&&(frame>1))

my_ch_changes++;
         my_ch_old=my_ch;

     my_2nd_ch=0; my_3rd_ch=0; I_am_gw[id]=0;

 for(i=0;i<N;i++)

  if((neighbor[i]==1)&&(I_am_ch[i])&&(my_ch!=i))

 { my_2nd_ch=i; I_am_gw[id]=1; break;}

     if(I_am_gw[id]==1)
{

 my_3rd_ch=0;
```



```
  for(i=0;i<N;i++)
   if((neighbor[i]==1)&&(I_am_ch[i])&&(my_ch!=i)&&(my_2nd_ch!=i))
   { my_3rd_ch=i; break; }
      if(my_3rd_ch!=0)
  {
   temp_ch=my_3rd_ch;
   for(j=1;j<=num_data_slots;j++)
if((data_slots[id][j].status!='i')&&(data_slots[id][j].code==my_ch))
  { temp_ch=0; break; }
       if(temp_ch!=0)
  {
     temp_ch=my_3rd_ch;
     for(j=1;j<=num_data_slots;j++)
   if((data_slots[id][j].status!='i')&&(data_slots[id]
        [j].code==my_3rd_ch))
     { temp_ch=0; break; }
          if(temp_ch==0)
     { temp_ch=my_ch; my_ch=my_3rd_ch; my_3rd_ch=temp_ch; }
         else if(rand()%100<50)
     { temp_ch=my_ch; my_ch=my_3rd_ch; my_3rd_ch=temp_ch; }
   }

   temp_ch=my_3rd_ch;
   for(j=1;j<=num_data_slots;j++)
if((data_slots[id][j].status!='i')&&(data_slots[id]
   [j].code==my_2nd_ch))
  { temp_ch=0; break; }
       if(temp_ch!=0)
   {
      temp_ch=my_3rd_ch;
      for(j=1;j<=num_data_slots;j++)
   if((data_slots[id][j].status!='i')&&(data_slots[id]
    [j].code==my_3rd_ch))
     { temp_ch=0; break; }
          if(temp_ch==0)
     { temp_ch=my_2nd_ch; my_2nd_ch=my_3rd_ch; my_3rd_ch=temp_ch; }
         else if(rand()%100<50)
     { temp_ch=my_2nd_ch; my_2nd_ch=my_3rd_ch; my_3rd_ch=temp_ch; }
   }
  }
 }
     for(j=1;j<=num_data_slots;j++)
 {
  if((data_slots[id][j].status=='i')||
 ((data_slots[id][j].code!=my_ch)&&((!I_am_gw[id])||(data_slots[id][j].code!=my_2nd_ch))&&((my_
)
      {
   data_slots[id][j].code=my_ch;
   if((I_am_gw[id])&&(my_3rd_ch==0)&&(rand()%100<50)) data_slots[id][j].code=my_2nd_ch;
   else if((I_am_gw[id])&&(my_3rd_ch!=0))
   {
if(((i=rand()%100)>34)&&(i<67)) data_slots[id][j].code=my_2nd_ch;
else if(i>=67) data_slots[id][j].code=my_3rd_ch;
   }
  }
```

```
    }
   }
   if(PR_ALOHA)

        for(j=1;j<=num_data_slots;j++)
     data_slots[id][j].code=1;
send ctrl_ph{id, I_am_ch[id],neighbor,routing_list,&data_slots
     [id][0]::(M+1)*sizeof(struct DATA_SLOTS)} to coid;

  }
    or  receive (move mv)
{

   if(MOVING_PATTERN[id]==RAND_MOVE)
   {
      x = x - (int)lrand48()%(MOVING_RANGE*2+1) + MOVING_RANGE;
      y = y - (int)lrand48()%(MOVING_RANGE*2+1) + MOVING_RANGE;
   }
   else if(MOVING_PATTERN[id]==TRAJ_MOVE)
   {
      if(MOVING_RANGE!=0)
      {
         x = x + traj[id].x * (ii=(int)(lrand48()%MOVING_RANGE+1));
         y = y + traj[id].y * ii;
       }
      else
      {
     x = x + traj[id].x;
     y = y + traj[id].y;
      }
      if((x<0)||(x>max_x)) traj[id].x=-traj[id].x;
      if((y<0)||(y>max_y)) traj[id].y=-traj[id].y;
   }
   if(x<0) x=0;
   if(y<0) y=0;
   if(x>max_x) x=max_x;
   if(y>max_y) y=max_y;
   send position_msg{id,x,y} to coid;
    mob_count[id] = mob_count[id]+1;

       if (CLUST_ALGOR3)
           for (i =0;i<N;i++)

           {
               if (mob_count[id] > 5)
                  mobi[id][i].my_mob = 0;
               else
                  mobi[id][i].my_mob = 1;

         send  my_stabil {id, mobi}  to coid;
           }
  }
   or receive (my_signal sigs)
        {
signal_stren  = sigs;
```



```
  for (i=0; i<N;i++){

signals[id][signal_stren.id].id= signal_stren.id;
signals[id][i].my_sig= signal_stren.sig[0][i].my_sig;
                  }
          }

   or receive (my_stabil stab)

      {
mobi_rate   =stab;
mobi[id][mobi_rate.id].id= mobi_rate.id;
mobi[id][mobi_rate.id].my_mob= mobi_rate.mob[id][mobi_rate.id].my_mob;

       }
or  receive (ctrl_end_msg crlem)

{

   for(i=1;i<=num_nodes;i++)
         if( (neighbor[i]==1)&& (
         simclock()-updated_time[i]>
         (clocktype)(num_nodes)*CTRL_SLOT_TIME) )
         {
        neighbor[i]=0;  Neighbors[id][i]=0;
    routing_list.next_addr[i]=0; routing_table[id].next_addr[i]=0;
   routing_list.hop[i]=INFINITE_HOP; routing_table[id].hop[i]=INFINITE_HOP;
    for(j=1;j<=num_nodes;j++)
       if(routing_list.next_addr[j]==i)
       {
          routing_list.next_addr[j]=0; routing_table[id].next_addr[j]=0;
          routing_list.hop[j]=INFINITE_HOP;
                routing_table[id].hop[j]=INFINITE_HOP;
             }
        }
   for(j=1;j<=num_data_slots;j++)
   {
        if(  ((PR_ALOHA)&&(data_slots[id][j].inter_source==id)&&
       (data_slots[id][j].status=='a'))||
       ((!PR_ALOHA)&&(data_slots[id][j].inter_source==id)&&
       ((data_slots[id][j].status=='a')||
        ((data_slots[id][j].status=='r')&&
         ((routing_list.next_addr[data_slots[id][j].dest]!=data_slots
        [id][j].inter_dest)||
         (data_slots[data_slots[id][j].inter_dest][j].code!=
        data_slots[id][j].code)||
        (data_slots[data_slots[id][j].inter_dest][j].status!='r')||
        (data_slots[data_slots[id][j].inter_dest][j].source!=
         data_slots[id][j].source)||
        (data_slots[data_slots[id][j].inter_dest][j].dest!=
         data_slots[id][j].dest)||
        (try[j]>=3)
        )))
        )
 {
```

```
data_slots[id][j].status='i';
if( (!PR_ALOHA)&&(neighbor[data_slots[id][j].inter_dest]==1)&&
       (data_slots[data_slots[id][j].inter_dest][j].status=='r')&&
    (data_slots[data_slots[id][j].inter_dest]
       [j].source==active_q[j].pkt.source)&&
    (data_slots[data_slots[id][j].inter_dest]
       [j].dest==active_q[j].pkt.dest)&&
    (data_slots[data_slots[id][j].inter_dest]
       [j].pkt_num==active_q[j].pkt.pkt_num) )
 temp2_PRMA_q=PRMA_q;
 while(temp2_PRMA_q!=NULL)
 {
  if((temp2_PRMA_q->pkt.type==active_q[j].pkt.type)&&
     (temp2_PRMA_q->pkt.source==active_q[j].pkt.source)&&
     (temp2_PRMA_q->pkt.dest==active_q[j].pkt.dest))
   break;
         temp2_PRMA_q=temp2_PRMA_q->next;
 }
 if(temp2_PRMA_q!=NULL)
 {
  if(temp2_PRMA_q==PRMA_q)
     PRMA_q=PRMA_q->next;
         else
  {
   temp3_PRMA_q=PRMA_q;
   while(temp3_PRMA_q->next!=temp2_PRMA_q) temp3_PRMA_q=
    temp3_PRMA_q->next;
   temp3_PRMA_q->next=temp2_PRMA_q->next;
  }
  temp2_PRMA_q->next=first_PRMA_q;
  first_PRMA_q=temp2_PRMA_q;
 }
}
else if(active_q[j].pkt.pkt_num>0)
{
 if((temp1_PRMA_q=(struct PRMA_Q *)malloc(sizeof
    (struct PRMA_Q)))==NULL)
 { filei=fopen("ERROR","w"); fclose(filei); }
 temp1_PRMA_q->pkt.type=active_q[j].pkt.type;
 temp1_PRMA_q->pkt.source=active_q[j].pkt.source;
 temp1_PRMA_q->pkt.dest=active_q[j].pkt.dest;
 temp1_PRMA_q->pkt.pkt_num=active_q[j].pkt.pkt_num;
 temp1_PRMA_q->pkt.time_gen=active_q[j].pkt.time_gen;
 temp1_PRMA_q->next=first_PRMA_q;
 first_PRMA_q=temp1_PRMA_q;
}
else if((CONG_CTRL)&&(!PR_ALOHA)&&
    (active_q[j].pkt.pkt_num<=0)&&
    (CALL_ON[active_q[j].pkt.source][active_q[j].pkt.dest]>0))
{
 temp2_PRMA_q=PRMA_q;
 while(temp2_PRMA_q!=NULL)
 {
  if((temp2_PRMA_q->pkt.type==active_q[j].pkt.type)&&
     (temp2_PRMA_q->pkt.source==active_q[j].pkt.source)&&
```



```
     (temp2_PRMA_q->pkt.dest==active_q[j].pkt.dest))
   break;
         temp2_PRMA_q=temp2_PRMA_q->next;
  }
  if(temp2_PRMA_q!=NULL)
  {
   if(temp2_PRMA_q==PRMA_q)
      PRMA_q=PRMA_q->next;
          else
   {
    temp3_PRMA_q=PRMA_q;
    while(temp3_PRMA_q->next!=temp2_PRMA_q) temp3_PRMA_q=
       temp3_PRMA_q->next;
    temp3_PRMA_q->next=temp2_PRMA_q->next;
   }
   temp2_PRMA_q->next=first_PRMA_q;
   first_PRMA_q=temp2_PRMA_q;
         }
 }
}
else if((data_slots[id][j].status!='i')&&
    (data_slots[id][j].inter_dest==id)&&
    ((routing_list.hop[data_slots[id][j].inter_source]!=1)||
     (data_slots[data_slots[id][j].inter_source][j].inter_dest!=id)||
     (data_slots[data_slots[id][j].inter_source][j].status=='i')||
     (data_slots[data_slots[id][j].inter_source]
        [j].code!=data_slots[id][j].code)||
     (data_slots[data_slots[id][j].inter_source]
        [j].source!=data_slots[id][j].source)||
     (data_slots[data_slots[id][j].inter_source]
        [j].dest!=data_slots[id][j].dest)))
     data_slots[id][j].status='i';
       }
if(DELAY_REQ_ON)
{
 temp1_PRMA_q=PRMA_q;
 while(temp1_PRMA_q!=NULL)
 {
  if(frame-temp1_PRMA_q->pkt.time_gen>=DELAY_REQ-1)
  {
   if(temp1_PRMA_q->pkt.pkt_num>0)
   {
    pkt_dropped_PRMA++;
    pkt_status[temp1_PRMA_q->pkt.source]
        [temp1_PRMA_q->pkt.dest].dropped++;
    if((pkt_status[temp1_PRMA_q->pkt.source]
        [temp1_PRMA_q->pkt.dest].lost[(int)temp1_PRMA_q->
        pkt.pkt_num%((int)DELAY_REQ+1)]==temp1_PRMA_q->
        pkt.pkt_num)||
       (pkt_status[temp1_PRMA_q->pkt.source][temp1_PRMA_q->
       pkt.dest].got[(int)temp1_PRMA_q->pkt.pkt_num%
       ((int)DELAY_REQ+1)]==temp1_PRMA_q->pkt.pkt_num))
       pkt_status[temp1_PRMA_q->pkt.source][temp1_PRMA_q->
       pkt.dest].repeat_dropped++;
    pkt_status[temp1_PRMA_q->pkt.source][temp1_PRMA_q->
```

```
        pkt.dest].lost[(int)temp1_PRMA_q->pkt.pkt_num%(
        (int)DELAY_REQ+1)]=temp1_PRMA_q->pkt.pkt_num;
            }

    if(temp1_PRMA_q==PRMA_q)
    {
     PRMA_q=PRMA_q->next;
     free(temp1_PRMA_q);
     temp1_PRMA_q=PRMA_q;
    }
    else
    {
     temp2_PRMA_q=PRMA_q;
     while(temp2_PRMA_q->next!=temp1_PRMA_q) temp2_PRMA_q=
         temp2_PRMA_q->next;
     temp2_PRMA_q->next=temp1_PRMA_q->next;
     free(temp1_PRMA_q);
     temp1_PRMA_q=temp2_PRMA_q->next;
    }
   }
   else
    temp1_PRMA_q=temp1_PRMA_q->next;
  }
}

if(DELAY_REQ_ON)
    {
 temp1_PRMA_q=first_PRMA_q;
 while(temp1_PRMA_q!=NULL)
 {
  if(frame-temp1_PRMA_q->pkt.time_gen>=DELAY_REQ-1)
  {
   if(temp1_PRMA_q->pkt.pkt_num>0)
   {
    pkt_dropped_PRMA++;
    pkt_status[temp1_PRMA_q->pkt.source][temp1_PRMA_q->
        pkt.dest].dropped++;
    if((pkt_status[temp1_PRMA_q->pkt.source][temp1_PRMA_q->
       pkt.dest].lost[(int)temp1_PRMA_q->pkt.pkt_num%(
       (int)DELAY_REQ+1)]==temp1_PRMA_q->pkt.pkt_num)||
       (pkt_status[temp1_PRMA_q->pkt.source][temp1_PRMA_q->
       pkt.dest].got[(int)temp1_PRMA_q->pkt.pkt_num%((int)
       DELAY_REQ+1)]==temp1_PRMA_q->pkt.pkt_num))
       pkt_status[temp1_PRMA_q->pkt.source][temp1_PRMA_q->
       pkt.dest].repeat_dropped++;

   pkt_status[temp1_PRMA_q->pkt.source][temp1_PRMA_q->
      pkt.dest].lost[(int)temp1_PRMA_q->pkt.pkt_num%(
      (int)DELAY_REQ+1)]=temp1_PRMA_q->pkt.pkt_num;
          }
   temp2_PRMA_q=PRMA_q;
   while(temp2_PRMA_q!=NULL)
   {
    if((temp2_PRMA_q->pkt.type==temp1_PRMA_q->pkt.type)&&
      (temp2_PRMA_q->pkt.source==temp1_PRMA_q->pkt.source)&&
```



```
       (temp2_PRMA_q->pkt.dest==temp1_PRMA_q->pkt.dest))
         break;
     temp2_PRMA_q=temp2_PRMA_q->next;
    }
    if(temp2_PRMA_q!=NULL)
    {
     temp1_PRMA_q->pkt.pkt_num=temp2_PRMA_q->pkt.pkt_num;
     temp1_PRMA_q->pkt.time_gen=temp2_PRMA_q->pkt.time_gen;
           if(temp2_PRMA_q==PRMA_q)
     {
      PRMA_q=PRMA_q->next;
      free(temp2_PRMA_q);
     }
     else
     {
      temp3_PRMA_q=PRMA_q;
      while(temp3_PRMA_q->next!=temp2_PRMA_q) temp3_PRMA_q=temp3_PRMA_q->next;
      temp3_PRMA_q->next=temp2_PRMA_q->next;
      free(temp2_PRMA_q);
     }
    }
    else
    {
     if(temp1_PRMA_q==first_PRMA_q)
     {
      first_PRMA_q=first_PRMA_q->next;
      free(temp1_PRMA_q);
      temp1_PRMA_q=first_PRMA_q;
     }
     else
     {
      temp2_PRMA_q=first_PRMA_q;
      while(temp2_PRMA_q->next!=temp1_PRMA_q) temp2_PRMA_q=
         temp2_PRMA_q->next;
      temp2_PRMA_q->next=temp1_PRMA_q->next;
      free(temp1_PRMA_q);
      temp1_PRMA_q=temp2_PRMA_q->next;
     }
           }
   }
   else
    temp1_PRMA_q=temp1_PRMA_q->next;
  }
    }

if((CONG_CTRL)&&(!PR_ALOHA))
{
  for(i=1;i<=num_nodes;i++)
    for(j=1;j<=num_nodes;j++)
    {
      if((CALL_ON[i][j]>0)&&(my_cong_flag[i][j]>0)&&(cong_flag
         [i][j]==0))
     my_cong_flag[i][j]=0;
      if((CALL_ON[i][j]>0)&&(my_cong_flag[i][j]==0))
      {
```

```
    q_len=0;
    temp1_PRMA_q=PRMA_q;
    while(temp1_PRMA_q!=NULL)
    { if((temp1_PRMA_q->pkt.source==i)&&(temp1_PRMA_q->
      pkt.dest==j)&&(temp1_PRMA_q->pkt.pkt_num>0))
        q_len++;
     temp1_PRMA_q=temp1_PRMA_q->next;
    }
    if((q_len>=(int)DELAY_REQ*3/10)||((i==id)&&(q_len>=
                  (int)DELAY_REQ*2/10)))
    {  my_cong_flag[i][j]++; cong_flag[i][j]++; }
      }
      else if((CALL_ON[i][j]>0)&&(my_cong_flag[i][j]>0))
      {
    q_len=0;
    temp1_PRMA_q=PRMA_q;
    while(temp1_PRMA_q!=NULL)
    { if((temp1_PRMA_q->pkt.source==i)&&(temp1_PRMA_q->
         pkt.dest==j)&&(temp1_PRMA_q->pkt.pkt_num>0))
       q_len++;
      temp1_PRMA_q=temp1_PRMA_q->next;
    }
    if(q_len<=(int)DELAY_REQ*1/10)
    {  my_cong_flag[i][j]--; cong_flag[i][j]--; }
      }
    }
}
if(PR_ALOHA)
   for(j=1;j<=num_data_slots;j++)
      data_slots[id][j].status='i';
      ii=0;
for(j=1;j<=num_data_slots;j++)
 if((data_slots[id][j].status=='i')&&
    (((data_slots[my_ch][j].status=='i')&&(data_slots[id]
         [j].code==my_ch))||
     ((my_2nd_ch!=0)&&(data_slots[my_2nd_ch][j].status=='i')
         &&(data_slots[id][j].code==my_2nd_ch))||
     ((my_3rd_ch!=0)&&(data_slots[my_3rd_ch][j].status=='i')
         &&(data_slots[id][j].code==my_3rd_ch))
    ))
      ii++;
        j1=0;
        for(j=1;j<=num_data_slots;j++)
   if((data_slots[id][j].status=='i')&&(data_slots[my_ch]
       [j].status=='i')&&(data_slots[id][j].code==my_ch))
      j1++;
        if(I_am_gw[id])
{
         j2=0;
         for(j=1;j<=num_data_slots;j++)
    if((my_2nd_ch!=0)&&(data_slots[id][j].status=='i')&&
        (data_slots[my_2nd_ch][j].status=='i')&&
        (data_slots[id][j].code==my_2nd_ch))
       j2++;
         if(my_3rd_ch!=0)
```



```
{
          j3=0;
          for(j=1;j<=num_data_slots;j++)
     if((data_slots[id][j].status=='i')&&(data_slots[my_3rd_ch][j].status=='i')&&(data_s
        j3++;
        }
 else
    j3=j1;
}
else
{ j2=j1; j3=j1; }

       j1=j1/3+1;
j2=j2/3+1; j3=j3/3+1;
if(PR_ALOHA){ii=num_data_slots; j1=ii/3; j2=j1; j3=j1;}
temp1_PRMA_q=first_PRMA_q;
while(((j1>0)||(j2>0)||(j3>0))&&(ii>0)&&(temp1_PRMA_q!=NULL))
{

 if(!PR_ALOHA)
 {
   i=0;
   for(j=1;j<=num_data_slots;j++)
      if(((temp_addr=routing_list.next_addr[temp1_PRMA_q->pkt.dest])!=0)&&
     (data_slots[id][j].status=='i')&&
     (data_slots[temp_addr][j].status=='i')&&
     (((data_slots[my_ch][j].status=='i')&&(my_ch==data_slots[id][j].code))||
      ((my_2nd_ch!=0)&&(data_slots[my_2nd_ch][j].status=='i')&&(my_2nd_ch==data_slots[id]|
      ((my_3rd_ch!=0)&&(data_slots[my_3rd_ch][j].status=='i')&&(my_3rd_ch==data_slots[id]|
     )&&
     (data_slots[id][j].code==data_slots[temp_addr][j].code))
       i++;
           i1=0;
   for(j=1;j<=num_data_slots;j++)
      if((data_slots[my_ch][j].status=='i')&&(data_slots[id][j].status=='i'))
     i1++;
           if(!I_am_gw[id]) {i2=i1; i3=i1; }
   else
   {
            i2=0;
    for(j=1;j<=num_data_slots;j++)
       if((data_slots[my_2nd_ch][j].status=='i')&&(data_slots[id][j].status=='i'))
      i2++;
           if(my_3rd_ch!=0)
    {
            i3=0;
     for(j=1;j<=num_data_slots;j++)
        if((data_slots[my_3rd_ch][j].status=='i')&&(data_slots[id][j].status=='i'))
       i3++;
           }
    else   i3=i1;
   }
           if((i!=0)&&((temp1_PRMA_q->pkt.pkt_num>0)||
              ((CALL_ON[temp1_PRMA_q->pkt.source][temp1_PRMA_q->
              pkt.dest]>0)&&(temp1_PRMA_q->pkt.pkt_num==0))||
```

```
            ((i1>=num_data_slots/2+1)&&(i2>=num_data_slots/2+1)
            &&(i3>=num_data_slots/2+1)&&
          (((!I_am_gw[id])&&(!I_am_gw[temp_addr]))||
         (((I_am_gw[temp_addr])||(I_am_gw[id]))&&(i>=
         num_data_slots/4+1)) )) ))

   i=(int)(rand()%i+1);
  else i=num_data_slots+1;
}
        else if(PR_ALOHA)
  i=(int)(rand()%ii+1);
if((I_am_gw[id])&&(!PR_ALOHA)) i=i/2+1;
for(j=1;j<=num_data_slots;j++)
{
 if(((temp_addr=routing_list.next_addr[temp1_PRMA_q->pkt.dest])==0)
   ||(temp_addr==temp1_PRMA_q->pkt.source))
 {
    if(NO_ROUTE_COUNT)
    {
     if(temp1_PRMA_q->pkt.pkt_num>0)
      {
    pkt_status[temp1_PRMA_q->pkt.source][temp1_PRMA_q->
        pkt.dest].no_route_pkts++;
        pkt_dropped_PRMA++;
        pkt_status[temp1_PRMA_q->pkt.source][temp1_PRMA_q->
           pkt.dest].dropped++;
        if((pkt_status[temp1_PRMA_q->pkt.source][temp1_PRMA_q->
           pkt.dest].lost[(int)temp1_PRMA_q->pkt.pkt_num%((int)
           DELAY_REQ+1)]==temp1_PRMA_q->pkt.pkt_num)||
           (pkt_status[temp1_PRMA_q->pkt.source][temp1_PRMA_q->
           pkt.dest].got[(int)temp1_PRMA_q->pkt.pkt_num%((int)
           DELAY_REQ+1)]==temp1_PRMA_q->pkt.pkt_num))
           pkt_status[temp1_PRMA_q->pkt.source][temp1_PRMA_q->
           pkt.dest].repeat_dropped++;
        pkt_status[temp1_PRMA_q->pkt.source][temp1_PRMA_q->
           pkt.dest].lost[(int)temp1_PRMA_q->pkt.pkt_num%(
           (int)DELAY_REQ+1)]=temp1_PRMA_q->pkt.pkt_num;
              }
              temp2_PRMA_q=PRMA_q;
      while(temp2_PRMA_q!=NULL)
      {
     if((temp2_PRMA_q->pkt.type==temp1_PRMA_q->pkt.type)&&
        (temp2_PRMA_q->pkt.source==temp1_PRMA_q->pkt.source)&&
        (temp2_PRMA_q->pkt.dest==temp1_PRMA_q->pkt.dest))
                     break;
                else
      temp2_PRMA_q=temp2_PRMA_q->next;
      }
      if(temp2_PRMA_q!=NULL)
      {
    temp1_PRMA_q->pkt.pkt_num=temp2_PRMA_q->pkt.pkt_num;
    temp1_PRMA_q->pkt.time_gen=temp2_PRMA_q->pkt.time_gen;
    if(temp2_PRMA_q==PRMA_q)
    {
     PRMA_q=PRMA_q->next;
```



```
     free(temp2_PRMA_q);
    }
    else
    {
     temp3_PRMA_q=PRMA_q;
     while(temp3_PRMA_q->next!=temp2_PRMA_q)
        temp3_PRMA_q=temp3_PRMA_q->next;
                temp3_PRMA_q->next=temp2_PRMA_q->next;
     free(temp2_PRMA_q);
    }
       }
       else
       {
    if(temp1_PRMA_q==first_PRMA_q)
    {
     first_PRMA_q=first_PRMA_q->next;
     free(temp1_PRMA_q);
     temp1_PRMA_q=first_PRMA_q;
     i=0;
    }
    else
    {
     temp2_PRMA_q=first_PRMA_q;
     while(temp2_PRMA_q->next!=temp1_PRMA_q)
        temp2_PRMA_q=temp2_PRMA_q->next;
                temp2_PRMA_q->next=temp1_PRMA_q->next;
     free(temp1_PRMA_q);
     temp1_PRMA_q=temp2_PRMA_q;
    }
       }
     }
     break;
}

if((PR_ALOHA)&&(data_slots[id][j].status=='i')) i--;
else if((data_slots[id][j].status=='i')&&
   (((data_slots[my_ch][j].status=='i')&&(my_ch==data_slots[id][j].code))||
    ((my_2nd_ch!=0)&&(data_slots[my_2nd_ch][j].status=='i')&&
          (my_2nd_ch==data_slots[id][j].code))||
    ((my_3rd_ch!=0)&&(data_slots[my_3rd_ch][j].status=='i')&&
          (my_3rd_ch==data_slots[id][j].code))
          )&&
          (data_slots[id][j].code==data_slots[temp_addr][j].code)&&
   (data_slots[temp_addr][j].status=='i'))
    i--;
        if(i==0)
{
 if(PR_ALOHA)
 { j1--; j2--; j3--; }
 else
 {
  if((j1>0)&&(data_slots[id][j].code==my_ch))
  { j1--; if(my_2nd_ch==0)j2--; if(my_3rd_ch==0) j3--;}
  else if ((my_2nd_ch!=0)&&(j2>0)&&(data_slots[id][j].code==my_2nd_ch))
  { j2--; if(my_3rd_ch==0)j3--;  }
```

```
        else if((my_3rd_ch!=0)&&(j3>0)&&(data_slots[id][j].code
        ==my_3rd_ch)) j3--;
        else
        { temp1_PRMA_q=temp1_PRMA_q->next;  break; }
       }
       ii--;
       data_slots[id][j].status='a';
       data_slots[id][j].source=temp1_PRMA_q->pkt.source;
       data_slots[id][j].inter_source=id;
       data_slots[id][j].inter_dest=temp_addr;
       data_slots[id][j].dest=temp1_PRMA_q->pkt.dest;
       data_slots[id][j].pkt_num=temp1_PRMA_q->pkt.pkt_num;
       active_q[j].inter_source=id;
       active_q[j].inter_dest=temp_addr;
       active_q[j].code=data_slots[id][j].code;
       active_q[j].pkt.type=temp1_PRMA_q->pkt.type;
       active_q[j].pkt.source=temp1_PRMA_q->pkt.source;
       active_q[j].pkt.dest=temp1_PRMA_q->pkt.dest;
       active_q[j].pkt.pkt_num=temp1_PRMA_q->pkt.pkt_num;
       active_q[j].pkt.time_gen=temp1_PRMA_q->pkt.time_gen;
       if(temp1_PRMA_q==first_PRMA_q)
       {
        temp2_PRMA_q=first_PRMA_q;
        first_PRMA_q=first_PRMA_q->next;
        temp1_PRMA_q=temp1_PRMA_q->next;
        free(temp2_PRMA_q);
             }
       else
         {
        temp2_PRMA_q=first_PRMA_q;
        while(temp2_PRMA_q->next!=temp1_PRMA_q) temp2_PRMA_q=
                temp2_PRMA_q->next;
                temp2_PRMA_q->next=temp1_PRMA_q->next;
        temp2_PRMA_q=temp1_PRMA_q;
        temp1_PRMA_q=temp1_PRMA_q->next;
        free(temp2_PRMA_q);
       }
       break;
        }
        }
         if(i!=0) temp1_PRMA_q=temp1_PRMA_q->next;
       }
          }
     or  receive (ctrl_ph ctrlp)
    {
    ctrl = ctrlp;

    I_am_ch[ctrl.id]=ctrl.iamch;
    neighbor[ctrl.id] =1;
    Neighbors[id][ctrl.id]=1;

if(ROUTING_PRO1)
    {
      for(i=0;i<N;i++)
      {
```



```
       other_routing_lists[ctrl.id].next_addr[i]=ctrl.r_list.next_addr[i];
       other_routing_lists[ctrl.id].hop[i]=ctrl.r_list.hop[i];

         }
       }
       for(i=0;i<N;i++)
       {
         if( (PR_ALOHA)||
         ((I_am_ch[i])&&((neighbor[i]==1)||(i==id))
         &&((ctrl.neighbor[i]==1)||(i==ctrl.id)))
              )
         {
        routing_list.hop[ ctrl.id]=1;
        routing_list.next_addr[ctrl.id]=ctrl.id;
         }
       }
       if((i>N)&&(routing_list.hop[ctrl.id]==1))
       {
          routing_list.hop[ctrl.id]=INFINITE_HOP;
          routing_list.next_addr[ctrl.id]=0;
          for(j=0;j<N;j++)
         if(routing_list.next_addr[j]==ctrl.id)
         {
            routing_list.next_addr[j]=0;
            routing_list.hop[j]=INFINITE_HOP;
          }
         }
       if(routing_list.hop[ctrl.id]==1)
       {
          for(i=0;i<N;i++)
          {
            if((routing_list.next_addr[i]==ctrl.id)&&
           (routing_list.hop[i]!=ctrl.r_list.hop[i]+1))
                {
           if((ctrl.r_list.hop[i]==INFINITE_HOP)||(ctrl.r_list.hop[i]+1
               ==INFINITE_HOP)||(ctrl.r_list.next_addr[i]==id))
           { routing_list.next_addr[i]=0; routing_list.hop[i]=INFINITE_HOP;}
           else
             routing_list.hop[i]=ctrl.r_list.hop[i]+1;
            }
            if((routing_list.hop[i]>ctrl.r_list.hop[i]+1)&&
                (ctrl.r_list.next_addr[i]!=id))
            {
           routing_list.next_addr[i]=ctrl.id;
           routing_list.hop[i]=ctrl.r_list.hop[i]+1;
          if(routing_list.hop[i]==INFINITE_HOP)
              routing_list.next_addr[i]=0;
            }
              }
       }
if(ROUTING_PRO1)
       {
     for(i=0;i<N;i++)  {
   if((routing_list.hop[i]==INFINITE_HOP)  &&(i!=infinity))
```

```
    for(j=0;j<N;j++)

        routing_list.hop[i]=other_routing_lists[j].hop[i];
    routing_list.next_addr[i]=j;

      if((j!=i)&&(j!=id)&&(routing_list.hop[j]==1)&&
     (other_routing_lists[j].next_addr[i]!=id)&&
     (other_routing_lists[j].hop[i]<routing_list.hop[i]-1))
             {
        routing_list.hop[i]=other_routing_lists[j].hop[i]+1;
        routing_list.next_addr[i]=j;
        if(routing_list.hop[i]==INFINITE_HOP)
        routing_list.next_addr[i]=0;
      }
             }
  }
for(i=0;i<N;i++)
  for(j=0;j<N;j++)
    if((i!=id)&&(j!=i)&&(j!=id)&&(routing_list.hop[j]==1)&&
       (routing_list.hop[i]>other_routing_lists[j].hop[i]+1)&&
       (other_routing_lists[j].hop[i]!=INFINITE_HOP)&&
       (other_routing_lists[j].hop[i]!=INFINITE_HOP-1))
           {
      temp_head=other_routing_lists[j].head[i];
      for(ii=0;ii<=other_routing_lists[j].hop[i];ii++)
      {
        temp_head=other_routing_lists[j].head[temp_head];
        if((other_routing_lists[j].hop[temp_head]!=other_routing_lists[j].hop[i]-ii)||
       (temp_head==j)||(temp_head==id))
          break;
      }
      if((temp_head==j)&&(temp_head!=id)&&(ii==other_routing_lists[j].hop[i]))
      {
        routing_list.hop[i]=other_routing_lists[j].hop[i]+1;
        routing_list.next_addr[i]=j;
        routing_list.head[i]=other_routing_lists[j].head[i];
      }
    }
  }
  for(i=0;i<N;i++)
  {
     routing_table[id].next_addr[i]=routing_list.next_addr[i];
     routing_table[id].hop[i]=routing_list.hop[i];
  }
  for(j=1;j<=num_data_slots;j++)
  {

   data_slots[ctrl.id][j].status=ctrl.d_slots[j].status;
   data_slots[ctrl.id][j].code=ctrl.d_slots[j].code;
   data_slots[ctrl.id][j].source=ctrl.d_slots[j].source;
   data_slots[ctrl.id][j].inter_source=ctrl.d_slots[j].inter_source;
   data_slots[ctrl.id][j].inter_dest=ctrl.d_slots[j].inter_dest;
   data_slots[ctrl.id][j].dest=ctrl.d_slots[j].dest;
   data_slots[ctrl.id][j].pkt_num=ctrl.d_slots[j].pkt_num;
```



```
   if((data_slots[id][j].status=='a')&&
  (data_slots[ctrl.id][j].status=='r')&&
  (data_slots[id][j].code==data_slots[ctrl.id][j].code)&&
  (data_slots[id][j].source==data_slots[ctrl.id][j].source)&&
  (data_slots[id][j].inter_source==data_slots[ctrl.id][j].inter_source)&&
  (data_slots[id][j].inter_dest==data_slots[ctrl.id][j].inter_dest)&&
  (data_slots[id][j].dest==data_slots[ctrl.id][j].dest)&&
  (data_slots[ctrl.id][j].pkt_num==data_slots[id][j].pkt_num))
        {      data_slots[id][j].status='r';  try[j]=0;
         }
   else if((!PR_ALOHA)&&(data_slots[id][j].status=='r')&&
  (data_slots[ctrl.id][j].status=='r')&&
  (data_slots[id][j].code==data_slots[ctrl.id][j].code)&&
  (data_slots[id][j].source==data_slots[ctrl.id][j].source)&&
  (data_slots[id][j].inter_source==data_slots[ctrl.id][j].inter_source)&&
  (data_slots[id][j].inter_dest==data_slots[ctrl.id][j].inter_dest)&&
  (data_slots[id][j].dest==data_slots[ctrl.id][j].dest)&&
  (data_slots[ctrl.id][j].pkt_num==data_slots[id][j].pkt_num))
      {    try[j]=0;}

   else if((!PR_ALOHA)&&(data_slots[id][j].status=='r')&&
  (data_slots[ctrl.id][j].status=='r')&&
  (data_slots[id][j].code==data_slots[ctrl.id][j].code)&&
  (data_slots[id][j].source==data_slots[ctrl.id][j].source)&&
  (data_slots[id][j].inter_source==data_slots[ctrl.id][j].inter_source)&&
  (data_slots[id][j].inter_dest==data_slots[ctrl.id][j].inter_dest)&&
  (data_slots[id][j].dest==data_slots[ctrl.id][j].dest)&&
  (data_slots[ctrl.id][j].pkt_num!=data_slots[id][j].pkt_num))
        {     try[j]++;
   }
  }
 }
or  receive (data_in datai)

   {
  datum=datai;
  if((data_slots[id][datum.slot_no].source >0))
  if((data_slots[id][datum.slot_no].status=='i')&&
  (data_slots[id][datum.slot_no].code==datum.pkt_in.code)&&
  (datum.pkt_in.inter_dest==id))

     {
   data_slots[id][datum.slot_no].status='r';
   data_slots[id][datum.slot_no].source=datum.pkt_in.pkt.source;
   data_slots[id][datum.slot_no].inter_source=datum.pkt_in.inter_source;
   data_slots[id][datum.slot_no].inter_dest=datum.pkt_in.inter_dest;
   data_slots[id][datum.slot_no].dest=datum.pkt_in.pkt.dest;
   data_slots[id][datum.slot_no].pkt_num=datum.pkt_in.pkt.pkt_num;
   if(data_slots[id][datum.slot_no].dest <datum.pkt_in.pkt.pkt_num)

   {
    if((CONG_CTRL)&&(!PR_ALOHA))

    {
```

```
 }
 if(datum.pkt_in.pkt.pkt_num==-1) {
  CALL_ON[datum.pkt_in.pkt.source][datum.pkt_in.pkt.dest]++;
        }
     else if(datum.pkt_in.pkt.pkt_num>0)
 {
  pkt_got_PRMA++;
  pkt_status[datum.pkt_in.pkt.source][datum.pkt_in.pkt.dest].received++;
  if(pkt_status[datum.pkt_in.pkt.source][datum.pkt_in.pkt.dest].got
         [(int)datum.pkt_in.pkt.pkt_num%((int)DELAY_REQ+1)]==
         datum.pkt_in.pkt.pkt_num)
         {
           pkt_status[datum.pkt_in.pkt.source]
           [datum.pkt_in.pkt.dest].received++;
       else
  {
     pkt_status[datum.pkt_in.pkt.source][datum.pkt_in.pkt.dest]
        .got[(int)datum.pkt_in.pkt.pkt_num%((int)DELAY_REQ+1)]
        =datum.pkt_in.pkt.pkt_num;

     if(pkt_status[datum.pkt_in.pkt.source][datum.pkt_in.pkt.dest]
         .lost[(int)datum.pkt_in.pkt.pkt_num%((int)DELAY_REQ+1)]
         ==datum.pkt_in.pkt.pkt_num)
    pkt_status[datum.pkt_in.pkt.source][datum.pkt_in.pkt.dest]
         .repeat_dropped++;
      }

   if(pkt_status[datum.pkt_in.pkt.source][datum.pkt_in.pkt.dest]
       .last_pkt_heard+1!=datum.pkt_in.pkt.pkt_num)
        pkt_status[datum.pkt_in.pkt.source][datum.pkt_in.pkt
       .dest].out_of_seq++;
   pkt_status[datum.pkt_in.pkt.source][datum.pkt_in.pkt.dest]
       .last_pkt_heard=datum.pkt_in.pkt.pkt_num;
      }
}
 else
 {

if((temp1_PRMA_q=(struct PRMA_Q *)malloc(sizeof(struct PRMA_Q)))==NULL)
  { file1=fopen("ERROR","w"); fclose(file1); }
  temp1_PRMA_q->pkt.type=datum.pkt_in.pkt.type;
  temp1_PRMA_q->pkt.source=datum.pkt_in.pkt.source;
  temp1_PRMA_q->pkt.dest=datum.pkt_in.pkt.dest;
  temp1_PRMA_q->pkt.pkt_num=datum.pkt_in.pkt.pkt_num;
  temp1_PRMA_q->pkt.time_gen=datum.pkt_in.pkt.time_gen;

  if(datum.pkt_in.pkt.pkt_num>=0)
  {
  temp1_PRMA_q->next=first_PRMA_q;
  first_PRMA_q=temp1_PRMA_q;
  }
  else
  {
   temp1_PRMA_q->next=NULL;
```



```
   if(first_PRMA_q!=NULL)
   {
temp2_PRMA_q=first_PRMA_q;
while(temp2_PRMA_q->next!=NULL)
        temp2_PRMA_q=temp2_PRMA_q->next;
temp2_PRMA_q->next=temp1_PRMA_q;
   }
   else
first_PRMA_q=temp1_PRMA_q;
  }
 }
}

else if((data_slots[id][datum.slot_no].status=='r')&&
    (data_slots[id][datum.slot_no].code==datum.pkt_in.code)&&
    (data_slots[id][datum.slot_no].source==datum.pkt_in.pkt.source)&&
    (data_slots[id][datum.slot_no].inter_source==
    datum.pkt_in.inter_source)&&
    (data_slots[id][datum.slot_no].inter_dest==id)&&
    (datum.pkt_in.inter_dest==id)&&
    (data_slots[id][datum.slot_no].dest==datum.pkt_in.pkt.dest))
    {
 data_slots[id][datum.slot_no].pkt_num=datum.pkt_in.pkt.pkt_num;
 if(data_slots[id][datum.slot_no].dest==id)
 {
  if((CONG_CTRL)&&(!PR_ALOHA))
  {

  }
  if(datum.pkt_in.pkt.pkt_num==-1)
 CALL_ON[datum.pkt_in.pkt.source][datum.pkt_in.pkt.dest]++;
      else if(datum.pkt_in.pkt.pkt_num>0)
  {
   pkt_got_PRMA++;
   pkt_status[datum.pkt_in.pkt.source]
        [datum.pkt_in.pkt.dest].received++;
   if(pkt_status[datum.pkt_in.pkt.source]
        [datum.pkt_in.pkt.dest].got[(int)
        datum.pkt_in.pkt.pkt_num%((int)DELAY_REQ+1)]==
        datum.pkt_in.pkt.pkt_num)
  pkt_status[datum.pkt_in.pkt.source]
      [datum.pkt_in.pkt.dest].received++;
       else
  {
     pkt_status[datum.pkt_in.pkt.source][datum.pkt_in.pkt.dest].got
       [(int)datum.pkt_in.pkt.pkt_num%((int)DELAY_REQ+1)]=
       datum.pkt_in.pkt.pkt_num;
     if(pkt_status[datum.pkt_in.pkt.source]
       [datum.pkt_in.pkt.dest].lost[(int)datum.pkt_in.pkt.pkt_num%
       ((int)DELAY_REQ+1)]==datum.pkt_in.pkt.pkt_num)
     pkt_status[datum.pkt_in.pkt.source]
       [datum.pkt_in.pkt.dest].repeat_dropped++;
       }

   if(pkt_status[datum.pkt_in.pkt.source][datum.pkt_in.pkt.dest]
```

```
      .last_pkt_heard+1!=datum.pkt_in.pkt.pkt_num)
  pkt_status[datum.pkt_in.pkt.source][datum.pkt_in.pkt.dest]
       .out_of_seq++;
   pkt_status[datum.pkt_in.pkt.source][datum.pkt_in.pkt.dest]
        .last_pkt_heard=datum.pkt_in.pkt.pkt_num;
      }
  }
 else
 {
  if((temp1_PRMA_q=(struct PRMA_Q *)malloc(sizeof(struct PRMA_Q)))
      ==NULL)
  { file1=fopen("ERROR","w"); fclose(file1); }
  temp1_PRMA_q->pkt.type=datum.pkt_in.pkt.type;
  temp1_PRMA_q->pkt.source=datum.pkt_in.pkt.source;
  temp1_PRMA_q->pkt.dest=datum.pkt_in.pkt.dest;
  temp1_PRMA_q->pkt.pkt_num=datum.pkt_in.pkt.pkt_num;
  temp1_PRMA_q->pkt.time_gen=datum.pkt_in.pkt.time_gen;
  temp1_PRMA_q->next=NULL;
  for(j=1;j<=num_data_slots;j++)
if((data_slots[id][j].status!='i')&&(data_slots[id][j].inter_source
   ==id)&&
   ('r'==datum.pkt_in.pkt.type)&&
   (data_slots[id][j].source==datum.pkt_in.pkt.source)&&
   (data_slots[id][j].dest==datum.pkt_in.pkt.dest))
      break;
      temp3_PRMA_q=first_PRMA_q;
  while(temp3_PRMA_q!=NULL)
  {
if((temp3_PRMA_q->pkt.type==datum.pkt_in.pkt.type)&&
   (temp3_PRMA_q->pkt.source==datum.pkt_in.pkt.source)&&
   (temp3_PRMA_q->pkt.dest==datum.pkt_in.pkt.dest)) break;
else
   temp3_PRMA_q=temp3_PRMA_q->next;
  }
      if((j>num_data_slots)&&(temp3_PRMA_q==NULL))
  {
temp1_PRMA_q->next=first_PRMA_q;
first_PRMA_q=temp1_PRMA_q;
  }
  else
  {
if((datum.pkt_in.pkt.pkt_num>0)&&(!FIFO))
{
      temp1_PRMA_q->next=PRMA_q;
      PRMA_q=temp1_PRMA_q;
        }
    else if(PRMA_q!=NULL)
    {
     temp2_PRMA_q=PRMA_q;
     while(temp2_PRMA_q->next!=NULL) temp2_PRMA_q=temp2_PRMA_q->next;
     temp2_PRMA_q->next=temp1_PRMA_q;
    }
    else
     PRMA_q=temp1_PRMA_q;
if((FIFO)&&(!PR_ALOHA)&&(datum.pkt_in.pkt.pkt_num>0))
```



```
    {

     temp2_PRMA_q=PRMA_q;
     while(temp2_PRMA_q!=NULL)
     {
      if((temp2_PRMA_q->pkt.pkt_num==0)&&
         (temp2_PRMA_q->pkt.type==datum.pkt_in.pkt.type)&&
         (temp2_PRMA_q->pkt.source==datum.pkt_in.pkt.source)&&
         (temp2_PRMA_q->pkt.dest==datum.pkt_in.pkt.dest))
              {
       if(temp2_PRMA_q==PRMA_q)
       {
        PRMA_q=PRMA_q->next;
        free(temp2_PRMA_q);
        temp2_PRMA_q=PRMA_q;
       }
       else
       {
        temp3_PRMA_q=PRMA_q;
        while(temp3_PRMA_q->next!=temp2_PRMA_q)
           temp3_PRMA_q=temp3_PRMA_q->next;
                temp3_PRMA_q->next=temp2_PRMA_q->next;
        free(temp2_PRMA_q);
        temp2_PRMA_q=temp3_PRMA_q->next;
       }
      }
              else
       temp2_PRMA_q=temp2_PRMA_q->next;
     }
    }
   }
  }
}
     data_slots[id][datum.slot_no].status='r';
     data_slots[id][datum.slot_no].source=datum.pkt_in.pkt.source;
     data_slots[id][datum.slot_no].inter_source=datum.pkt_in.inter_source;
     data_slots[id][datum.slot_no].inter_dest=datum.pkt_in.inter_dest;
     data_slots[id][datum.slot_no].dest=datum.pkt_in.pkt.dest;
     data_slots[id][datum.slot_no].pkt_num=datum.pkt_in.pkt.pkt_num;
     temp1_PRMA_q->pkt.type=datum.pkt_in.pkt.type;
     temp1_PRMA_q->pkt.source=datum.pkt_in.pkt.source;
     temp1_PRMA_q->pkt.dest=datum.pkt_in.pkt.dest;
     temp1_PRMA_q->pkt.pkt_num=datum.pkt_in.pkt.pkt_num;
     temp1_PRMA_q->pkt.time_gen=datum.pkt_in.pkt.time_gen;
 }
 or receive  (the_data_slot datas)
    {
        the_d_slot=datas;
       {
       }
       else if((data_slots[id][the_d_slot.no].status=='r')&&
      (data_slots[id][the_d_slot.no].inter_source==id))
       {
     temp1_PRMA_q=PRMA_q;
     while(temp1_PRMA_q!=NULL)
```

```
    {

     data_slots[id][the_d_slot.no].pkt_num=temp1_PRMA_q->pkt.pkt_num;
     active_q[the_d_slot.no].pkt.type=temp1_PRMA_q->pkt.type;
     active_q[the_d_slot.no].pkt.source=temp1_PRMA_q->pkt.source;
     active_q[the_d_slot.no].pkt.dest=temp1_PRMA_q->pkt.dest;
     active_q[the_d_slot.no].pkt.pkt_num=temp1_PRMA_q->pkt.pkt_num;
     active_q[the_d_slot.no].pkt.time_gen=temp1_PRMA_q->pkt.time_gen;
         active_q[the_d_slot.no].inter_source=id;
         active_q[the_d_slot.no].inter_dest=data_slots[id]
         [the_d_slot.no].inter_dest;
     active_q[the_d_slot.no].code=data_slots[id][the_d_slot.no].code;
     send data_out{the_d_slot.no, active_q[the_d_slot.no]} to coid;
      hold(1);
     if((data_slots[id][the_d_slot.no].source==temp1_PRMA_q->pkt.source)&&
    (data_slots[id][the_d_slot.no].dest==temp1_PRMA_q->pkt.dest)) break;
     else
     temp1_PRMA_q=temp1_PRMA_q->next;
    }
    if(temp1_PRMA_q!=NULL)
    {
     data_slots[id][the_d_slot.no].pkt_num=temp1_PRMA_q->pkt.pkt_num;
     active_q[the_d_slot.no].pkt.type=temp1_PRMA_q->pkt.type;
     active_q[the_d_slot.no].pkt.source=temp1_PRMA_q->pkt.source;
     active_q[the_d_slot.no].pkt.dest=temp1_PRMA_q->pkt.dest;
     active_q[the_d_slot.no].pkt.pkt_num=temp1_PRMA_q->pkt.pkt_num;
     active_q[the_d_slot.no].pkt.time_gen=temp1_PRMA_q->pkt.time_gen;
         active_q[the_d_slot.no].inter_source=id;
         active_q[the_d_slot.no].inter_dest=data_slots[id]
         [the_d_slot.no].inter_dest;
     active_q[the_d_slot.no].code=data_slots[id][the_d_slot.no].code;
       if(temp1_PRMA_q==PRMA_q)
     {
      PRMA_q=PRMA_q->next;
         }
     else
     {
      temp2_PRMA_q=PRMA_q;
      while(temp2_PRMA_q->next!=temp1_PRMA_q) temp2_PRMA_q=
          temp2_PRMA_q->next;
          temp2_PRMA_q->next=temp1_PRMA_q->next;
   free(temp1_PRMA_q); }
   send data_out{the_d_slot.no, active_q[the_d_slot.no]} to coid;
   the_d_slot.no, id, active_q[the_d_slot.no].pkt.source,
   active_q[the_d_slot.no].inter_source,active_q[the_d_slot.no].
       inter_dest,active_q[the_d_slot.no].pkt.dest,active_q
       [the_d_slot.no].pkt.pkt_num,active_q[the_d_slot.no].code);
    }
    else
    { data_slots[id][the_d_slot.no].status='i';
          }
    }
  }
or  receive (call_req callr)
  {
```



```
    c_req=callr;
  if((temp1_PRMA_q=(struct PRMA_Q *)malloc(sizeof(struct PRMA_Q)))==NULL)
  { file1=fopen("ERROR","w"); fclose(file1); }
    temp1_PRMA_q->pkt.type=c_req.first_pkt.type;
    temp1_PRMA_q->pkt.source=c_req.first_pkt.source;
    temp1_PRMA_q->pkt.dest=c_req.first_pkt.dest;
    temp1_PRMA_q->pkt.pkt_num=c_req.first_pkt.pkt_num;
    temp1_PRMA_q->pkt.time_gen=c_req.first_pkt.time_gen;
    temp1_PRMA_q->next=NULL;
    if(first_PRMA_q!=NULL)
    {
     temp2_PRMA_q=first_PRMA_q;
     while(temp2_PRMA_q->next!=NULL) temp2_PRMA_q=temp2_PRMA_q->next;
     temp2_PRMA_q->next=temp1_PRMA_q;
    }
    else
     first_PRMA_q=temp1_PRMA_q;
      }
 or receive(call_trfc_gen callg)
 {
    c_trfc=callg;
if((temp1_PRMA_q=(struct PRMA_Q *)malloc(sizeof(struct PRMA_Q)))==NULL)
    { file1=fopen("ERROR","w"); fclose(file1); }
    temp1_PRMA_q->pkt.type=c_trfc.pkt.type;
    temp1_PRMA_q->pkt.source=c_trfc.pkt.source;
    temp1_PRMA_q->pkt.dest=c_trfc.pkt.dest;
    temp1_PRMA_q->pkt.pkt_num=c_trfc.pkt.pkt_num;
    temp1_PRMA_q->pkt.time_gen=c_trfc.pkt.time_gen;
    temp1_PRMA_q->next=NULL;
        if(PR_ALOHA)
    {
       if(first_PRMA_q!=NULL)
       {
        temp2_PRMA_q=first_PRMA_q;
        while(temp2_PRMA_q->next!=NULL) temp2_PRMA_q=temp2_PRMA_q->next;
        temp2_PRMA_q->next=temp1_PRMA_q;
       }
       else
        first_PRMA_q=temp1_PRMA_q;
    }
    else
    {
       for(j=1;j<=num_data_slots;j++)
     if((data_slots[id][j].status!='i')&&(data_slots[id][j].source==id))
        break;
           temp3_PRMA_q=first_PRMA_q;
       while(temp3_PRMA_q!=NULL)
       {
     if(temp3_PRMA_q->pkt.source==id)
       break;
            else
       temp3_PRMA_q=temp3_PRMA_q->next;
       }
       if((j>num_data_slots)&&(temp3_PRMA_q==NULL))
       {
```

```
 temp1_PRMA_q->next=first_PRMA_q;
 first_PRMA_q=temp1_PRMA_q;
  }
  else
  {
if((c_trfc.pkt.pkt_num>0)&&(!FIFO))
{
      temp1_PRMA_q->next=PRMA_q;
      PRMA_q=temp1_PRMA_q;
        }
    else if(PRMA_q!=NULL)
    {
     temp2_PRMA_q=PRMA_q;
     while(temp2_PRMA_q->next!=NULL) temp2_PRMA_q=temp2_PRMA_q->next;
     temp2_PRMA_q->next=temp1_PRMA_q;
    }
    else
     PRMA_q=temp1_PRMA_q;

if((FIFO)&&(!PR_ALOHA)&&(c_trfc.pkt.pkt_num>0))
{
 temp2_PRMA_q=PRMA_q;
 while(temp2_PRMA_q!=NULL)
 {
  if((temp2_PRMA_q->pkt.pkt_num==0)&&
     (temp2_PRMA_q->pkt.type==c_trfc.pkt.type)&&
     (temp2_PRMA_q->pkt.source==c_trfc.pkt.source)&&
     (temp2_PRMA_q->pkt.dest==c_trfc.pkt.dest))
          {
   if(temp2_PRMA_q==PRMA_q)
   {
    PRMA_q=PRMA_q->next;
    free(temp2_PRMA_q);
    temp2_PRMA_q=PRMA_q;
   }
   else
   {
    temp3_PRMA_q=PRMA_q;
    while(temp3_PRMA_q->next!=temp2_PRMA_q)
       temp3_PRMA_q=temp3_PRMA_q->next;
          temp3_PRMA_q->next=temp2_PRMA_q->next;
    free(temp2_PRMA_q);
    temp2_PRMA_q=temp3_PRMA_q->next;
   }
  }
        else
    temp2_PRMA_q=temp2_PRMA_q->next;
       }
     }
   }
 }
}
}
```



```
// COLLECT SIMULATION RESULTS

finalize
{
   fprintf(file1," Number of  Clusterhead  changes = %d \n", changes);
   fprintf(file1,"frame=%d, ch_changes=%lf, my_ch_changes=%lf\n",
   frame,(double)ch_changes/frame, (double)my_ch_changes/frame);
   fprintf(file1,"mean_no_of_clusters=%lf\n",mean_no_of_clusters);
   fprintf(file1," SOURCE AND DESTINATION INFORMATION\n\n");
   for(i=0;i<N;i++)
    for(j=0;j<N;j++)
       if((pkt_status[i][j].received!=0)||(pkt_status[i][j].dropped!=0))
          fprintf(file1,"(%d,%d)=%d pkts rcved(%d duplicate), %d dropped(%d
              dup,%d no_route_pkts,%d no_route_frames,%d loops),%d
              outofseq,%d idle\n",
              i,j,pkt_status[i][j].received,pkt_status[i][j].repeat_received,
              pkt_status[i][j].dropped,pkt_status[i][j].repeat_dropped,
              pkt_status[i][j].no_route_pkts, pkt_status[i][j].no_route_frames,
              pkt_status[i][j].idle);

    fclose(file1);

    }
}
```