Investigation of Mechanisms for Routing in Mobile Ad-hoc Network

Humayun Bakht

A thesis submitted in partial fulfilment of the requirements of Liverpool John Moores University for the degree of Doctor of Philosophy

School of Computing and Mathematical Science
Liverpool John Moores University
August 2006

ACKNOWLEDGEMENT

All praise is due to GOD the almighty who has showered his mercy and helped me in various forms thus made me able to complete my research program of study.

I would like to pay special tributes to my mother Rizwan Fatima Iqbal and all other family members in supporting me throughout my research degree program. I am also thankful to my brothers, sisters and other relatives specially Skinder Bakht and Afroze Bakht who given me all types of assistance and have directed me to accomplish the set goals.

I have no doubt about the expertise of my supervisors Prof Madjid Merabti and Dr Robert Askwith. The completion of this research program is a direct result of their efforts. They were always eager to provide me all types of necessary help wherever and whenever needed. From the beginning till the end of this research thesis I tried to be in touch with them on a regular basis to get direction of further progress. I received different research related instructions from Professor Madjid Merabti while my technical writing skills have greatly improved through the positive feedback from Dr Robert Askwith. Not only the time I have spent with them is memorable but also it helped me to develop research skills for further research in this area. I am therefore, from the depth of my heart thankful for their support and constructive instructions which made me able to complete my research.

I am also very thankful to all the staff members, other colleagues who encouraged me specially by attending monthly research seminars and with their advice. In particular the technicians team who provided me all necessary technical supports whenever it required.

At last but not least, I am also thankful to Professor David Gewritz, Chairman and Editor In-chief to ZATZ publishing USA. He encouraged me to write articles about different aspects related to my research area. It is a very useful experience in improving both technical and non technical writing skills and to introduce my self to the general audience of various backgrounds. Experience gained through articles writing boasted me writing more technical and non technical publications.

TABLE OF CONTENTS

LIST OF FIGURES

List of Tables

Appendix

Abstract

Mobile networks allow a more flexible model of communication than traditional networks. Unlike cellular mobile networks, mobile ad-hoc networks do not have any fixed communication infrastructure. For an action connection, the end host as well as the intermediate nodes can be mobile and therefore routes are subject to frequent disconnections. The goal of mobile ad-hoc networking is to extend mobility into the chain of autonomous mobile domains, where a set of nodes form the network routing infrastructure in an ad-hoc manner. Absence of a fixed infrastructure in mobile ad-hoc network poses a number of different types of challenges to this area. Typical challenges include limited bandwidth, power constraint, security, routing, hidden terminal problems and corroboration of mobile devices.

Routing in mobile ad-hoc networks is achieved through the mutual cooperation of mobile devices that form a chain of routes between two mobile nodes. Frequent topology changes, limited bandwidth and battery power of mobile nodes make routing a challenging issue in mobile ad-hoc networks. Internet protocols [109] are considered efficient to support routing in fixed mobile networks. However, current research shows their weak performance over mobile ad-hoc networks. Design of routing protocols for mobile ad-hoc networks have so far taken approaches similar to the routing protocols used in wired networks. Variations of link-states or source routing protocols or combinations of both have been reported. Most of these protocols are based on one of two techniques i.e. tables driven or on demand routing protocols [102]. Current literature [44] shows different weaknesses in the existing schemes.

Taking the routing problem as the basis, this project investigated routing mechanisms for mobile ad-hoc networks. This thesis proposed the Mobile ad-hoc on Demand Data Delivery protocol (MAODDP) as a routing solution for mobile ad-hoc networks. The key features of MAODDP is the establishment of route and data delivery simultaneously one after the other. MAODDP deals some of the routing related issues along side routing. It is designed to minimize reaction to topological changes and uses a combination of sequence numbers and broadcast ID to ensure the freshness of routes. MAODDP is loop-free, self-starting, and can scales to different sizes of mobile ad-hoc networks. MAODDP offers quick adaptation to dynamic link conditions, low processing and memory overhead, low network utilization, and determines routes to the destinations within the mobile ad-hoc network.

MAODDP functional model has been developed in Java and evaluated on the Scalable Wireless Network Simulator (SWANS). Seven different sets of experiments were conducted under various network environments. MAODDP found to be applicable in all types of network. The Protocol showed an impressive data delivery ratio with minimum consumption of available memory. MAODDP compared against AODV and DSR, where it performed well in comparison with each of these schemes.

List of Abbreviations

Acknowledge Message	ACK
Active Time	AT
Ad-hoc On-Demand Distance Vector Routing	AODV
Associatively Based Routing	ABR
Broadcast ID	B.ID
Code Division Multiple Accesses	CDMA
Core Extraction Distributed Ad-hoc Routing	CEDAR
Cluster Based Routing Protocol	CBRP
Clusterhead Gateway Switch Routing	CGSR
Destination Distance Sequence Vector Routing	DSDV
Destination IP Address	D.IP.ADD
Destination Sequence Number	D.S.NO
Distance Routing Effect Algorithm for Mobility	DREAM
Distributed Dynamic Routing Algorithm	DDR
Dynamic Source Routing	DSR
Greedy Parameter Stateless Routing	GPSR
Global State Routing	GSR
Global Positioning System	GPS
Fisheye State Routing	FSR
Hop Counter	H.COUNT
Internet Engineering Task Force	IETF
Internet Protocol Address	IP.ADD

Inter Zone Protocol	IERP
Intra Zone Routing Protocol	IARP
Medium Access Control Layer	MAC
Message Life Time i.e. Status Time	ST
Multicast Joining Request	MJReq
Open Shortest Path First Routing protocol	OSPF
Routing Information Protocol	RIP
Temporary Ordered Routing Algorithm	TORA
Topology Broadcast Based on Reverse Path Forwarding	TBRPF
Transmission Control Protocol	TCP
Route Error	RER
Route Request Data Delivery	RQDD
Signal Stability Routing Protocol	SSR
Source IP Address	S.IP.ADD
Sequence Number	S.NO
Source Sequence Number	S.S.NO
Landmark Routing	LANMAR
Listening Time	LT
Location Aided Routing	LAR
Wireless Routing Protocol	WRP
Witness Aided Routing	WAR
Zone Routing Protocol	ZRP

CHAPTER 1. INTRODUCTION

1.1 Mobile Ad-hoc Network

Ad-hoc is a Latin word, which means "for this or for this only." A mobile ad-hoc network is a collection of mobile nodes establishing a mobile ad-hoc or short lived temporary network in the absence of any fixed network[1]. In other words they are local area or personal area networks, with temporary plug-in wireless connections, in which some of the network devices are part of the network only for short duration of a communications session. Nodes in mobile ad-hoc networks are free to move and organize themselves in an arbitrary fashion. Each user is free to roam about while in communication with others. The path between each pair of users may have multiple links and the radio between them can be heterogeneous. This allows an association of various links to be a part of the same network.

1.1.1 Applications of Ad-hoc Networking

There are quite a number of uses for mobile ad-hoc networks [126]. A mobile ad-hoc network is suited to situations where an infrastructure is unavailable or to deploy one is not cost-effective. Some of the possible uses of a mobile ad-hoc network are some business environments such as in a business meeting outside the office to brief clients on a given assignment. Mobile ad-hoc networks can also be used to provide crisis management services applications, such as in a disaster recovery, where the entire communication infrastructure is destroyed and restoring communication quickly is crucial[4]. By using a mobile ad-hoc network an infrastructure could be set up in hours instead of weeks as is required in the case of wired line communication. Some of the common applications of mobile ad-hoc networks are as follows.

Conferencing

This scenario envisages a group of people gathering in the same area and exchanging shared information using the multi-hop characteristics of a mobile ad-hoc network. The alternative is to connect every node with a central network which at times might be unavailable or the overhead might be too costly when all that is required is the sharing of small amounts of data. Mobile ad-hoc networks under such situations could ease the job and provide a better solution to meet the user requirements.

Ubiquitous Computing

In future almost any type of devices will be able to join spontaneously in an established network and thereby exchange data with other devices within their close vicinity, in a transparent and seamless fashion[5]. Blue-tooth is an emerging standard to backup such a vision and is supported by some of the famous companies such as Ericsson Inc, IBM, Intel, Microsoft and Nokia. Bluetooth could be a suitable technology to establish a mobile ad-hoc network for transferring information without utilizing a fixed infrastructure.

Data Gathering

Another application where a mobile ad-hoc network can prove useful is the collection of data from remote areas. A network of sensors contains small, inexpensive, short-range radio transmitters that can collectively gather and forward data towards a base station. Current alternatives include either storing data for later collection or using large or expensive satellite transmitters. But for cases like animal tracking where small transmitters are essential, mobile ad-hoc networks can prove to be useful too, provided there is a reasonable coverage of the specific area with tracking devices.

Ad-hoc Sensor's Network

A mobile ad-hoc sensor network follows a broader sequence of operational scenarios, thus demanding a less complex setup procedure [6]. A mobile ad-hoc sensor or hybrid ad-hoc network consists of a number of sensors spread in a geographical area. Each sensor is capable of mobile communication and has some level of intelligence to process signals and to transmit data. In order to support routed communications between two mobile nodes, the routing protocol determines the node connectivity and routes packets accordingly. This makes mobile ad-hoc sensor networks highly adaptable so that they can be deployed in almost all environments.

Combat operations

Military operations are often spontaneous i.e. with little or no fixed network infrastructure[7]. Therefore, these operations require a communications solution which is spontaneous too. A mobile ad-hoc network can support the built-in geographical location by using an extremely accurate form of triangulation. In a mobile ad-hoc network, readings are faster than the geographical positioning systems because the soldiers don't have to wait for multiple satellites to acquire an integrated security.

A mobile ad-hoc network also allows devices to transmit at a lower output power to the neighbours. This benefits the overall network by lowering the probability of detection and by increasing the battery life of the participating devices. Therefore if the device is captured, the soldiers can list that device to maintain the integrity of the network.

Through anti-jamming mechanisms the soldiers in military operations are neither dependent on a single frequency nor are constrained to any military band. A mobile ad-hoc network is best suited to jamming

because noise can now be routed around problem areas. This could reduce the probability of jamming. That could make a mobile ad-hoc network a suitable choice for short term military operations.

1.1.2 Characteristics of Mobile Ad-hoc Network

A mobile ad-hoc network was defined with characteristics such as purpose-specific, autonomous and dynamic[8]. The control and management of mobile ad-hoc networks is distributed among the participating nodes as shown in Fig.2.13. Mobile devices in an ad-hoc network may have low CPU process capability and small memory sizes, thus affecting the capability of the mobile ad-hoc network to reach other devices. In the following section some of the important characteristics of mobile ad-hoc networks are discussed.

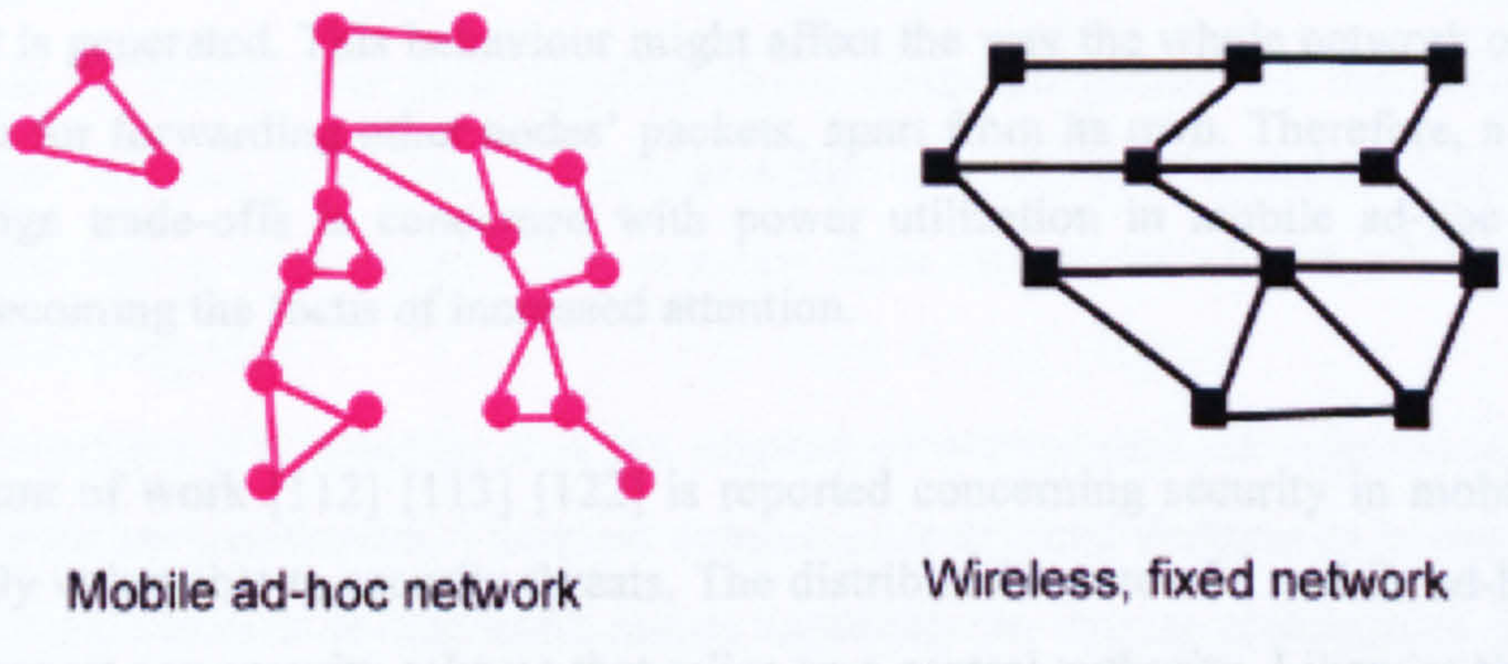


Figure 1.1. Node mobility in mobile ad-hoc network

Node Mobility

Node mobility introduces certain scalability problems in protocols for mobile ad-hoc networks [9]. When the network topology changes frequently control messages have to be sent between nodes so that new routes are found and propagated throughout the network. In such environments, it is reasonable to expect that when the topology changes happen there might be a short period where lots of control messages will propagate across the network to distribute the new destination paths. Therefore, the protocol designer should make provision for highly dynamic and fast adapting algorithms that minimise control messages and attempt to utilize long-lived routes to the maximum extent.

Control Messages

Information dissemination in mobile ad-hoc networks is different then other networks [103], moreover, Minimising control messages is also essential because of the additional load that they place in the bandwidth constrained wireless links. There is typically at least an order of magnitude difference between wired and wireless data rates with the former having a standardised 1 G-bit/sec in local Ether-net while the latter having in the best case, a nominal bit rate of 54 M-bit/sec. The techniques used to reduce control messages must

strike a balance between minimum amounts of messages by keeping network state information in each node and flooding the network each time the topology changes. The former has the side effect that in an ever-changing network topology stale routes will appear often, while when the latter is applied to a network containing nodes with high rates of mobility, it might result in control messages consuming all the available bandwidth. This could also lead to very slow network convergence where nodes contain either incomplete or out-of-date views of the network topology.

Battery Life

Saving battery power of the participating nodes in a mobile ad-hoc network is an important aspect to be seen. Mobile nodes in a mobile ad-hoc network rely on exhaustible means for providing energy i.e. batteries [11] [12]. For these nodes, energy conservation becomes an important design decision. Nodes with low battery power may decide to enter a power saving mode when they have nothing to send or until another high priority event is generated. This behaviour might affect the way the whole network operates, since each node is responsible for forwarding other nodes' packets, apart from its own. Therefore, a multitude of other problems and design trade-offs is concerned with power utilization in mobile ad-hoc networks and the particular area is becoming the focus of increased attention.

Security

Good amount of work [112] [113] [122] is reported concerning security in mobile ad-hoc network which can be highly vulnerable to security threats. The distributed nature of a mobile ad-hoc network makes it difficult to implement any security scheme that relies on a central authority. Likewise there is an increased possibility for eavesdropping, denial-of-service and malicious attacks. In addition to that a more secure architecture may be slower and more cumbersome. That makes security in a mobile ad-hoc network some what problematic; nevertheless, the decentralised architecture provides a more resilient approach to single points of failure.

1.1.3 Critical Ad-hoc Networking Features

The whole life cycle of mobile ad-hoc networks depends on a number of quantitative and qualitative features [16]. In the following section a list and description of each of these features is covered.

Quantitative Features: Quantitative features deal with all those aspects of mobile ad-hoc networks that are related with the network formation and maintenance.

- **Network Settling Time:** It is the time required for a collection of mobile nodes to automatically organize themselves and transmit messages reliably. The network settling time is one of the key factors in deploying or reorganizing a pre-established mobile ad-hoc network. It plays an important role in the

successful implementation of one of many sub-schemes to support different network control functions of a mobile ad-hoc network.

- **Network Join Time:** The time required for an entering node or group of nodes to become integrated into the mobile ad-hoc network is referred to as the network join time. This factor has its due importance in particular to assist mobile nodes forming mobile ad-hoc networks or for nodes to join an already established network. In general the shorter the join time to establish or to join the network could yield better network performance.
- **Network Depart Time:** The time required for the network to recognize the loss of one or more nodes and reorganize itself to route around the departed nodes. It is directly related to the overall efficiency of mobile ad-hoc networks. In short the quicker a network can reorganize after any changes in the network topology the better.
- **Network Recovery Time:** The time required for a collapsed portion of the network due to traffic overload or node failures, to become functional again once the load is reduced or the nodes become operational. Therefore, a network with low recovery time can produce better efficiency than those that take a longer recovering time.
- **Frequency of Updates (Overhead):** It is the number of control packets required in a given period to maintain proper network operation. The most common use of these features is in routing protocols[124] for a mobile ad-hoc network. It is a useful metric especially for routing protocols to determine the upper limit to maintain and support such schemes to assist in a smooth network operation.
- **Memory Requirement:** It is the storage space requirements in bytes including routing tables and other management tables. This feature enables a routing protocol to estimate efficiency of a routing algorithm by adjusting the total size of storage space. It could also give a complete insight as to what extent a routing algorithm can rely to maintain such information in a mobile ad-hoc network.
- **Network Scalability:** It is measured in terms of the number of nodes that the mobile ad-hoc network can scale to and reliably preserves communication. It is important especially for routing protocols to adjust various routing functions accordingly. Moreover, through this feature an upper bound of the maximum number of mobile nodes in a mobile ad-hoc network can be implemented. Such type of limit can be useful in expanding different routing mechanisms, schemes to reduce network overhead and to obtain an increased network data delivery.

Qualitative Critical Features: Qualitative features are related with the performance of the network. In this context most of these features focused on supporting different network operations. Short descriptions of these features are as follows.

- **Knowledge of Nodal Locations:** Does the routing algorithm require local or global knowledge of the network? Routing schemes of mobile ad-hoc networks rely on different mechanisms to obtain topology and routes information. There are some which require local information and others that require global information. How successful these algorithms are remained a question. However this feature could serve as a basic principle in an algorithm design and in its further development to address different routing issues of mobile ad-hoc networks.
- **Effect of Topology Changes:** Does the routing algorithm need complete restructuring or only incremental updates? This feature further explains how the routing algorithm handles the dynamic nature of mobile ad-hoc networks. In both cases where it depends on incremental updates or complete restructuring a number of different issues such as limited battery power and bandwidth constraint are involved. Thus this feature could possibly help in determining the effectiveness of a routing solution.
- **Adaptation to Radio Communication Environment:** Do nodes use estimated knowledge of fading, shadowing of multi-user interference on links in their routing decisions. The practical implementation of this feature varies from protocol to protocol. At present it can be observed that different protocols[17] or schemes rely on various mechanisms for this purpose.
- **Power Consciousness:** Does the network employ routing mechanisms that consider the remaining battery life of a node? This is an important factor as in general mobile ad-hoc networks suffer with limited battery power. Therefore measures should be taken by routing algorithms to preserve these resources.
- **Single or Multi-channel:** Does the routing algorithm utilize a separate control channel? In some applications, multi channel execution may make the network vulnerable to countermeasures. Besides other benefits, this feature can help routing algorithms in introducing several other mechanisms to establish secure transmission between various mobile hosts.
- **Bi-directional or Unidirectional Links:** Does the routing algorithm perform efficiently on unidirectional links e.g. if bi-directional links become unidirectional? A better routing efficiency is expected from a routing algorithm that operates on bi-direction basis.
- **Preservation of Network Security:** Do routing and MAC layer policies support the survivability of the network, in terms of low probability of detection, low probability of intercept and security? This feature is helpful in the implementation of security policy over mobile ad-hoc networks.
- **QoS Routing and Handling of Priority Messages:** Does the routing algorithm support priority messaging and reduction of latency for delay sensitive real time traffic? Can the network send priority messages/voice even when it is overloaded with routine traffic levels? This feature is linked with the

ability of the network such that how consistent the network is in delivering priority messages especially when it is overloaded.

- **Real-time Voice and Video Services:** Can the network support simultaneous real-time multicast voice while supporting traffic loads associated with situation awareness and other routine services? It is an important factor as mobile ad-hoc networks at present lack support for such services.

1.1.4 Problems, Constraints and Challenges of Mobile Ad-hoc Network

In the near future the next generation mobile system will include both infrastructure mobile networks and infrastructure-less mobile ad-hoc networks. Moreover, researchers [110] [111] also investigating the extenstion of mobile ad-hoc network to support wider area coverage. At present there are a number of different issues that restrict use of mobile ad-hoc networks at a wider level. Some of these constraints are as follows.

- **Dynamic Topologies**

The network topology of a mobile ad-hoc network may become very dynamic as mobility of nodes or membership of nodes is very random and rapid. There is a lot of work focused on routing protocols to support mobility is reported in [10]. This emphasizes the need for routing solutions to be dynamic. The network topology may change rapidly and unpredictably and the connectivity among the terminals may vary with time. The mobile nodes in the network dynamically establish routing among themselves as they move about; moreover a user in the mobile ad-hoc network may not only operate within the mobile ad-hoc network, but may require access to a public fixed network. The mobile ad-hoc network therefore should be able to adapt the traffic and propagation conditions as well as the mobility patterns of the mobile network nodes.

- **Bandwidth-constrained Variable Capacity Links**

It is expected that mobile ad-hoc network capacity grow with the area it cover [18]. However, at present wireless links will continue to have significantly lower capacity than their hardwired counterparts. In addition the realized data delivery of wireless communications after accounting for the effects of multiple access, fading, noise, and interference conditions is often much less than a radio's maximum transmission rate.

One effect of the relatively low to moderate link capacities is that congestion is typically the normal rather than the exception, i.e. aggregate application demand will likely approach or exceed network capacity frequently. A mobile network is often simply an extension of the fixed network infrastructure; it is expected that mobile ad-hoc users will demand similar services. These demands will continue to increase as multimedia computing and collaborative networking applications rise.

- **Energy-constrained Operation**

A major issue in ad-hoc networks is energy consumption since nodes are usually mobile and battery operated [29]. For these nodes, the most important system design criteria for optimisation may be energy conservation. For most of the light-weight mobile terminals, the communication-related functions should be optimized to save unnecessary power consumption. A mobile ad-hoc network poses different challenges for designing power efficient systems. Due to the absence of an infrastructure, each node in a mobile ad-hoc network also acts as a router. For a mobile ad-hoc network to exist, nodes have to be at least in the reception mode most of the time. A mobile ad-hoc network should be able to balance traffic load among nodes such that power constrained nodes can be put into a sleep mode while traffic is routed through other nodes.

- **Limited Physical Security**

A mobile network is generally more prone to physical security threats than a wired network . Therefore increased possibility of eavesdropping, spoofing, and denial-of-service attacks should be carefully considered. Existing link security techniques are often applied within wireless networks to reduce security threats. The disintegrated nature of network control in mobile ad-hoc networks provides additional robustness against the single points of failure of more integrated approaches. At present mobile ad-hoc networks do not adopt any standard security policy. However, research proposing a security model for mobile ad-hoc network is reported in [39].

- **Quality of Services (QoS)**

Quality of service refers to the ability of a network to provide a more reliable service to the selected network traffic. The inherent stochastic nature of communications quality in a mobile ad-hoc network makes it difficult to offer fixed guarantees on the services offered to a device. An adaptive quality of service must therefore be implemented over the traditional resource reservation to support the multimedia services in a mobile ad-hoc network.

In a mobile ad-hoc network, a number of different routes with various levels of node capacity and power may be available for a source to transmit data to the destination. However, not all routes are capable of providing the same level of quality of service that can meet the requirements of mobile users. Moreover, even if the selected route between a source and the destination meets the user requirements, the network error characteristics are expected to vary with time due to the dynamic nature of the mobile ad-hoc network. Therefore, providing quality of service levels in a constantly changing environment like a mobile ad-hoc network is a challenge issue.

- **Capacity of Mobile Ad-hoc Network**

A mobile ad-hoc network could be regarded as a small web of routers, or mobile terminals, providing packet-forwarding services to each other. A good amount of work [18] [42] concerning this issue has been done. It is known that larger the size of a mobile ad-hoc network, the lesser the data delivery that may be obtained from the network. Likewise, it is said that high mobility in a mobile ad-hoc network could be a means of getting better data delivery in some situations. In mobile ad-hoc networks, an intermediate node may receive the same transmission many times during an active session through different channels. In most of the cases, a multi-hop path is required for a source node to transmit data to the destination node. This concurrent wireless transmission results in a decrease in the capacity of the mobile ad-hoc network.

- **Service Awareness in Mobile Ad-Hoc Network**

Service discovery plays an important role in present wireless communication systems [19]. In mobile ad-hoc networks all clients' services are provided through the mutual agreement among the participating nodes. Unlike mobile networks, the central registry cannot be found in advance in a mobile ad-hoc network; therefore, finding the location of the central registry becomes the primary concern in the service discovery mechanism. In mobile ad-hoc networks, mechanisms like service advertisement and service query have been used as solutions to this problem. However, the dynamic nature of mobile ad-hoc networks makes service advertisement and service query process a difficult task.

- **Security in Mobile Ad-hoc Network**

Attacks against mobile ad-hoc routing protocols can be classified as active or passive. A passive attack does not disrupt the operation of the protocol, but tries to discover valuable information by listening to traffic. An active attack injects arbitrary packets and tries to disrupt the operation of the protocol in order to limit the availability, gain authentication, or attract packets destined for other nodes. Different security mechanisms have been considered for mobile ad-hoc networks. However, at present this area lacks a standard security policy.

- **Group Communication in Mobile Ad-hoc Network**

In a mobile ad-hoc network group communication issues differ from those in a wired network . It is due to the variable and unpredictable nature of the wireless medium, where the signal strength and propagation varies with the time and the environment. In a mobile ad-hoc network, an efficient group communication model can ease effective communication among various groups in the network. At present, multicasting routing [20] [41] [115] is gained by adopting one of two approaches: flooding and tree-based routing. Flooding offers the lowest control overheads with very high data traffic, while tree-based routing reduces data traffic in the network but requires many control data exchanges. Study shows less efficient

performance of these techniques in mobile ad-hoc network. That makes multicasting a challenging issue for mobile ad-hoc networks.

- **Routing Misbehaviour in Mobile Ad-hoc Network**

Routing misbehaviour is one of the interesting issues in the context of a mobile ad-hoc network. There are situations when a node becomes 'selfish' [13] and refuses to act as an intermediate node in an active communication session. This issue requires a solution where a selfish node could be identified and thereby damages to the entire network operation could be reduced.

- **Distributed Management Services in Mobile Ad-hoc Network**

Information access and sharing is difficult in a mobile ad-hoc network because of their dynamic nature, scarce resources, and heterogeneous user devices. These issues also make middleware services unsuitable for synchronous communication because they are too vulnerable to communicate disruptions. Therefore, a mobile ad-hoc network needs an effective distributed management solution to perform various tasks under different scenarios.

- **Routing in Mobile Ad-hoc Network**

Routing in a mobile ad-hoc network is achieved through the mutual cooperation of mobile devices that form a chain or route between two mobile nodes. Existing Internet protocols [109] are considered unsuitable for routing in mobile ad-hoc networks [44]. Thus the developments of efficient routing strategies for mobile ad-hoc network are required.

1.2 Scope and Objectives

Routing forms the basis of network formation. The current protocols for mobile ad-hoc networks can generally be categorized into two groups: pro-active and re-active protocol types. Pro-active protocols[23] follow an approach similar to the one used in wired routing protocols. By continuously evaluating the known and attempting to discover new routes, pro-active protocols try to maintain the most up-to-date view of the network. This allows them to efficiently forward packets as the route is known in advance before the transmission. In contrast reactive protocols[25] [28] determine the proper route only when required, that is, when a packet needs to be forwarded. In this instance, the node floods the network with a route-request and builds the route on demand from the responses it receives. This eliminates the need to maintain a consistent overview of the network.

Recent research shows a number of different problems in most of the existing protocols. Some of the main problems include limitation: most of the well known protocols in this area are limited to particular scenarios i.e. do not perform well in all environments; lack of analytical studies: insufficient work has been done to evaluate their performance with respect to the other techniques of similar types. Moreover, proposed

schemes focus on routing without considering their effects on some other issues. Taking routing in mobile ad-hoc networks as one of the main research topics this thesis defined the following scopes and objectives.

- The specification and design of a novel routing algorithm for mobile ad-hoc networks. This algorithm should takes into account bandwidth consumption, faster and reliable data packet delivery; reduced network overhead and battery power of the participating nodes.

- The design and implementation of the proposed routing algorithm in a simulation framework.

- Verifying the correctness in the implementation of the simulated algorithm by providing a set of tests and interpreting the results. This is different from the evaluation of the performance of the algorithm in that we are trying to verify that individual modules of the algorithm work as intended.

- Evaluate the performance of the proposed algorithm by constructing different mobility scenarios with varying parameters such as the number of nodes, their mobility patterns, velocity and transmission range and test the protocol under these scenarios. The data generated by these test rounds would then be aggregated and suitable analysis should be conducted. It would also be interesting to apply this routing algorithm to the same kind of scenarios as other mobile ad-hoc routing protocols and make a comparative analysis in terms of efficiency, overhead, bandwidth consumption, faster and reliable data delivery and power consumption.

- Simulation-based comparison of the the proposed algorithm against some of the dominant routing schemes of mobile ad-hoc networks. This will give a clear understanding of the protocol's efficiency against selected schemes. Moreover, it will also verify the proposed algorithm's benefits over other schemes in a simulation environment.

1.3 Novelty of the Work

This thesis investigated routing mechanisms for mobile ad-hoc networks and proposed Mobile Ad-hoc On-Demand Data Delivery Protocol (MAODDP) as a routing solution for a mobile ad-hoc network. The most important novel aspect of the work is the integrated approach [27] to solve a range of routing problems in the mobile ad-hoc networking domain. Unlike other protocols which typically focus on the core delivery problem or a subset of related issues such as security, our protocol addresses a range of issues in an integrated way. Some of the novel features of MAODDP are as follows.

- Integrated approach: Our research identified different issues as interrelated with routing. MAODDP deals with some of those issues alongside routing. Research findings[27] concluded that an effective routing solution could be established if some other routing related problems i.e. bandwidth, battery power and security, were dealt with alongside routing. In most of the existing routing techniques [23][24] these factors have not been addressed fully. Research shows [40] weak performance of existing techniques on addressing these issues.
- Simultaneously route establishment and data delivery: MAODDP focuses on simultaneous route establishment and data delivery one after the other. In table driven protocols continuous network updates could result in unnecessary network overhead, thereby resulting in reduction of network efficiency [40]. Likewise too many query packets prior to data delivery could yields the same effect [34]. MAODDP neither requires any regular updates nor depends on route establishment prior to the data delivery.
- Power aware operation: MAODDP conserves battery power of mobile nodes with its own power saving mechanisms. Mobile nodes are allowed to switch in between one of two modes i.e. sleep state or active state. Nodes are required to be awake only during an active transmission and they are allowed to go into sleep mode if they are not the receivers or the senders of MAODDP packets after a particular time interval. Moreover with the addition of a specific listening time (LT) each node can switch back into listening mode after a time period of (LT). Once switched back into active mode if the node does not find itself in any active transmission it can switch back into sleep mode. This sort of strategy could be very helpful in reducing unnecessary power consumption.
- Security: A mobile ad-hoc network does not have any standard security policy. This could possibly lead active attackers to easily exploit or possibly disable a mobile ad-hoc network. Apart from some of the protocols [88] which were designed as secure protocols, Initial specification of some of the famous protocols including DSDV[23][84] and AODV[25][117] are silent over security issues. Unlike these schemes, MAODDP offers secure routing and has its own secure mechanisms.
- Selection of Shortest route: MAODDP adopts a totally new strategy for selecting the shortest and the best path from a source to a destination. A hop-counter has been introduced inside the route query and data delivery (RQDD) packet headers. Whenever a source node broadcasts RQDDs it sets the hop-counter value to zero. This counter will increase by one from the previous value each time it reaches any intermediate node before it reaches the intended destination. Each intermediate node then automatically becomes aware of the exact hop-counts from itself to the destination of interest. This information could also be used in any future communication.

- Guaranteed Data Delivery. Destination sequence distance vector routing[23] and ad-hoc on-demand data delivery routing protocol[25] are considered as some of the main and the earliest protocols of table driven and on-demand type protocols respectively. DSDV[23] suggests a straightforward mechanism to establish routing but does not guarantee anything about the actual data being sent[34]. Similarly AODV[26] introduced a new way of establishing routing in mobile ad-hoc networks but its specification is silent on this issue[34]. MAODDP introduced a special mechanism of broadcasting acknowledge packets by the destination node destined for the source node of RQDD. In this regard status time plays an important role in finding out the message status i.e. if the transmission failed in the last attempt.
- Support of real time application: Mobile nodes in MAODDP do-not need to wait for the route establishment prior to data transfer, thus MAODDP could offer support to real time communication.
- Bandwidth conservation: The MAODDP approach is helpful in reducing bandwidth consumption. Unlike [43] which is designed specifically to address bandwitdth consumption, In comparison with MAODDP, existing routing techniques[23][24][25] show no specific procedure to save unnecessary bandwidth consumption. Table driven protocols[23] due to their specific routing approach consume more bandwidth. Similarly in on-demand protocols [25] [125] with too many query packets, route replies could result in the same [34]. Moreover issues like network congestion and network speed are related with the available bandwidth. A network with higher available bandwidth could result in better performance and less chance of network reduction. MAODDP addresses most of these deficiencies in the available schemes.
- Implementation and Evaluation: Several protocols have been citied in the available literature. Besides their technical classification, these schemes can also be categorized into two main types i.e. those that have been implemented and evaluated and those without implementation and evaluation. In this regards, MAODDP outclassed all such schemes that have not been implemented and evaluated. Evaluation experiments resulted in higher message activities with an accumulated average data delivery of 79.07%. Moreover, MAODDP conserved an accumulated average of 67.50 % available memory. A separate set of experiments was conducted to measure effectiveness of the MAODDP power aware mechanisms. Findings showed that almost 22 % of available memory was saved under power saving mode with an increased data delivery of 5%. MAODDP was compared against AODV[25] and DSR[28]. It was found 2.52 times and 7.92 times more effective in conserving memory and handling active data packets than AODV. A separate set of experiments revealed that in

a varying mobility environment, MAODDP perform better than DSR with 52.5 % more data delivery than DSR.

1.4 Structure of Thesis

A set pattern was followed to accomplish the above mentioned objectives and goals. The background of this thesis mainly relied on the available literature which was also helpful to design and achieve the final objective of our investigation. The thesis has been organized in the same pattern. Following is a brief introduction to each of the chapters and structure of the thesis.

In chapter 2 an introduction to routing, design objectives and different metrics are given. The main aim is to introduce some of the main Internet protocols and highlight how some of these approaches were followed in mobile ad-hoc networks. Six different types of protocols for mobile ad-hoc networks are cited in the available literature. In this chapter an introduction to each of these types and related protocols are also covered.

Chapter 3 presents a comprehensive critical study of existing routing protocols of mobile ad-hoc networks. Thirty one protocols falling under various categories introduced in chapter 2 are critically evaluated. This chapter has been divided into two different sections. Section one critically evaluates all those protocols that are currently under review by IETF. In section 2 protocols which followed the closest approach to MAODDP are evaluated before others. For the sake of clarification critiques have been provided in a separate subsection of the main section of each scheme. This chapter provides a basis of our project work and highlights various weaknesses in the existing work.

In chapter 4 the specification of Mobile Ad-hoc On-Demand Data Delivery Protocol (MAODDP) is given. Different terminologies, protocol functions and various features of MAODDP are parts of this chapter. Brief comparisons with some other techniques are also provided alongside various operations. That is to compare various functions of MAODDP with respect to some other techniques of this area.

Chapter 5 presents implementation processes of MAODDP. The protocol functional model has been developed in JAVA. This functional model was later evaluated in a simulation environment i.e. Scalable Wireless Network Simulator (SWANS). Different standard classes and classes related to protocol implementation are explained inside this chapter.

Chapter 6 presents evaluation results. In this chapter a brief introduction to the selected simulation framework for the simulation study and its benefit over other simulation software are also covered. In total seven different sets of experiments each comprising nine tests with various simulation parameters and under different simulation environments were conducted. This chapter contains results of different experiments and

graphical representation of various results alongside general results charts of each different set of experiments.

Chapter 7 presents a comprehensive comparison study of MAODDP against some of the famous routing protocols. MAODDP was compared against AODV [25] and DSR [28] via simulation experiments. DSDV and DSR have not been implemented under SWANS therefore for the former a theoretical comparison has been done and for the latter results from some other study are used.

Chapter 8 concludes the research findings and further work. The main aim of this chapter is to provide a summary of all the work that has been done, goals that have achieved and to highlight some of the possible future works.

CHAPTER 2. ROUTING PROTOCOLS AND TECHNIQUES

2.1 Introduction

A mobile ad-hoc network has different routing requirements than a mobile fixed network. Unlike a fixed network where various hardware are used to route packets from one station to another station , a mobile ad-hoc network relies on mobile nodes to route packets in the network. It has been mentioned before that there are some factors which directly affect routing mechanisms of a mobile ad-hoc network. However, as a matter of fact routing protocols [23] [24] for mobile ad-hoc networks follow the same basic design strategies as in fixed networks.

In this chapter an introduction to routing, routing design objectives and different metrics are covered. This chapter also highlights network protocols alongside descriptions of some of the famous Internet protocols. At present protocols for a mobile ad-hoc network could be classified into one of six types [104]. An introduction to each of these types will be given. The main focus of this chapter is to extend the relation between protocols for fixed and mobile ad-hoc networks and to highlight different routing protocol types for mobile ad-hoc networks.

2.2 Routing

Routing is the process of information exchange between two hosts in a packet switched network [44]. It involves two basic activities, determining optimal routing paths and transporting packets through a network. Routing is achieved via routing protocols. Routing protocols use different metrics to evaluate the path best suited for a packet to travel. A metric is defined as a standard of measurement, such as path bandwidth, that is used by routing algorithms to determine the optimal path to a destination. Most of the routing algorithms initialise and maintain routing tables, which contain route information . Route information varies depending on the routing algorithm used.

Routing algorithms fill routing tables with different routing related information. Destination or next hop associations tell a router that a destination could be reached by sending the packet to a particular router representing the next hop on the way to the final destination . When a router receives an incoming packet, it checks the destination address and attempts to associate this address with a next hop. Routers compare metrics to determine optimal routes, and these metrics differ depending on the design of the routing

algorithm used. The similar design principle has been followed by table driven protocols for mobile ad-hoc networks.

Conventional networks use routers to communicate with one another. Maintaining routing tables is an issue which differs between fixed networks and mobile ad-hoc networks. In conventional networks routers transmit different routing related information to maintain an updated routing table. The similar schemes were followed [23] in mobile ad-hoc networks. However, each of these schemes differs on the amount of information that is included inside update messages. Link state advertisement is another example of a message sent between routers to inform others about the state of the sender's links. Link information can also be used to build a complete picture of network topology for routers to determine optimal routes to the network destinations.

Switching algorithms are another type of routing algorithm. In switching algorithms, a host determines a router's address if it has a packet to send. The source host sends a packet addressed specifically to a router's physical address i.e. medium access control layer (MAC) address along with the protocol network layer address of the destination host. When the packet reaches to the router, the router examines if it has the information of the next hop, if it finds one it forwards this packet otherwise it discards the packet.

This approach is very similar to on-demand protocols [25] of mobile ad-hoc networks. These protocols have some differences in comparison with the switching algorithm for conventional networks. Due to the nature of mobile ad-hoc networks and the way they are normally formed on-demand protocol depends on some extra functions. An example of one of such function is route error messages. Storage of information is another factor which differs from protocol to protocol. Some of these protocols [23] [25] use routing tables while others [28] focus on caching related address .In this context, some routing protocols of mobile ad-hoc networks are based on the division of the entire network into intra-domain and inter domain end systems. In simple intra domain end systems are those end systems that can communicate within routing domains. While inter domain end systems are those end systems that communicate both within and between routing domains.

2.2.1 Routing Algorithms Design Goals

Routing protocols tend to follow different design approaches which can best suit the requirements of a specific network. Some of these design principles are as follows.

- Optimality
- Simplicity and low overhead
- Robustness and stability
- Rapid convergence
- Flexibility

A routing algorithm should be able to offer the best possible route. This feature of a routing algorithm is known as optimality. Routing algorithms use different metrics and metric weightings to calculate the best route. In mobile ad-hoc networks different routing protocols uses various metrics for such calculations.

The routing algorithm should capable of offering its functionality efficiently with a minimum of software and utilization overheads. Efficiency of a routing algorithm has due role in particular when the software implementing the routing algorithm has to run on a computer with limited physical resources. It is an important factor especially in the context of mobile ad-hoc networks which suffer with low network resources. Protocols for mobile ad-hoc networks follow different strategies to implement this factor. There are some protocols [12] [13] which have been specifically designed to reduce energy consumption. Likewise, some others focus on saving the limited bandwidth [11][43].

Another factor which should be taken into account during the design stage of a routing algorithm is robustness. Routing algorithms should be capable of addressing issues such as hardware failures, high load conditions, and incorrect implementations. Among these issues the most important for mobile ad-hoc networks is high load condition. Hardware failures of mobile nodes would not necessarily damage the entire network but could have some limited effects. Issues like high load conditions have a direct impact on the performance of mobile ad-hoc networks. At present most of the routing protocols for mobile ad-hoc networks pay less attention to address high load conditions.

Convergence is the process of agreement by all routers on an optimal path. Routing algorithms that converge slowly can cause routing loops. Therefore routing algorithms with fast network convergence are considered good routing algorithms. Convergence is particularly important in the context of mobile ad-hoc networks where topology changes happen frequently and unexpectedly. Routing protocols of mobile ad-hoc networks use different mechanisms [23][25] to ensure an alternative route is available in case if a previously known route becomes unknown, expired or broken.

A final design factor is flexibility. Routing algorithms are expected to be flexible. It is expected that they should react quickly and accurately to handle various network circumstances. Due to various constraints associated with mobile ad-hoc networks flexibility is an important factor. It is expected that routing algorithm should be flexible enough to address routing in mobile ad-hoc networks

2.2.2 Algorithm Types

Routing algorithms for mobile ad-hoc networks extract various characteristics from those for fixed networks. Their classification into different types is based on two factors i.e. individual approach and design factors. In this section a brief explanation of different types of algorithms is presented. These algorithms wherever appropriate have been referenced with the related schemes of mobile ad-hoc networks .A list of different algorithm types and their details is as follows.

- Single-path versus multi-path
- Flat versus hierarchical
- Host-intelligent versus router-intelligent
- Intra-domain versus inter-domain
- Link-state versus distance vector

Static versus Dynamic

Static routing algorithms are table mapping algorithms and are established by an administrator. One major weakness in static routing algorithms is their inability to respond to network changes. Dynamic routing algorithms can adjust network circumstances via incoming routing update messages. Therefore these routing algorithms are considered stable in comparison with static routing algorithms.

Dynamic Routing Algorithms

Dynamic routing algorithms e.g.[28][66] are considered as interoperable with static routes. These algorithms can be allocated to act as a repository for all un-routable packets. Therefore they ensure that all messages are at least addressed in some manner.

Single-Path versus Multi-path

Unlike single path algorithms, some routing protocols [46] [47] support multiple paths to the same destination. These multi-path algorithms allow traffic multiplexing over multiple lines and can offer better data delivery and reliability.

Flat Vs Hierarchical

Some existing routing algorithms are operated in a flat space, while some others use routing hierarchies [87]. In a flat routing system the routers are peers of all others. In a hierarchical routing system [51] [72], some routers form a routing backbone. Packets from non-backbone routers travel to the backbone routers, where they are sent through the backbone until they reach the general area of the destination. At this point they travel from the last backbone router through one or more non-backbone routers to the final destination.

Hierarchical routing mimics the organization of most companies and thus supports their traffic patterns efficiently. In hierarchical routing intra-domain routers need to know only about other routers within their domain. This algorithm can further be simplified depending on the routing algorithm being used and the routing update traffic can be reduced accordingly.

Host-Intelligent Vs Router-Intelligent

Some routing algorithms use source routing in which the source node determines the entire route [25], while other algorithms assume that hosts have no information about routes. In these algorithms routers determine the path through the network based on their own calculations.

Intra-domain vs. Inter-domain

Some routing algorithms work only within domains, others [53] [54] work within and between domains. Their nature of operations differs from each other. This further emphasizes that an optimal intra-domain routing algorithm would not necessarily be an optimal inter-domain routing algorithm.

Link-State vs. Distance Vector

Link state algorithms [11] are based on link-state strategy flooding routing information to all other nodes. Each router sends only the portion describing the state of its own links of the routing table. In distance vector algorithms such as in Bellman-Ford algorithms each router sends all or some portion of its routing table but only to its neighbours. Therefore, distance vector algorithms know only about their neighbours. Link-state algorithms cover more quickly therefore they are less open to routing loops and are considered more scalable than distance vector protocols. However, link state algorithms require more CPU power and memory and can also be more expensive to implement and support.

2.2.3 Routing Metrics

Routing algorithms uses routing information to calculate routing metrics. These metrics are used to calculate the best possible route to a destination. Information to calculate such metrics varies from one algorithm [23] to the other [25]. In this regards some routing information for mobile ad-hoc networks differ from those in fixed networks. Following is the list of the information used in metrics calculation both in fixed and mobile ad-hoc network.

- Path length
- Reliability
- Delay
- Bandwidth
- Load
- Communication cost

Path length is one of the most common routing metrics. Routing protocols for fixed networks allow network administrators to assign arbitrary costs to each network link. In such situations path length is the sum of the costs associated with each link traversed. Protocols of mobile ad-hoc networks [23] [25] tend to follow a different approach for such calculation. An example [23] is the use of a hop-count for such calculation. The hop count is the number of passes through internetworking devices a packet passes through between a source and a destination.

Reliability is the bit-error rate of each network link. It is the ratio between the numbers of links that can recover quickly against the number of links which take time to recover after a network failure. It differs from one networking environment to the other.

Routing delay is the length of time required to move a packet from source to destination in a network. It depends on available bandwidth and the physical distance between the source and the destination and the processing time at each route.

Bandwidth is the available traffic capacity of a link. A higher amount of available bandwidth has many benefits but cannot guarantee a better route for data transmission.

Load is the degree to which a network resource such as a router could be busy. It can be calculated in different manners such as CPU utilization and packets processed per second.

Communication cost depends on the type of network hardware. It is considered as an important metric for all networks.

2.3 Network protocols

Routed packets are transported by the routing protocol across the network. These network protocols perform different functions required for routing which vary among protocol suites. Network protocols occur at the upper five layers of the OSI reference model [62]; the network layer, the transport layer, the session layer, the presentation layer and the application layer. In this section a brief overview of some of the famous Internet protocols is presented.

2.3.1 Open Shortest Path First Routing protocol (OSPF)

OSPF [60] is a link state or shortest path first protocol which is classified as a dynamic, adaptive protocol. It can adjust to the problems in the network and provides short convergence periods to stabilize the routing table. OSPF supports subnet addressing and type of service (TOS) routing. Routing decisions of OSPF are based on one of two fields of IP datagram i.e. the destination IP address and the TOS. Once the decision is made on the IP datagram, the datagram is routed without additional headers.

OSPF Operations

Each router stores different in a directory known as database. Some of this information is interfaces at the router that are operable, status information about each neighbour to a router and topology of the network with a directed graph. Route broadcast this information on periodic basis. A router in OSPF considers itself as a root node and calculates the shortest path to all the other routers in the autonomous system. Each cost metric for each TOS is calculated separately. If the two paths have equal value OSPF will distribute the traffic equally on these paths.

OSPF [60] supports one to many routing and offers its own security mechanism by including authentication mechanism. Therefore, routers must go through some simple procedure to authenticate the traffic between them.

OSPF works with directed graphs which contain values between two points i.e. network or gateway. These values are the weighted shortest path value with the router established as the root. Therefore the router

calculates the shortest path tree from itself to any other point in the Internet. This calculation specifies the next hop to the destination in the hop-to-hop forwarding process. Information received via periodic advertisement from their neighbours is used for such calculations. Different methods known as external routing are used to obtain routing information from outside autonomous system. OSPF uses some other protocol such as EGP for communication outside autonomous systems. One other possibility is to use static routes between the autonomous systems.

OSPF supports two types of external capabilities. In type 1 the external metrics are the same as the internal OSPF link state metric and in type 2 the external metrics only use the cost of the router to the external autonomous system. Of these two types, type 2 follows a simple method. It assumes that the routing between autonomous systems is the major cost of routing the packet which in some actual operation not true. This approach removes conversion between internal link state metrics of OSPF and their external costs.

2.3.2 Routing Information Protocol

The Routing Information Protocol, or RIP [63], is one of the famous Internet protocols. It is also an easily confused protocol due to the existence of a variety of RIP-like routing protocols. Moreover these protocols are based on the same set of algorithms that use distance vectors for various calculations.

Routing Updates

RIP updates are broadcast at regular intervals and when the network topology changes. An update message which includes changes to an entry is added inside the routing tables. Routers only maintain the route with the lowest metric value to a destination. Once the routing tables are updated the router transmits this information to all other routers via special updates.

RIP Routing Metric

A single routing metric or hop count is used to measure the distance between the source and a destination network. Each hop in a path from source to destination is given a hop count value which is typically 1. For all updates with new or changed destination network entry is received at router. The router adds 1 to the metric value indicated in the update and enters the details in the routing table. The IP address of the sender is used as the next hop.

RIP Stability Features

Looping is avoided in RIP through implementation of a limit on the number of hops allowed in a path from the source to a destination. The maximum number of hops in a path is fifteen. Any destination which exceeds this limit is considered as unreachable. However this stability feature limits the maximum diameter of a RIP network to less than 16 hops. RIP also implements the split horizon and hold-down mechanisms to prevent incorrect routing information from being propagated.

RIP Timers

RIP regulates its performance via numerous timers. Three different timers are used for various purposes. The routing-update timer sets the interval between periodic routing updates. It is helpful to prevent congestion which could result from all routers simultaneously updating their neighbours. Attached with routing tables the route timeout timer is another important timer used to mark routes invalid as it expires. However, the route timeout timer is retained in the table until the route-flush timer expires.

2.4 Transmission Control Protocol and Mobile Ad-hoc Network

Due to different routing requirements of mobile ad-hoc networks, Internet protocols are considered less efficient for such an environment. TCP/IP [21] is a window based protocol which enables two hosts in a network to form a connection and exchange streams of data. TCP is a connection oriented protocol which guarantees data delivery and the delivery of packets in the same order in which they were sent. Different benefits are possible through such schemes for fixed networks where per-flow bandwidths can go up to several megabits per second. This design principle can also be used to avoid maintenance of any fine-grained timers on a per-flow basis.

A significant amount of work [22] [107] has considered TCP support for mobile ad-hoc networks. However at present TCP lacks this functionality. In general, the dynamic nature of mobile ad-hoc networks makes it harder for Internet protocols to support routing at different layers of the protocol stack. Internet protocols are tailored for wireless environments at the medium access control layer, where carrier sense multiple access with collision avoidance can easily affect the performance of these protocols.

Wireless networks suffer from high link error rate which TCP often misinterprets as packet loss caused by link congestion. TCP as a result reduces wireless network efficiency by invoking congestion controls mechanisms. In mobile ad-hoc networks where route changes occur quite frequently this problem becomes worse. Route failures cause packet drops during an active transmission at the intermediate nodes. This packet drop is often misinterpreted again as congestion loss by TCP. Moreover, route changes can also introduce frequent out of order delivery which affects the TCP control mechanism. Moreover, use of a window based transmission mechanism such as TCP in mobile ad-hoc networks normally results in the problem of burstiness in packet transmission.

TCP is based on a slow start mechanism during connection initiation and when it recovers from heavy congestion in the network. A slow start mechanism uses an exponential increase of the congestion window size. The increase mechanism is still non-aggressive by design as it can take several route reply periods before a connection operates on its true available bandwidth. It could decrease data stream rate in mobile ad-hoc networks due to the overhead associated with the request-to-send, clear-to-send and acknowledge

packets[22]. These packets are used by the CSMA/CA protocol sent by TCP from the destination to the source. Similarly if forward and reverse paths are the same the acknowledge traffic in the reverse direction will contend with the data stream on the forward path. This results in the reduction of the rate of data stream and could slow down the mobile ad-hoc network.

2.5 Classification of Routing Protocols for Mobile Ad-hoc Network

Variations of tables driven and on-demand protocols have been proposed for mobile ad-hoc networks. These protocols follow different approaches using various design patterns and routing metrics. On the basis of their individual characteristics these protocols can be classified into one of the following six types.

- Single Channel vs. Multichannel protocols
- Uniform Vs Non-Uniform protocols
- Hierarchical Topology /Clustered based routing
- Position based protocols
- Proactive vs. on Demand routing protocols
- Hybrid protocols
- Full vs. Reduced topology information

2.5.1 Single Channel vs. Multichannel Protocols

Single channel protocols use one shared channel for communication and multi-channel protocols rely on Code Division Multiple Accesses (CDMA), Frequency Division Multiple Access (FDMA), or Time Division Multiple Access (TDMA) to establish communication channels. In multi-channel protocol channels are assigned by a distinguished controlling station. Some of the protocols that fall in this categories are Cluster-head Gateway Switched routing(CGSR) protocols , Associativey Based Routing (ABR) [64] [100], Cluster Based Routing Protocol (CBRP) [73], Distance Routing Effect Algorithm for Mobility (DREAM) [47], Destination Sequence Distance Vector Routing (DSDV)[40], Distributed Spanning Tree Protocol (DST) [58], Flow oriented protocol (FORP) [56], Fisheye State routing (FSR) [24][45], Geographic Distance Routing (GEDIR)[79], Global State Routing (GSR) [65], Lightweight Mobile Routing (LMR) [88, 89], Topology Broadcast Based on Reverse Path Forwarding (TBRPF) [59][68], Witness Aided Routing (WAR) [61] and Wireless Routing Protocol (WRP) [69].

2.5.2 Uniform Vs Non Uniform

A uniform protocol does not assign any special roles to any node. In a non-uniform protocol some nodes may be assigned a special role which needs to be performed in a distributed fashion. Typically clustering protocols are non-uniform. An example of non uniform protocols is the Cluster Based Routing Protocol (CBRP) [73]. It forms clusters and thus requires cluster-heads which are distinguished nodes.

Cluster-head Gateway Switched routing protocols additionally define gateway nodes which form a backbone type network and requires a special role for the nodes which are part of the core. DST[58] also creates a backbone on the stable regions of the network. Likewise HSR forms clusters as in CBRP [73] and CGSR however in HSR there are no gateway nodes but multilevel clusters and cluster-heads. LANMAR [70] requires landmark nodes for each group of nodes while OLSR [93] require the selection of MPR nodes which has also a special role.

2.5.3 Hierarchical Topology/Cluster Based Routing

Clusters in a hierarchical topology or cluster based routing [73] are usually represented by a dedicated node known as the cluster-head. These nodes form the cluster and attached nodes use the cluster head to describe their association with a specific cluster. Clusters can also be formed hierarchically such that there are multiple layers of clusters. The cluster-heads manage communications within a cluster and are informed about joining and leaving nodes. In addition to cluster-heads gateway nodes are also suggested in CGSR and in HSR. These gateway nodes transmit information from one cluster to the other and therefore may be part of more than one cluster. Since cluster formation and selection of cluster-heads is usually a significant effort in terms of signaling traffic as is the removal and addition of nodes from and to a cluster. Cluster-stability has become one important aspect of clustering algorithms.

Clustering in general suffers from some drawbacks especially with very stable clusters. Since the cluster-head and also the gateway nodes have to do the routing and managing work they can easily create a bottleneck. The communication load will certainly be higher for a cluster-head or a gateway node than for an ordinary node thus consuming more energy which can lead to an early outage of these nodes due to the limited battery power.

There are some other hierarchical properties that should be taken into account in this class. Some protocols like FSR [24] and DREAM [47] introduce a set of scopes for routing information. In any of these protocols, close, fast moving nodes receive more information more frequently than others. Further there are routing protocols, which use different routing strategies, depending where and how far a packet has traveled, like zone routing protocols.

Some of the routing protocols that use clustering or a hierarchical structure is as follows. The Cluster routing protocol (CBRP) [73] defines clusters and cluster-heads; the Fisheye State Routing(FSR) [24] [105] define scopes for dissemination of routing information and thus also can be considered hierarchical but not clustered; the Hierarchical State Routing Protocol(HSR) does physical and logical clustering. It is also capable of multilevel clustering; LANMAR, since a landmark can be considered a representative node on a higher level. The Zone Routing Protocol (ZRP) [53] [54] [55] defines a routing zone, this is also regarded as type of clustering.

2.5.4 Position Based Protocols

The advances in the development of the Global Positioning System (GPS) make it possible to provide location information with a precision in the order of a few meters. Location information through such systems can be used for directional routing in distributed ad-hoc systems. The universal clock in these systems can provide global synchronizing among GPS equipped nodes. It is said that geographical location information can improve routing performance in mobile ad-hoc networks. Where there are many benefits of such schemes one of the main drawbacks lies in the fact that the locations may not be accurate by the time the information is used. Therefore, most of the protocols (GPSR)[77] and DREAM [47] assume that the nodes know their positions. One other protocol falls in the same category is reported in [75]. Likewise a detailed report about various existing position based mechanisms is cited in [76]. Some of the other benefits of position based algorithms are minimizing routing overheads to some extent as no routing tables are involved. Moreover, no route discovery and route maintenance procedures are needed therefore the chance of network overhead is reduced.

These systems use an internal route discovery mechanism which imposes an overhead to maintain the position information. Some protocols in this type such as DREAM [47] and LAR [78] use a flooding approach. In these schemes packets are not sent to all neighbors but to only those in the right direction of the target. DREAM [47] utilizes an all-for-all position service such that each node carries a location table for each other node and requests are forwarded in the right direction.

Hybrid/hierarchical schemes and Terminode routing are some other protocols which fall in this category. In these protocols long distances are covered by a greedy directional routing scheme. If the packet is close enough some non directional mechanism will guide the packet to the destination.

One other protocol that uses directional routing is GEDIR [79]. In GEDIR to obtain the right direction the locations of source, target and intermediate nodes must be known. It is not specified how the location information should be determined. Usually this consists of a broadcast message from the originating node indicating the desired route. Nodes which have the required information will respond to the originating node which will eventually choose a route from the replies it received. The broadcast may be limited to travel only a few hops first before a network wide broadcast will be issued. This requires the route request and selection process must be finished before the message can be sent. This procedure further leads to an initial setup delay for messages if their route is not known to the node. To limit the impact of this delay most protocols use a route cache for once established routes. However, the information in this cache will run out since in a mobile environment the routes will be invalid after some time. It is essential for applications using an on-demand routing protocol to be tolerant for such an initial setup delay.

2.5.5 Tables Driven Routing Protocols

Table-driven protocols are one of the very first routing schemes that were proposed for mobile ad-hoc networks. These protocols maintain a consistent overview of the network. This could be achieved in one of two ways i.e. event driven and regular updated protocols. Event driven protocols will not send any routing update packets if no change in topology occurs, while protocols that are updated in regular intervals will always send their topology information to other nodes at regular intervals. Each node uses routing tables to store the topology information of the network. This information is used to transfer data among various nodes of the network. To ensure the freshness of the routing tables, these protocols adopt different sorts of mechanisms. One of the adopted methods is broadcasting "hello", a special message containing topology information, at fixed intervals of time. On receiving this message, each node updates its routing tables with fresh information of other nodes. Some of the examples of the event driven protocols are Cluster Based Routing Protocol (CBRP) [73], Clusterhead Gateway Switch Routing (CGSR), Destination Distance Sequence Routing (DSDV) [23], Global State Routing (GSR) [65] and Wireless Routing Protocol (WRP)[69] [114] while some of the regular updates protocols are Distributed Dynamic Routing Algorithm (DDR)[48],Fisheye State Routing (FSR)[24], Greedy Parameter Stateless Routing (GPSR) [77], Landmark Routing (LANMAR) [70] [74] and Topology Broadcast Based on Reverse Path Forwarding (TBRPF) [59][68].

2.5.6 On-Demand or Reactive Routing Protocols

In on-demand protocols [95], if a source node requires a route to the destination for which it does not have route information, it initiates a route discovery process which goes from one node to the other until it reaches the destination or an intermediate node has a route to the destination. The source node uses established routes for data transmission to the destination node. Some of the better known on-demand protocols are Ad-hoc On-demand Distance Vector routing (AODV) [25][118], Dynamic Source Routing (DSR) [28] [89] [90] and Temporary Ordered Routing Algorithm (TORA) [80] [96]. Other examples of on-demand protocols are Associatively Based Routing (ABR) [64][101], Ad-hoc on Demand Distance Vector (AODV) [25], Core Extraction Distributed Ad-hoc Routing (CEDAR) [81] [108], Distance Routing Effect Algorithm for Mobility (DREAM) [47], Location Aided Routing (LAR) [78], Signal Stability Routing Protocol (SSR) [82] and Witness Aided Routing (WAR) [61].

2.5.7 Hybrid Protocols

Hybrids protocols involve the combination of table driven and on-demand routing protocols. Adaptive distance vector routing (ADV) is an example of such a protocol. In ADV [83] routes are maintained proactively but only to certain nodes and the size and frequency of the updates are adapted. Terminode routing consists of on-demand location based components i.e. Anchored Path Geodesic Packet (AGPF)

forwarding and a proactive local routing component, which works similar to IARP from ZRP. The Zone routing protocol also consists of a proactive Intra Zone Routing Protocol (IARP) and an on-demand Inter Zone Protocol (IERP).

2.6 Summary

Most of the routing protocols for mobile ad-hoc networks adhere to the basic design principles of routing protocols for fixed networks. This chapter explored the relation between the routing mechanisms for fixed and mobile ad-hoc networks. In this context, wherever appropriate some of those protocols have been referenced in the above discussion. Moreover, different protocol types of mobile ad-hoc networks were also focused. Brief discussions of the TCP performance over mobile ad-hoc networks along with some of the problems were presented. The following chapter is an extension of this chapter in that some of the famous routing mechanisms of mobile ad-hoc networks will be critically evaluated.

CHAPTER 3. ROUTING PROTOCOLS FOR MOBILE AD-HOC NETWORK

3.1 Introduction

In the previous chapter we presented a brief discussion of different types of routing protocols for fixed and mobile ad-hoc networks. In this chapter different routing protocols for mobile ad-hoc networks will be critically evaluated. This chapter has been divided into two main sections. Section one studies some of the protocols currently under consideration by Internet Engineering Task Force (IETF). Besides a general introduction of these protocols, this section also covers critiques of these schemes. For the sake of clarification critiques have been provided in a separate subsection of the main section of each scheme.

The same procedure is followed in section 2 which studied a number of other protocols. In section 2 all those protocols whose operations are similar in some aspect to mobile ad-hoc on-demand data delivery protocol (MAODDP) are evaluated before the protocols that fall in any other category or follow different operational patterns. Most of the findings presented in this chapter are based on some of our published work[2][3][27][40].

3.2 Protocols Under Review by IETF

Destination distance sequence vector (DSDV) [23] of table driven , Ad-hoc on-demand distance vector (AODV) [25] of on-demand and Zone routing protocol(ZRP) [53] of the hybrid type are under consideration by IETF. The main aim of IETF is to organize ongoing research efforts on a single, large platform. One of the tasks of the working group is to standardize IP routing protocol functionality for wireless routing applications, both within static and dynamic types of topologies. Other sets of tasks include exploration of mobile ad-hoc network problems on a large scale, related performance issues, and further development of related proposed protocols. Recently, other modifications have been made to the initial charter of this working group. These changes focus on the operation of mobile ad-hoc networks under a reduced scope by targeting the promotion of a number of different routing protocol specifications to experimental status. Although some understanding and implementation of these protocols already exists, more operational experimentation is required. It is important to mention that most of these protocols have been implemented to different extent. Among them DSDV and AODV follow the nearest mode of operation to MAODDP. Therefore in the following section these two protocols will be critically evaluated before ZRP.

3.2.1 Destination-Sequenced Distance-Vector Routing Protocol (DSDV)

The destination sequenced distance vector routing protocol (DSDV)[23]is an extension of classical bellman ford routing mechanism . DSDV maintains a consistent network view via periodic routing updates. Routing information is stored inside routing tables maintained by each node. An example of a routing table is shown in Table.3.1. Each entry is marked with a sequence number assigned by the destination node. The sequence numbers enable the mobile nodes to distinguish stale routes from new ones thereby avoiding the formation of routing loops.

Destination	Next	Metric	Seq.Number	Install-Time	Stable Data
F	F	0	F-670	002000	Ptr_F
G	G	1	G-780	004000	Ptr_G
H	G	3	H-890	006000	Ptr_H

Table 3.1 DSDV Structure of routing table

New route broadcasts contain the address of the destination, the number of hops to reach the destination, the sequence number of the destination and a new sequence number unique to the broadcast. The route labelled with the most recent sequence number is always used. In the event that two updates have the same sequence number, the route with the smaller metric is used in order to optimise the path. An example of a route advertised table is shown is Table. 3.2.

Destination	Sequence number	Metric
F	0	F-670
G	1	G-780
H	3	H-890

Table 3.2 Advertised route table in DSDV

The receiver also advertises route received in its periodic broadcast. The receivers also add an increment to the metric before advertising the route since further incoming packets will require one more hop to reach the destination.

The most recent measurement of the settling time is calculated by maintaining the weighted average over the most recent updates of the route for each destination. A parameter is also selected to indicate how long a route has to remain stable before it is counted as truly stable. Any route more stable than this parameter will cause a triggered update if it is ever replaced by another route with different next hop or metric.

Broken links are described by ∞ metric, infinity, which could be any value greater then the maximum allowed metric . Whenever a link to the next hop is broken any route through the next hop is assigned ∞ metric and an updated sequence number. Therefore the only situation when a mobile node generates a sequence number is when a link breaks. Mobile nodes receiving new broadcasts can find the fresh route by comparing receiving information with those that are previously recorded in their routing table. A route with a recent sequence number is considered as a fresh route while the older sequence number routes are discarded. If sequence numbers are found to be the same then the route with better metric will be selected .

Critique of DSDV

DSDV requires nodes to periodically transmit routing table update packets regardless of the network traffic . When the number of nodes in the network grows the size of the routing tables and the bandwidth required to update them also grows[34]. This overhead is considered as the main weakness of DSDV [34]. DSDV also poses a period of convergence before which routes will not be known and packets will be dropped [34]. This could also limit the number of nodes that can connect to the network since the overhead grows as O (N^2) [34]. Moreover, DSDV works only with bidirectional links [34].

DSDV is mentioned in several different simulation studies [37]. The results were mixed but later papers show results where DSDV is not performing well compared to the other protocols. DSDV is well known for its poor performance at high mobility rate. It is known that DSDV performed well for fairly static topologies but became unreliable as node mobility and the number of traffic sources increased. Likewise DSDV optimal values for parameters like settling time are not easy to determine [34]. That might result in unnecessary bandwidth consumption [34]. Moreover in DSDV routing loops can occur while the network is reacting to a change in the topology.

DSDV uses distance vector shortest-path routing as the underlying routing protocol. It has a high degree of complexity especially during link failure and additions [34]. Maximum settling time is difficult to determine in DSDV. DSDV does not support multi-path routing[34]. Fluctuation is another problem of DSDV. DSDV is much more conservative in terms of routing overhead but because link breakages are not detected quickly more data packets are dropped[34]. The specification of DSDV is silent over security issues [40]. DSDV assumes that all nodes are trustworthy and cooperative. Once the false sequence has been established the attacker will continuously send out new packets to update the value. So more hosts will be cheated [34] therefore a single misbehaving node can pose a serious threat for the entire network [34].

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

AODV is a combination of both DSR [28] [92] [93] and DSDV [23]. It inherits the basic on-demand mechanism of route discovery and route maintenance from DSR and the use of hop-by-hop routing sequence numbers and periodic beacons from DSDV. AODV[25] provides both multicast, and unicast connectivity in a mobile ad-hoc environment. The main feature of AODV is quick response to link breakage in an active route .

AODV[25] builds routes using a route request and route reply query cycle. For destination source nodes with no prior information it broadcasts a route request (RREQ) packet. Nodes receiving RREQ update their information and set up backward pointers to the source node. In addition to the source node's IP address, current sequence number and broadcast ID, RREQ also contains the most recent sequence number of the destination of which the source node is aware. A node receiving the RREQ may send a route reply RREP in two situations, if it is the destination or if it has a route to the destination with corresponding sequence number greater then or equal to that contained in the RREQ. If this is the case it unicasts a RREP back to the source. Otherwise, it rebroadcasts the RREQ.

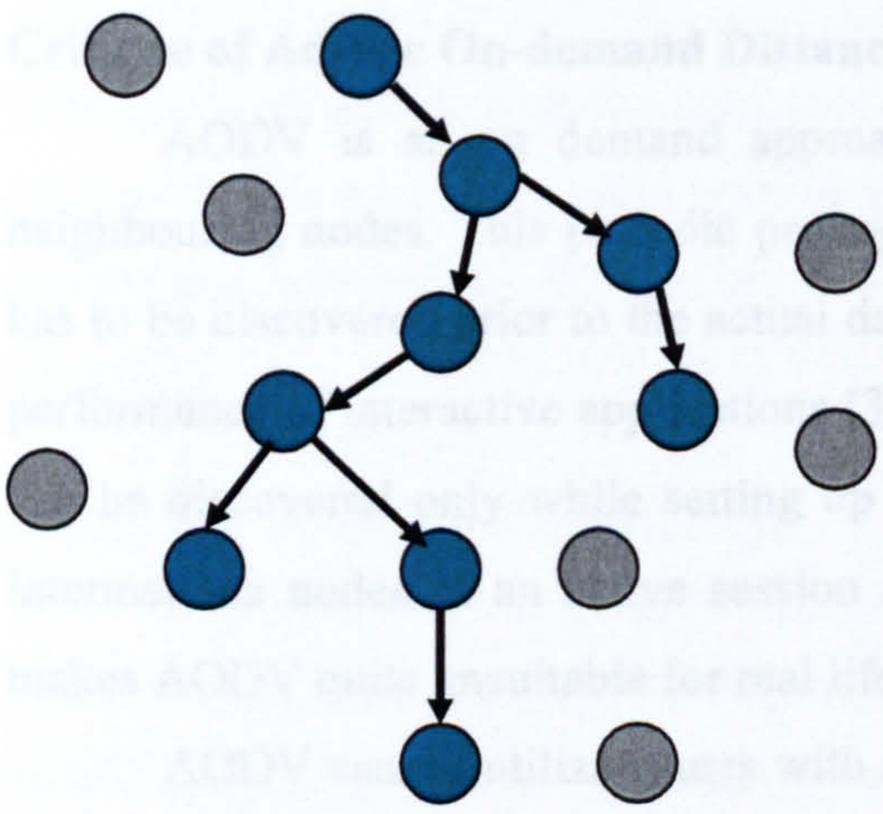


Figure 3.1 A sample multicast tree AODV

When the source node receives the RREP it begins to forward data packets to the destination. If the source later receives a RREP containing a greater sequence number or same sequence number with a smaller hop count it can update its routing information. The source can then use this route for that destination. AODV defines route discovery, route maintenance and route error messages types as shown in figures 3.3, 3.4 and 3.5 respectively. AODV uses 'hello' messages to check the status of neighbouring nodes. If a node in

AODV fails to send 'hello' message within the prescribed time limit it will be considered an inactive node and the link will be considered broken.

RREQ ID	Hop Count	Destination IP Address	Destination Sequence Number	Source IP Address	Source Sequence Number

Table 3.3 Route request AODV

Hop Count	Destination IP Address	Destination Sequence Number	Source IP Address	Life Time

Table 3.4 AODV route reply

N Flag	Destination Count	Unreachable Destination1 IP Address	Unreachable Destination1 Sequence Number	Other destination as needed

Table 3.5 Route error AODV

Critique of Ad-hoc On-demand Distance Vector Routing (AODV)

AODV is an on demand approach but still use periodic broadcast of 'hello message' to track neighbouring nodes. This periodic propagation causes network overhead in AODV [34]. In AODV a route has to be discovered prior to the actual data packet transmission. This initial search latency may degrade the performance of interactive applications [34]. Similarly the quality of path is not known prior to call set-up. It can be discovered only while setting up the path. Moreover the quality of path must be monitored by all intermediate nodes in an active session at the cost of additional latency and overhead penalty [34]. That makes AODV quite unsuitable for real life applications.

AODV cannot utilize routes with asymmetric links between nodes and thus requires symmetric links [34]. Nodes in AODV store only routes that are needed. Nodes use the routing caches to reply to route queries. These results in 'uncontrolled' replies and repetitive updates in hosts' caches yet early queries cannot stop the propagation of all query messages which are flooded all over the network.

3.2.3 Zone Routing Protocol (ZRP)

The Zone Routing Protocol (ZRP) is a hybrid routing protocol [53]. It combines both proactive and reactive routing techniques. Each node has a predefined zone centered at itself in terms of number of hops. For nodes within the zone it uses proactive routing protocols to maintain routing information. For those nodes outside of its zone it does not maintain routing information on a permanent basis. Instead, on-demand routing strategy is adopted when inter-zone connections are required.

The ZRP protocol consists of three components. In the zone proactive Intra-zone Routing Protocol (IARP) is used to maintain routing information. IARP can be link state routing or distance vector routing depending on the implementation. For nodes outside the zone, reactive Inter-zone Routing Protocol (IERP) is performed. IARP provides a route to nodes within a node's zone. IERP uses the route query (RREQ) route reply (RREP) packets to discover a route very similar to some on-demand routing protocols[25].

When the intended destination is not known at a node i.e. not in its IARP routing table that node must be outside of its zone. Thus, a RREQ packet is broadcast via the nodes on the border of the zone. Such a RREQ broadcast is called Broadcast Resolution Protocol (BRP). Route queries are only broadcast from one node's border node to other border nodes until one node knows the exact path to the destination node i.e. the destination is within its zone.

Critiques of Zone Routing Protocol (ZRP)

ZRP limits the proactive overhead to only the size of the zone. It also limits reactive search overhead to only select border nodes. Potential inefficiency may occur when flooding of the RREQ packets goes through the entire network. To some extent this protocol can provide a better solution in terms of reducing communication overhead and delay. But this benefit is subjected to the size of a zone and the dynamics of a zone. ZRP does not provide an overall optimized shortest path if the destination has to be found through IERP [87]. Moreover with the increase of network size ZRP could create unpredictable large overhead. In ZRP each path to a destination may be suboptimal. This also means that each node will have higher level topological information. This poses a higher memory requirement and an extra burden on the network resources.

3.3 Other Routing Algorithms

Besides the above mentioned protocols there are some other relevant routing protocols [23][25] well known in this field. Similarly for some others [57] [58] no further work is reported. These protocols fall into one of different protocols types which have been discussed earlier in chapter 2 of this thesis. Among these protocols Dynamic source routing (DSR) [28][94] and Temporary ordered routing algorithm(TORA)[97] to some extent falls in the same category as MAODDP. In this section besides introducing some other routing techniques a critical evaluation of each of these schemes will be presented.

3.3.1 Dynamic Source Routing (DSR)

Dynamic source routing protocol [28][37] is a reactive protocol. DSR requires no periodic updates of any kind at any level within the network. DSR uses source routing through which the sender knows the complete hop-by-hop route to the destination. These routes are stored in a route cache. A data packet carries the source route in the packet header. DSR makes very aggressive use of source routing and route caching.

No special mechanism to detect routing loops is needed. Likewise any forwarding node caches the source route in a packet forwards for possible future use.

The DSR [28] protocol consists of two mechanisms, route discovery and route maintenance. Route discovery mechanism is initiated when a source desires a route to a destination for which it does not have any prior information. Route discovery process functions by flooding the network with route request (RREQ) packets. Each node receiving a RREQ packet rebroadcasts it unless it is the destination or it has a route to the destination. RREQ and RREP packets are also source routed. The route carried back by the RREP packet is cached at the source for future use. For route maintenance whenever a link on a source route is broken the source node is notified using a route error (RER) packet. The source removes any route using this link from its cache. An intermediate node can use an alternative route from its own cache when a data packet meets a failed link on its source route. A source node receiving an RER packet piggybacks the RER packet in the following RREQ.

This is helpful in cleaning up the caches of other nodes in the network that may have the failed link in one of the cached source routes. When a node overhears a packet not addressed to itself it checks if the packet could be routed via itself to gain a shorter route. If so, the node sends a gratuitous RREP to the source of the route with this new and better route. Promiscuous listening helps a node to learn different routes without directly participating in the routing process.

Critique of Dynamic Source Routing (DSR)

DSR is not designed to track topology changes occurring at a high rate [34]. Two sources of bandwidth overhead in DSR are route discovery and route maintenance. These occur when new routes need to be discovered or when the network topology changes. In DSR this overhead can be reduced by employing intelligent caching techniques in each node at the expense of memory and CPU resources. The remaining source of bandwidth overhead is the required source route header included in every packet. This overhead cannot be reduced by techniques outlined in the protocol specification [34].

DSR is based on source routing therefore it requires considerably greater routing information. In DSR a route has to be discovered prior to the actual data packet transmission. This initial search latency may degrade the performance of interactive applications [34]. Moreover, the quality of the path is not known prior to the call setup. It can be discovered only while setting up the path. This quality of path needs monitoring by all intermediate nodes during a session. It increases the cost of additional latency and overhead penalty [87].

Due to source routing DSR has major scalability problem. Nodes use routing caches to reply to route queries. This results in 'uncontrolled' replies and repetitive updates in hosts' caches. Moreover, early queries cannot stop the propagation of all query messages which are flooded all over the network. Therefore when

the network becomes larger, the control packets and message packets also become larger. This could degrade the protocol performance after a certain amount of time.

DSR is cited in some simulation studies. In an analytical study of the probabilities of successful deliveries and the total amount of traffic generated for a successful delivery is presented. It is argued that an end-to-end recovery mechanism does not scale if the routing path lengths increase. Instead a local recovery mechanism is suggested that gives much better results. It is reported that DSR performs better at high mobility rates due to the overhead of AODV's route discovery messages.

3.3.2 Temporary Ordered Routing Algorithm (TORA)

TORA [80] [98] is a distributed routing protocol which is based on a link reversal algorithm. TORA establishes routes quickly by localizing algorithmic reaction to topological changes whenever possible. Shortest path routes are considered secondary and importance is given to longer routes to avoid the overhead of discovering new routes. Timing is another important factor for TORA because the height metric is dependent on the logical time of a link failure. TORA assumes all nodes have synchronized clocks.

TORA is designed to discover routes on demand. At each node in the network a separate copy of TORA is run for each destination. When a node needs a route it broadcasts a query request to all other nodes. This query packet contains the address of the destination for which it requires a route. This packet propagates throughout the network until it reaches either to the destination or to the closest node having a route to the destination. This node than broadcasts an update packet listing its height with respect to the destination. When this reply packet propagates through the network each node that receives the update sets its height to a value greater than the height of the neighbour node from which the update was received. It has the effect of creating multiple links from one node to the other.

Critiques of Temporary Ordered Routing Algorithm (TORA)

TORA is one of the largest protocols thus it requires extra memory for different operations. Each node must maintain a structure describing the node's height as well as the status of all connected links per connection supported by the network. TORA requires each node to be in constant coordination with neighbouring nodes, to detect topology changes and coverage which pose high bandwidth and CPU requirements. Similar to DSDV [23] routing loops can occur while the network is reacting to a change in the topology.

It is known that TORA is one of the complex algorithms among other algorithms of similar types. The main drawback of TORA is the exorbitant assumptions that it makes. Not only does it require bi-directional links and a link-level protocol but it actually depends on correct and in-order transmission of all packets.

TORA was used in some simulation studies such as in [98] where its performance was not impressive. TORA performs well for fairly static topologies but become unreliable as node mobility and the number of traffic sources increase. Some of the simulation results presented in [133, 134] showed that TORA is best suited to a large, densely packed collection of nodes with very low node mobility.

Between the two papers reporting simulation results only [98] actually simulates node mobility. In [98] TORA is encumbered by its layering on top of IMEP. Likewise it was found that IMEP caused considerable congestion when TORA was trying to converge in response to node mobility. This resulted in TORA requiring between one and two orders of magnitude more routing overhead than other mobile ad-hoc routing protocols investigated in [98].

TORA uses internodal co-ordination and it exhibits instability behaviour similar to "count-to-infinity" problem in distance vector routing protocols. Thus there is a potential for oscillations to occur especially when multiple sets of coordinating nodes are concurrently detecting partitions, erasing routes, and building new routes based on each other. Though such oscillations are temporary and route convergence will ultimately occur, it poses real threats to utilize TORA at its full.

3.3.3 Associativity Based Routing (ABR)

Associativity based routing [100] is a new and different approach which claims to be free from loops, deadlock and packet duplicates. It defines a routing metric for mobile ad-hoc network. This metric is known as the degree of association stability. A route is selected based on the degree of association stability of mobile nodes. All nodes generate a beacon to signify their existence. When received by neighbouring nodes this beaconing causes their associatively tables to be updated.

For each beacon received the associativity tick of the current node with respect to the beaconing node is incremented. Association stability is defined by connection stability of one node with respect to another node over time and space. A high degree of association stability may indicate a low state of node mobility while a low degree may indicate a high state of node mobility. Associativity ticks are reset when the neighbours of a node or the node itself moves out of proximity. ABR relies on one of three functions i.e. broadcast query and await-reply (BQ-REPLY) to discover a route for the intended destination; on-demand route recovery phase to handle break or expired links and on-demand route deletion phase to delete unused routes. Most of these functions operate very similar as some of the other on-demand protocols such as AODV [25] and DSR [28]. The main concept of ABR is adopted in [116] to support on-demand multicast operation.

Critique of ABR

ABR adopts the basic idea of maintaining routing information via continuous beacon updates. It is fairly well known that such schemes are not very impressive due to extra burden they pose on certain network resources. Moreover, due to the nature of mobile ad-hoc networks, it is highly unlikely to maintain strong link connectivity among mobile nodes. ABR has used in some of the simulation studies. In results were mixed while in ABR showed weak performance in comparison with other simulated protocols.

3.3.4 Signal Stability Routing (SSR)

Signal Stability based adaptive protocol (SSR) [82] is an on-demand protocol. SSR selects routes based on the signal strength between nodes and on the node's location stability. SSR can further be split into two cooperative protocols i.e. the dynamic routing protocol (DRP) and the static routing protocol (SST). DRP is responsible for maintaining the signal stability table (SST) and routing table (RT). SST records the signal strength of neighbouring nodes. This signal strength is obtained by periodic beacons from the link layer of each neighbouring node. Signal strength is either marked as a strong or weak channel. All transmissions received by DRP are processed. After updating the appropriate table entries the DRP passes the packet to the SRP.

The SRP passes the packet up the stack if it is the intended receiver. If not, it looks up the destination in the routing table (RT) and forwards the packet. If no entry for the destination is found in the RT, SRP initiates a route-search process to find a route. To avoid looping, route-request packets are forwarded to the next hop only if they are received over strong channels and have not been previously processed. The destination chooses the first arriving route-search packet to send back. It is expected that the packet arrived over the shortest and/or least congested path. The DRP reverses the selected route and sends a route-reply message back to the source of the route-request. The DRP of the nodes use the route search process to update their RTs accordingly.

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 than 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.

Critique of Signal Stability Routing (SSR)

A partial route discovery mechanism is not valid to SSR. Therefore if a link failure is detected route discovery has to be initiated from the source. Broken links are locally detected but not repaired and the multiple flooding of RouteRequest messages restricts the bandwidth. One other weakness of SSR is the failure of the intermediate nodes to reply to route requests which are forwarded towards the destination. This drawback adds more delay during the route discovery process. SSR does not suggest any mechanism to address those packets which receive over the weak channel. In a mobile ad-hoc network environment it is expected that channel strength could vary and maintaining strong signals on a consistent basis is not easy. In

SSR the absence of mechanisms to differentiate between different types of packets could results in large number of packets being dropped.

3.3.5 Wireless Routing Protocol (WRP)

The Wireless Routing Protocol (WRP) [69] [114] maintains routing information among all nodes in the network. Each node maintains four tables i.e. distance table, routing table, link-cost table and message retransmission list (MRL) table. Each entry of the MRL contains the sequence number of the update message a retransmission counter and an acknowledgement required flag vector with one entry per neighbour and a list of updates sent in the update message. The MRL records updates in an update message that needs to be retransmitted and which neighbours should acknowledge their transmission.

Nodes inform each other of link changes, loss of link between two nodes through the use of update messages. An update message is sent only between neighbouring nodes and contains a list of updates as well as a list of responses indicating which node should acknowledge the update. The neighbours then update their distance table entries and check for new possible paths through other nodes. Any new paths are relayed back to the original nodes so that they can update their tables accordingly.

Critique of Wireless Routing Protocol (WRP)

Nodes in WRP maintain four tables and thus require sufficiently higher memory then some other table driven protocols. WRP also use 'hello packets' to keep updated routing information. It has been mentioned before those messages consume different network resources. Overall latency associated in routing is comparatively less in WRP as it maintains separate tables. However, it is of more use when a link failure occurs. WRP use distance vector shortest-path routing as the underlying routing protocol and it has certain degree of complexity during link failure and additions.

In the light of current specification of WRP it could be well understood that such a scheme is best suited for a small network. It is expected that the protocol efficiency could drop down as the network grows. WRP focuses on broadcasting packets to the nodes in close vicinity, it may be concluded that nodes may not have adequate information about nodes not in their vicinity. Therefore, it limits effective data transmission to a small area. It could be said that for WRP there is no route discovery mechanism. Route management between nodes far from each other is still an issue.

Update messages are limited to the neighboring nodes. This limits the network view for nodes not operating in the close vicinity. Due to the dynamic nature of mobile ad-hoc networks it is expected that link changes will occur quite frequently. This requires such update packets to be broadcast more frequently. It will have two negative impacts on the network. First one is that node reaction will be slow for immediate replies. Likewise, such frequent update transmission can put an extra burden on the overall network resources such as bandwidth etc. Moreover, it could also lead towards network overhead.

3.3.6 Cluster-head Gateway Switch Routing Protocol (CGSR)

The Cluster-head gateway switch routing protocol (CGSR) is a clustered multi-hop mobile wireless network with several heuristic routing schemes. CGSR can greatly reduce the routing table size compared to distance vector protocols. Only one entry is needed for all nodes in the same cluster.

In CGSR a cluster head controls a group of mobile nodes. A framework for code separation and channel access through which routing and bandwidth allocation is achieved. A cluster head selection algorithm is utilized to select a node as the cluster head using a distributed algorithm within the cluster. The disadvantage of having a cluster head scheme is that frequent cluster head changes can adversely affect routing protocol performance since nodes are busy with cluster head selection rather than packet relaying. Hence instead of invoking cluster head reselection every time the cluster membership changes a clustering algorithm is introduced. Using LCC cluster-heads only change when two cluster heads come into contact or when a node moves out of contact of all other cluster-heads. The main problem is transmission power limited by the number of cluster head changes in mobile ad-hoc networks.

CGSR also proposes some methods to improve the performance which include path reservation priority token scheduling etc. The CGSR is the only table driven protocol that follows a hierarchical routing philosophy and does not use any hello messages.

Critique of Cluster-head Gateway Switch Routing Protocol (CGSR)

Maintaining cluster structure in a mobile environment is the main drawback of this scheme. Therefore its efficiency in such an environment is a questionable issue. Moreover LCC clustering algorithm introduces additional overhead and complexity in the formation and maintenance of clusters [87]. Frequent cluster head change is another weakness of this scheme. Frequent cluster head changes can adversely affect performance since most of the time it spends in selecting the cluster-head or the leader. Cluster head tables also pose additional requirement to the memory.

Routing performance is dependent on the status of specific nodes time complexity of a link failure. This link failure is associated with a cluster head. It is higher then DSDV given the additional time complexity of a link failure associated with a cluster head reselection. CGSR use distance vector shortest-path routing as the underlying routing protocol. It has the certain degree of complexity during link failure and additions. In CGSR cluster heads and gateway nodes have higher computation and communication load then other nodes. The network reliability may also be affected due to single points of failure of these critical nodes.

3.3.7 Global State Routing (GSR)

Global State Routing (GSR) [65] improves link state routing by avoiding flooding of routing messages. Each node maintains a Neighbours list, a Topology table, a Next Hop table and a Distance table. Neighbours list of a node contains the list of its neighbours where all nodes that can be heard by a node are assumed to be its neighbours. For each destination node, 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. On receiving a routing message the node updates its topology table if the sequence number of the message is newer than the sequence number stored in the table. After this the node reconstructs its routing table and broadcasts the information to its neighbours.

Critiques of Global State Routing (GSR)

The update message size in GSR is relatively large compared to those in some other schemes. Large message size and propagation delay wastes a considerable amount of network bandwidth. That makes it difficult to predict GSR performance on different sizes of network.

It is not clear why routing information in GSR is stored inside three tables besides maintaining neighbour list. This approach is different from traditional link state routing protocols such as DSDV [23] which use a single table for same purpose. Keeping information inside three different tables limits node performance to a certain extent. Not limited to route or address management, these tables have their due affects on battery life of mobile nodes. Efficient retrieval of already stored addresses requires a search operation. Having distributed information could slow down the whole search process. Likewise storing new information could yield the same affect.

3.3.8 Fisheye State Routing (FSR)

Fisheye State Routing (FSR) is an improvement on GSR. In FSR, each update message does not contain information about all nodes, thereby reducing the size of the messages and saving a considerable amount of bandwidth. Instead, it exchanges information about closer nodes more frequently then it does about farther nodes thus reducing the update message size. So each node gets accurate information about neighbours. However, details and accuracy of information decreases as the distance from the node increases. The scope is defined in terms of the nodes that can be reached in a certain number of hops. The centre node has most accurate information about all nodes in the white circle and so on. Even though a node does not have accurate information about distant nodes the packets are routed correctly because the route information becomes more and more accurate as the packet moves closer to the destination.

Critique of Fisheye State Routing (FSR)

It is clear from the above description that FSR could show better results in a small network. However, its efficiency could be reduced as the network grows. In other words accuracy of information decreases as the distance between the nodes increases. Having an integrated node contain wider information than other nodes can reduce the response ability of other nodes in the network. It also reduces the view of the other nodes in comparison with the centre node. Moreover, this semi integrated structure is not suitable for mobile ad-hoc network environments. FSR in its current specification relies on the frequent propagation of routing packets. Mobile ad-hoc networks suffer with frequent topology changes. It is highly unlikely to achieve such a consistent packet propagation which also reduces the efficiency of FSR.

3.3.9 Source Tree Adaptive Routing (STAR)

In STAR [57] each node maintains a source tree which consists of its preferred links to each destination. The source tree is calculated on the information of its own links and the source trees reported by its neighbors. Any changes in a source tree are reported to the neighbors in an incremental manner. The source tree and neighbor information establish the partial topology information in each node. This information is used by a route selection algorithm to obtain the route table with destination and next hop. In STAR information is updated with link state updates. An update message contains information of several link state updates (LSUs). It reflects changes in the node's source-tree sequence numbers. This update message is used to distinguish current from outdated information. The link state information does not time out, thus removing the need for a periodic update. STAR can operate in several modes but there are two main modes namely optimum routing approach and the least overhead routing approach.

Critique of Source Tree Adaptive Routing (STAR)

STAR requires new neighbors and leaving neighbors are detected in finite time. This could also limit the overall scope of this scheme. Likewise the protocol requires a link layer capable of transmitting local broadcast messages without hidden terminal interference. Unlike some other link state protocols STAR does not follow any approach to clear outdated information from the routing table. This leaves a number of different side effects on the protocol performance. Over time routing tables will grow bigger. No doubt it will have its own negative impact on the available resources such as memory. Likewise it could also degrade node performance. In situations where in an already established network nodes have to look for destination of interests an extra amount of time is added to the initial node search process. Moreover, if nodes decided to search for a suitable route, the same response query packet will be received at all intermediate receiver's nodes. As a consequence the whole network will be slowed down. Chances are as time passes the network performance will reach such an extent where rebooting the entire network becomes necessary.

In STAR the link state information does not time out which makes it difficult to predict anything about the stability of the recorded links. STAR claims to reduce the routing overhead but the protocol specification is silent about its effect on network resources such as bandwidth and battery power. Lastly, not enough literature highlighting STAR performance or comparison with other schemes is available. This also limits the possibility of gaining a wider understanding about the protocol's working and its performance in different networking environments.

3.3.10 Optimized Link State Routing (OLSR)

Optimized Link State Routing [71] is another proactive link state protocol which is claimed to work best in large dense networks. In OLSAR each node selects a set of Multipoint Relays (MRP) from its neighbors. The radio range of the MRP is set such that it should cover all two hops neighbors. Each node has the knowledge as to for which node it acts as a MRP. Thus OLSR requires bidirectional links. OLSR utilizes UDP to distribute routing packets. Each routing packet contains one or more OLSR messages. Messages exist for neighbors by the same originator as the route and send its reply via the reversed hop list in the received request.

Critique of Optimized Link State Routing (OLSR)

OLSR is suitable for networks where frequent communication take place in collection of nodes rather than as a whole. It is not clear what criteria nodes use to form Multipoint Relays (MRP). Each routing packet in OLSR can have more than one message. Therefore more effective measures are required to differentiate different messages in a routing packet. OLSR use User Datagram Protocol (UDP) as a communication medium. UDP provides very few error recovery services and it uses a direct way to send and receive datagram's over an IP network. Due to the nature of mobile ad-hoc network it is expected that network transmission would meet different types of error. Absence of effective error recovery mechanisms could make it difficult to utilize OLSR at best.

3.3.11 Distance Routing Effect Algorithm for Mobility (DREAM)

Distance Routing Effect Algorithm for Mobility[47] is a table driven protocol. It is designed to provide distributed loop-free and multi-path routing. DREAM is also able to adapt to mobility. For routing update DREAM introduces two new mechanisms i.e. frequency and message life time. The principles are distance effect and mobility rate. Each node can sense these two principles with the help of information obtained from GPS. In general, for the first principle i.e. distances effect; the greater the distance separating two nodes the slower they appear to be moving with respect to each other. With the mobility rate the faster a node moves the more frequently it needs to advertise its new location. Using the location information obtained from GPS each node can realize the two principles in routing.

In DREAM, each node records location information in a Location Table. The table records locations of all the nodes. Like most of the table driven protocols, each node broadcasts a control packet to inform all

other nodes about their locations. The distance effect is realized by sending messages more frequently to nodes that are more closely positioned. In addition the frequency of sending a control packet is adjusted based on its moving speed. With the location information stored at routing tables, data packets are partially flooded to nodes in the direction of the destination, and then it selects a set of one-hop neighbors that are located in the direction. If such steps are empty the data is flooded to the entire network. Otherwise, the set is enclosed in the data header and transmitted with the data. Only nodes specified in the header are qualified to receive and process the data packet. They repeat the same procedure by selecting their own packet. They repeat the same procedure by selecting their own set of one-hop neighbor updating the data header and sending the packet out. When the destination receives the data it responds with an ACK to the source in a similar way. However, the destination will not issue an ACK if the data is received via flooding. The source, if it does not receive an ACK for data sent through a designated set of nodes, retransmits the data again by pure flooding.

Critique of Distance Routing Effect Algorithm for Mobility (DREAM)

DREAM is claimed to be a loop free since the messages travel away from the node into a specific direction. This could be questioned since in a network with very high mobility the target direction can change even back to a node that has sent the message already. Another problem is that location table entries may be stale and that no close neighbor in the required direction can be found. DREAM requires each node to be equipped with a GPS system. This additional requirement has several drawbacks. Normally a GPS system is available under certain scenarios such as in battle field or in a disaster recovery. Availability of such a system among normal users is not common. That not only limits the operational scope of DREAM but also poses a limit on its further practical implementation.

There are different conditions imposed by the protocol for routine network operations. It is a common observation that normal network operation becomes complex due to excess of conditions. Conditions such as issuing an acknowledge message only if the packet is received via flooding pose an additional requirement. A node now has to discover first how the packet is received. It could add to the waiting time for packet in the queue. Likewise it could also delay in responding to those packets which requires immediate action. Environments such as battle fields etc require smooth and effective transmission. These conditions could results in significant drops of protocol performance. Finally no further work on DREAM has been reported in the cited literature but other routing schemes such as LAR or FSR did pick up some concepts of DREAM.

3.3.12 Zone-based Hierarchical Link State Routing Protocol (ZHLS)

In Zone-based Hierarchical Link State Routing Protocol (ZHLS), the network is divided into non-overlapping zones. ZHLS defines two levels of topologies – node level and zone level. A node level topology

tells how nodes of a zone are connected to each other physically. A virtual link between two zones exists if at least one node of a zone is physically connected to some node of the other zone. Zone level topology tells how zones are connected together.

There are two types of Link State Packets (LSP) – node LSP and zone LSP. A node LSP of a node contains its neighbour node information and is propagated with the zone whereas a zone LSP contains the zone information and is propagated globally. So each node has full node connectivity knowledge about the nodes in its zone and only zone connectivity information about other zones in the network. Therefore with zone id and the node id of a destination the packet is routed based on the zone id till it reaches the correct zone. Then in that zone, it is routed based on node id. A zone id of the destination is sufficient for routing so it is adaptable to changing topologies.

Critique of Zone-based Hierarchical Link State Routing Protocol (ZHLS)

ZHLS could perform better in specific zones but it is difficult to maintain consistency across the network. The protocol to some extent can provide a better solution in terms of reducing communication overhead and delay, but this benefit is subject to the size of a zone and the dynamics of a zone. It is expected that an increase in the size of network ZHLS could create an unpredictable large overhead.

Efficient connectivity among various zones is itself an issue. Therefore if connectivity among mobile nodes in a zone is sound, it could be expected that the situation in other zone or the worst case in neighbouring zone is not good enough. ZHLS proposed two different types of link state packets. In order to keep all nodes updated frequent propagation of this information is needed. Therefore, nodes should be capable of differentiating among various types of packets. That makes whole issue a bit complicated for the nodes. Engaging nodes in more jobs could affect their ability to respond to various network packets and consume node resources.

3.3.13 Hierarchical State Routing (HSR)

The characteristic feature of Hierarchical State Routing (HSR) is multilevel clustering and logical partitioning of mobile nodes. The network is partitioned into clusters and a cluster-head elected as in a cluster-based algorithm. In HSR, the cluster-heads again organize themselves into clusters and so on. The nodes of a physical cluster broadcast their link information to each other. The cluster-head summarizes its cluster's information and sends it to neighbouring cluster-heads via a gateway. These cluster-heads are members of the cluster on a level higher and they exchange their link information as well as the summarized lower-level information among each other and so on. A node at each level floods to its lower level the information that it obtains after the algorithm has run at that level. So the lower level has hierarchical topology information. Each node has a hierarchical address. One way to assign a hierarchical address is the cluster numbers on the way from root to the node. A gateway can be reached from the root via more than one

path so a gateway can have more than one hierarchical address. A hierarchical address is enough to ensure delivery from anywhere in the network to the host.

In addition, nodes are also partitioned into logical sub-networks and each node is assigned a logical address. Each sub-network 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. The transport layer sends a packet to the network layer with the logical address of the destination. The network layer finds the hierarchical address of the hierarchical address of the destination LMS from its LMS and then sends the packet to it. The destination LMS forwards the packet to the destination. Once the source and destination know each others hierarchical addresses they can bypass the LMS and communicate directly. Since logical address/hierarchical address are used for routing it is adaptable to network changes.

Critique of Hierarchical State Routing (HSR)

Continuously changing hierarchical addresses makes it difficult to locate and keep track of nodes[87]. This makes it difficult to achieve routing at a lower expense. It is expected that most of the time nodes will be busy locating different addresses. This also requires nodes to advertise their routes on a frequent basis. It has been mentioned before that such scheme adds an extra burden on available network resources. Moreover, absence of efficient maintenance and error recovery mechanisms could also pose additional requirements in the address management of HSR.

3.3.14 Cluster Based Routing Protocols (CBRP)

In Cluster Based Routing protocol (CBRP) [73] the nodes are divided into clusters. When a node comes up, it enters the undecided state, starts a timer and broadcasts a hello message. When a cluster-head gets this hello message it responds with a triggered hello message immediately. When the undecided node gets this message it sets its state to member. If the undecided node times out then it makes itself the cluster-head if it has bi-directional link to some neighbour otherwise it remains in undecided state and repeats the procedure again.

Each node maintains a neighbour table. For each neighbour, the neighbour table of a node contains the status of the link (uni- or bi-directional) and the state of the neighbour (cluster-head or member). For each neighbour cluster the table has entry that contains the gateway through which the cluster can be reached and the cluster-head of the cluster. When a source has to send data to a destination it floods route request packets but only to the neighbouring cluster-heads. On receiving the request a cluster-head checks to see if the destination is in its cluster. If it finds one, it sends the request directly to the destination else it sends it to all its adjacent cluster-heads. In CBRP routing is done using source routing. In forwarding a packet if a node detects a broken link it sends back an error message to the source and then uses a local repair mechanism. In

the local repair mechanism when a node finds the next hop is unreachable it checks to see if the next hop can be reached through any of its neighbours or if hop after next hop can be reached through any other neighbour.

Critique of Cluster Based Routing Protocols (CBRP)

CBRP [73] and all those who focus on achieving routing in small partitions of a network face the same type of problems. One important issue is connectivity among individual clusters. Network formation in such design is another issue i.e. how nodes will be allocated to different clusters or in zones such as in ZRP. It is mentioned in the specification of CBRP that a new node joining inside a cluster is based on broadcasting a message. But it is not clear how a node know in advance which cluster it wants to join. Moreover if the node receives replies from more than one clusters then how it will make its joining decision. Likewise in the case of clusters what scheme CBRP utilizes to enable all the cluster-heads to be aware about all other cluster-heads in the network. The specification details some error recovery mechanism but is silent about issues such as link satiability between clusters. CBRP like any other protocols for mobile ad-hoc network needs to focus on some conventional issues besides routing. Some well known issues are limited resources and dynamic nature. However, the specification of CBRP is silent on discussing the impacts of these issues.

3.3.15 Hybrid Routing Protocol

Hybrid routing protocols divides a set of nodes into zones in the network topology. Then, the network is partitioned into zones and a proactive approach is used within each zone to maintain routing information. Hybrid routing adopts a reactive approach to route packets between different zones. Therefore, in hybrid schemes a route to a destination that is in the same zone is established without delay while a route discovery and a route maintenance procedure is required for destinations that are in other zones. The zone routing protocol (ZRP) zone-based hierarchical link state (ZHLS) routing protocol and distributed dynamic routing algorithm (DDR)[48] are three hybrid routing approaches.

Critique of Hybrid Routing Protocol

The hybrid protocols can provide a better solution in terms of reducing communication overhead and delay. But this benefit is subject to the size of a zone and the dynamics of a zone. Therefore with the increase of network size HRP could create an unpredictable large overhead. This poses a limitation to the overall adaptability of HRP. Ideally zone could be bound to have some specific number of nodes to obtain consistent results. But this is not possible in a more practical environment. The hybrid approach provide a compromise on scalability issue in relation to the frequency of end-to-end connection, the total number of nodes and the frequency of topology change. Thus, the hybrid approach may not be a suitable approach for routing in some types of network.

3.3.16 Distributed Dynamic Routing Algorithm (DDR)

Distributed dynamic routing protocol (DDR) [48] constructs a network from a network topology where each tree of the constructed network has to be optimal. Each tree of the constructed network forms a zone. Once the network is partitioned into a set of non over-lapping dynamic zones each node calculates periodically its zone ID independently. Each zone is connected via the nodes that are not in the same tree but they are in the direct transmission range of each other. So the whole network can be seen as a set of connected zones. Thus each node from a zone can communicate with another node from another zone. Depending on features like node density rate of network connection and disconnection, node mobility and transmission power the size of zone increases and decreases dynamically. Mobile nodes can either be in a router mode or non-router mode regarding its position in its tree. This allows a more efficient energy consumption strategy. Each node is assumed to maintain routing information only to those nodes that are within its zone and information regarding only its neighboring zones.

Critique of Distributed Dynamic Routing Algorithm (DDR)

One of the main drawbacks of DDR is the absence of root concept which can prevent single points of failure. In DDR nodes could either be in router or non router mode. However its specification is silent about how nodes chose one of these modes, likewise, what measure the protocol introduces to assure smooth performance of tree and then the entire network. Finally efficient connectivity among zones is itself an issue.

3.3.17 Landmark Ad Hoc Routing Protocol (LANMAR)

Landmark routing protocol [70] [74] is designed for a mobile ad-hoc network that exhibits group mobility. Namely, one can identify logical subnets in which the members have a commonality of interests and are likely to move as a group. The protocol uses the notion of landmarks to keep track of such logical groups. Each logical group has one dynamically elected node serving as a landmark. A global distance vector mechanism propagates the routing information about all the landmarks in the entire network.

LANMAR works in symbiosis with a local scope routing scheme. The local routing scheme can use the flat proactive protocols mentioned previously. When a node need to relay a packet to a destination within its scope it uses the routing tables directly. Otherwise, the packet will be routed towards the landmark corresponding to the destination's logical subnet, which is read from the logical address carried in the packet header. When the packet arrives within the scope of the destination it is routed using the local tables possibility without going through landmark.

Critique of Landmark Ad Hoc Routing Protocol (LANMAR)

The main limitation of LANMAR is the assumption of group mobility[87].LANMAR guarantees shortest paths only when destinations are within the scope. It poses certain limitations and could perform well in small groups of mobile nodes. That makes it unsuitable for larger network. For remote nodes, though data

packets are first routed towards remote landmarks through shortest paths, extra hops may be traversed before a destination is found. LANMAR is cited in few papers but no further analysis or evaluation was done. Connectivity among different groups is a problem. Likewise node joining and leaving requires separate functions. That further poses additional requirements to the basic structure of the protocol. It is commonly known that network and communication overhead could be reduced in such a scheme. But this is normally possible either in smaller section or in number of groups. Therefore it is difficult to have the same efficiency in different sections of the network.

3.3.18 Geographic Addressing and Routing (GAR)

Geographic addressing and routing allows sending messages to all nodes in a specific geographical area using geographic information instead of logical node addresses. Geocast [106] is designed for group reception, multicast groups for receiving geographic messages are maintained at the GeoNodes. The incoming geographic messages are stored for a life time and during the time they are multicast periodically through assisted multicast address. Clients at GeoHosts tune into the appropriate multicast address to receive the messages.

A geographic destination address is expressed in three ways point circle and polygon. A circle is taken with centre point and radius and polygon is denoted with a list of points. A point is represented by geographic coordinates i.e. latitude and longitude. When the destination of a message is a polygon/circle every node within the geographic region of the polygon/circle will receive the message. A geographic router calculates its service area as the union of the geographic areas covered by the network attached to it. This service area is approximated by a single closed polygon. GeoRouters exchange service area polygons to build routing tables. This approach builds a hierarchical structure consisting of GeoRouters. The end users can move freely about the network.

Critique of Geographic Addressing and Routing (GAR)

GAR used geographical area instead of logical node addresses. For that geographic information is needed. No doubt, additional equipment such as a GPS system is required to gather this information. Therefore, this scheme is suitable for specific situations. Examples of such situations might be a battlefield or in a disaster discovery. Generally mobile users do not have GPS or similar system installed on their system. That limits the scope of GAR and its implementation at a wider scale.

3.3.19 Location Aided Routing (LAR)

The Location-Aided Routing (LAR) [78] protocol is a source based on-demand protocol. It utilizes location information to limit the area for discovering a new route to a smaller "request zone". LAR provides two schemes to determine the request zone. Under scheme one; the source estimates a circular area i.e. an expected zone in which the destination is expected to be found at the current time. The position and the size

of the circle is calculated based on the knowledge of the previous destination location, the time instant associated with the previous location record and the average moving speed of the destination. The smallest rectangular region that includes the expected zone and the source is the request zone. The coordinates of the four corners of the zone are attached to a route request by the source. During the route request floods, only nodes inside the request zone forward the request message.

In scheme two the source calculates the distance to the destination based on the destination location known to it. This distance along with the destination location is included in a route request message and sent to neighbours. When a node receives the request it calculates its distance to the destination. A node will relay a request message only if its distance to the destination is less than or equal to the distance included in the request message.

Critique of Location Aided Routing (LAR)

In scheme one where the expected zone of the destination node is determined on the basis of previous destination information is questionable. It is commonly known that nodes in mobile ad-hoc networks suffer with frequent topology changes where nodes join and leave the network at any time. Therefore there are wider chances that the expectation of finding such a circle gives no results. Under such situations it will then become necessary to repeat the whole procedure until a suitable or the required route is established. Therefore an extra burden will be added on the network. One important factor is node efficiency and their resources. Both efficiency and resource will have negative impact if the nodes are overloaded with various searches, replies, forwarding and receiving packets of various types. Moreover, schemes focusing on attaining routing via network partitioning lack some fundamental issues. Examples of such issues include information consistency and maintenance across the entire network. Likewise coordination among various sections of the network is not something which is easy to achieve. This scheme could be improved by adapting the request zone on the fly by the intermediate nodes of the route request. More flexible forms of request zones may be used. Similarly location information can be piggy backed to any node to keep location information more accurate within the network.

3.3.20 Greedy Perimeter Stateless Routing (GPSR)

Greedy Perimeter Stateless Routing (GPSR) [77] reduces the routing load on per node routing state by using only neighbor location information in forwarding data packets. In GPSR, each node gets informed about its neighbors through beacon messages which are broadcast on a periodic basis. GPSR assumes that sources can determine through separate means the location of destinations and include such location in the data packet header. A node makes forwarding decisions based on the relative position of destination and neighbors.

The routing operation of GPSR is based on two data forwarding schemes i.e. greedy forwarding and perimeter forwarding. Data forwarding scheme is the primary forwarding mechanism while perimeter forwarding is used in the regions where the primary one fails. Greedy forwarding works this way: when a node receives a packet with the destination's location it chooses from its neighbors the node that is geographically the closest to the destination and then forwards the data packet to it. This local optimal choice repeats at each intermediate node until the destination is reached. When a packet reaches a dead end i.e. a node whose neighbors are all farther away from the destination than itself, the perimeter forward is performed.

Critique of Greedy Perimeter Stateless Routing (GPSR)

At the basic level GPSR follows the same pattern as link state routing. GPSR reduces the routing load by storing only next hop neighbors information. No immediate mechanism for route maintenance is proposed. Therefore, absence of any active route maintenance mechanism could degrade the whole idea and efficiency of the protocol. GPSR like many other link state protocols requires neighboring nodes broadcasting link state information at regular intervals. It could be seen as a means of creating network congestion

It is well known that nodes in a mobile ad-hoc network operate on low battery power. That makes conserving battery power as one of the important considerations in protocol design. Under the situation when a node exhausts its battery life it could distract communication structure and affect the normal network operation. GPSR broadcasts packets at regular intervals and thus requires nodes to be awake most of the time. It could produce a negative impact on network resources. Having two different mechanisms for data forwarding adds extra burden on a node. That could result in slowing down node responses to different requests and therefore affect network data delivery and performance.

3.3.21 Adaptive Distance Vector Routing (ADV)

ADV, the adaptive Distance Vector Routing Algorithm[83] is known as a combination of proactive and reactive type of protocol. The main characteristic is proactive since routes are maintained all the time. ADV is based on two key characteristics to implement its on-demand character i.e. only routes to active receivers are maintained and the frequency and size of routing updates is adapted to the current network conditions.

In ADV active receivers must be announced in a broadcast like manner. If a node ceases to be a receiver this must be announced too. Every node keeps a receiver flag for each destination in its routing table to reflect the status of this node. Each node keeps a trigger node in order to adapt the frequency and contents of routing updates to the network load and mobility. This variable can be increased in certain steps depending on the events that the nodes receive. There are two thresholds the first is a dynamic threshold which is

computed on the recent past and the role of the node. If this dynamic threshold is exceeded a partial update is scheduled. The second threshold is a fixed constant TRGMeterFull which will trigger a full update if it is reached. The trigger meter is reset after each update.

Critique of Adaptive Distance Vector Routing (ADV)

Unlike some of the above discussed protocols ADV adopts more complex structure. It is a mixture of proactive and reactive protocols. ADV maintains a consistent overview of the network as it is common in many proactive protocols. Therefore the drawbacks to some extent are similar as for table driven protocols. However, instead of maintaining full view only routes to active receivers are stored. There are certain limitations due to the design structure of ADV. It is not necessary that active receivers for one transmission will be available for another one. The harm is if a packet is routed via a stored list, it may or may not lead to the destination. In other words transmission via this approach could yield the same result as it is common in some other protocols of similar types.

One final point is consumption of node battery power as in most of the cases it is already limited. Addition of update packets apart from regular data packets will force mobile nodes to be in receiving mode all the time. Likewise a node could meet a situation when it decided to act selfishly by discarding a genuine request. It could be for any reason such as a node may decide to save its power. Under such situations it also poses a security hole for an active attacker and can put the whole network at risk. It may be noted in most of the schemes discussed so far less attention has been paid to the security issue.

3.3.22 Core Extraction Distributed Ad Hoc Routing (CEDAR)

In CEDAR a backbone of the network is formed through the selection of a subnet of the nodes in the network. This structure is later used to broadcast messages therefore no flooding is required. The message sent over the core network increases waves and decreases waves which notify about an increase or decrease of available bandwidth. The propagation of these waves depends on the amount of available bandwidth and dynamically adjusted. This helps in quality of service which is disseminated in an efficient way. A node contacts its dominator with a route request that contains source destination and required bandwidth. The dominator calculates a quality of service route if this is feasible and then continues to establish it. This includes possible discovery of the dominator of the destination and a core path to it.

Critique of Core Extraction Distributed Ad Hoc Routing (CEDAR)

In CEDAR [81] the selection of nodes for the sub-net could be a problematic issue. Moreover on one hand it could create considerable delay before a network is formed. On the other hand, there is no guarantee that through such schemes the entire network could be covered. Likewise, a specific mechanism is required to handle all joining and leaving requests from individual nodes. It has to be done through packet transmission. It could also result in addition of extra update or similar type of packets. These packets could

be a means to add a further burden on available bandwidth, which could be a means to create network overhead. One final point is that most of the schemes that are based on network partitioning to achieve routing suffer with one or more similar problems. One such problem is consistency. Ideally, this sort of scheme is more suitable for a small network of few nodes.

3.3.23 Distributed Spanning Tree Protocol (DST)

DST [58] considers the variation of different regions in mobile ad-hoc networking environments. DST proposed the establishment of a backbone network in the stable regions using a spanning tree algorithm. For the unstable regions a flooding or a shuttling approach is used to transmit the packet to the destination even through a very unstable area.

Critique of Distributed Spanning Tree Protocol (DST)

DST provides routing only in stable area. Moreover, it requires time before a clear view about the stable region could be established. In most of the cases, nodes require connection with other nodes or at least with nodes of interest. It is not possible in DST as the selection of stable regions requires time. DST is described in [58] and have compared in some places against pure flooding. However there was no comparison with other protocols. Moreover, the comparison focuses on some of the small protocols and no comparisons have been done with some of the prominent protocols. Therefore it is difficult to add any further comments.

3.3.24 Flow Oriented Protocol (FORP)

FORP [56] is deigned for real time traffic flows. Like on-demand protocols, traffic flow is requested first and can be used after. In FORP, each link has a Link Expiry Time (LET) and the minimum of all LETs for all links in a route gives the Route Expiry Time (RET). The destination sends a Flow-HANDOFF message which triggers another Flow-REQUEST thus finding a new route over which the current flow can be rerouted without interrupting it.

Critique of Flow oriented protocol (FORP)

FORP [56] is very similar to some other on-demand protocols. Therefore the draws back in the general sense are the same as in some other on-demand protocols. No specific procedure is followed to reduce the power consumption which otherwise will be busy in receiving and forwarding flow requests. Likewise no precautions have been taken to avoid message looping. Moreover, the whole scheme of flow requests without proper check could cause network overhead. Finally, no further work outlining FORP's performance or comparison with other protocols is reported in the scientific literature.

3.3.25 Fuzzy Sighted Link State Algorithms (FSLS)

FSLS also focuses on the problem of limited dissemination of link state information. Links state information is sent with dynamically limited time-to-live and in certain intervals. It further depends on the number of hops the updates can travel. Far reaching link state information messages are sent much less

frequently than short reaching link state information messages. Also these messages are only created if the state of a link has changed within the scope of the LSU (Link state unit). The length of the intervals and scope of the LSUs is the design parameter of the class of FSLS algorithms. An extreme case is the discrete link state algorithm DLS in which each LSU is sent through the whole network. It differs from standard link state only in the fact that the LSU is not sent immediately after a link status changes but at the beginning of the next interval.

Critique of Fuzzy Sighted Link State Algorithms (FSLS)

It would be difficult to establish stable routing throughout the network via FSLS. Maintaining limited information could also mean offering limited routing. Moreover, it is always an issue to achieve same data delivery in different sections of the network. To some extent the protocol also relies on updates. In the case of a mobile ad-hoc network where topology changes happen quite frequently, it is hard to maintain updated topology information without generating network overhead. Moreover, this sort of schemes could also cause mobile nodes to be engaged all the time. Engaging nodes throughout the network life could result in nodes exhausting battery power, an extra burden on the available bandwidth and degradation both in nodes efficiency and data delivery. No further work and comparison of FSLS is reported in the cited literature.

3.3.26 Lightweight Mobile Routing (LMR)

Lightweight mobile routing (LMR) [159] is a link reversal routing protocol. Its operation depends on three basic messages i.e. query, reply and failure query. A query message is sent by the source node via limited broadcast. The source then waits for a reply packet which is issued by a node which has route to the destination. The directed flood caused by the reply messages forms a directed acyclic graph rooted in the originator of the reply. The route itself and the up and down stream links formed depend on the order of the reply transmissions. If a node loses its last route to the destination and it has upstream neighbors a failure query is broadcast to erase invalid routes. On reception of a failure query the node may either transmit a reply or another failure query if its last link was erased by the first failure query. So instead of a direct link reversal LMR erases the links and sets new links. Loop freedom is ensured by marking previous unassigned links as downstream-blocked if the node has already an upstream link. These markers time out after a while but it may happen that a downstream link cannot be used because of possible loop formation. Likewise to avoid deadlock a similar mechanism is used.

Critique of Lightweight Mobile Routing (LMR)

Limited broadcast in LMR may also mean that routing is in a limited area. To some extent it could also improve different performance metrics. But LMR limits the network coverage and is not well suited for a larger network. Moreover, too many route queries could pose additional load on the network. Likewise the same factor could also be seen an additional burden on the limited network resources. LMR is cited in some

of the available literature but mainly as a reference. The protocol lost interest with the development of TORA as a successor.

3.3.27 Link Reversal Routing (LRR)

LRR [99] is designed specifically to aid routing in a highly dynamic network. One of the main objectives is to minimize the amount of overhead. In situations when topology changes need to be announced the maintained topology is reduced to a directed acyclic graph rooted in the destination. This graph is used to direct each link as either upstream or downstream to the destination. If a node in the graph becomes a local minimum i.e. it has no downstream, one of its links is reversed. To achieve this, the notion of height is introduced thus the problem is similar to flows in a graph. The height of the minimum node is raised such that it is higher then the lowest of its neighbors thus reversing the direction of this link. The reversal can cause another node to become a minimum and the process continues.

Critique of Link Reversal Routing (LRR)

In LLR no node knows about the distance of itself to the destination. Therefore optimizing metrics used in distance vector or link state algorithms cannot be used. This limits the adaptability of this approach at a wider level. Moreover, in the light of current specification of LLR, it could easily be concluded that the scheme could produce results in a small area of few nodes. However, it's difficult to predict anything about a network with wider coverage. Thus LLR is well suited for small networks.

3.3.28 Topology Broadcast Based on Reverse Path Forwarding (TBRPF)

Based on the reverse path forwarding algorithm TBRPF [59][68] is one of the table driven or proactive link state protocols. Unlike traditional table driven protocols, TBRPF maintains a spanning tree in each node for each other node as the source. Each parent of the source node is responsible for this tree formation. A list of parents is kept at each node for every other node as well as full topology table including cost and sequence number of each link the node is aware of. The topology update messages are sent along these spanning trees but in the reverse direction. TBRPF supports only bidirectional links. The topology updates are transmitted reliably. Very similar to table driven protocols, a hello message is used for neighbor's detection. This hello message also contains a list of router IDs and a sequence number such that each node can maintain its neighbor table. TBRPF also transmits updated information which contains details of any changes in the router list.

Critique of Topology Broadcast Based on Reverse Path Forwarding (TBRPF)

The main problem in most of the schemes similar to TBRPF is the formation of the spanning tree. A considerable amount of time is required to form the spanning tree in each node. Moreover extra efforts are needed to maintain all such trees. Another aspect is the little consideration that has been given to address the dynamic nature of mobile ad-hoc networks. Use of hello messages in TBRPF could reduce a node's individual performance. Likewise it could also be a means of reducing node and network limited resources.

3.3.29 Terminode Routing (TLR/TRR/AGPF)

Routing between terminodes [161-163] is a hybrid process that routes packets based on the geographic position. The destination address is called location dependent address (LDA). From this LDA the closest friend-node is calculated and the packet is delivered to it. Terminodes use the concept of a virtual home region which is the same for some approach. In other words for each node there exists such a home region which is specified by a fixed position and a radius. The region can be calculated by a hash function over the node's id. Each node within the virtual home region of a certain node must maintain the current position of this node so that other nodes can obtain it.

The position based routing method is called Anchored Geodesic Packet Forwarding (AGPF). To avoid running into a maximum the route is oriented on set anchors along the path. An anchor is just a specific location. The anchored path is determined by the source using Friend Assisted Path Discovery (FAPD) and included into the packet. FAPD is based on small world graphs. Alternatively the path can be determined by Data Requirement Delivery (DRD) which just sends the packet to a set of neighbors whose angle is the smallest to the right direction. The local routing method is no longer based on position information but only on a unique node identifier of the target id. Two hops neighborhood information is maintained by each node by using hello packets. If the neighborhood is known and a packet can utilize local routing target to the node which received the packet, a path discovery is initiated to direct the packet to the destination.

Critique of TLR/TRR/AGPF

The protocol utilizes a number of different concepts of some of the earlier proposed schemes to offer routing. It uses hello messages to maintain two hops neighbor information; similarly it relies on a path discovery mechanism to direct the packet to the destination. Chances are both of these functions will be used extensively due to the dynamic nature of mobile ad-hoc networks. It could result in unnecessary network resource consumption. Likewise it could also drop overall network data delivery and thereby efficiency. It is clear from the above mentioned specification that the scheme suits smaller networks. The main reason is the difficulty to disseminate information across the network within the design frame of this protocol.

3.3.30 Witness Aided Routing (WAR)

Witness Aided routing [61] makes use of the possibility to overhear a transmission in range of a node on a wireless channel in a unique way. A node capable of overhearing a transmission from one mobile host to another over a relay can act as a passive witness for that transmission. In situations when a relay is not able to reach the destination witness node i.e. node can overhear transmission becomes an active witness and tries to deliver the packet on behalf of the relay node, thus saving the packet even if the original route failed. Because many nodes can be witnesses of a certain transmission special care is taken to avoid contention.

The goal is to perform just one single successful delivery. To achieve this each witness host which intends to deliver the packet must get permission from the destination host. To get this permission the node sends a request to the destination host. If the target host did receive the packet before by the relay the request will be rejected otherwise the set of witnesses will be polled by the target until the packet could be successfully delivered.

The route discovery mechanism of WAR is very similar to DSR with the enhancement of multiple route selection criteria. The target can be instructed to wait a certain amount of route requests or to wait for a certain time period and then choose the route to answer the route discovery according to some specified criteria. Alternate routes can be remembered to have them ready if the first choice breaks. Similar to DSR, WAR uses source routing to forward packets. Only that delivery is regarded as successful for which a forwarding node receives an acknowledgement from either the intended relay node or from any witness. If not the route is considered broken and a route discovery process is initiated. Just like DSR the source route information in a relayed packet can be used to update local information.

Critique of WAR

The main difficulty is the information about nodes that can overhear a transmission. Even if the node is identified there are many reasons as to why this scheme might not work well. There are reasons as to why a node refuses to act as an intermediate node. One of those reasons is its own interest by conserving limited battery power for personal use. However, at present the protocol features are silent to address this issue. For instance if a node is agreed to perform such a service, it is an issue as to how long such a node can act. Moreover, having single or few nodes to cover the entire network is not easy to achieve specially in the context of mobile ad-hoc networks. WAR also makes use of a route discovery process which may be a means of generating extra network overhead. These factors pose a limit to the overall performance of WAR.

3.3.31 Geographic Distance Routing (GEDIR)

GEDIR uses an approach based on progress to select the set of neighbors. This set of neighbors is then use to forward the message to describe a set of related geographic routing protocols.

Critique of Geographic Distance Routing (GEDIR)

Topology and efficiency of both mobile nodes and networks as a whole varies throughout the network life of mobile ad-hoc networks. Any attempt to record such information would be a costly issue. Moreover, establishing routing based on the stability of mobile nodes may not be an impressive idea in the context of mobile ad-hoc networks. GEDIR also requires extra hardware which posed additional requirements to the protocol.

3.4 Summary

This chapter presented a critical evaluation of different routing schemes for mobile ad-hoc networks. In section one all those protocols that have been currently under consideration by IETF were evaluated. Section 2 presented protocols that follow the nearest approach to MAODDP before other schemes. There are different conclusions that can be drawn from the discussions made in this chapter. Among these conclusions some are of general type while the rest vary from one to the other. In general most of the schemes lack practical implementations. Moreover, those who have been implemented are limited to a particular environment. Lack of studies about these schemes is also an issue. Apart from some of the main schemes existing literature are silent about most of the schemes discussed in this chapter. That makes it harder to evaluate these schemes in comparison with some of the schemes that follow some operational pattern. This fact also poses an additional obstacle in their further development. It has been mentioned in the chapter background research of this thesis that mobile ad-hoc networks suffer with different issues. Some of the most prominent issues are bandwidth constraints and limited power of mobile devices. Most of the schemes mentioned above clearly lack in handling this and some other issues. Therefore there is definitely a need of a routing solution that can not only offer a better routing but also address some of the other routing related issues.

In the following chapter the specification of Mobile Ad-hoc on-Demand Data Delivery Protocol (MAODDP) will be presented. Details of various terminology, operations and features of the protocols are also discussed. In this regard, various operations are presented in a manner which can illustrate the difference between MAODDP and some other famous schemes.

CHAPTER 4. MOBILE AD-HOC ON-DEMAND DATA DELIVERY PROTOCOL (MAODDP) SPECIFICATION

4.1 Introduction

This chapter presents the specification of the Mobile Ad-hoc On-Demand Data Delivery Protocol (MAODDP). In this chapter some of the issues interrelated with routing are also highlighted. Our research findings [3] show that most of the existing routing mechanisms [25] address routing without considering their effects on some other issues such as the bandwidth and the limited battery power of mobile devices in mobile ad-hoc networks. The formation of MAODDP[2] is based on this very same idea. It is a simple multi-hop routing protocol to establish routing while considering some other routing related issues as shown in Fig.4.1. In Fig.4.1 the basic model of MAODDP routing has been taken as an integrated problem while some others problems have been shown as interrelated with the routing.

MAODDP offers self starting; loop free routing among various hosts of a mobile ad-hoc network. The key feature of MAODDP is the route establishment and data delivery one after the other [41, 164]. MAODDP requires no periodic updates of any kind at any level within the network. MAODDP enables mobile nodes to identify route breakage or expire routes so that such routes could be marked as invalid using the route error message.

This chapter has been organized as follows. Section 2 presents an overview of MAODDP which includes an introduction to different terminology. In section 3 a detailed specification of MAODDP is presented. Section 4 highlights protocol operations and features while a brief summary of this chapter is given in section 5.

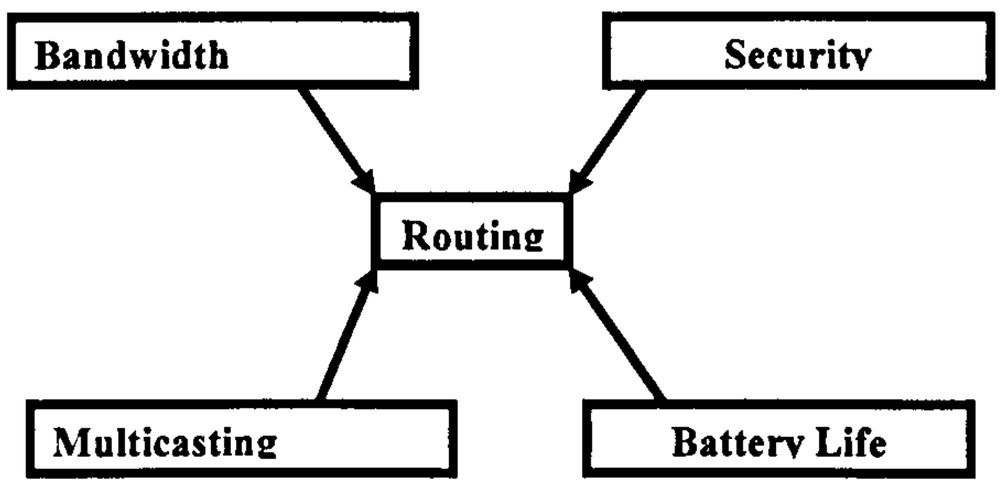


Figure 4.1 Basic design model of MAODDP

4.2 Protocol Overview

MAODDP, as shown in figure 4A, follow a specific pattern of operations to support routing in mobile ad-hoc network.

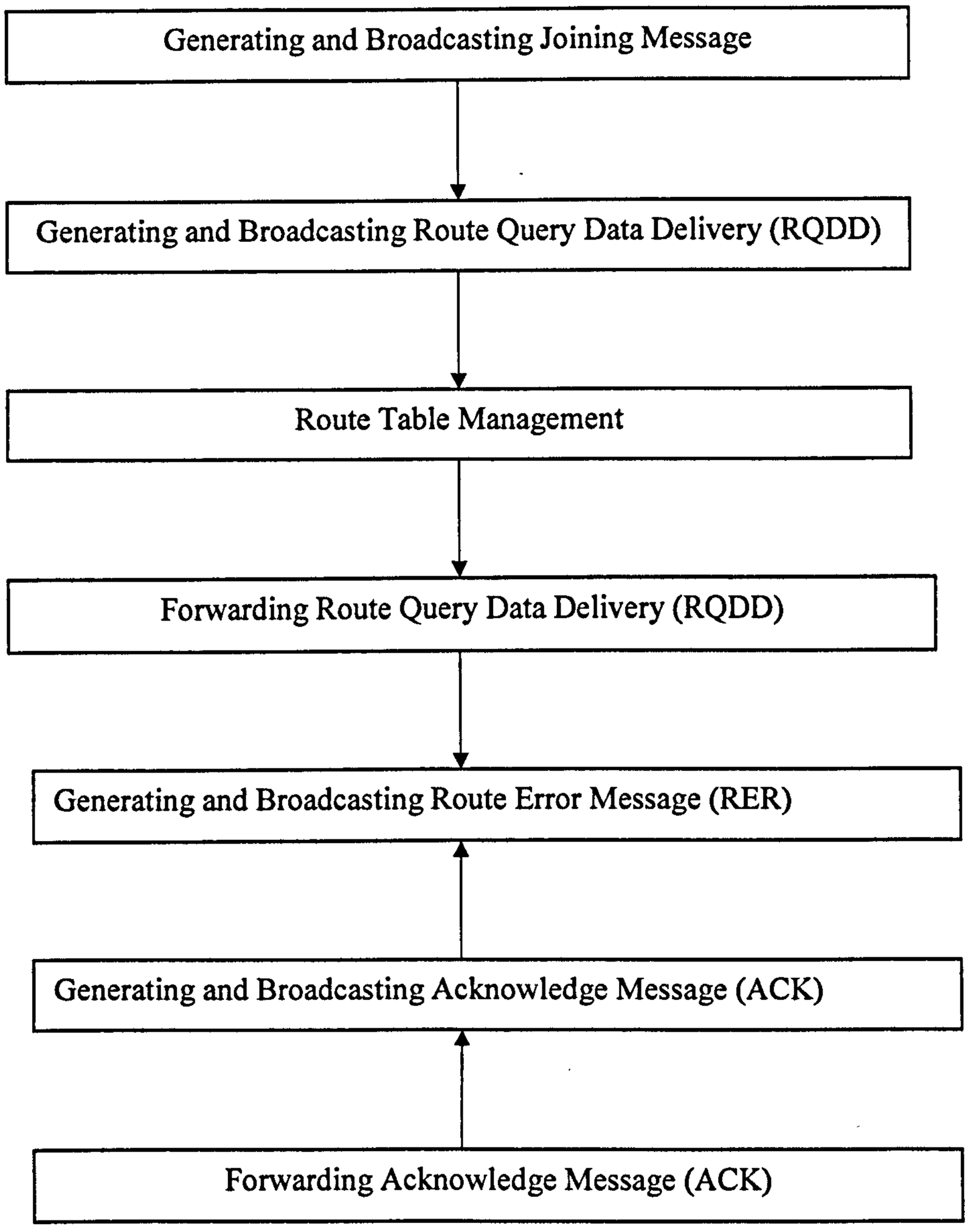


Figure 4A. MAODDP operations model

MAODDP is pure reactive or on-demand protocol [41, 50], as shown in Fig.4.2. Where source node 'S' broadcasts route request data delivery (RQDD) packets for destination D.

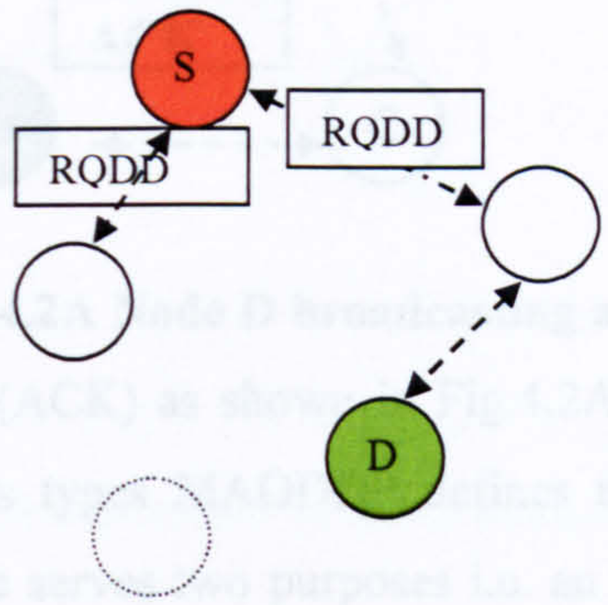


Figure.4. 2. Node S broadcasting RQDD packets for nodes D

In MAODDP, a Joining message is broadcast to form a mobile ad-hoc network. All nodes who want to be part of the mobile ad-hoc network are required to broadcast this message. Information such as node sequence number, IP address, route expiry time and hop-counter fields are part of the joining message. Information contained in the joining message serves as a starting point for initializing routing tables.

The hop-counter inside the 'joining message' assists mobile nodes to locate their next-hop neighbour and the distance between two nodes in the mobile ad-hoc network. The hop-counter value increases as it reaches another node in the network. Therefore, if there are four nodes in a mobile ad-hoc network as shown in Fig.4.2. with node S being the first and node D is the last then by the time the joining messages broadcast by node D reaches node S the hop-counter value should be '2'. It is due to the fact that the hop-counter is assigned the value '0' at the time of initializing any broadcast packet.

Data gathered through the "Joining message" if needed could also be used to transmit information from one node to the other node as long as the route is valid. However for destinations where the source node finds either no route or an expired route, it broadcasts a route query and data delivery packet (RQDD). From the application point of view MAODDP regards the RQDD packet as a part of its route query and data delivery process. Details of RQDD and its structure are covered in the later section of this chapter.

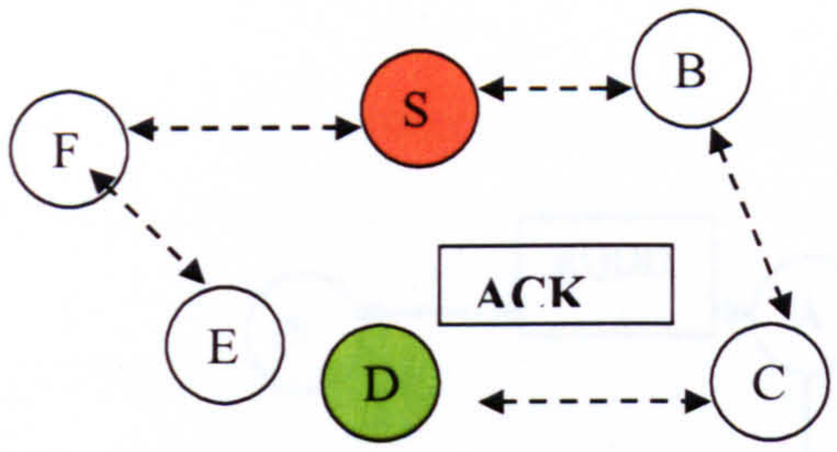


Figure.4.2A Node D broadcasting acknowledge message

The Acknowledge message (ACK) as shown in Fig.4.2A and the route error message as shown in Fig.4.2B are some of the messages types MAODDP defines to support different routing operations. In MAODDP an acknowledge message serves two purposes i.e. an indication of successful route delivery and updating routing tables. Route maintenance in MAODDP is achieved through route error (RER) messages very similar to some other techniques[25] of mobile ad-hoc networks.

The route error (RER) message as shown in Fig.4.2B is used to track down different expired, broken or invalid routes. In Fig.4.2B Node F on discovering link breakage to Node E broadcast RER message. MAODDP uses a combination of message broadcast ID and sequence number to avoid message looping. Therefore, nodes are bound to issue a new broadcast ID for each new broadcast. This broadcast ID along with node sequence number is used to determine freshness of the routing packet.

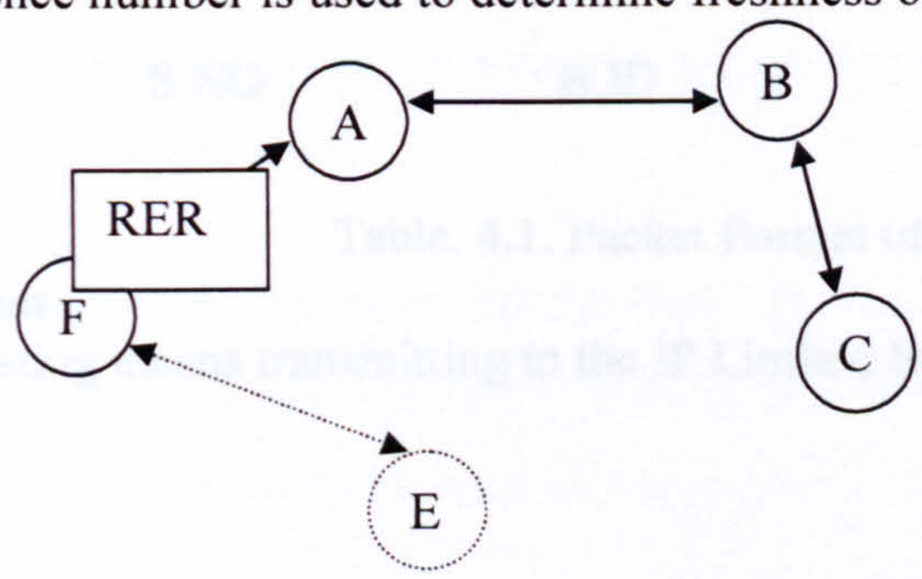


Figure.4.2B Node F broadcasting route error message

4.3 MAODDP Specification

4.3.1 MAODDP Terminology

Different technical terms are used in the specification of MAODDP and wherever possible comparison is made against routing protocols of similar types. In this section a detailed description of each of these terms is presented.

- **Source Node**

Source node is the node which initiates a MAODDP message. For instance, the node initiating a RQDD and flooding the RQDD message is called the source node of the RQDD. In Fig.4.5A source 'S' is broadcasting RQDD to destination 'D'.

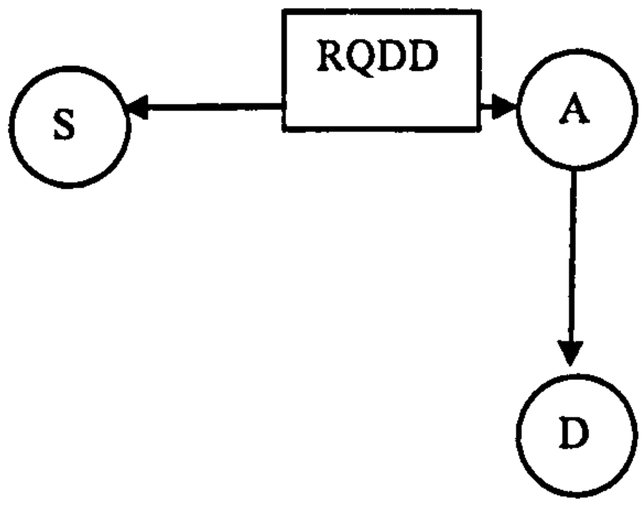


Figure.4.5A.Source S broadcasting RQDD to destination D

- **Joining Message**

When a mobile node wishes to join an already established mobile ad-hoc network or a set of mobile nodes want to form a mobile ad-hoc network they broadcast a 'Joining message'. The Joining message contains information such as node sequence number, broadcast ID which should be zero for all the nodes already joined or joining, the message and hop counter field as shown in Table.1.

In table.4.1. S.NO is the node sequence number, B.ID is broadcast ID, IP.ADD is node IP address , ST is status time which represent message expiry time and H.COUNT is hop counter which calculates the number of mobile nodes between any two nodes.

S.NO	B.ID	IP.ADD	H.COUNT

Table. 4.1. Packet Format of Joining Message

- **Broadcast**

Broadcasting means transmitting to the IP Limited Broadcast address. Broadcasting is useful to enable flooding.

- **Flood**

A Protocol for mobile ad-hoc networks is responsible for the information to be spread throughout the network. That could be done through flooding. In MAODDP a message is flooded across the network along with a broadcast ID.

- **Route Query and Data Delivery Packet (RQDD)**

A RQDD contains route request information and data packets to deliver. Route request information contains the following fields.

- Source Sequence number
- Source IP Address
- Broadcast ID
- Destination IP Address
- Destination Sequence Number

- Hop Count (number of hops needed to reach destination)

-Message life time i.e. status time (ST)

- Hop counter

S.S.NO	S.IP.ADD	B.ID	D.IP.ADD	D.S.NO	H.COUNT	ST	H.COUNTER

Table.4.2. Packet format of RQDD

In Table.4.2. S.S.NO is source sequence number; S.IP is source IP address; B.ID is broadcast ID; D.IP. ADD is destination IP address; D.S.NO is destination sequence number known to source; H.COUNT is the last hop-count known to the source; ST is the message expiry or status time and H.COUNTER is the hop counter.

- **Broadcast ID**

  Each time a node broadcasts a message it generates the broadcast ID i.e. a new broadcast ID must be added for every new message.

- **Sequence Number**

  Sequence number is the number which is generated by the source node at the time of broadcasting various messages. Each broadcast message has broadcast ID and a sequence number attached to it. Broadcast ID and the sequence number are helpful in determining the freshness of the route and to avoid message looping. Therefore, a RQDD with same sequence number and broadcast ID is discarded and is not forwarded to any other node of the network.

- **Status Time**

  Status time is the message expiry time. It helps the node determine the status of an active transmission i.e. if the node does not receive ACK from the destination node before the expiry of the status time it regards the previous transmission as unsuccessful. Under this situation if it wishes it can rebroadcast the RQDD packet.

- **Listening Time (LT) and Active Time (AT)**

  Energy conservation is one of the important aspects of mobile ad-hoc networks[13, 15]. At present different routing schemes adopt various mechanisms to reduce energy consumption. In MAODDP each node is allowed to switch into one of two modes i.e. sleep state and/ or active state.

A node can switch into sleep mode if it does not hear from other mobile node within a time limit known as Listening Time (LT). Likewise, after a specific time interval known as Active Time a node can switch back to active state.

- **Routing Tables**

  Routing tables were first introduced in table driven protocols. In the available literature[40, 45] a number of criticisms are made on routing tables for storing routing information. It is reported[37, 38] that maintaining routing tables could increase extra burden on the network bandwidth. Fortunately most of these

criticisms have been done in the context of table driven protocols[23]. It is well known that these protocols focus on maintaining a consistent view of the network via continuous network updates[34]. It has been mentioned before that MAODDP does not require any periodic updates and all functions of MAODDP work entirely on-demand[27]. Therefore, MAODDP relies on routing tables to store different routing information.

In MAODDP, Information like destination ID, node sequence number; next hop neighbors; hop count to other nodes of the network and message broadcast ID are stored in the routing tables. Among these information broadcast ID is deleted from the routing table if the node has received the acknowledge packet for the message whose ID is recorded inside the routing table or if the node does not hear anything before the expiry of the status time.

- **Neighbor Node**

Node A is said to be a neighbor of node B if node A can hear node B on some interface. Node C is considered as a neighbor node for both A and B if it has a bidirectional link between A and B as shown in Fig.4.6. In Fig.4.6 Node A is a neighboring node of node B while Node C is a neighboring node for both A and B as it has bidirectional links to both A and B. Likewise, Node D is a neighboring node of Node C and vice versa.

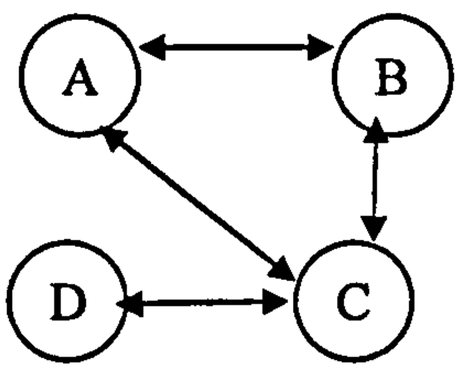


Figure.4.6. A sample ad-hoc network (1)

- **Intermediate Forwarding Node**

An Intermediate forwarding node is the node which agrees to forward packets destined for another destination node, by retransmitting them to a next hop which is closer to the uni-cast destination along a path which has been set up using routing control messages.

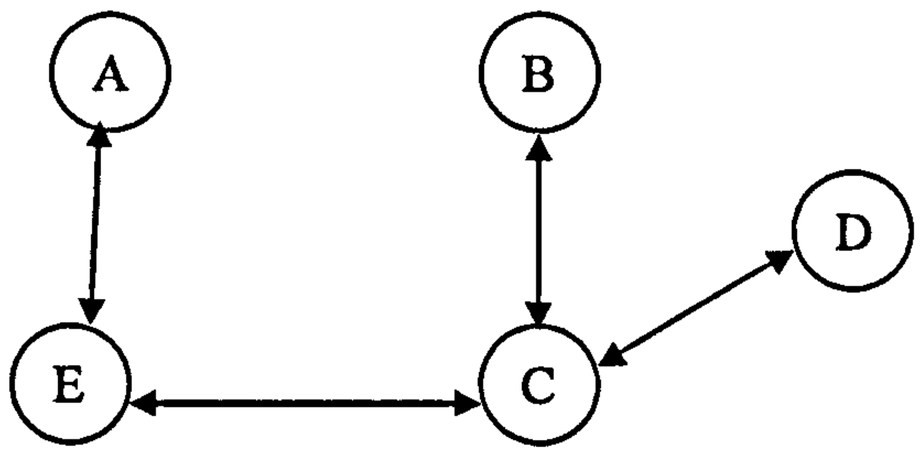


Figure.4.7. A simple ad-hoc network (2)

In Fig.4.7. Node E is an intermediate node in between Node A and Node C. Similarly Node C is an intermediate node for Node B and Node D.

- **Active Route**

An active route is the routing table entry with a finite metric in the hop-count field. A routing table may contain entries that are not active. They have an infinite metric in the hop-count field. Only active route entries can be used to forward data packets. Invalid entries are eventually deleted.

- **Forward Route**

A route set-up to send data packets from a source to a destination is called a forward route. In Fig.4.7. A forward route from node A to node D is possible through node E and node C.

- **Reverse Route**

A reverse route is the route, which is set up to forward an acknowledged message (ACK) back to the source node. In Fig.4.7. A reverse route from node D to node A is possible through node D and node E.

- **Destination Node**

Destination node is the receiver node of RQDD. In Fig.4.7. if node A wants to transfer some data to node D then node D is said to be the destination node.

- **Acknowledge Packet (ACK)**

Packet issued by the destination node on reception of RQDD.

- **Route Error Packet (RER)**

Packet issued by any node of the network informing about a possible link break is known as a route error packet (RER).

- **MAODDP Messages Types**

MAODDP defines four message types. Joining message broadcast for network formation, Route query and data delivery message (RQDD) which a source node broadcasts to find a suitable route to the destination along with a data packet, Acknowledge message (ACK) broadcast by the destination node when it receives the RQDD message destined for it, Route error (RER) message could be broadcast by any of the nodes to inform others about link breakage or a expired route.

- **Multicast Joining Request (MJReq)**

MAODDP supports one of two situations for multicast operations. Details of these operations are covered in the later section of this chapter under multicast operation of MAODDP. Multicast joining request is the process through which a node forwards a request to join a multicast tree as shown in Fig.4.8. In Fig.4.8. Node 'A' sends MJReq to node 'D'.

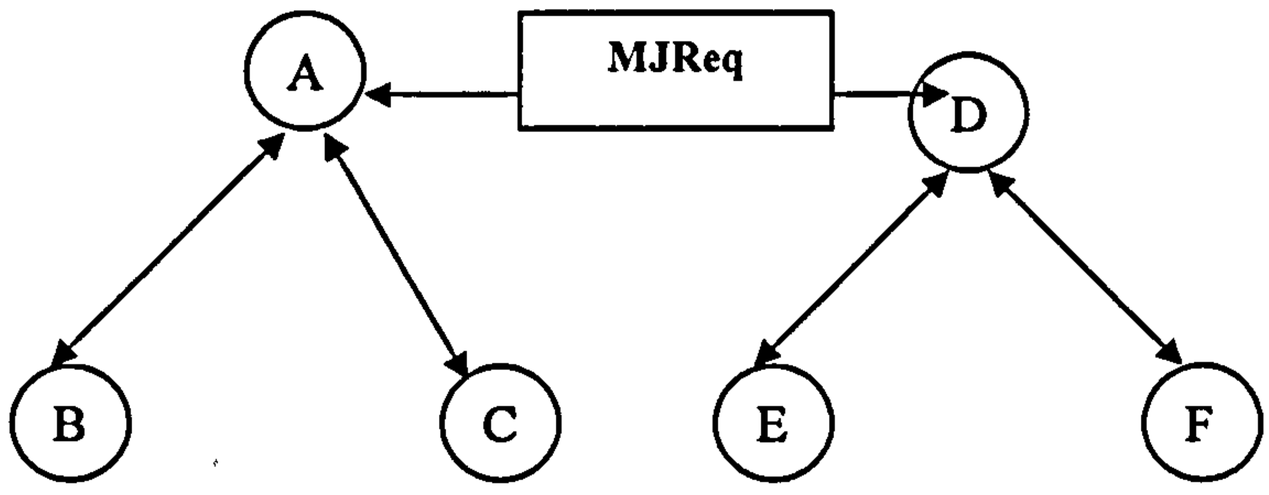


Figure.4.8. Node A sending MJReq to node D

- **Joining Query**

This query is broadcast when a node wishes to join a multicast tree for which it has no valid route.

4.4.2 MAODDP Operations

MAODDP supports different routing operations . Some of these operations are related with network establishment while others support route establishment and route maintenance. Details of these operations are as follows.

- **Broadcasting Joining Message**

Joining message is broadcast in one of two situations. In situation one, as shown in Fig.4.9a. if a number of mobile nodes wants to form mobile ad-hoc network they broadcast join messages which serve as an initial point of contact for the other nodes in the network. In situation two, when a mobile node wants to join an established mobile ad-hoc network, it must broadcast a join message which serves as an indicator that a new node wants to join the network. In Fig.4.9b. Node D is broadcasting Jmessage to join Network A.

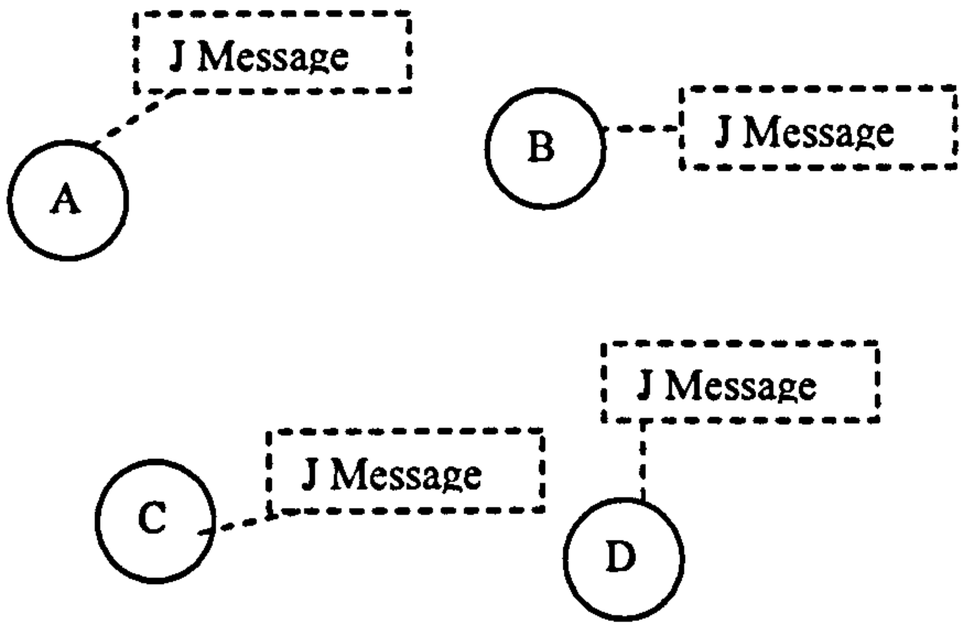


Figure.4.9a. Mobile nodes broadcasting Jmessage to setup mobile ad-hoc network

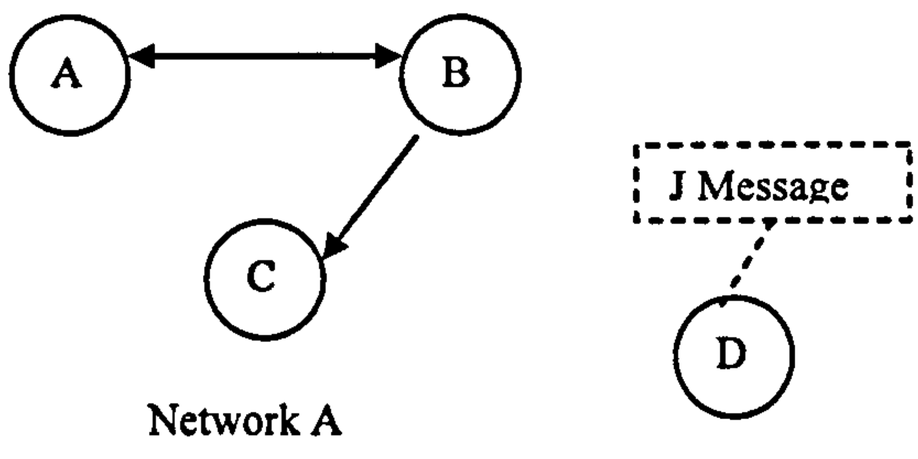


Figure.4.9b. Node D broadcasting Jmessage to join the networkA.

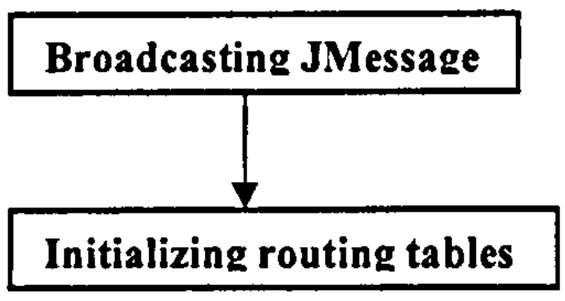


Figure.4.10. Joining message initializing routing tables

Each node on receiving Jmessage initializes routing tables with the information of other nodes as shown in Fig.4.10.

- **Generating Route Query and Data Delivery Packet (RQDD)**
  A node floods RQDD which contain both route query information and data when it wants to deliver packets to a destination for which it has no route information available in its routing table. This can happen if the destination is previously unknown to the node or if a previously valid route to the destination expires or is broken. In the situation when a node does not have any previous information about the destination node,

information such as sequence number gathered through the joining message is used and is considered as the last known destination sequence number for this destination.

The source sequence number in the RQDD is the node's own sequence number. A source node often expects to have bidirectional communications with a destination node. Intermediate nodes in between the source and the destination copy the source sequence number and destination sequence number from the RQDD. That establishes a reverse path from the destination to the source of the RQDD. In Fig.4.11 Source 'A' is sending RQDD via node 'B' to the destination 'C' which is sending the ACK back to source 'A' through the reverse path developed via the RQDD.

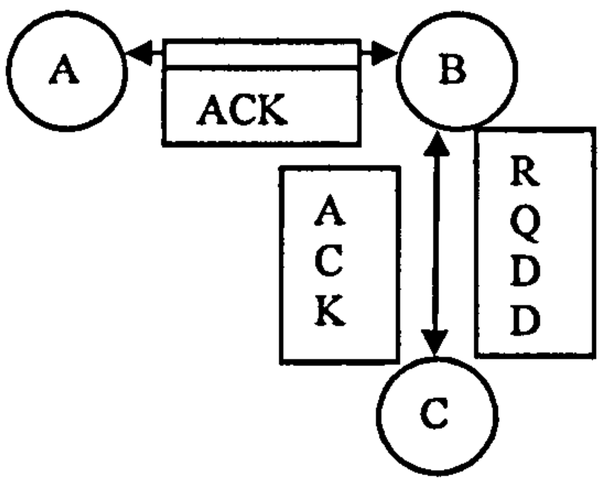


Figure.4.11 Node 'A' broadcasting RQDD for Node 'C' and receiving ACK from Node 'C'

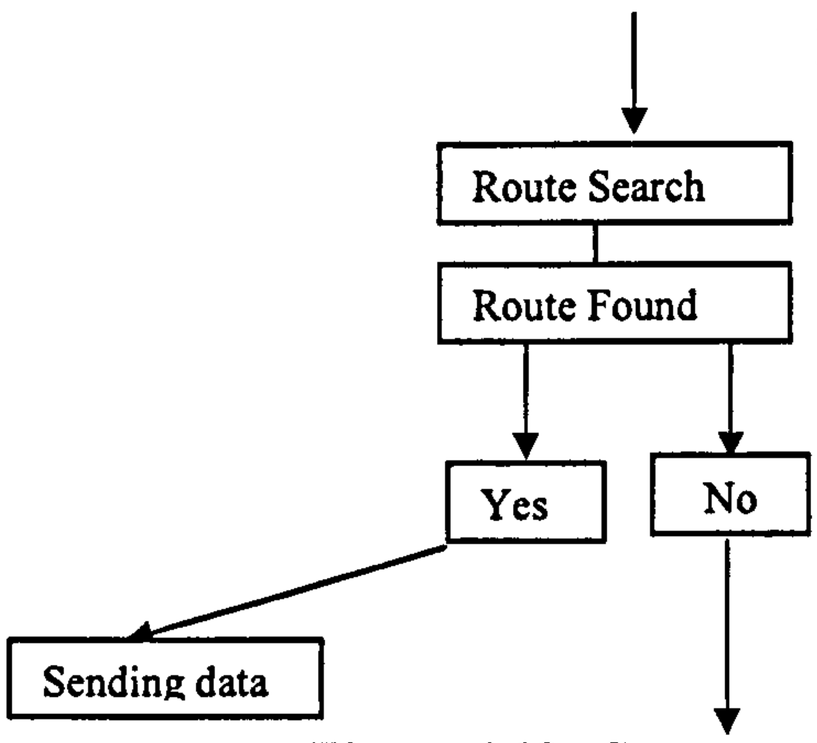


Figure.4.12a Source node performs initial route search

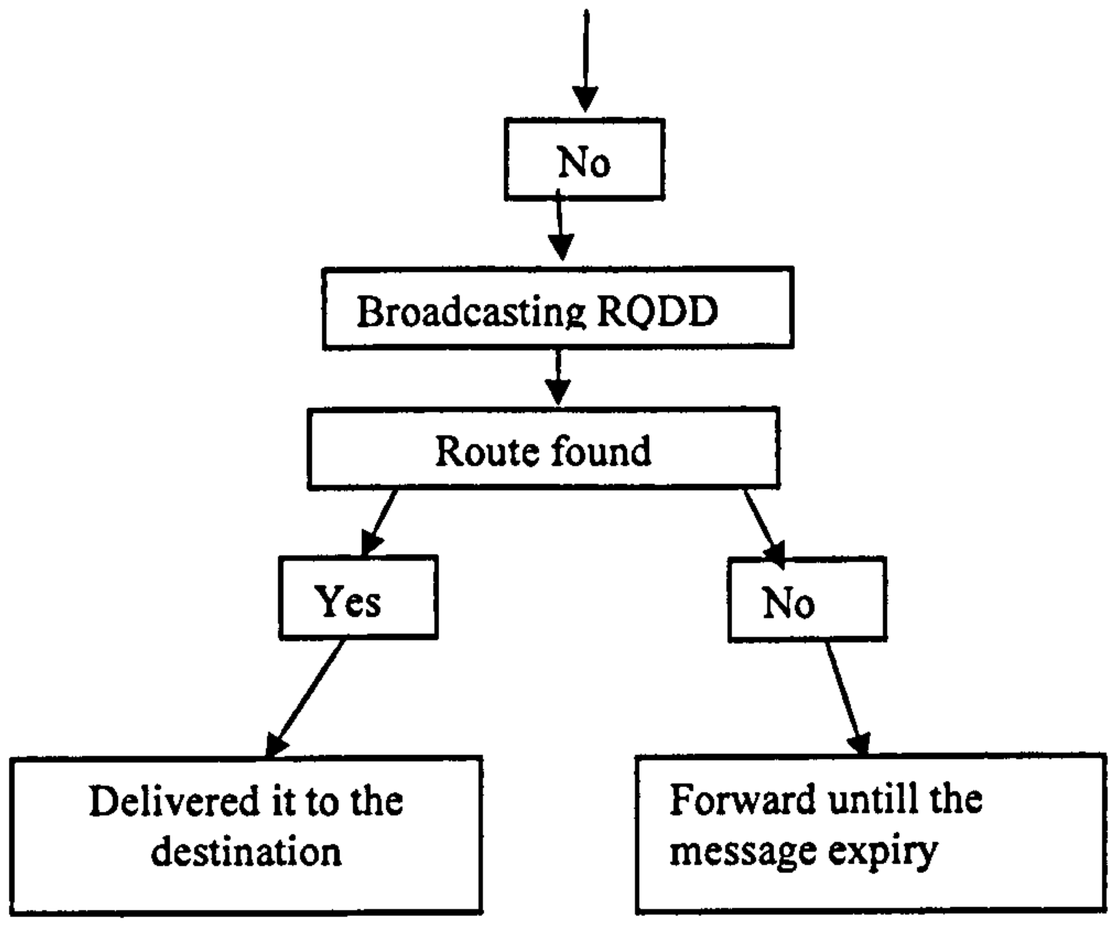


Figure4.12b. Source node broadcasting RQDD

In Figure.4.12a the source node first performs an initial route search that is the search for a route in its routing table. If the route is found the same route is used to transfer data.

In Fig4.12b the source node broadcasts a RQDD if no previous recorded route is found in its routing table. If a route to destination is found in the routing table of an intermediate node it forwards the data via this route to the destination otherwise the process repeat until the RQDD expires.

- **Forwarding Route Query and Data Delivery Packet (RQDD)**

When a broadcast RQDD reaches any intermediate node it must perform number of actions in sequence. Legitimacy of RQDD: whether or not the same packet has arrived before. To find out the freshness of received RQDD, the receiver or intermediate node checks the sequence number and the broadcast ID of the RQDD with the information already stored inside its routing table. If the sequence number and broadcast ID matches the one stored inside the receiver routing table, the node takes no further action and simply discards the packet, as shown in Fig.4.13a.

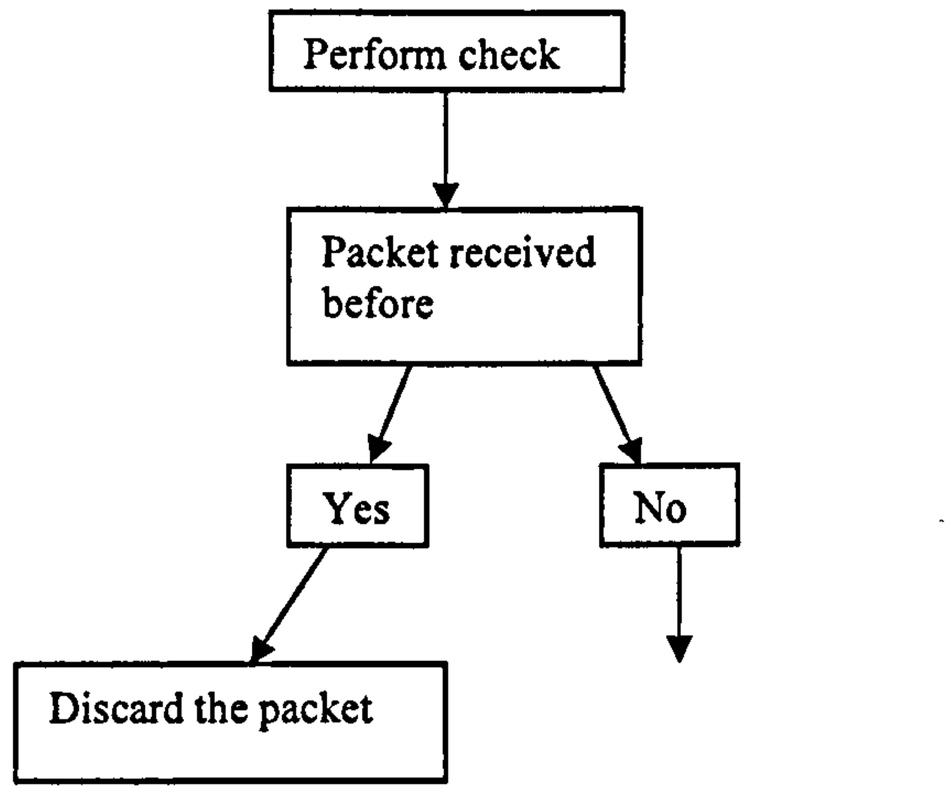


Figure.4.13a. Receiving node of RQDD performing check

Updating routing information: A node updates routing information if it finds no previous information about the received RQDD inside its routing table or if the RQDD contains latest information about the source and the destination node. The whole process is summarized in Fig.4.13b.

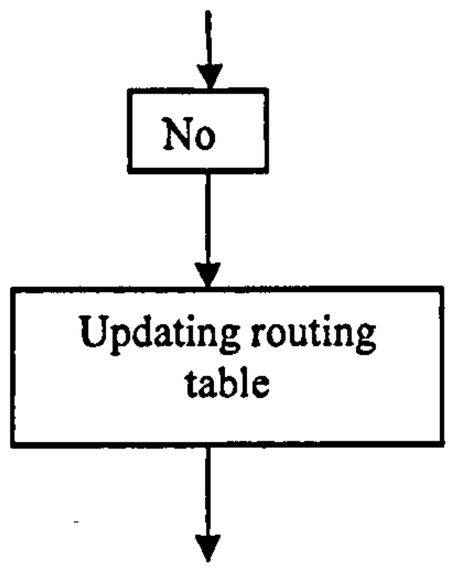


Figure.4.13b. Receiver node of RQDD updates routing table

Forwarding RQDD: In this stage the RQDD receiving node chooses one of three options. If the destination node is its next hop neighbour it forwards the RQDD to the destination as shown in Fig.4.14a. The second option if the node has a fresh enough route to the destination, it forwards the RQDD as shown in Fig.4.14b. Finally if the intermediate node has no information about the destination sought, it forward RQDD to its next hop neighbour as shown in Fig.14c.

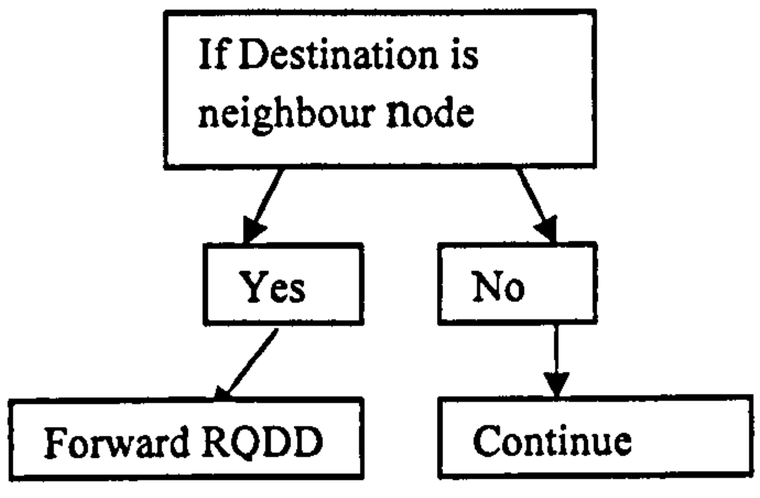


Figure.4.14a. Receiver node forward RQDD destination neighbouring node

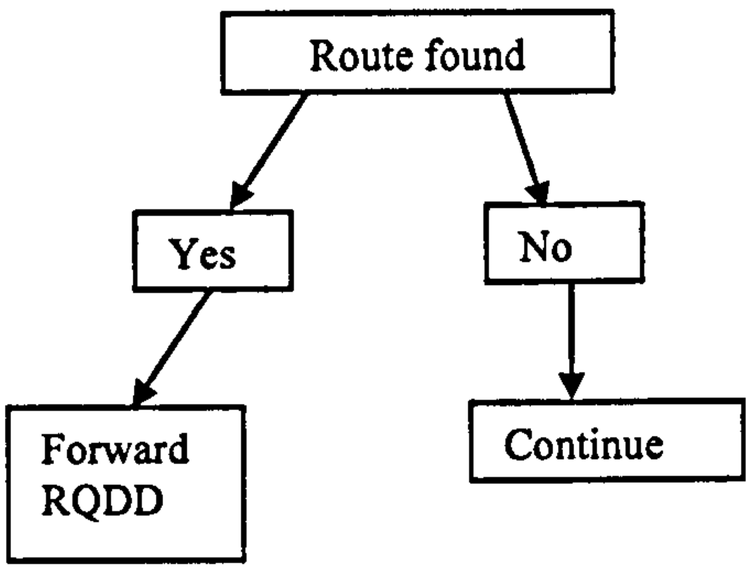


Figure.4.14b. Receiving node forward RQDD using available route

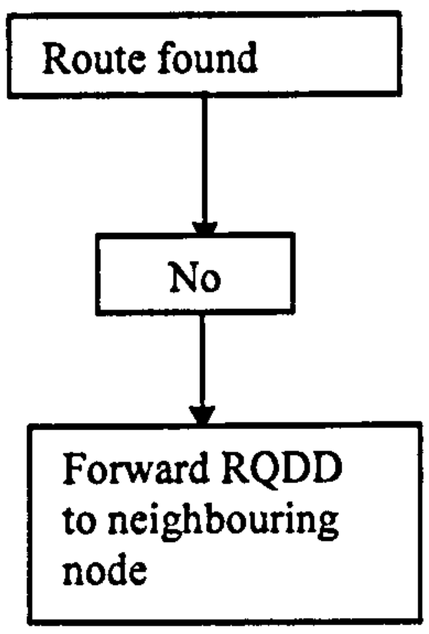


Figure.4.14c. Receiving node forward RQDD to its neighbour

- **Creating Reverse Routes**

All the nodes through which the RQDD passes before its reaches to destination determine a reverse route from destination node to the source of the RQDD. This route is used to transfer the ACK from the destination to the source of the RQDD.

- **Broadcasting and Forwarding Acknowledge Message (ACK)**

In MAODDP, the destination node broadcasts an acknowledge packet once it receives a RQDD. The Acknowledge message has been introduced to inform the source node about the successful data delivery. It can also help in maintaining fresh routes to various nodes from source to the destination. Through ACK

mobile nodes can be informed about various other nodes, hops counts from a particular node to another node and the shortest path between one and the other node. It could be seen as one of some benefits of using such scheme.

- **Managing Sequence Number**

In MAODDP each node is responsible for maintaining its own sequence number. This is required to ensure loop free routing. This sequence number in combination with message broadcast ID can be used to avoid message looping. In MAODDP the sequence number of a node increases either when a node broadcast a RQDD or when a destination node broadcast an acknowledge message back to the source node, it increases its own sequence number to the maximum of its current sequence number.

In MAODDP, all the mobile nodes store the latest available information about the sequence number for the IP address of the destination node for which the route table entry is maintained. It is updated whenever a node receives new information about the sequence number through RQDD packet, ACK packet, or RER messages.

- **Route Table Management and Maintaining Route Utilization Record**

Effective management of route tables and route utilization records is an important factor in achieving effective routing across different mobile terminals. In this context, it is also important to store only those bits of information inside routing tables without which it may not be possible to structure the routing mechanism. In an environment like a mobile ad-hoc network it is fairly important to keep records of those entire routes that have been formed before or are in use. It has a direct effect on a number of different factors. Examples of some of these factors include error detection, identification of valid routes, identification of route that have expired etc.

In MAODDP, each route life is updated whenever the route is used to transmit a data packet. The new route time out should be the sum of the time the route is used for data transmission and the active route time out. Since the route between each source and destination pair are expected to be unicast, the lifetime for the previous hop, along the reverse path back to the IP source, is also updated to be no less than the sum of the time the route is used to transmit the data packet and the active-route-timeout.

- **Error Detection and Broadcasting Error Messages**

In MAODDP, error detection occurs in one of the two conditions. When a node detects a link breakage for the next hop or neighbouring node in an active route in its routing table or it receives a RER message from one of the neighbours indicating a route failures, an inactive node in one or more active routes. Normally all of these errors could be detected in normal routing operations i.e. route discovery and data

delivery process or broadcasting acknowledge message. In figure 4.15 Node C is broadcasting an error message.

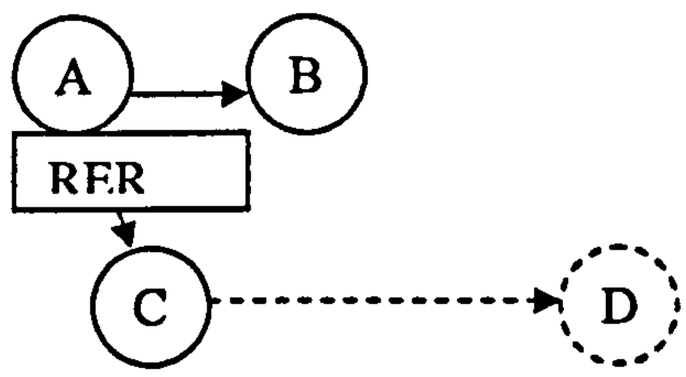


Fig.4.15. Node C broadcasting RER message

- **Power Saving Mode**

Nodes are required to switch between one of two states i.e. soft (or sleep) or active state. A node can switch into sleep mode if it does not hear from other mobile node within a time limit known as Listening Time (LT). Likewise, after a specific time interval known as the Active Time a node can switch back to active state.

- **Multicast Joining Request**

A Multicast operation of MAODDP is limited to one of two possible scenarios protocol meet during its normal multicast operations. If a node wishes to join a multicast tree for which it is not a member, it can invoke a multicast joining request by broadcasting a Joining query (MJR) to the multicast group. The IP address of a member of the multicast tree, hop counter, status time, node sequence number, current multicast group number and broadcast ID are part of MJR. In addition it also contains a "J" flag. If a node attempts to join a multicast tree and is successful it will receive an acknowledge message back from the destination node with the "J" flag at high. In this situation if the node wishes to join this new multicast tree it should disassociate itself from the current group by broadcasting a message indicating its new association.

- **Multicasting**

Unlike [46] [49] [67] which are designed as multicast routing protocols, MAODDP deal multicasting as a part of integrated approach [123]. A MAODDP multicast operation depends on the combination of flooding and the formation of a multicast tree structure as shown in figure 4.16. It is evident that flooding is suitable for high data traffic and offers lowest control overheads while tree-based routing reduces data traffic in the network but requires many control data exchanges. MAODDP focuses on maintaining only those routes that are active. Expired or invalid route entries are automatically deleted from the route table. For multicast operations a multicast tree is maintained for the life of the multicast group. There are two main scenarios which can explain different aspects of multicast operations of MAODDP.

MAODDP Multicasting Scenario one

Nodes join a multicast tree to transfer some data to a node which is a part of a multicast tree for which the node is not a member. This situation could further be categorized into one of two sub scenarios. The first

part of this situation deals with finding a suitable route to the multicast tree and the second part deals with the node joining request with this multicast tree.

A node will initiate MJR by flooding routing a JR packet with an additional flag "J" defining the node wish joining the multicast tree. The JR packet will either go through a number of intermediate nodes before it makes its way to the specified multicast tree. Similar to RQDD process the reverse path will automatically be generated in between the source node and the multicast tree. If the group is ready to accept the requested node it issue ACK back to the source node with the status of J flag at high. It indicates that the group is ready to accept requested node as a new member of multicast tree. Once accepted the node can join the multicast tree but must disassociate itself from any other association. A node must inform about its new association to its old group members.

MAODDP Multicasting Scenario 2

There is a possibility that a node wants to share some information with a node that is a part of some other multicast tree. Under this situation, if a route to the destination does not exist the node will initiate multicast route query and data delivery process by broadcasting a route query and data delivery packet (MRQDD). Once it has been done the node has to wait till either it receives an acknowledge back from the destination node or no acknowledge as received before the expiry of the status time. As has been mentioned before a reverse path will automatically be set up through the intermediate node during the route query and data delivery process.

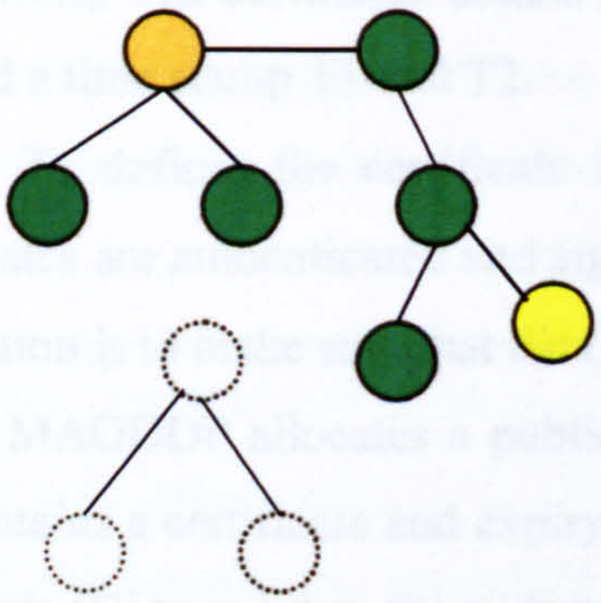


Figure.4.16 Multicasting in MAODDP

- **Security**

  Mobile nodes in an ad-hoc network meet two types of security challenges i.e. import and export authorization[166]. Import authorization requires a node acting as a router to take a decision whether or not it should modify its routing information when it receives information from somewhere outside. Export authorization requires a router to take a decision whenever it receives a request for routing information. In general, security challenges of mobile ad-hoc networks can be summarized as follows

Import authorization: refers to the authority about routing messages regarding a certain destination node.

Source authentication: we need to be able to verify that the node is the one it claims to be.

Integrity: In addition, we need to be able to verify that the routing information that it is being sent to us has arrived safely.

From a routing protocol prospective there are two types of message i.e. data and data routing message, which have unique requirements. Data messages are point to point and could be well protected by adopting existing mechanisms such as IPSec. Routing message impose a greater challenge for routing protocols as some parts of these messages always change.

In general, routing messages carry two types of information: mutable and non-mutable. It is necessary to protect mutable information in routing messages in a way that no trust in intermediate nodes is required. Otherwise securing the mutable information will be much more expensive in computation; moreover the overall security of the system will greatly decrease.

The MAODDP security mechanism uses trusted certificate server C which can be formed by two or more mobile nodes on a mutual basis. This trusted server public key is known to all valid nodes of the network. Keys are priority generated and are exchanged through mutual relationship between C and each node. Each node obtains a certificate with exactly a single key from the trusted certificate server on joining the network. The certificate details different aspects of the connecting node such as node addresses, a public key and a time stamp T1 and T2.

T1 defines the certificate issue time and T2 stands for the expiry time of the certificate. These certificates are authenticated and signed by the server C. The goal of communication between source and the destination is to make sure that data reaches the destination safely.

MAODDP allocates a public key to all the mobile nodes at the joining of the network. The public key contains a certificate and expiry time. For each RQDD the receiver node extracts the public key from the certificate 'C' to validate the signature and to make sure that the certificate has not expired and it's still valid. The same process is repeated in forwarding ACK from destination to the source node.

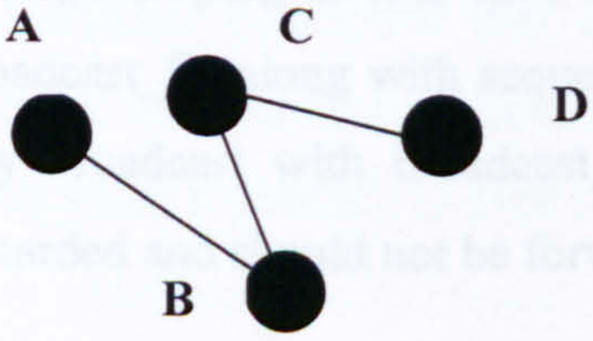


Figure 4.17 Data transmission from A to D through B

In Fig.4.17 Node B on receiving the RQDD verifies the public key and the certificate valid time by extracting this information from the attached certificate. B then removes the A certificate signature, records B as predecessor, signs the contents of the message originally broadcast by A, appends its own certificate and forwards the broadcast message to the neighbouring node till it reaches D.

4.4.3 Features of MAODDP

Routing was the first area which attained focus with the emergence of commercial mobile ad-hoc networks. Initially it was proposed to achieve routing by using conventional routing mechanisms. The same ideas were evolved into tables driven or link state routing mechanisms[23,24]. Later another type of routing protocols known as On-demand routing protocol[25] was proposed. It has been mentioned before that throughout this thesis our main focus will be on evaluating MAODDP against two main protocols i.e. DSDV and AODV each from the above mentioned type respectively. It is due to the fact that these protocols are currently under review by IETF. Moreover, they follow closely related operational patterns as MAODDP. In the following section different features of MAODDP will be highlighted against the above two mentioned schemes.

- **On-Demand Routing**

MAODDP is an on-demand protocol. On-demand routing protocols have been demonstrated better by producing lower overheads than table driven protocols types.[23, 24] On-demand protocols are able to react quickly too many changes that may occur in node connectivity. It is known that these types of protocols are also able to reduce routing overhead in periods or areas of the network in which topology changes are less frequent.

- **Loop Free Routing with the use of Sequence Number and Broadcast ID**

Due to the nature of mobile ad-hoc networks it is highly likely that mobile nodes could receive the same query or data packet again [176, 177]. This problem is known as "Message Looping". Message looping could create unnecessary bandwidth and power consumption as well as an extra network overhead. It is important to introduce measures through which different types of packet could be differentiated.

In MAODDP the combination of broadcast ID and node sequence number is utilized to avoid message looping as it is used in [25]. Each node issues a new broadcast ID for every new RQDD. The Broadcast_ID along with sequence number is used by all other intermediate nodes to identify data packets. Any broadcast with broadcast_ID and sequence number similar to the one received before should be discarded and should not be forwarded to any other node in the network.

- **Bandwidth Efficient**

It has been mentioned before that a mobile ad-hoc network operates with limited bandwidth. Reducing bandwidth consumption in this type of network could be seen as one of the important aspects [9]. Existing techniques [23, 24] show no specific procedure to save unnecessary bandwidth consumption. In table driven protocols[23] where routing is established but techniques of maintaining a consistent network overview via continuous network updates put extra burden on the available bandwidth[40]. Similarly on-demand protocols[25] involving with too many query packets and route replies could produce the same effects. Moreover issues like network congestion, network speed are related to the available bandwidth. A network with higher available bandwidth could result in better performance and less chance of network reduction. MAODDP addressed most of these deficiencies in the available schemes.

MAODDP does not require network updates apart from the regular joining message at the time of network formation. Instead of route queries and route replies MAODDP establishes the route and delivers the data simultaneously at the same time one after the other[164]. I.e. MAODDP does not require routes to be established prior to the data transfer.

- **Conserving Battery Power of Mobile Nodes**

Nodes in a mobile ad-hoc network operate on low battery power. Routing protocols [14] should be capable of introducing measures which can help reduce unnecessary power consumption. In table driven protocols[23] mobile nodes are required to be in the awake state all the time and in on-demand protocols[25] [28]. Too many queries and route replies could also result in sleep state reduction of mobile nodes.

MAODDP adopts an integrated approach[27]. Nodes are allowed to switch between one of two modes i.e. sleep state or active state. Nodes are required to be awake only during the time of an active transmission and they are allowed to go into sleep mode if they are not the receivers or the senders of RQDD packets after a particular time interval. Moreover with the addition of a specific listening time (LT) each node can switch back into listening mode after a time period (LT). Once switched back into active mode if the node does not find it self in any active transmission it can switch back into sleep mode. This sort of strategy could be very helpful in reducing unnecessary power consumption.

- **Faster Network Converge**

Table driven protocols could offer a better network speed as the route is found in advance prior to a transmission. MAODDP with establishing route and delivering data simultaneously at the same time could increase the overall data transmission speed. Not limited to a smaller network further work on MAODDP could make it possible to offer same or better speed to the larger mobile ad-hoc networks.

- **Guaranteed Data Delivery to the Desired Destination**

Most of the earlier reported work[34, 68, 86] in this area are focused on achieving routing with very less attention given to successful data delivery to the destination. Some of the famous table driven[23] and on-demand protocols[25,28] are silent on this issue. Destination sequence distance vector routing[23] and ad-hoc on-demand data delivery routing protocols[25] are considered to be as some of the main and the earliest proposed protocols of table driven and on-demand types protocols of mobile ad-hoc network respectively. DSDV[23] suggests a straightforward mechanism to establish routing but does not guarantee anything about the actual data being sent. Similarly AODV[25] introduces a new way of establishing routing in ad-hoc networks but its specification is silent on this issue[34].

MAODDP introduces a special mechanism of broadcasting an acknowledge packet by the destination node of the RQDD[34]. In this regard status time plays an important role in finding out the message status i.e. if the transmission is failed in the last attempt.

- **Integrated Approached**

Research findings[27] conclude that an effective routing solution could be established when some other routing related problem i.e. limited bandwidth , power and security, is dealt with alongside routing. In most of the existing routing techniques[23] [25] these factors have not been addressed in full. Research shows[27] weak performance of these techniques on addressing some of the above mentioned issues. The basic model of mobile ad-hoc on-demand data delivery protocol reflects the integrated approach of MAODDP. In short MAODDP offers a routing solution which is capable of addressing most of the routing associated issues along with routing.

- **Simultaneous Route Discovery and Data Delivery**

It is one of the key features of MAODDP that it establishes the route and delivers data simultaneously at the same time. In table driven protocols continuous network updates could result in unnecessary network overhead, thereby resulting in a reduction of network efficiency. Likewise too many query packets prior to data delivery could yield the same effect[40]. MAODDP neither requires any regular updates nor does it depends on route establishment prior to the data delivery.

- **Automatic mechanism to find out if the transmission has failed**

It is another important feature of MAODDP which makes source nodes able to decide if the last transmission was successful. This feature does not exist in any of the previously proposed schemes[46, 47]. MAODDP introduces a 'status-time' ST, which is the time the source node should wait to receive an ACK from the destination node. This time has been added as an additional information field inside the RQDD.

- **Systematic Procedure for Corroboration of Mobile Devices**

In a mobile ad-hoc network each nodes relies on others to forward data packets to the other nodes in the network[178]. There is good number of reasons why mobile nodes in an ad-hoc network would prefer not

to cooperate. When nodes do cooperate they establish the necessary ad-hoc structure that can make multi-hop communication possible. This allows traffic flow from a node to reach those destinations that would either require a significant amount of transmission energy using single hop communication or simply not be possible without routing the traffic through other nodes. This further means that nodes must be willing to forward traffic for other nodes, and in this way spend energy without receiving any direct benefit.

If a node only considers its own short term live period, as they themselves operate on low power battery devices, then it may not choose to participate within the network. Thus the concept of introducing measures which can force participating nodes to collaborate into the architecture of a mobile ad-hoc network becomes one of the important issues. This combination of nodes must be able to focus on the dynamics of the nature of a mobile ad-hoc network and must also be able to take many similar factors into account.

It is always an issue how various participating nodes can be organized systematically or in other words localized when there is no command control system available. As location information provides the context to receive data, a reliable solution to node localization is a critical routing component of an mobile ad-hoc network and has therefore received a good amount of recent attention in present literature on mobile ad-hoc networks[94, 108]. Currently, there are various solutions to node localization[43, 179]. Some of these solutions include, positioning system, use of various types of model containing the characteristics of initialization phase of such networks, asynchronous wake-up and scarce knowledge about the topology of the network graph.

MAODD proposes that a mobile node should be selected as the administrator node. The selection of administrator node depends on the participating nodes of a mobile ad-hoc network. There are a number of measures which then could be taken to identify selfish nodes. Some measures might include broadcasting the sequence number of such nodes to all other nodes of the network. This node can then be excluded and the network can be rebooted.

- **Security**

Protecting data transformation in a mobile ad-hoc network is an important aspect to be seen. At present a mobile ad-hoc network does not have any standard security policy. This could possibly lead active attackers to easily exploit or possibly disable a mobile ad-hoc network. The initial specification of some of the famous protocols including DSDV[41, 46] and AODV are silent over security issues. However, an extension of DSDV addressing security is reported in [15]. Moreover, a simulation studies [38] addressing the same issue is also cited in the available literature. It has been mentioned before in this chapter that MAODDP deals with security besides routing and has its own security mechanism[34].

- **Selection of Shortest Path in between the Source and the Destination**

The battery life of the mobile nodes and the network bandwidth are some of the vulnerable assets of a mobile ad-hoc network. Selecting the shortest path to the destination could play an important role in saving these resources. MAODDP has adopted a totally new strategy for selecting the shortest and the best path from a source to a destination. A hop counter has been introduced inside the RQDD packet header. Whenever a source node broadcasts RQDDs it set the hop-counter value to zero. This counter will increase by one from the previous value each time it reaches any intermediate node before it reaches the intended destination. Each intermediate node automatically becomes aware of the exact hop-count from itself to the destination of interest. This information could also be used in any future communication.

4.5 Summary

In this chapter the functional specification of MAODDP has been presented. The MAODDP specification included the introduction to various terminology, operational details and MAODDP features. The aim was to highlight different aspects of MAODDP against some of the other schemes in this area. The functional model of MAODDP has been written in Java. In the following chapter details of various classes and a brief description of some of the functions are also covered.

CHAPTER 5. MAODDP IMPLEMENTATION

5.1 Introduction

Developing the functional model of MAODDP was important to verify various concepts in a practical environment. The MAODDP model has been developed in JAVA. This chapter presents a brief account of the implementation process and has been organized as follows. An introduction to the MAODDP implementation is given in section 2 and chapter summary is covered in section 4.

5.2 MAODDP Implementation

The MAODDP implementation is one of the crucial stages of this project. The main idea was to implement MAODDP in a manner which makes it possible to evaluate protocol performance in a simulation framework. Most of the main functions of the protocol have been written in Java. Some of these functions are Route query data delivery, acknowledge message and power aware operation. A partial implementation of route error functions can also be found in program codes of MAODDP. A single file representing all classes is included inside the appendix of this thesis. In this section a brief introduction to the Java language along with some of the standard classes will be presented before explanation of the MAODDP local classes.

5.2.1 Implementation Language

MAODDP was implemented in Java. It was due to the prior knowledge and experience of Java and our plan to evaluate the protocol on a Java based simulator i.e. Scalable Wireless Network Simulator (SWANS). In the following chapter a brief introduction to SWANS and reasons for its selection are discussed. Java is a general-purpose, concurrent, class-based object-oriented language. Its specification clearly distinguishes between the compile-time errors that can and must be detected at compile time, and those that occur at run time. Compile time normally consists of translating Java programs into a machine-independent byte-code representation. Run-time activities include loading and linking of the classes needed to execute a program, optional machine code generation and dynamic optimization of the program, and actual program execution. "RouteMaoddp.java" depends on some of the standard Java classes defined inside the util package of Java. Short descriptions of each of these classes are as follows.

- **Class Iterator**

  Iterators provide an abstract mechanism to access elements in collection classes without revealing their underlying representations. This class is mainly used to add or remove elements inside or from various lists.

- **Class Map**

The Map class is used specially in recording various information inside MAODDP routing tables. It is one of the most used collection class in java.util. Maps provide a more general way of storing elements. The map collection type allows storing pairs of elements, termed "keys" and "values", where each key maps to one value.

- **Class HashMap**

Similar to MAP, the class HashMap is also used in the MAODDP routing table structure. Class HashMap provides Hash table based implementation of the map interface. This implementation provides all of the optional map operations, and permits null values and the null key.

- **Class Linked List**

Linked list is an implementation of the list interface and it is used in defining various lists within the MAODDP implementation. It implements all optional list operations, and permits all elements including null. In addition to implementing the list interface, the LinkedList class provides uniformly named methods to get, remove and insert an element at the beginning and end of the list. These operations allow linked lists to be used as a stack, queue, or double-ended queue (deque).

- **Class Set**

Class set is used in various places in MAODDP implementation. It represents a set as a collection that contains no duplicate elements. More formally, sets contain no pair of elements e1 and e2 such that e1.equals (e2), and at most one null element.

5.2.2 MAODDP Classes

Implementation of MAODDP is done in fourteen different classes. Among them some are declared as public while others are as private. Having public class declaration means that the variables inside this class are accessible for the rest of the classes in the same environment. Declaring some class as private means variables will be treated as locals to the class where they are defined. Functions within the private class are accessible through an object of class type.

Three different message types of MAODDP namely "route query data delivery", "acknowledge message" and "route error" are implemented in three separate classes. Function broadcasting has been implemented in a separate class. The Routing table's structure is defined inside class "Route Table".

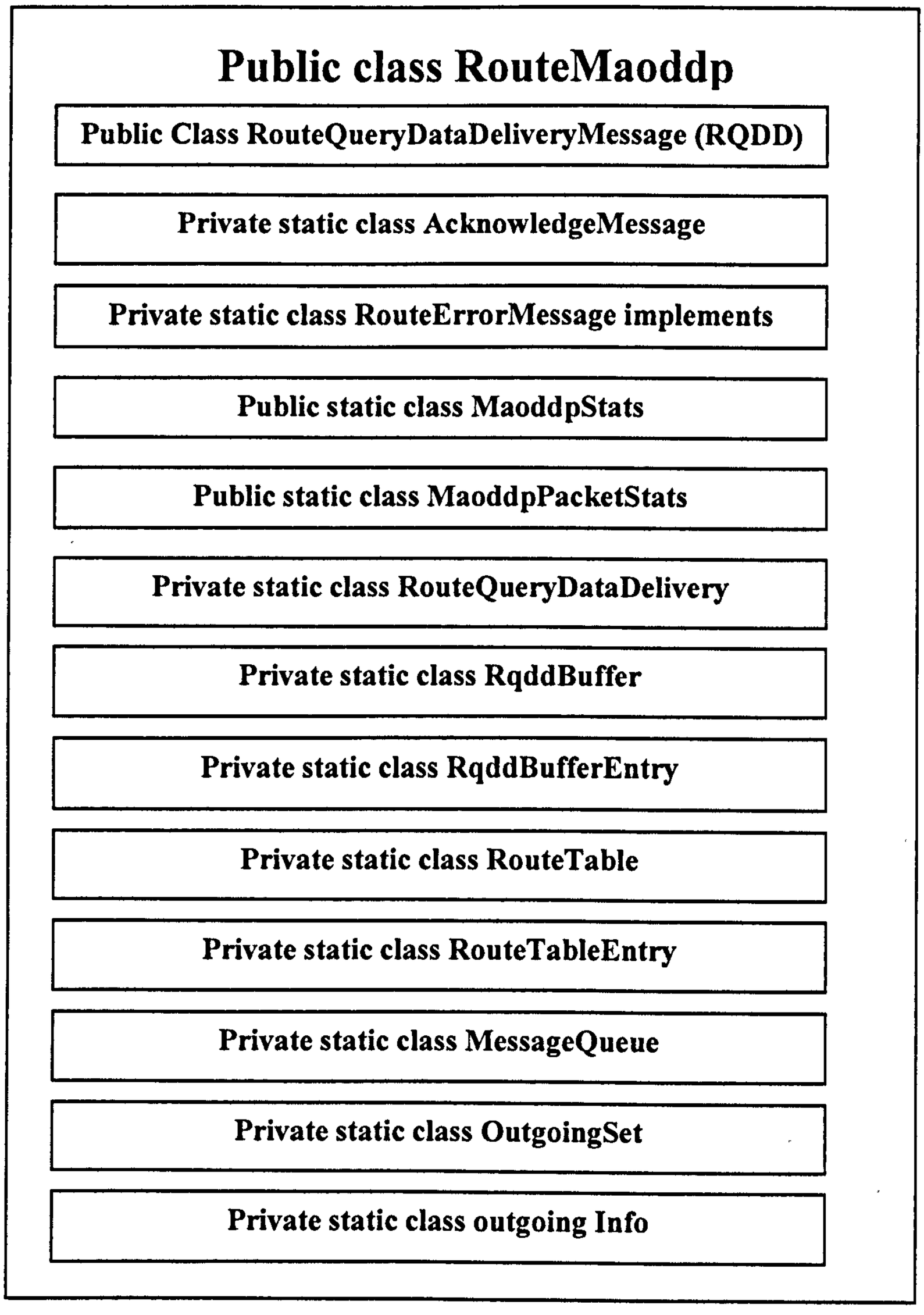


Figure 5.1. Implementation file structure

Addition and deletion of entries from the routing table is handled by class "Route Table Entry". Messages are queued in a buffer before being forwarded to the mobile nodes. This whole process is handled by classes "messagequeue" and "RqddBuffer". Different information of class RqddBuffer are stored inside the class "RqddBufferentry". Besides these classes constants and variables are also declared both on public and private types. The description and brief explanations of various functions of these classes are as follows.

- **Public Class RouteMAODDP**

RouteMAODDP being the main class has been declared public. Various constants are defined inside the class "RouteMaddop". These constants can be set within the MAODDP code and can be used to monitor MAODDP performance for different simulation environments. A brief description of some of these constants is as follows.

Constants

- DEBUG_MODE (Boolean) – if true, debugging statement are printed. Default is false.
- SEQUENCE_NUMBER_START (int) – Starting sequence number of each node. Default is 0.
- RQDD_BUFFER_EXPIRE_TIME (long) – Maximum duration of an entry may reside in the RQDD buffer before it may be removed. Default is 5 seconds.
- MAX_BUFFER_SIZE (int) – Strict maximum size of node's RQDD buffer.
- RQDD_TIMEOUT_BASE (long) – Constants term for RQDD timeout duration. Default is 1 second.
- RQDD_TIME_PER_TTL (long)-Variable term for RQDD timeout duration, which depends on the TTL value of the RQDD message. Default is 500 milliseconds (per TTL).
- TRANSMISSION_JITTER (long) – The maximum delay before sending any packet.

- **Public Class RouteQueryDataDeliveryMessage (RQDD)**

Class RouteQueryDataDeliveryMessage (RQDD) is the implementation of "Route Query Data Delivery" message of MAODDP. It also Implements the jist.swans.misc.Message interface of SWAN. Fields that are part of RQDD are defined inside this class as private variables. Some of these variables are rqddId which is route query data delivery identification number, destIp an object of NetAddress class serves as destination node IP address, origIP of object NetAddress class acts as source node IP address, destSeqNum is the destination node last known sequence number, origSeqNum which is source node sequence number and hopcount which is hop-count from source node to the destination node.

Route query data delivery message object is created via the method RouteQueryDataDeliveryMessage (RQDD) followed by constructing a copy of an existing RQDD. Each of the above mentioned variables are made part of RQDD and a number of different procedures are defined to return the value of each of these

fields. Example of one of such function is getRqddId () which returns the RQDD broadcast-ID number and incHopCount () which record the hop count from source node to the destination node. The Java codes of these functions are as follows.

```
Public int getRqddId () {
                    return rqddId;
                                }
        returns broadcast-ID of RQDD.
```

```
public void incHopCount()
    {
                hopCount++;
                          }
```

- **Private static class AcknowledgeMessage**

The AcknolwedgeMessage class implements the jist.swans.misc.Message interface of SWAN. It represents implementation of "Acknowledge Message" of MAODDP. The various fields of acknowledge message are represented by different variables. Some of these variables are destIp an object of NetAddress class stores the destination node IP address, origIp an object of NetAddress class acting as a source node Ip address and hopCount representing the hopCount field. Method "AcknoweldgeMessage" is used to construct a new acknowledges message. Next various methods are written to return values of each of the above explained fields. Example of some of these methods are NetAddress getDestIp() which returns the destination address and int getDestSeqNum() which returns the destination sequence numbers. Java code of these functions are as follows.

```
Public NetAddress getDestIp() {
            return destIp;
                 }
        public int getDestSeqNum() {
            return destSeqNum;
                 }
```

- **Private static class RouteErrorMessage implements Message**

This class is an implementation of "Route Error" message of MAODDP. It also implements the jist.swans.misc.Message interface of SWAN. In this class variable "Message_Size" holds the message size of "route error" message in bytes. An object "unreachableList" of LinkedList class is also constructed. A new route error message object with an empty unreachable list was created via method "RouteErrorMessage" followed by a new route error message object with a given unreachable list was constructed through the same

method by passing a list of LinkedList type as an argument. Java codes of both of these methods are as follows.

```
Public RouteErrorMessage()
                {
                   this(new LinkedList());

                 }
    public RouteErrorMessage(LinkedList list)
          {
              this.unreachableList= list;
                        }
```

Unreachable node net addresses are obtained via method getUnreachableList and a new unreachable node is added by method addUnreachable.

- **Public static class MaoddpStats**

The Class MaoddpStats supports global statistical information for a simulation. A stats object could be referenced to this class. Stats object should be instantiated once by the simulation driver program, and each MAODDP node should contain a reference to this object. The reference can be set using the setStats () method. Different variables and procedures are used to achieve this. Some of these variables are rqddOrig which holds the number of total route query data delivery packets, ackOrig to store total number of acknowledge messages generated and rqddSucc for total number of routes established. Java code of function clear of this class is given below.

```
    Public void clear()
{
   send.clear();
   recv.clear();
   netMsgs =0;
   rqddOrig= 0;
   ackOrig=0;
   rqddSucc = 0;
}
  }
```

- **Public static class MaoddpPacketStats**

Class MaoddpPacketStats stores different pieces of information related to each individual packets for class MaoddpStats. Some of the variables of this class are maoddpPackets which stores maoddp packets types, rqddPackets to hold route query data delivery packets, ackPackets for acknowledge packets and RER packets holding all route error packets. Function clear is the only function of this class and Java codes of this function are as follows.

```
Public void clear()
{
    maoddpPackets = 0;
    rqddPackets = 0;
    ackPackets=0;
    RERPackets=0;
    }
  }
```

- **Private static class RouteQueryDataDelivery**

This class handles route query data delivery packets (RQDD) broadcast by a node. In this class a single node can repeatedly broadcast RQDD message, with increasing time to live (TTL) value, until an acknowledge message is received. TTL value starts at TTL start value and is incremented by TTL increment method but it does not exceed TTL threshold value. Some of the variables declared inside this class are destIp to stored net address of the destination; rqddId serves for route query data delivery identifier number and ttl to represent "time to live". A new RQDD object is created via method "RouteQueryDataDelivery" followed by different methods to obtain values from RQDD. The method to increase "time to live (ttl)" is also a part of this class. Java code of this method is as follows.

```
Public void incTtl()
    {
     ttl=(byte)Math.min(ttl+TTL_INCREMENT, TTL_THRESHOLD);
              }
```

The method broadcast is an important method of this class. It saves RQDD information in a buffer so that it should not be forwarded if it receives again. This method also checks routing tables entries and whether or not an acknowledge has already been received. In the case that no acknowledge message has been received then it rebroadcasts RQDD with a new sequence number. Java code of the method broadcast is given below.

```
Public void broadcast () {
                    int destSeqNum =0;
        unknownDestSeqNum=false ;
        RouteTableEntry routeEntry= (RouteTableEntry)
thisNode.routeTable.lookup (this.destIp);
        if (routeEntry = null)
        {
          unknownDestSeqNum=true;
             }
        else
        {
          unknownDestSeqNum=false;
          destSeqNum=routeEntry.getDestSeqNum();
           }
RouteTableEntry selfEntry = thisNode.routeTable.lookup(thisNode.netAddr);
selfEntry.setDestSeqNum(thisNode.seqNum);
thisNode.routeTable.printTable();

RouteQueryDataDeliveryMessage rqddMsg= new RouteQueryDataDeliveryMessage
(
  rqddId, destIp,thisNode.netAddr,
destSeqNum,thisNode.seqNum,unknownDestSeqNum,0);

NetMessage.Ip rqddMsgIp = new NetMessage.Ip(rqddMsg,thisNode.netAddr,
NetAddress.ANY,
 Constants.NET_PROTOCOL_MAODDP,Constants.NET_PRIORITY_NORMAL,this.ttl);
  printlnDebug("Broacasting RQDD message with rqddId="+rqddId+",
ttl="+ttl, thisNode.netAddr);
 thisNode.self.sendIpMsg(rqddMsgIp,MacAddress.ANY);
 //stats
 if (thisNode.stats !=null)
```

```
{
      thisNode.stats.send.rqddPackets++;
      thisNode.stats.send.maoddpPackets++;
      } }
 }
```

- **Private static class RqddBuffer**

Class RqddBuffer keeps track of recently sent RQDD messages. It is to avoid resending RQDD with the same sequence number and ID. This class also monitors recent messages for which an acknowledge message has been received. Method "RqddBuffer" constructs a new route query data delivery buffer object where an entry to RQDD buffer is added via the method "addEntry". Method "addEntry" also delete all the expired entries and makes sure that the list should not exceed the maximum allowed list size. Entry inside RQDD buffer is checked by method contains and method "clearExpiredEntires" is used to remove all expired entries. Java code of methods "addEntry" and "clearExpiredEntires" are as follows.

```
Public void addEntry(RqddBufferEntry entry)
      {
      clearExpiredEntries();
        if (list.size() == MAX_RQDD_BUFFER_SIZE)
         {
              list.removeLast();
                                    }
         else if (list.size() > MAX_RQDD_BUFFER_SIZE)
        {
           throw new RuntimeException ("RQDD Buffer is larger than
allowed size !");
           }
 list.addFirst(entry);
              }
public void clearExpiredEntries()
    {
```

```
        while (!list.isEmpty() && JistAPI.getTime() >
((RqddBufferEntry)list.getLast()).getTimeSent() +
RQDD_BUFFER_EXPIRE_TIME)
      {
            printlnDebug("Removing Entry from RqddBuffer", localAddr);
            list.removeLast();
            }
   }
```

- **Private static class RqddBufferEntry**

A single entry of the RQDD Buffer is recorded inside Class RqddBufferEntry. This class records the RQDD ID for RQDD message that was sent, the net address of the source node of RQDD and the time when RQDD was stored inside the RQDD buffer. Several other variables are used to assist the main functions of this class. The method RqddBufferEntry is responsible for creating a new RQDD buffer entry object and the method getTimeSent() returns the time at which entry was created. This class also checks existing records before a new entry is created. It is done by method "equals". Java codes of the method RqddBufferEntry and equals are as follows.

```
Public RqddBufferEntry(int rqddId, NetAddress originIp)
      {
          this.rqddId = rqddId;
          this.originIp = originIp;
          this.timeSent = JistAPI.getTime();
             }

          public long getTimeSent() {
         return timeSent;
                 }
public neighbour equals(Object o)
      {
   if (o== null || (!( o instanceof RqddBufferEntry)))
     {
            return false;
     }
RqddBufferEntry entry = (RqddBufferEntry)o;
```

```
            return (this.rqddId ==  entry.rqddId &&
this.originIp.equals(entry.originIp));
            }

      public int hashCode()
    {
          return this.rqddId;
                                    }
}
```

- **Private static class RouteTable**

Class route table is the implementation of routing tables of MAODDP. The Route table stores route information in a HashMap, mapping NetAddress objects to RouteTableEntry objects. It contains methods for removing all routes through a given next hop, and for removing a list of route entries. Some other methods include addition of new entry, lookup for an existing entry and deletion of entry are also defined inside this class. RouteTable object is constructed via method "RouteTable". A separate method "add" is written to add new entries inside the routing table. The method remove is used to remove entries from the routing table and the method lookup is responsible to search for the existing entries. To print the routing table as required the method "printable" is written. Java code of the methods "add" and "neighbour" are given below.

```
Public void add(NetAddress key, RouteTableEntry value)
          {
                  table.put(key,value);
                                          }
 public void neighbour()
                {
                      Iterator itr = table.entrySet().iterator();
                      while (itr.hasNext())
                         {
                              Map.Entry mapEntry =
(Map.Entry)itr.next();
                                    NetAddress dest =
(NetAddress)mapEntry.getKey();
```

```
                                        RouteTableEntry route =
(RouteTableEntry)mapEntry.getValue();

printDebug("route_table:["+dest+,localAddr);
                          if (route !=null)
                           {
printlnDebug_plain("nextHop="+route.nextHop+" DSN="+route.destSeqNum+"
                                hopCnt="+route.hopCount+")]");
                                                }
                                   else
            {
                                printlnDebug_plain("null)]");
                                    }
                          }
                 }
```

- **Private static class RouteTableEntry**

Class RouteTableEntry stores various routing information. Some of the declared variables inside this class are 'nexthopaddress' which is 'MacAddress', 'destinationsequencenumber' and 'hopcount'. Besides variables different functions are defined inside this class. An example function is RouteTableEntry() which records all three variables containing respective information. Likewise the method getDestSeqNum returns the destination sequence number as required. Java code of these methods are given below.

```
Public RouteTableEntry(MacAddress nextHop, int destSeqNum, int hopCount){
      this.nextHop= nextHop;
      this.destSeqNum = destSeqNum;
      this.hopCount = hopCount;
              }
public int getDestSeqNum()
       {
        return destSeqNum;
       }
```

- **Private static class MessageQueue**

Class "MessageQueue" temporarily stores transport-layer messages while routes are being searched. When route information becomes available, the messages are then sent along the routes. A list object of LinkedList type is declared to store a list of IP messages. 'Message queue' object is constructed with an

empty list. Method "add" is responsible for adding an IP message to the queue. All queued messages are sent to a given destination via a given next hop. All repeated messages are removed from the queue via method "removeMsgsForDest". Java code of the methods "dequeueAndSend" and "removeMsgsForDest" are given below.

```
Public void dequeueAndSend(NetAddress dest, MacAddress nextHop)

                {
                 for (int i =0; i<list.size(); i++)
                {
                   if (((NetMessage.Ip)list.get(i)).getDst().equals(dest))
                      {
                NetMessage.Ip msg= (NetMessage.Ip)list.remove(i);
                printlnDebug("Routing IP message to "+nextHop,
thisNode.netAddr);
                thisNode.self.sendIpMsg(msg,nextHop);
                        }
                   }
                        }
```

```
public void removeMsgsForDest(NetAddress dest)
                    {
                        Iterator itr = list.listIterator(0);
                        while (itr.hasNext())
                          {
                              NetMessage.Ip msg =
(NetMessage.Ip)itr.next();
                               if (msg.getDst().equals(dest))
                            {
                                itr.remove();
                            }
                        }
                }
```

- **Private static class OutgoingSet**

Class OutgoingSet stores the set of neighbouring nodes through which the node routes messages. Each node is then mapped to a corresponding object of OutgoingInfo. The data structure for the outgoing node set is declared inside this class prior to an object localAddr of type NetAddress. An object of outgoingSet is created and an iterator of type iterator() is also defined. To add a new entry to the outgoing node set the method "adds" is used. The outgoing node info for a given MAC address is returned via method "outgoingInfo". Java code of the method "add" and "getInfo" are as follows.

```
Public void add(MacAddress m)
        {
                printlnDebug("Adding "+m+" to outgoing set",
localAddr);
                map.put(m,OutgoingInfo());
             }
public OutgoingInfo getInfo(MacAddress m)

          {  return (OutgoingInfo)map.get(m);
                          }
```

- **Private static class outgoing Info**
  Class OutgoingInfo stores information for each node in the class OutgoingSet. Various variables are declared in this class to store different fields. The Local network address is stored inside variable netAddr of type NetAddress, the variable 'seqNum' stores 'node sequence number', rqddIdSeqNum stores 'route query data delivery ID number', routetable is declared as an object of class RouteTable, rqddList an object of LinkedList is declared to store a list of pending route query data delivery originated by the node, rqddBuffer is an object of RqddBuffer and is used to store information about previously sent RQDD messages, msgQueue is an object of MessageQueue and is used to store messages that need routes ,outgoingSet is an object of OutgoingSet which represents the set of nodes that will be acting as intermediate nodes for RQDD from source to destination and statistics is accumulated via object stats of class MoaddpStats.

A new object RouteMaoddp of class "RouteMaoddp" was created. The main purpose of method RQDDtimeout is to broadcast RQDD with a new 'rqdd ID' and incremented 'time to live (TTL)' value. This timeout event gets scheduled for a future time whenever the node originates a RQDD message. When the timeout for a given route query data delivery occurs, for which no acknowledge has been received, then it sends another 'RQDD message' with an increased TTL, and schedules another 'RQDDtimeout()'. This process continues until the TTL cannot be further increased. In other words timeout is an event that gets

called for every 'MAODDP_TIMEOUT' for the duration of a simulation cycle. It clears expired entries in the 'rqddBuffer'.

The method 'Send' is called by the network entity to send messages over the network. If routing information is available it simply forwards the message to the appropriate next hop. Otherwise, the message is saved in the messageQueue and a 'routerequestdatadelivery' is originated.

The method peek is called by the network layer for every incoming packet. The method send is called by the network layer to request transmission of a packet that requires routing. It is the responsibility of the routing layer to provide transmission of this packet to an appropriate next hop by calling the network layer sending routines once this routing information becomes available.

The method "receive" is defined to receive messages from the network layer. The method "recieverouterequestdatadeliverymessage" processes an incoming RQDD message, updates routing tables and then either sends 'Acknowledge message' via 'generateAcknowledgeMessage' if it is the destination node or forwards the RQDD via forwardRouteQueryDataDeliveryMessage() to the next hop node.

The method "recieveacknowledgeMessage" processes acknowledge messages. Similar to these two methods, 'method recieveRouteErrorMessage' receives and processes route error messages. The RQDD message is forwarded to all other nodes via the method 'forwardRouteQueryDataDeliveryMessage'. The acknowledge message is generated and sent by the method 'generateAcknowledgeMessage'. The method 'shouldUpdateRouteToOrgin' decides whether a node receiving a RQDD message should update its route to the RQDD originator. The RQDD time out period is computed by method 'computeRQDDTimeout'.

Java code of the method 'recieverouterequestdatadeliverymessage' and 'recieverouterequestdatadeliverymessage' are given below.

```
Private void receiveRouteQueryDataDeliveryMessage(
RouteQueryDataDeliveryMessage rqddMsg, NetAddress src, MacAddress
lastHop, NetAddress dst, byte priority, byte ttl) {
                    if (DEBUG_MODE)
                      {
                      printlnDebug("Receiving RQDD:" + "rqddId=" +
rqddMsg.getRqddId() + "destIp="+ rqddMsg.getDestIp() + "origIp=" +
rqddMsg.getOrigIp() + "destSN=" + rqddMsg.getDestSeqNum() + "origSN=" +
rqddMsg.getOrigSeqNum() + "unkDSN=" + rqddMsg.getUnknownDestSeqNum() +
"hopCnt=" + rqddMsg.getHopCount());
                                                              }
```

```
        RouteTableEntry origRouteEntry =
routeTable.lookup(rqddMsg.getOrigIp());
neighbor updateRoute = shouldUpdateRouteToOrigin(rqddMsg,
origRouteEntry);
                if (updateRoute)
            {
             //add /update route to RQDD originator through previous hop
             routeTable.add(rqddMsg.getOrigIp(), new RouteTableEntry(lastHop,
rqddMsg.getOrigSeqNum(), rqddMsg.getHopCount()+1));
             routeTable.printTable();
              }
//check if this node is dest, or has a route to dest with higher SN
// if so, send back a RQDD; otherwise, forward RQDD to neighbours
isDest = rqddMsg.destIp.equals(this.netAddr);
RouteTableEntry destRouteEntry = routeTable.lookup(rqddMsg.destIp);
routeToDestExists = (destRouteEntry !=null && destRouteEntry.nextHop !=
null);
hasFreshRoute= routeToDestExists && !rqddMsg.getUnknownDestSeqNum() &&
destRouteEntry.getDestSeqNum() > rqddMsg.getDestSeqNum();
inRqddBuffer = rqddBuffer.contains(new
RqddBufferEntry(rqddMsg.getRqddId(), rqddMsg.getOrigIp()));
 if (isDest || hasFreshRoute)
   {
      if (!inRqddBuffer || updateRoute)
       {
          generateAcknowledgeMessage(rqddMsg, isDest, destRouteEntry);
            }
         }
  else
    {
        // check buffer first to see if this node has already sent this
RQDD before
  if (!inRqddBuffer)
```

```
        {
          byte newTtl = (byte) (ttl-1); // decrement ttl
        if (newTtl > 0)
        {
           printlnDebug("Forwarding RQDD");
          forwardRouteQueryDataDeliveryMessage(rqddMsg, newTtl,
destRouteEntry);
            }
    }
 }
      }
private    void    receiveAcknowledgeMessage(AcknowledgeMessage    ackMsg,
NetAddress src, MacAddress lastHop, NetAddress dst, byte priority, byte
ttl)
{
   printlnDebug("handling  ACK:"  +  "destIp="  +  ackMsg.getDestIp()  +
"destSN=" + ackMsg.getDestSeqNum() + "origIp=" + ackMsg.getOrigIp() +
"hopCnt=" + ackMsg.getHopCount());
RouteTableEntry entry= routeTable.lookup(ackMsg.getDestIp());
       if    (entry    ==    null    ||    (ackMsg.getDestSeqNum()    >
entry.getDestSeqNum()) ||
        (ackMsg.getDestSeqNum()==        entry.getDestSeqNum()        &&
ackMsg.hopCount<entry.getHopCount())){
routeTable.add(
 ackMsg.getDestIp(), new RouteTableEntry(lastHop, ackMsg.getDestSeqNum(),
ackMsg.getHopCount()+1));
 routeTable.printTable();
 outgoingSet.add(lastHop);
 }
if (this.netAddr.equals(ackMsg.getOrigIp()))
//go through rqddlist, setting routeFound=true, and removing them
  { Iterator itr =rqddList.iterator();
   while (itr.hasNext())
```

```
      {
                RouteQueryDataDelivery
rqdd=(RouteQueryDataDelivery)itr.next();
                if (rqdd.getDest().equals(ackMsg.getDestIp()))
                {
                 printlnDebug("Removing rqdd from rqddlist");
                 rqdd.setRouteFound(true);
                  //stats
                if (stats !=null)
                {
                  stats.rqddSucc++;
                  }
           itr.remove();
                }
                  }
msgQueue.dequeueAndSend(ackMsg.getDestIp(), lastHop);
}
  else
      {
        RouteTableEntry                                    origRouteEntry=
routeTable.lookup(ackMsg.getOrigIp());
        if (origRouteEntry !=null && (ttl > 0))
       {
              MacAddress nextHop= origRouteEntry.getNextHop();
              ackMsg.incHopCount(); //increment ack hop count
              NetMessage.Ip ackMsgIp =
              new       NetMessage.Ip(       ackMsg,       src,       dst,
Constants.NET_PROTOCOL_MAODDP,Constants.NET_PRIORITY_NORMAL, (byte) (ttl-
1));
              printlnDebug("Forwarding ack message to node "+nextHop);
              self.sendIpMsg(ackMsgIp, nextHop);
              // stats
    if (stats != null)
```

```
{
    stats.send.ackPackets++;
    stats.send.maoddpPackets++;
            }
 this.outgoingSet.add(nextHop);
       }
}
}
```

5.3 Summary

In this chapter a brief account of the MAODDP implementation process was presented. The MAODDP functional model was developed in JAVA. An introduction to various Java standard classes alongside classes private to the MAODDP implementation was also covered. Each class was explained in the same order as it appears inside the actual implementation file. Wherever appropriate a description of different variables declared inside each class and example functions along with Java code were also discussed. This model was later evaluated in a simulation framework. In the following chapter a brief account of the MAODDP evaluation process will be presented, besides a general introduction to the simulation framework used for evaluation study, The next chapter will also highlight simulation environments for different sets of experiments, fields for each individual experiment, input parameters, results and graphical presentations of different results.

CHAPTER 6. MAODDP EVALUATION

6.1 Introduction

MAODDP was implemented in Java and evaluated on the Scalable Wireless Network Simulator (SWANS). Seven sets of experiments each comprising nine different tests were conducted. The first six sets of experiments were carried out under the MAODDP power saving mode and the last one without the power saving mode. Several parameters which include the number of nodes and mobility type were used to design different simulation environments for each of these experiments. MAODDP was found to be able of produce impressive data delivery while saving considerable amount of available memory. Moreover the MAODDP power aware mechanism was found effective in saving more memory and generating higher data delivery.

In this chapter a detailed account of various evaluation stages of MAODDP will be presented. In this regard a brief introduction to SWAN is presented in section 2. In section 3 the simulation environment is introduced. This section also contains details of different experiments, results and discussion specific to a particular set of experiments. General discussion is covered in section 4 and a brief summary of this chapter is given in section 5.

6.2 Introduction to Scalable Wireless Network Simulator (SWANS)

SWANS is based on the Java simulation framework (JIST). It is organized as independent software components and can be composed to form complete wireless simulations. SWANS capabilities are similar to ns2 and GloMoSim. SWANS components implement different types of applications; networking, routing and media access protocols; radio transmission, reception and noise models; signal propagation and fading models; and node mobility models. Below are some of the details related to SWANS design principles and how the simulator eases simulation in comparison with NS2 and GloMosim. We have found SWANS simple and effective in terms of evaluating MAODDP.

6.2.1 Design Highlights

Every SWANS component acts as a JIST entity and stores its own local state. Interaction among SWANS components takes place via exposed event-based interfaces. Different types of components exist in SWANS for various purposes. Examples of some of the components are components for constructing a node stack and components for a variety of mobility models and field configurations. This type of design pattern is helpful for simplifying problems by creating meaningful and simple simulation scenarios with relatively small and event-driven components. Unlike the design of ns2 and GloMoSim, SWANS can also explicitly

partition the simulation state and the degree of inter-dependence between components. This feature allows components to be readily interchanged with suitable alternative implementations of the common interfaces. Likewise, it also makes each simulated node be able to independently configure.

SWANS supports dynamically created objects. Objects like packets can traverse many different controls paths and can have highly variable lifetimes. Freeing unused packets is handled entirely by the garbage collector. It simplifies the memory management protocol and eliminates a common source of memory leaks that can accumulate over long simulation runs. An additional feature is the support of distributed simulation via node partitioning into individual fine-grained entities. This is due to JIST capability of reconfiguration without any change to entities across physical hosts running the simulation. Therefore these entities can be adjusted dynamically according to simulation communication patterns and it does not need to be homogeneous.

One of the distinguishing features of SWANS is the ability of running regular, unmodified Java network applications over the simulated network. This feature allows inclusion of existing Java-based software which includes peer-to-peer applications and application-level multicast protocols. Far greater scalability is possible as applications do not merely send packets to the simulator from other processes. In fact, they operate in simulation time within the same JiST process space.

6.2.3 Efficient Signal Propagation Using Hierarchical Routing

Efficient Modelling signal propagation within the wireless network is essential for scalable wireless simulation. The SWANS field entity must deliver that signal transmitter by radio entity to all radios that could be affected. Prior to which it considers fading, gain and path-loss. Some small subset of the radios on the field will be within reception range and a few more radios will be affected by the interference above sensitivity threshold. The remaining majority of the radios will not be tangibly affected by the transmission.

NS2 and GloMoSim implements a naïve signal propagation algorithm, which uses a slow O (n), linear search through all the radios to determine the node set within the reception neighbourhood of the transmitter. This clearly does not scale as the number of radios increases. NS2 has recently been improved with a grid based algorithm. Both of these algorithms in addition to a new efficient hierarchical binning have been implemented in SWANS.

6.2.4 Benefits over Other Simulators

The ns2 network simulator has been extended to support mobility and wireless networking protocols. NS2 follows clever "split object" design and thus allows Tcl-based script configuration of C-based object implementations. Based on numerous published results it is not easy to scale ns2 beyond a few hundred

simulated nodes. It is commonly known that ns2 scale with difficulty and needs substantial hardware resources to simulate a few thousand nodes.

GloMoSim is a relatively new simulator written in Parsec designed specifically for scalable network simulation. Having evaluated the computational and memory performance JiST against NS2, GloMoSim and Parsec it was found the JiST outperforms these systems in both time and space. Parsec runs very quickly and there are a number of reasons for this. It is compiled not interpreted and uses a modified gcc compiler to produce highly optimized binaries. It also uses non-pre-emptive logical processes to avoid system switching overhead. However, this process-oriented model exacts a very high memory cost per entity since each entity must store a program counter and its stack.

The GloMoSim design compensates for the high per entity overhead of Parsec by aggregating multiple node states into a single entity and inserting a level of indirection in the message dispatch path. This approach reduces the number of entities in the system at a cost of a performance penalty. Moreover, it also eliminates the transparency and many advantages inherent to a language-based approach. Message queuing and dispatch are efficient in NS2. NS2 uses a split object model across C and Tcl to support dynamic simulation configuration. This forces a specific coding pattern and it comes at a performance cost of replicating data between the two memory spaces. Thus, it exacts a high memory overhead. NS2 code written in C still runs fast however, Tcl-based functionality is almost two orders of magnitude slower. Additionally, both ns2 and GloMoSim suffer performance loss from their approaches to simulate configuration that eliminates opportunities for static optimizations.

JiST is based on a concurrent object model of execution and does not need node aggregation. Since entities are objects all within the same heap, the memory foot print is small. Moreover, no context switching overhead on a per event basis is present which makes dynamic Java byte-code compilation and optimization result in high computational data delivery. Since Java is a dynamic language, JiST does not require a split object model for configuration. Instead, reflection can be used to directly observe or modify the same objects used to run the simulations. These features eliminate the performance gap and the additional memory requirements.

6.3 Network Environment and Simulation Results

The SuSE Linux 10.1 operating environment was used for all the conducted experiments. A fixed field size of 500X500 metres was used and a varied node placement of grid type ranging from 15x15 to 30x30 was defined. The number of nodes was one of the varied parameters and was selected from the range of 25 to 450. A fixed data rate of 1 minute was used for all the simulation experiments. A fixed data rate of 1minute means an average rate of 1.0 data packet per minute. It further explains that some nodes can send 1

packet per minute; some can send 2 packets per minute and vice versa. The simulation stop time ranges from 600 to 800 seconds with a fixed start and resolution time of 10 and 60 seconds respectively. Resolution time defines the time where the simulation ends after nodes stop sending messages. In some of the simulations mobility was kept as static while for others different mobility patterns were used. For the first few experiments packet loss was declared as none and for the others one default packet loss was used. It must be noted that adding packet lost to the simulation does not really test anything new, since the simulation already have packet loss even without specifying it. The first six sets of experiments were carried out under power saving mode and the seventh one was without power saving operation.

The selection of the parameters is based on the fact that these parameters have been widely used in the cited simulation studies. Therefore, each set of experiment represents a different situation. These experiments were designed so that they can represent mobile ad-hoc network environments. Mobile nodes in an-ad-hoc networks are not static; therefore a varied field size is used in most of the experiments. Scalability is an interesting issue in the context of protocols of mobile ad-hoc networks. It justified our selection of varying the number of mobile nodes. Time is another factor which was selected as a varying parameter. It is due to the fact that mobile ad-hoc networks are subject to frequent topology changes which can affect data delivery rate. By varying the time parameter we can measure the protocol's performance in terms of handling such situations i.e. how quickly the protocol can re-establish connection to maintain data delivery.

Table 6.1 summarizes upper and lower limits of all the fields used to create different simulation environments. Definition and explanation of conclusions drawn from the simulation results are as follows.

Data delivery defines the ratio between the number of ACK sent and broadcasted RQDD.

Route formed: Defines number of new route added.

Elapsed time: It defines the time period in between simulation start time and simulation stop time.

Total Memory: Memory available for a simulation run.

Bandwidth Used: Memory used in a simulation run.

Memory saved: It is the difference of total memory and memory used in a simulation cycle.

Power efficiency is measured in terms of memory saved under normal operation and when the power aware operation is turned off.

Parameter	Lower limit (X,Y)	Upper Limit
Nodes	5	1500
Area(Grid)	5x5	40x40
Fields	500x500	3000x3000
Starting time	10	25
Ending time	100 (seconds)	1600 (seconds)
Resolution time	60	600
Mobility	Static	Model
Packet loss	None	Default

Table 6.1 Upper and lower bonds of experiments fields.

6.3.1 Increasing Node Density in a Fixed Field

Node ranges from 25 to 450 mobile nodes were placed in a varied grid area of range from 5x5 to 25x25. Experimentation revealed that the range of a node in SWAN using default parameters for the field is around 700 metres. Therefore, a two dimension fixed field size of 500x500 metres could suffice to make maximum number of nodes available in the range of other. Mobility was taken as static with a constant data rate of one minute. Simulation started and ended at 10 and 600 seconds respectively with a fixed resolution time of 60 seconds. Packet lost was defined as none. Details of all the table fields from experiment number one to experiment number nine are given in fields table 6.3.1 A to table 6.3.1 I and screen shots of each of these experiment is included in the appendix.

Based on the results obtained the following conclusions were made. Message activity as shown in figure graph 6.3.1 (A) and 6.3.1(B) both in terms of RQDD broadcast and ACK sent increased with the addition of mobile nodes. In figure graph 6.3.1(C) message activity increased with time apart from at the middle of the simulation where a slight drop could be observed. One interesting results was about route formed in a simulation cycle. It could be seen in table 6.3.1 and in figure graph 6.3.1(D) that addition of mobile nodes also steps up the possibility of new route formation among the mobile nodes. It is difficult especially in mobile ad-hoc networks to obtain constant data delivery. This can be seen in figure graph 6.3.1(E) and 6.3.1(F). In both of these graphs a variation in data delivery could be observed. This variation in data delivery is also due to the variation of some other fields. Some of these fields are grid size and field area, as it was expected these two factors had their own impact on the over all data delivery ration. On the basis of different simulation results of this set of experiments the overall average data delivery calculated was 69.78 which further explain that out of 100 broadcast RQDD almost 70 were delivered to the destination. SWANS allocate total memory and memory used in a simulation cycle and is displayed at the end of each simulation run. The amount of memory consumed is increased with the addition of mobile nodes as shown in figure

graph 6.3.1(E). More mobile nodes may mean more RQDD broadcast by mobile nodes thus requiring more memory. Figure graph 6.3.1(F) and 6.3.1(G) further illustrates total memory, memory used and memory saved respectively. Calculation revealed that on average 49.26 % memory was saved in this set of experiments. Elapsed time increased linearly with the addition of mobile nodes which can be observed in figure graph 6.3.1(H).

Fields tables experiment no 6.3.1

Number of Nodes	25
Field Size	500x500
Grid size	5x5
Start time	10
End Time	600
Data Rate	1 min

Table 6.3.1A of exp no 6.3.1A

Number of Nodes	50
Field Size	500x500
Grid size	10x10
Start time	10
End Time	600
Data Rate	1 min

Table 6.3.1B of exp of 6.3.1B

Number of Nodes	75
Field Size	500x500
Grid size	10x10
Start time	10
End Time	600
Data Rate	1 min

Table 6.3.1C of exp of 6.3.1C

Number of Nodes	100
Field Size	500x500
Grid size	10x10
Start time	10
End Time	600
Data Rate	1 min

Table 6.3.1C of exp of 6.3.1D

Number of Nodes	125
Field Size	500x500
Grid size	15x15
Start time	10
End Time	600
Data Rate	1 min

Table 6.3.1D of exp of 6.3.1E

Number of Nodes	150
Field Size	500x500
Grid size	15x15
Start time	10
End Time	600
Data Rate	1 min

Table 6.3.1E of exp of 6.3.1F

Number of Nodes	250
Field Size	500x500
Grid size	20x20
Start time	10
End Time	600
Data Rate	1 min

Table 6.3.1F of exp of 6.3.1G

Number of Nodes	350
Field Size	500x500
Grid size	20x20
Start time	10
End Time	600
Data Rate	1 min

Table 6.3.1G of exp of 6.3.1H

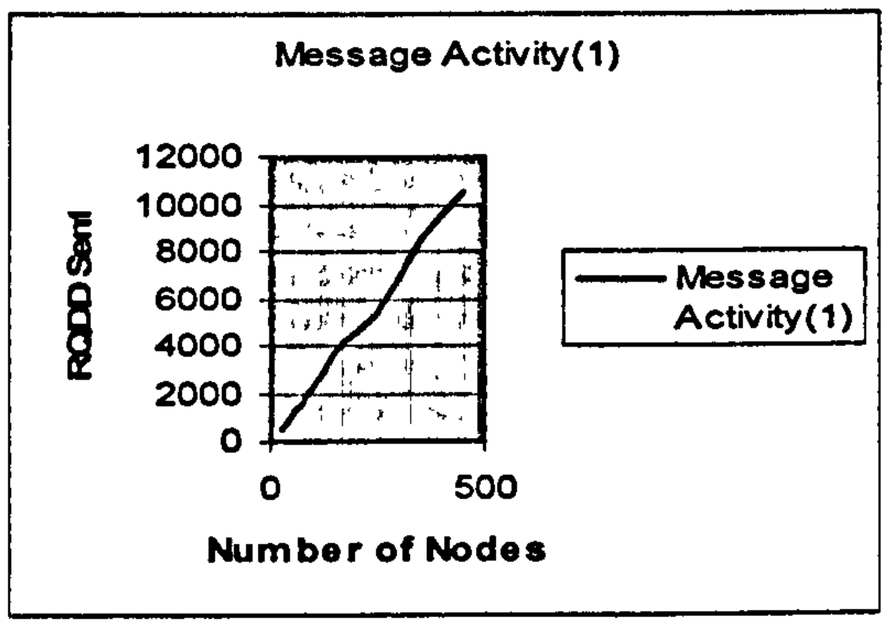
Number of Nodes	450
Field Size	500x500
Grid size	25x25
Start time	10
End Time	600
Data Rate	1 min

Table 6.3.1H of exp of 6.3.1I

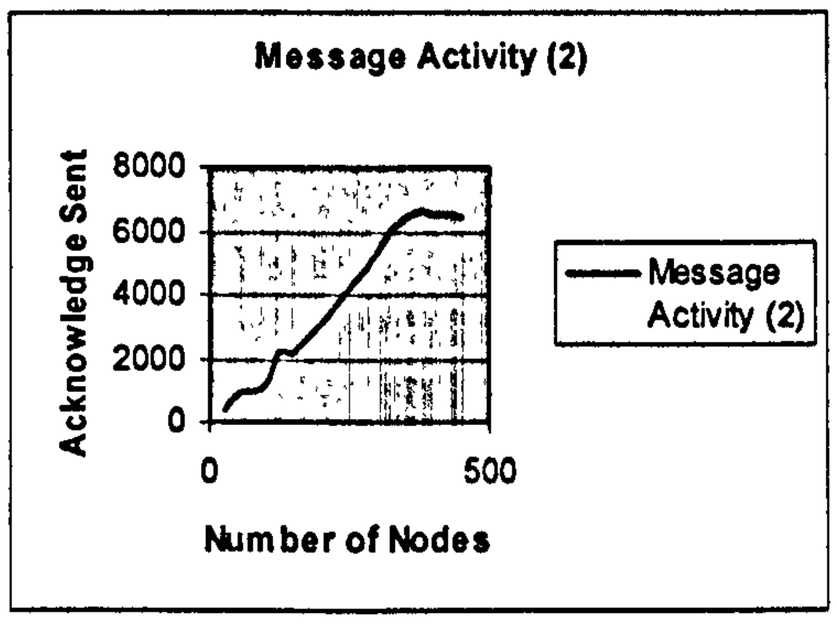
Nodes	RQDD Sent	ACK received	Fields	Data delivery %cnt	Routes formed	Elapsed Time	Total Memory	Memory used	Memory Saved
25	460	357	500, 500	77.60	46	6372	9150464	4900128	4250336
50	1073	831	500, 500	77.44	108	16136	2748416	1311056	1437360
75	1533	987	500, 500	64.38	153	38861	3158016	1407760	1750256
100	2260	1306	500, 500	57.78	225	87694	3649536	1785136	1864400
125	2944	2235	500, 500	75.91	295	215818	4255744	2198488	2057256
150	3766	2278	500, 500	60.48	378	321056	4935680	2598208	2337472
250	5509	4268	500, 500	77.47	678	1451020	9412608	4926616	4485992
350	8485	6452	500, 500	76.04	851	4137872	15515648	8123496	7392152
450	10516	6412	500, 500	60.97	1055	6691933	25112576	12290944	12821632
	Average(3009.55)	Average (2791.77)		Average(69.78)	Average 421				49.26 %cnt saved

Table 6.3.2 Results chart of experiments 6.3.1

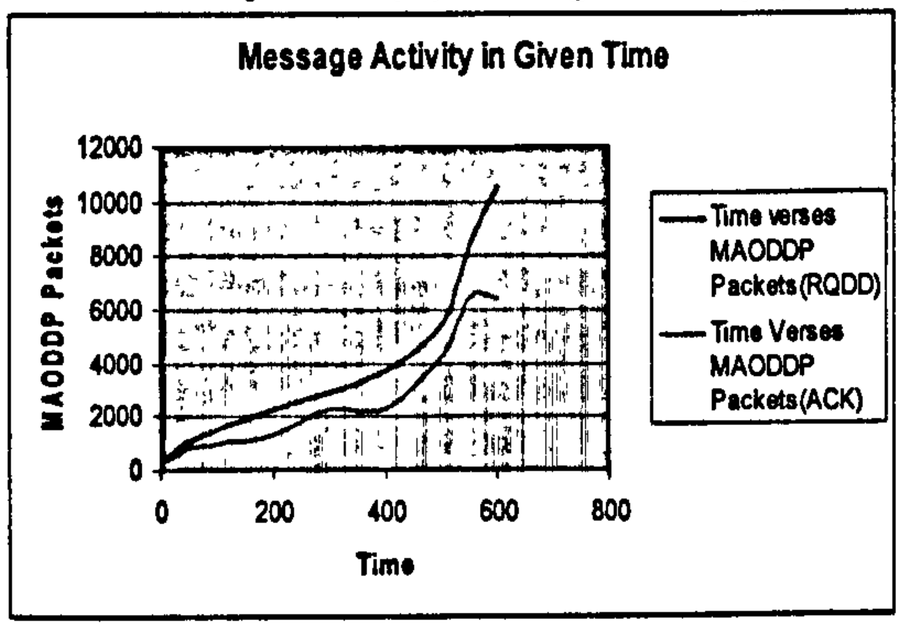
Figures results graphs of experiments 6.3.1



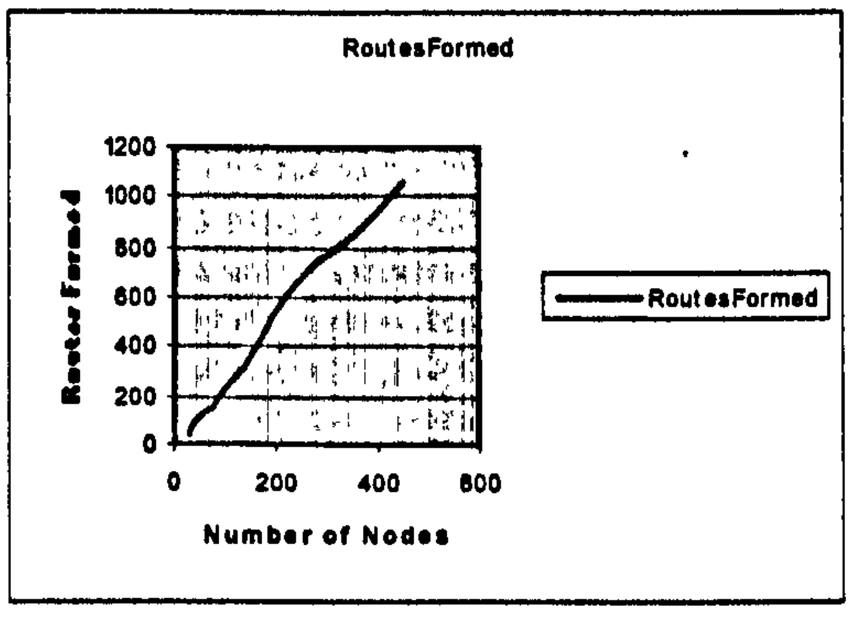
Graph 6.3.1(A) Message Activity (1)



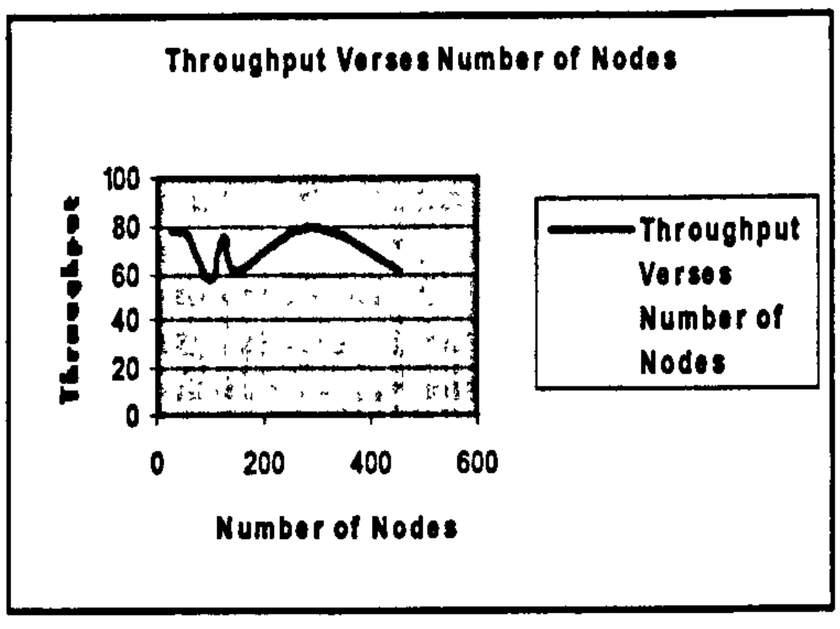
Graph 6.3.1(B) Message Activity (2)



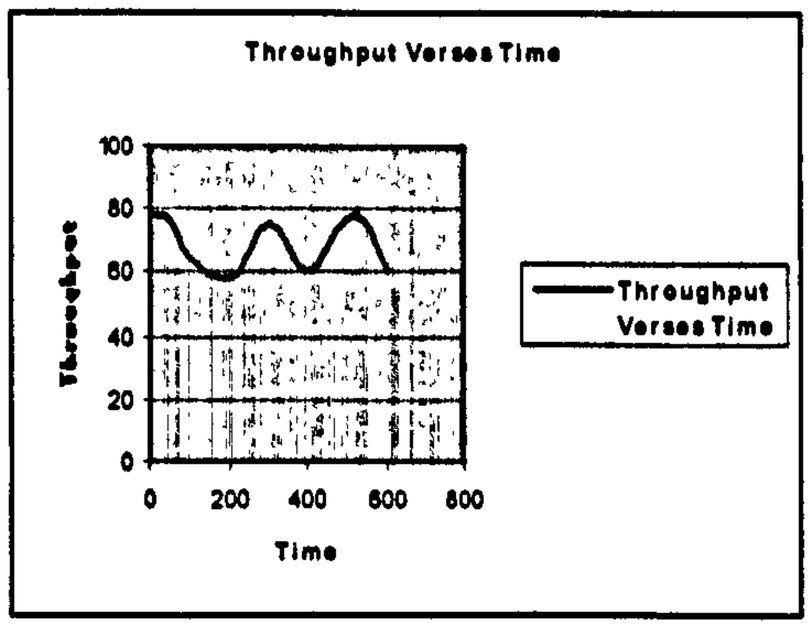
Graph 6.3.1© Message Activity in Given Time



Graph 6.3.1(D) Route Formed

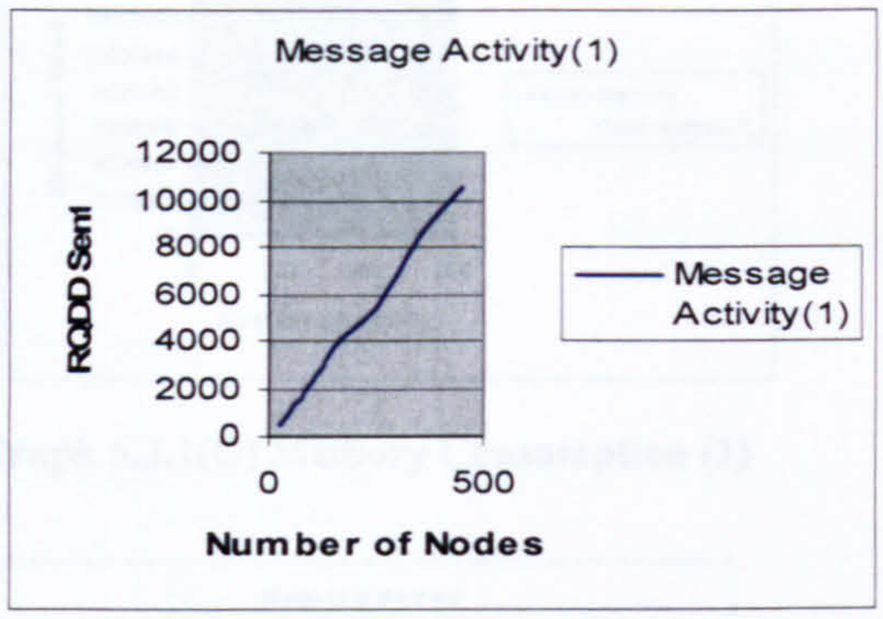


Graph 6.3.1(E) Data delivery Verses Number of Nodes

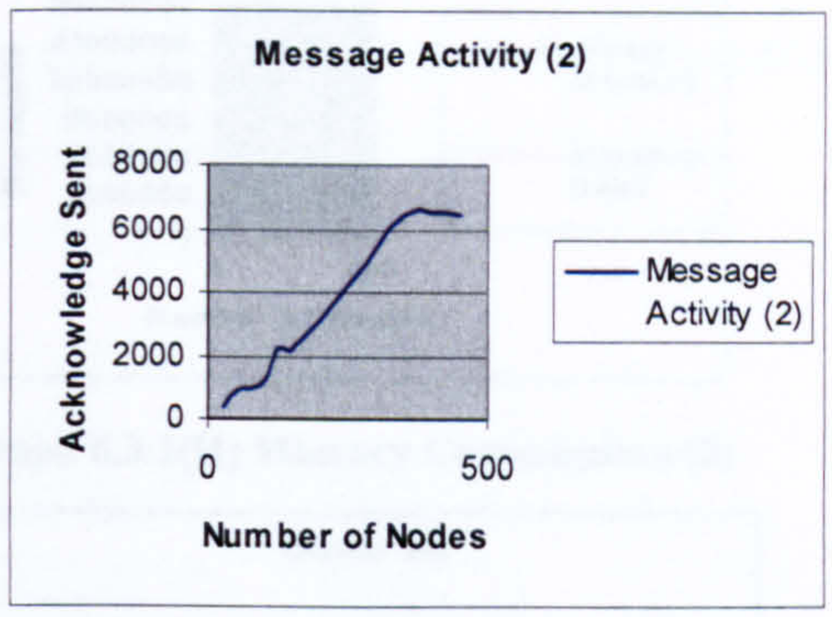


Graph 6.3.1(F) Data delivery Verses Time

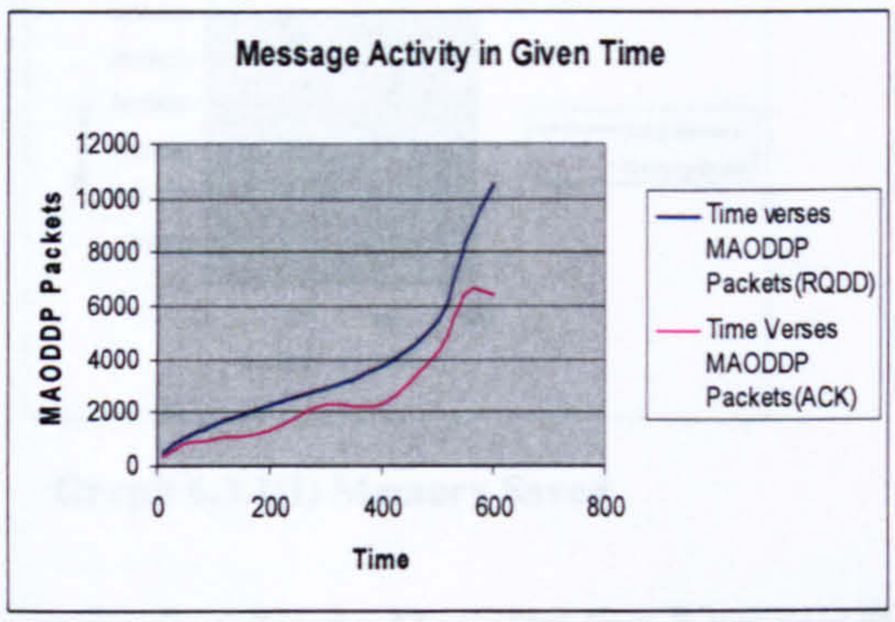
Figures results graphs of experiments 6.3.1



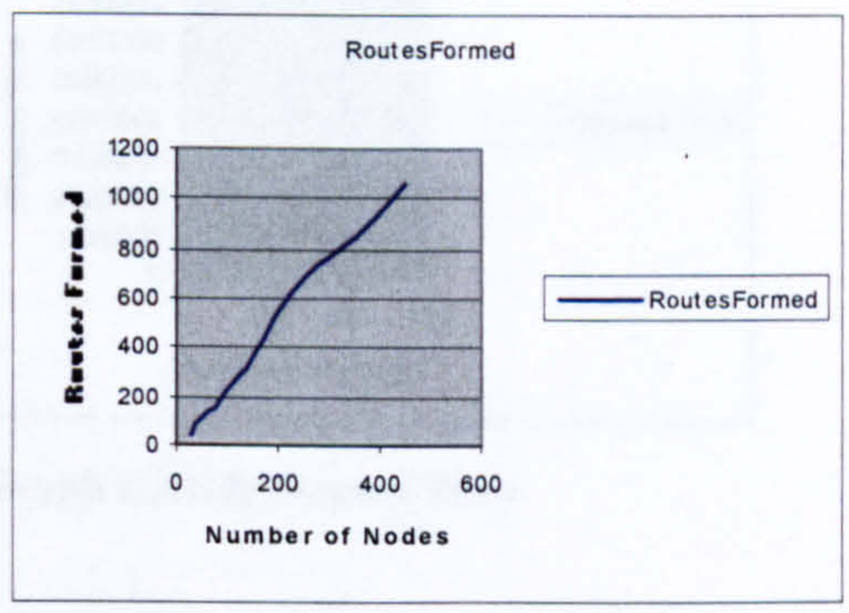
Graph 6.3.1(A) Message Activity (1)



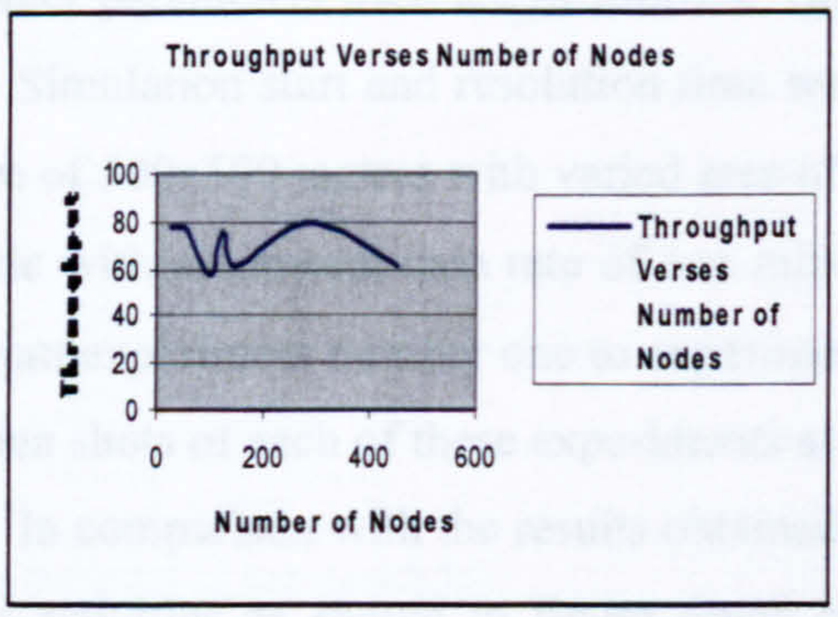
Graph 6.3.1(B) Message Activity (2)



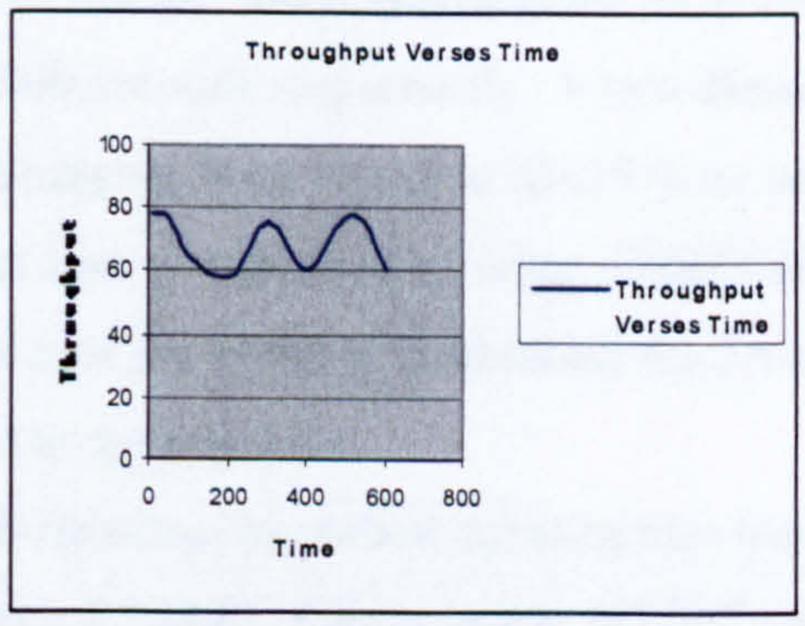
Graph 6.3.1© Message Activity in Given Time



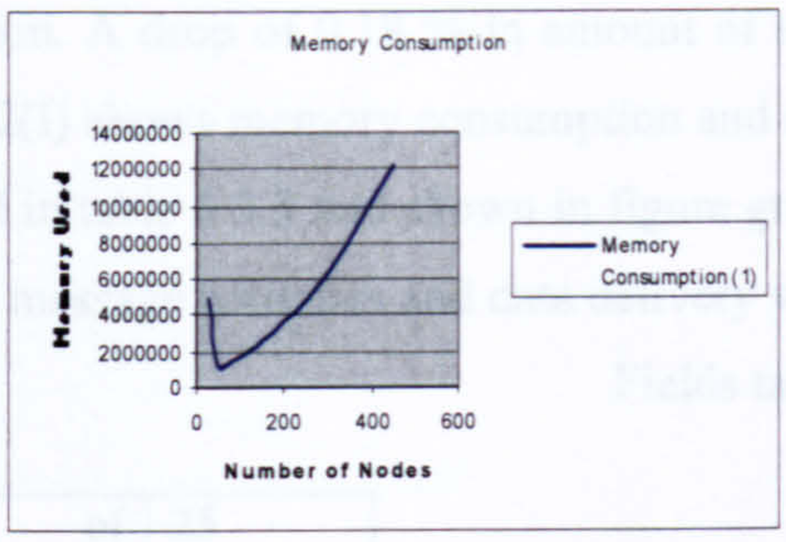
Graph 6.3.1(D) Route Formed



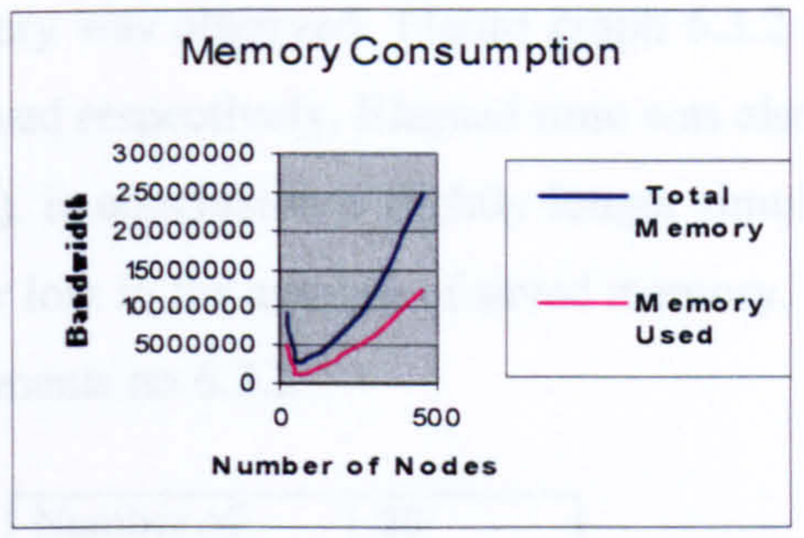
Graph 6.3.1(E) Data delivery Verses Number of Nodes



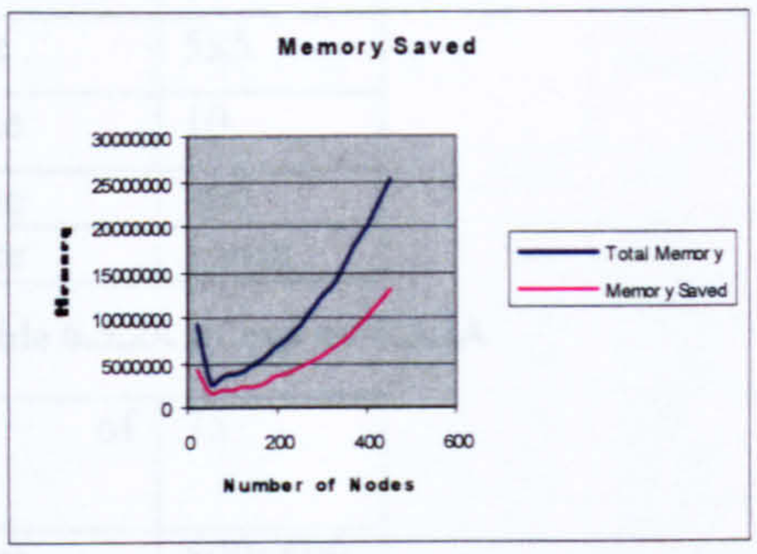
Graph 6.3.1(F) Data delivery Verses Time



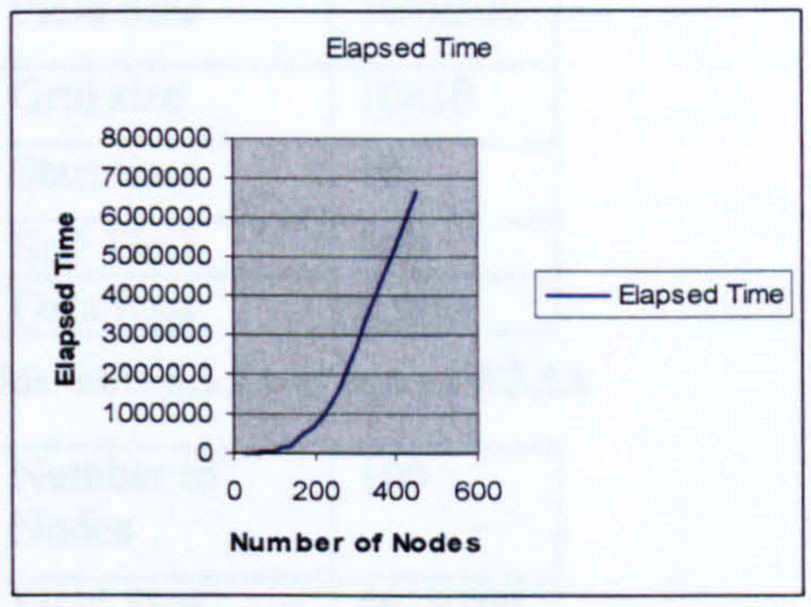
Graph 6.3.1(G) Memory Consumption (1)



Graph 6.3.1(H) Memory Consumption (2)



Graph 6.3.1(I) Memory Saved



Graph 6.3.1(J) Elapsed Time

6.3.2 Increasing Node Density for Longer Run

In this set of experiments the simulation ending time has increased by 200 seconds. Details of some of the other parameters used are as follows. The number of mobile nodes was selected from the range of 25 to 450. Simulation start and resolution time were 10 and 60 seconds respectively. A two dimensioned fixed field size of 500x500 metres with varied area of type grid ranging from 15x15 to 25x25 were used. Mobility was static with a constant data rate of one minute. Packet loss was defined as none. Details of all the table fields from experiment number one to experiment number nine are given in field tables 6.3.2A to table 6.3.2I and screen shots of each of these experiments are included in the appendix.

In comparison with the results obtained in 6.3.1 increasing simulation duration also increases overall message activities as shown in figure graph 6.3.2(A) and 6.3.2(B). Figure graph 6.3.2(C) shows a clear increment in message activities with respect to time. More routes were formed within the similar arrangement of each experiment as can also be observed in table 6.3.3 and in figure graph 6.3.2 (D). A sharp and dramatically increased data delivery both with respect to number of nodes and time were observed and can be observed in figure graph 6.3.2(E) and 6.3.2(F) respectively. From the statistics of table 6.3.3 average data delivery was calculated as 91.27, an increase of approximately 76% in comparison with 6.3.1. Having average data delivery of 91.27 means out of 100 RQDD packets almost 91 were successfully delivered to the

destination. A drop of 0.19 % in amount of saved memory was observed. Figure graph 6.3.2 (G), 6.3.2(H) and 6.3.2(I) shows memory consumption and memory saved respectively. Elapsed time was also increased as recorded in table 6.3.3 and shown in figure graph 6.3.3(J). In conclusion a slightly longer simulation run can increase message activities and data delivery with a minor loss in the amount of saved memory.

Fields tables experiments no 6.3.2

Number of Nodes	25
Field Size	500x500
Grid size	5x5
Start time	10
End Time	800
Data Rate	1 min

Fields table 6.3.2A of exp no 6.3.2A

Number of Nodes	50
Field Size	500x500
Grid size	10x10
Start time	10
End Time	800
Data Rate	1 min

Fields table 6.3.2A of exp no 6.3.2A

Number of Nodes	75
Field Size	500x500
Grid size	10x10
Start time	10
End Time	800
Data Rate	1 min

Fields table 6.3.2A of exp no 6.3.2A

Number of Nodes	100
Field Size	500x500
Grid size	10x10
Start time	10
End Time	800
Data Rate	1 min

Fields table 6.3.2B of exp no 6.3.2B

Number of Nodes	125
Field Size	500x500
Grid size	15x15
Start time	10
End Time	800
Data Rate	1 min

Fields table 6.3.2C of exp no 6.3.2C

Number of Nodes	150
Field Size	500x500
Grid size	15x15
Start time	10
End Time	800
Data Rate	1 min

Fields table 6.3.2D of exp no 6.3.2D

Number of Nodes	250
Field Size	500x500
Grid size	20x20
Start time	10
End Time	800
Data Rate	1 min

Fields table 6.3.2E of exp no 6.3.2E

Number of Nodes	350
Field Size	500x500
Grid size	20x20
Start time	10
End Time	800
Data Rate	1 min

Fields table 6.3.2E of exp no 6.3.2E

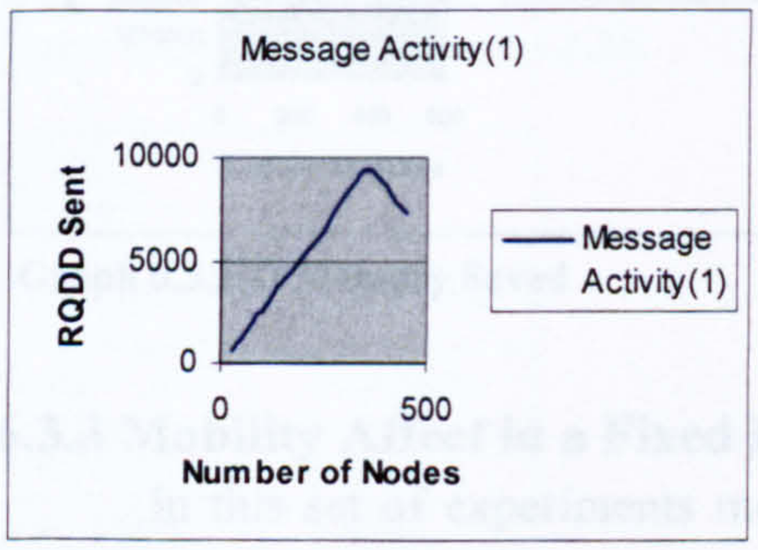
Number of Nodes	450
Field Size	500x500
Grid size	25x25
Start time	10
End Time	600
Data Rate	1 min

Fields table 6.3.2F of exp no 6.3.2F

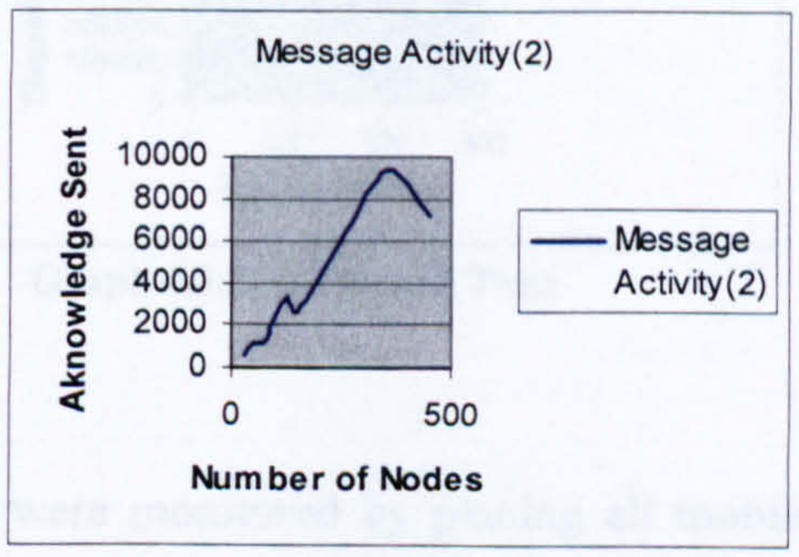
Table 6.3.3 Results chart of experiments no 6.3.2

Number of Nodes	RQDD Sent	ACK received	Fields X,Y	Data delivery %cnt	Routes Formed	Elapsed Time	Total memory	Memory used	Memory Saved
25	558	558	500, 500	100	48	5144	2498560	1145816	1352744
50	1249	1249	500, 500	100	144	24359	2768896	1313384	1455512
75	2033	1267	500, 500	62.32	183	52972	3215360	1418064	1797296
100	2533	2478	500, 500	97.82	243	151909	3600384	1652744	1947640
125	3400	3400	500, 500	100	310	325028	4329472	2202904	2126568
150	4066	2584	500, 500	63.55	405	380797	4984832	2599832	2385000
250	6455	6363	500, 500	98.57	623	2126693	9666560	4586512	5080048
350	9472	9400	500, 500	99.23	970	5850791	15597568	7631008	7966560
450	7258	7256	500, 500	99.97	1140	7798792	25513984	13728664	11785320
	4113.77	3839.44		**Average(91.27)**	451.77				49.7 % saved

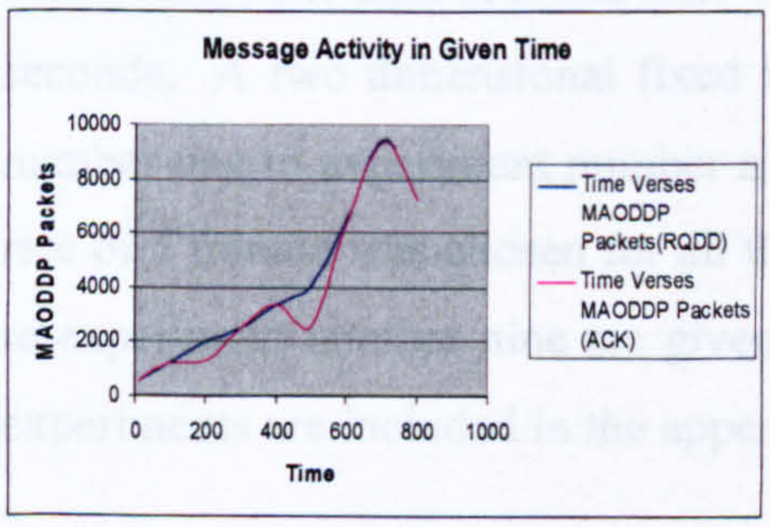
Figures results graphs of experiments no 6.3.2



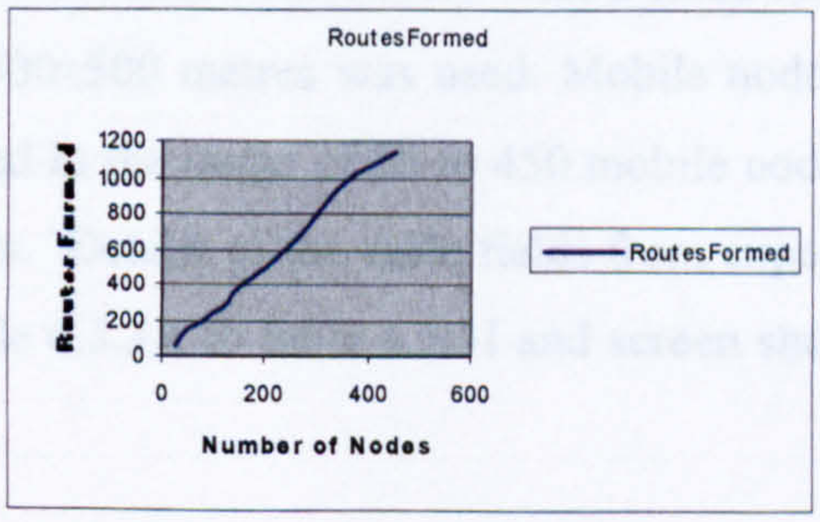
Graph 6.3.2(A) Message Activity (1)



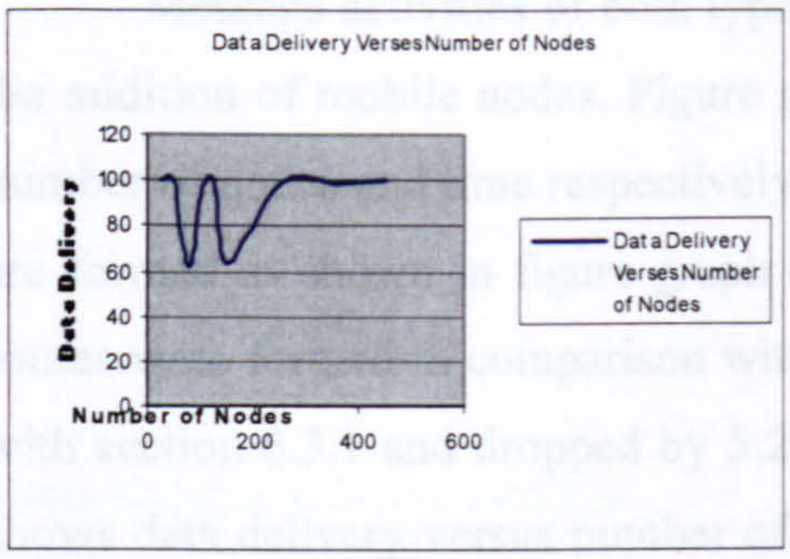
Graph 6.3.2(B) Message Activity (2)



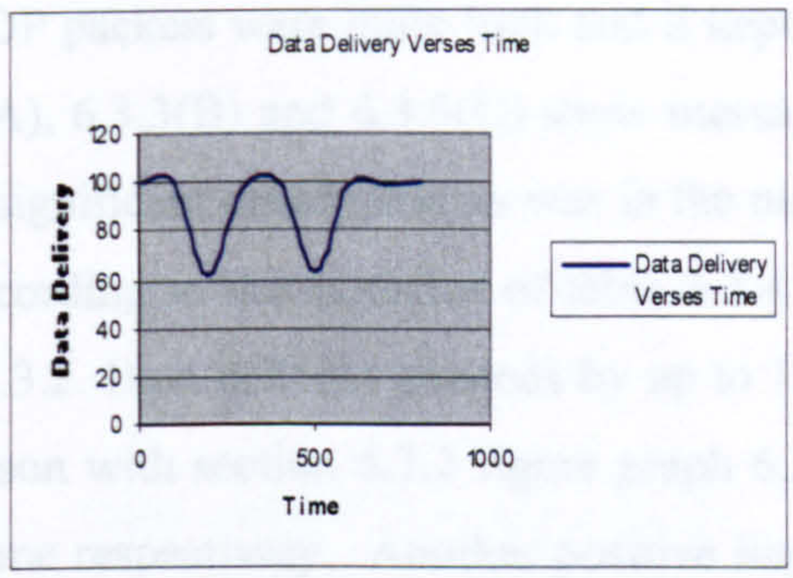
Graph 6.3.2© Message Activity in Given Time



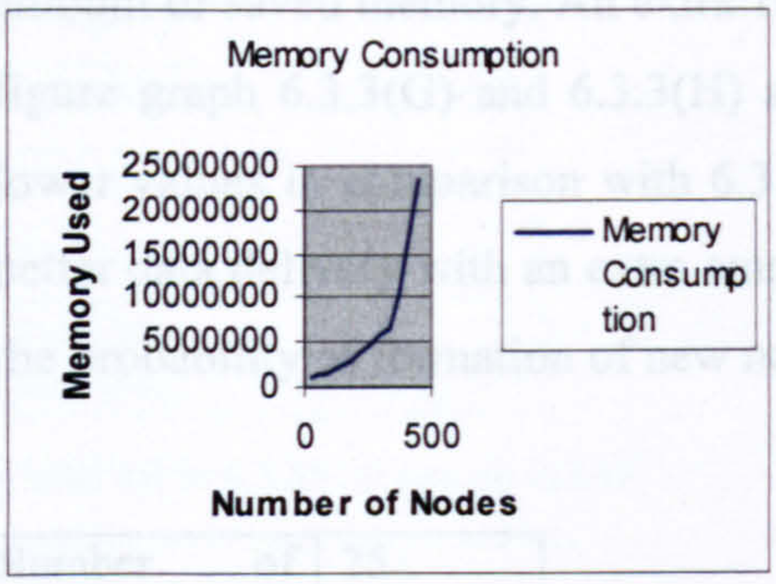
Graph 6.3.2(D) Route Formed



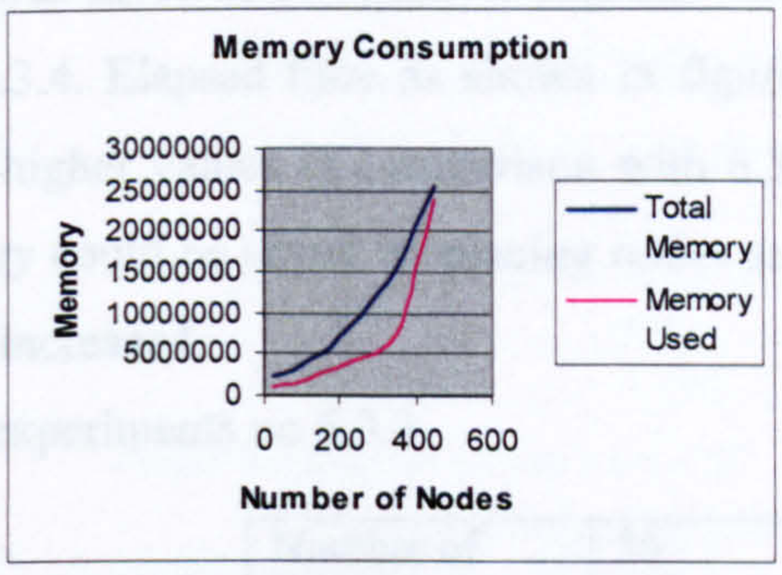
Graph 6.3.2(E) Data delivery Verses Number of Nodes



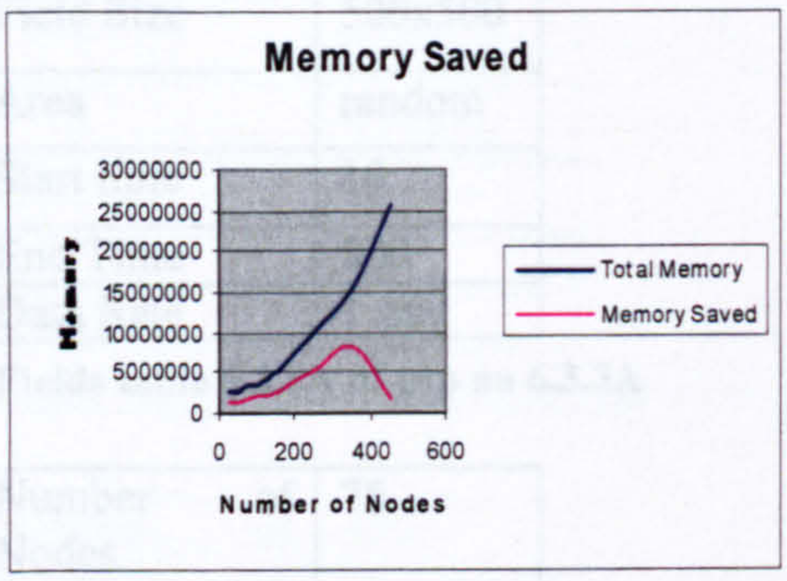
Graph 6.3.2(F) Data delivery Verses Time



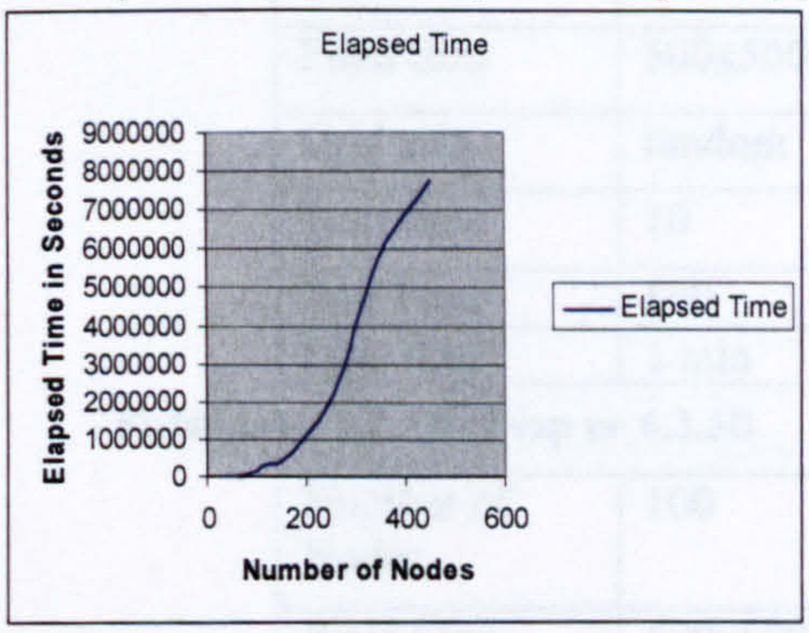
Graph 6.3.2(G) Memory Consumption (1)



Graph 6.3.2(H) Memory Consumption (2)



Graph 6.3.2(I) Memory Saved



Graph 6.3.2(J) Elapsed Time

6.3.3 Mobility Affect in a Fixed Field

In this set of experiments mobility effects were monitored by placing all mobile nodes at random. Simulation start time and end time was 10 and 800 seconds respectively with a fixed resolution time of 60 seconds. A two dimensional fixed field size of 500x500 metres was used. Mobile nodes from experiment number one to experiment number nine were varied in the range of 25 to 450 mobile nodes. A constant data rate of 1 minute was chosen for all the experiments. Details of the table fields from experiment number one to experiment number nine are given in fields table 6.3.3A to table 6.3.3I and screen shots of each of these experiments are included in the appendix.

Message activities of both types of MAODDP packets were quite high and it kept on increasing with the addition of mobile nodes. Figure graphs 6.3.3(A), 6.3.3(B) and 6.3.3(C) show message activities versus number of nodes and time respectively. One of the significant developments was in the number of routes that are formed as shown in figure graph 6.3.3(D). According to the statistics of table 6.3.4 higher numbers of routes were formed in comparison with 6.3.1 and 6.3.2. Data delivery exceeds by up to 16.36 in comparison with section 6.3.1 and dropped by 5.21 in comparison with section 6.3.2 figure graph 6.3.3(E) and 6.3.3(F) shows data delivery versus number of nodes and time respectively. Another positive improvement is in the amount of saved memory. An extra 16 % memory was saved in comparison with 6.3.1 and 6.3.2 as shown in figure graph 6.3.3(G) and 6.3.3(H) and in table 6.3.4. Elapsed time as shown in figure graph 6.3.3(I) has lower values in comparison with 6.3.2 and shows higher values in comparison with 6.3.1. In conclusion a better data delivery with an extra amount of memory could be saved by placing nodes at random. Moreover the probability of formation of new nodes was also increased.

Fields tables experiments no 6.3.3

Number of Nodes	25
Field Size	500x500
Area	random
Start time	10
End Time	800
Data Rate	1 min

Fields table 6.3.3A of exp no 6.3.3A

Number of Nodes	50
Field Size	500x500
Grid size	random
Start time	10
End Time	800
Data Rate	1 min

Fields table 6.3.3B of exp no 6.3.3B

Number of Nodes	75
Field Size	500x500
Area	random
Start time	10
End Time	800
Data Rate	1 min

Number of Nodes	100
Field Size	500x500
Area	random
Start time	10
End Time	800
Data Rate	1 min

Fields table 6.3.3C of exp no 6.3.3C

Number of Nodes	125
Field Size	500x500
Grid size	random
Start time	10
End Time	800
Data Rate	1 min

Fields table 6.3.3D of exp no 6.3.3D

Number of Nodes	150
Field Size	2000x2000
Grid size	40x40
Start time	10
End Time	800
Data Rate	1 min

Fields table 6.3.3E of exp no 6.3.3E

Number of Nodes	250
Field Size	500x500
Area	random
Start time	10
End Time	800
Data Rate	1 min

Fields table 6.3.3F of exp no 6.3.3F

Number of Nodes	350
Field Size	500x500
Grid size	30x30
Start time	10
End Time	800
Data Rate	1 min

Fields table 6.3.3G of exp no 6.3.3G

Number of Nodes	450
Field Size	500x500
Area	random
Start time	10
End Time	800
Data Rate	1 min

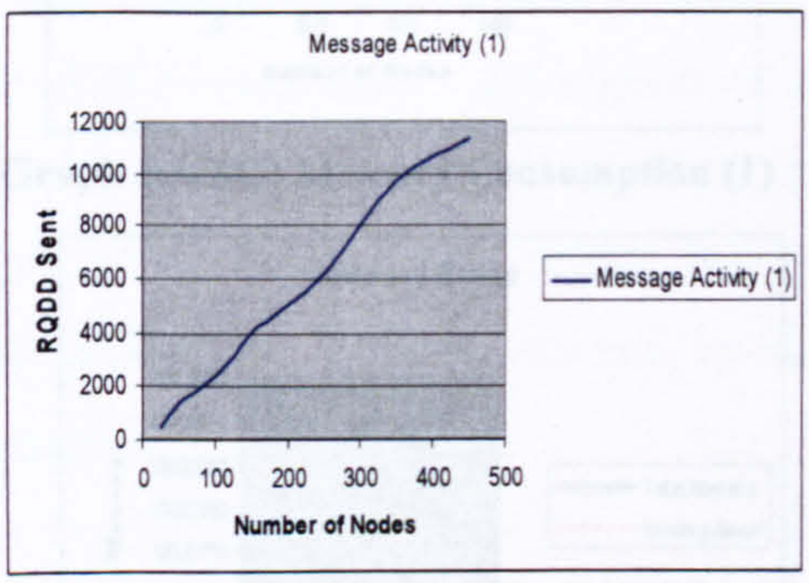
Fields table 6.3.3H of exp no 6.3.3H

Fields table 6.3.3I of exp no 6.3.3I

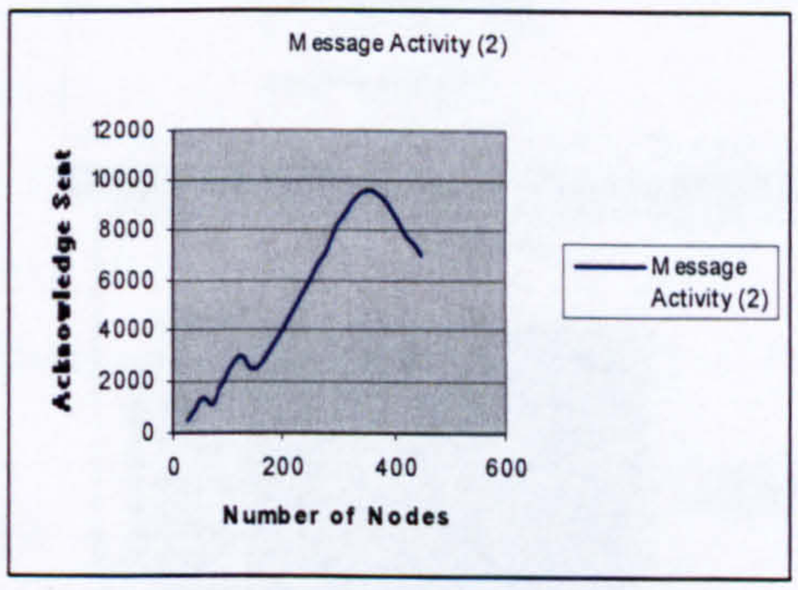
Number of Nodes	RQDD Sent	ACK received	Fields X,Y	Data delivery %cnt	Routes Formed	Elapsed Time	Total Memory	Memory used	Memory Saved
25	480	480	500, 500	100	86	4703	2498560	1144760	1353800
50	1437	1437	500, 500	100	171	26366	2764800	1308056	1456744
75	1894	1104	500, 500	58.28	284	45263	3203072	1413336	1789736
100	2435	2374	500, 500	97.49	349	174872	3608576	1674944	1933632
125	3128	3088	500, 500	98.72	434	303749	44378624	2225728	42152896
150	4069	2510	500, 500	61.68	638	363223	4960256	2604768	2355488
250	6219	6114	500, 500	98.31	1056	2058006	9654272	4616762	5037510
350	9747	9591	500, 500	98.39	1507	5922651	15491072	8285976	7205096
450	11386	6982	500, 500	61.32	2003	7468195	25739264	13452654	12286610
	4532.77	3742.22		**Average(86.021)**	725.333				67.29 %cnt saved

Table 6.3.4 Results chart of experiments No 6.3.3

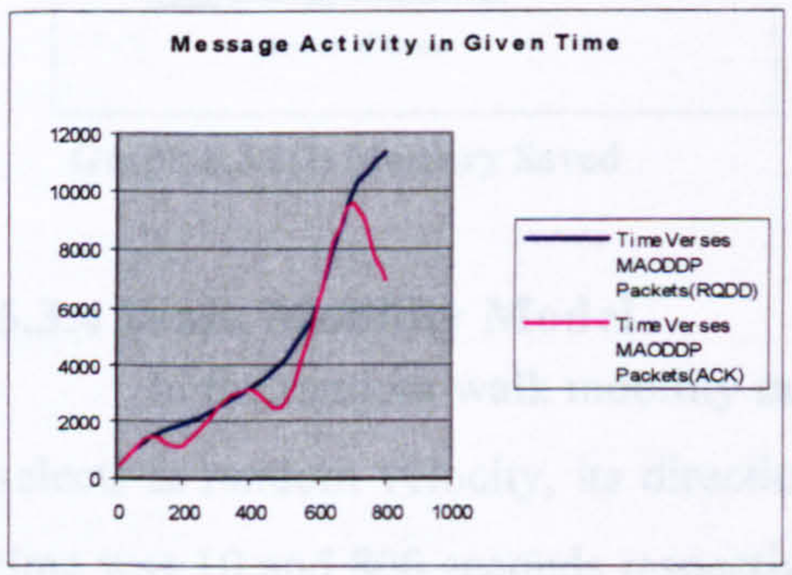
Figures results graphs of experiments no 6.3.3



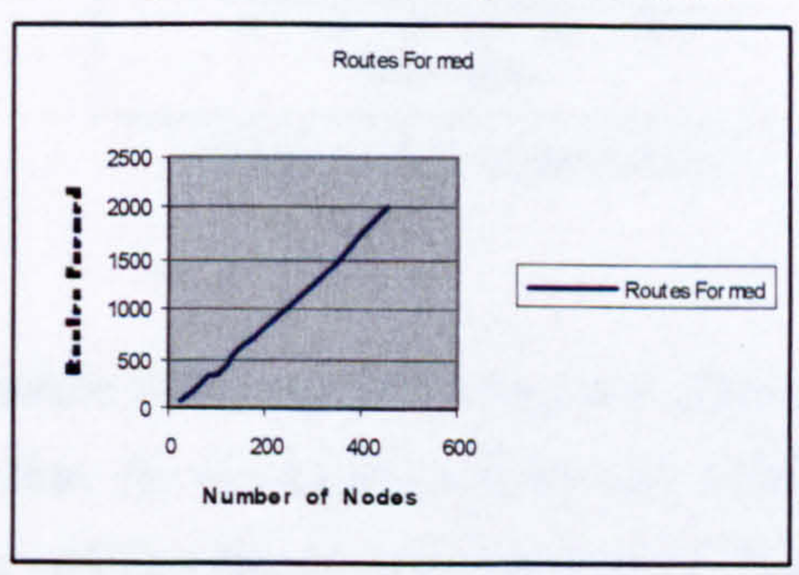
Graph 6.3.3(A) Message Activity (1)



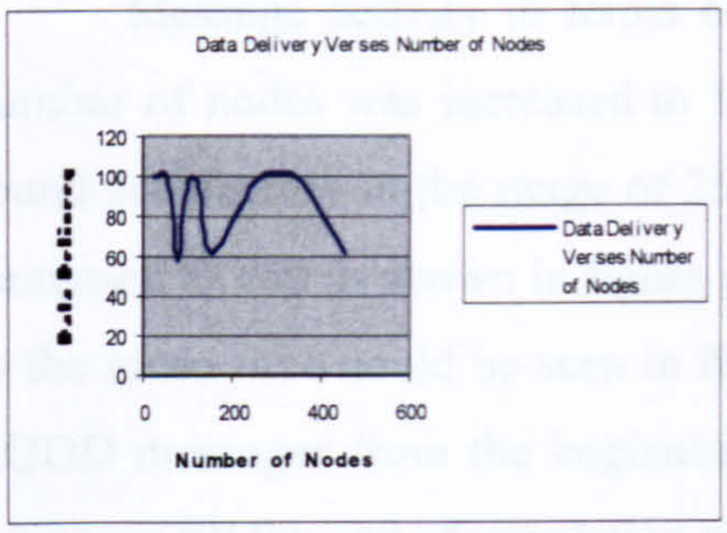
Graph 6.3.3(B) Message Activity (2)



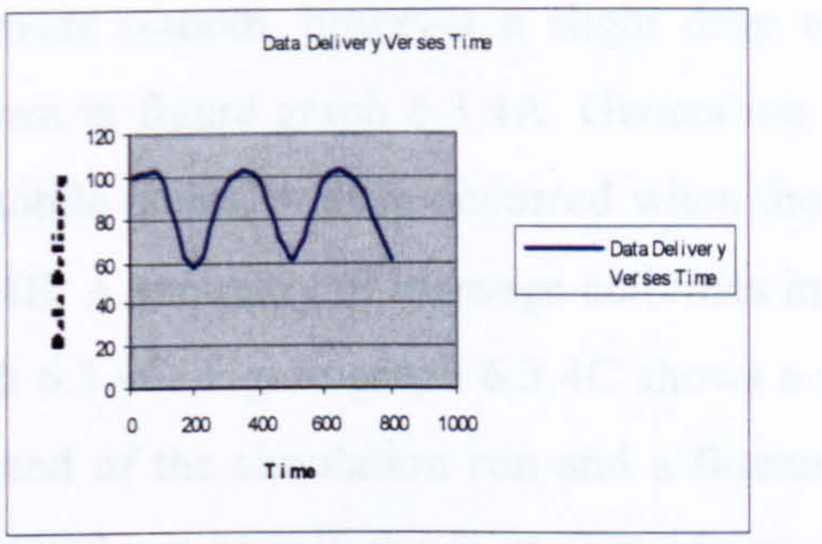
Graph 6.3.3© Message Activity in Given Time



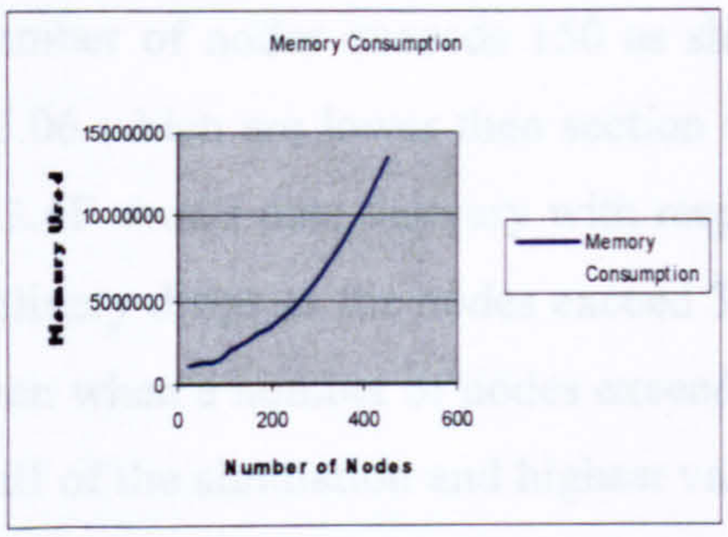
Graph 6.3.3(D) Routes Formed



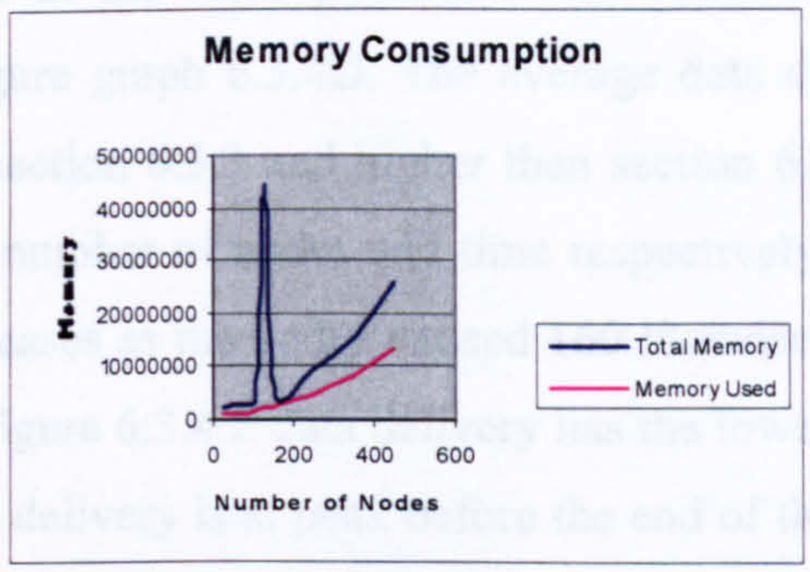
Graph 6.3.3(E) Data delivery Verses Number of Nodes



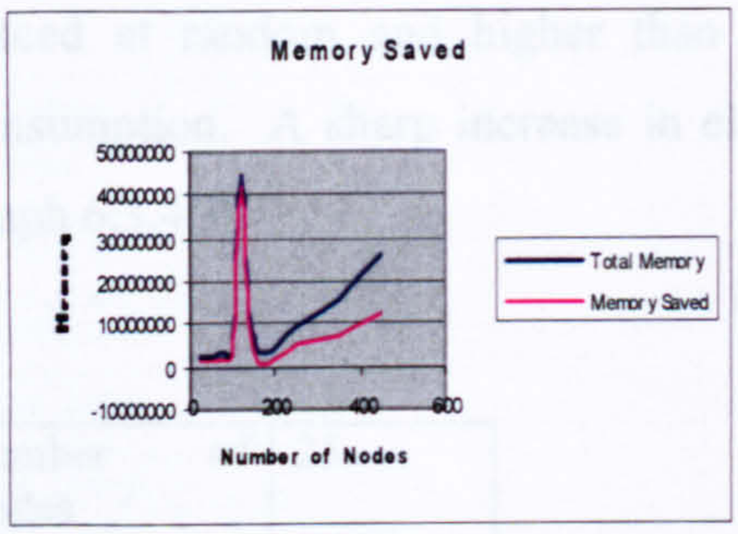
Graph 6.3.3(F) Data delivery Verses Time



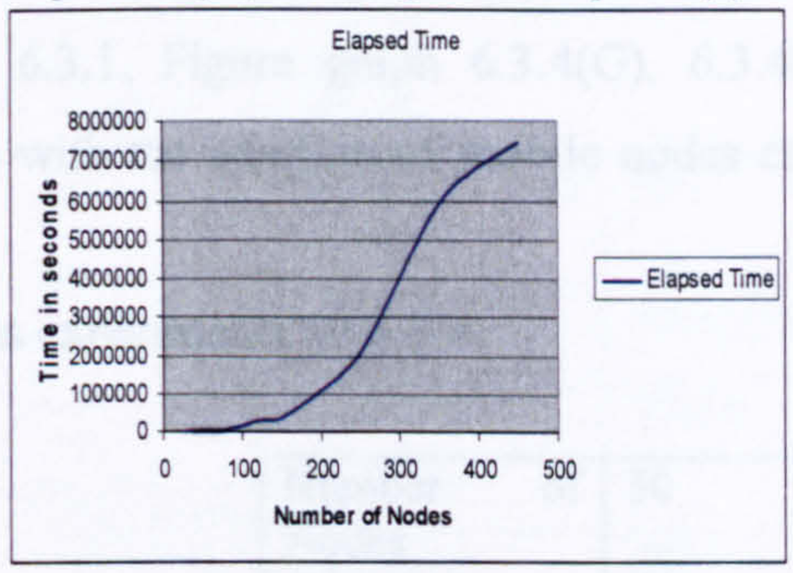
Graph 6.3.3(G) Memory Consumption (1)



Graph 6.3.3(H) Memory Consumption (2)



Graph 6.3.3(I) Memory Saved



Graph 6.3.3(J) Elapsed Time

6.3.4 Walk Mobility Model

In the random walk mobility model mobile nodes move in turn. Each node at the beginning of its turn selects at random velocity, its direction and time for the duration of its turn. Simulation start time and end time was 10 and 800 seconds respectively. In addition fixed resolution time of 60 seconds and pause time of 10 seconds were chosen for all experiments. Mobile nodes were placed in a fixed grid area of 30x30 within a two dimensional fixed field size of 500x500 metres. A varied value of mobile nodes in the range of 25 to 450 mobile nodes from experiment number one to experiment number nine with a constant data rate of one minute was used. Details of the table fields from experiment number one to experiment number nine are given in field table 6.3.4A to table 6.3.4I and screen shots of each of these experiments are included in the appendix.

Message activity in terms of RQDD were smooth, however a slight drop was observed when the number of nodes was increased to 120 as shown in figure graph 6.3.4A. Generation of ACK messages was found satisfactory in the range of 25 to 450 mobile nodes. A drop occurred when the numbers of nodes was increased to 450 as shown in figure graph 6.3.4B. A summary of message activities in this set of experiments in the given time could be seen in figure graph 6.3.4C. Figure graph 6.3.4C shows a smooth transmission of RQDD messages from the beginning till the end of the simulation run and a fluctuating response of ACK messages till the end of simulation time. MAODDP response in the formation of new routes was similar as in section 6.3.2 and section 6.3.3. A slight drop in the route formation was observed especially when the number of nodes exceeds 150 as shown in figure graph 6.3.4D. The average data delivery calculated was 75.06 which are lower then section 6.3.2 and section 6.3.3 and higher than section 6.3.1. Figure 6.3.4E and 6.3.4F shows data delivery with respect to the number of nodes and time respectively. In figure 6.3.4E data delivery drops as the nodes exceed 30 and increases as the nodes exceed 160 likewise another drop could be seen when a number of nodes exceeds 400. In figure 6.3.4.F data delivery has the lowest value just before the half of the simulation and highest value of data delivery is at peak before the end of the simulation. Statistics in table 6.3.5 showed that 56.90 % memory was saved. This figure is lower than 6.3.3 where nodes were placed at random and higher than 6.3.2 and 6.3.1. Figure graph 6.3.4(G), 6.3.4(H) showed memory consumption. A sharp increase in elapsed time with the addition of mobile nodes could be seen in figure graph 6.3.4(I).

Fields tables experiments no 6.3.4

Number of Nodes	25
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.4A of exp no 6.3.4A

Number of Nodes	50
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.4A of exp no 6.3.4A

Number of Nodes	75
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.4A of exp no 6.3.4A

Number of Nodes	100
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.4A of exp no 6.3.4A

Number of Nodes	125
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.4A of exp no 6.3.4A

Number of Nodes	150
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.4A of exp no 6.3.4A

Number of Nodes	250
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.4A of exp no 6.3.4A

Number of Nodes	350
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.4A of exp no 6.3.4A

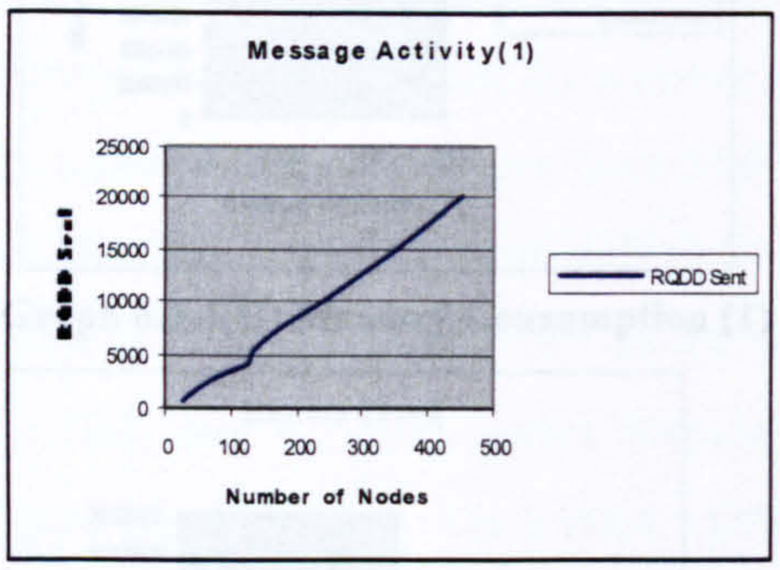
Number of Nodes	250
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.4A of exp no 6.3.4A

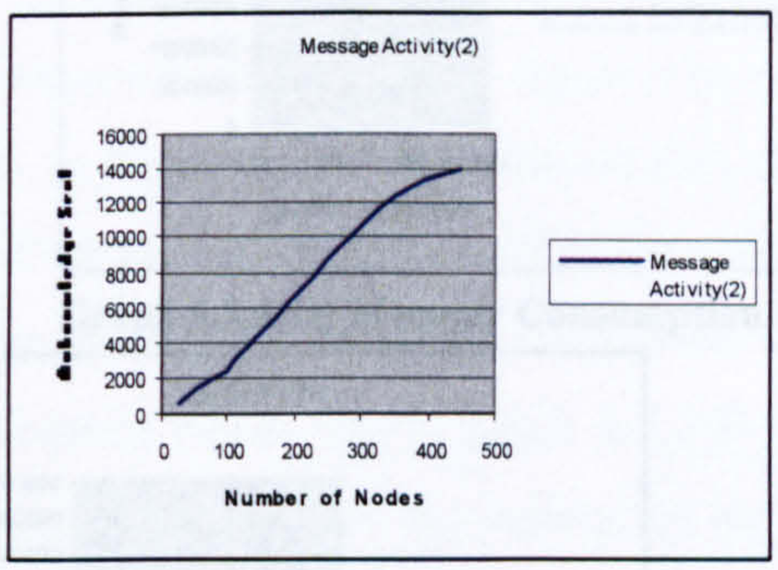
Number of Nodes	RQDD Sent	ACK received	Fields	Data delivery %cnt	Routes Formed	Elapsed Time	Total Memory	Memory used	Memory Saved
25	860	674	500, 500	78.37	86	6410	2482176	1135720	1346456
50	1710	1390	500, 500	81.28	171	30808	3330048	1339320	1990728
75	2846	1960	500, 500	68.86	284	85854	3911680	1547968	2363712
100	3490	2292	500, 500	65.67	349	172100	4706304	1908776	2797528
125	4339	3567	500, 500	82.20	434	385961	5578752	2323088	3255664
150	6387	4455	500, 500	69.75	638	674209	6496256	2741048	3755208
250	10553	8796	500, 500	83.35	1056	3271415	12296192	5291760	7004432
350	15046	12411	500, 500	82.48	1507	8798934	19607552	8847960	10759592
450	20098	13947	500, 500	69.39	2003	16300637	30212096	13057056	17155040
	7258.77	5499.11		**Average(75.70)**	Average 725.33				56.90 % save

Table 6.3.5 Results chart of experiments No 6.3.4

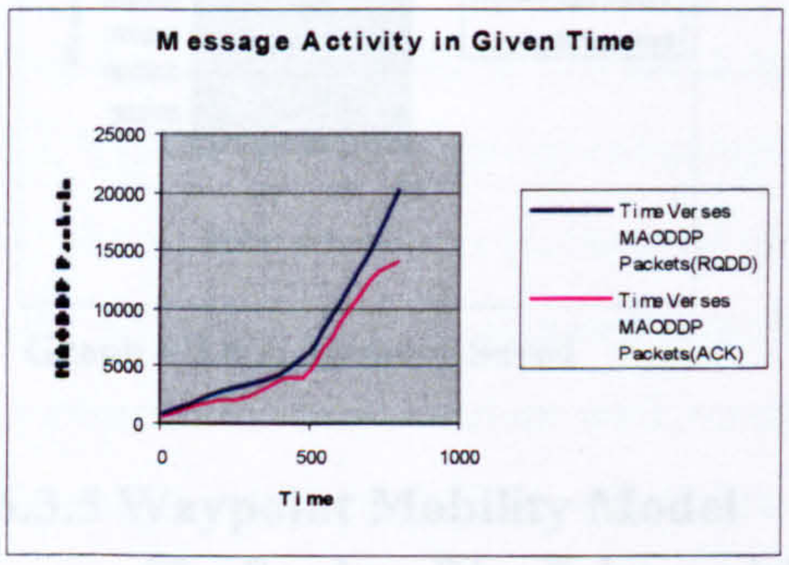
Figures results graphs of experiments no 6.3.4



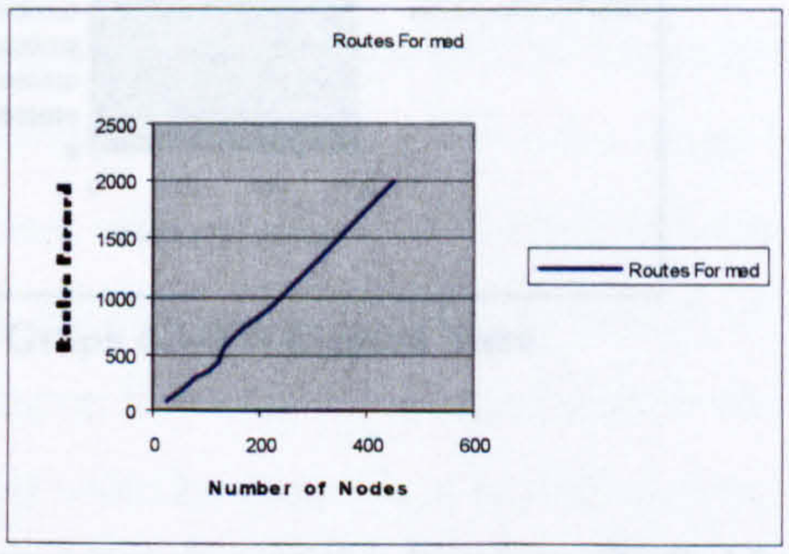
Graph 6.3.4(A) Message Activity (1)



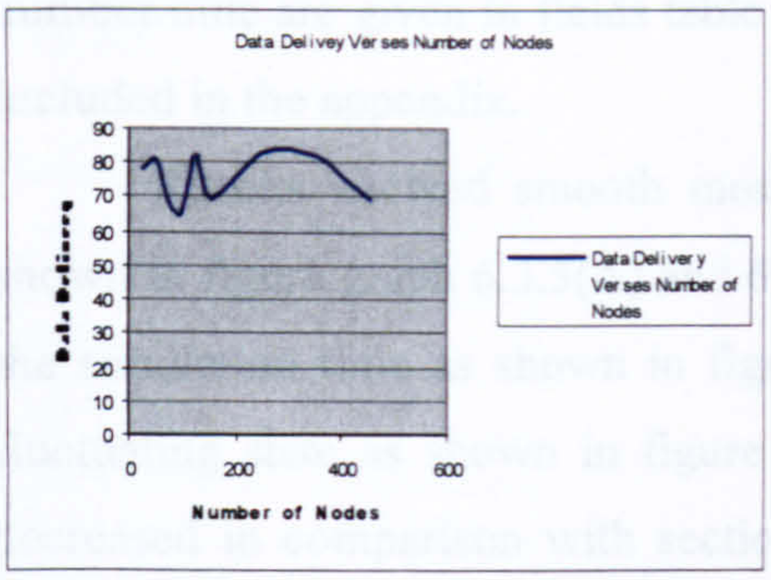
Graph 6.3.4(B) Message Activity (2)

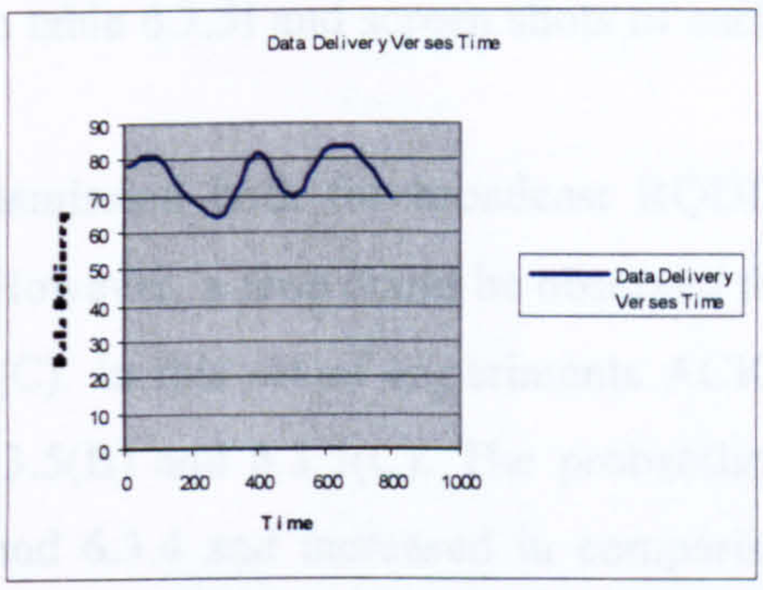


Graph 6.3.4© Message Activity in Given Time

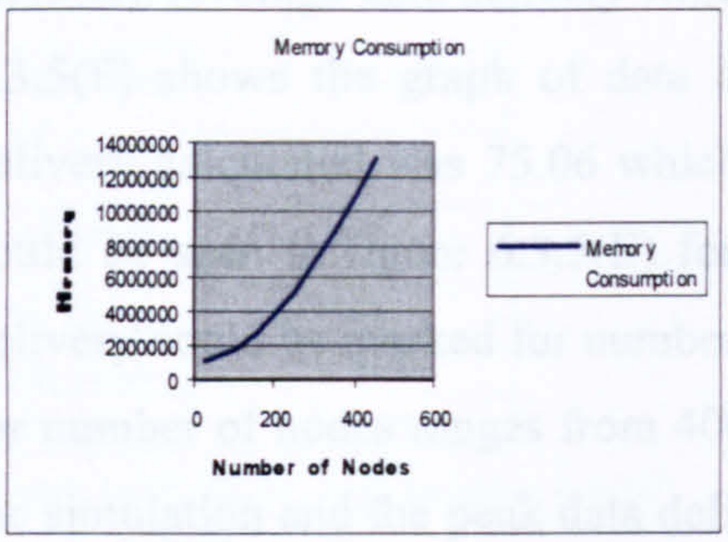


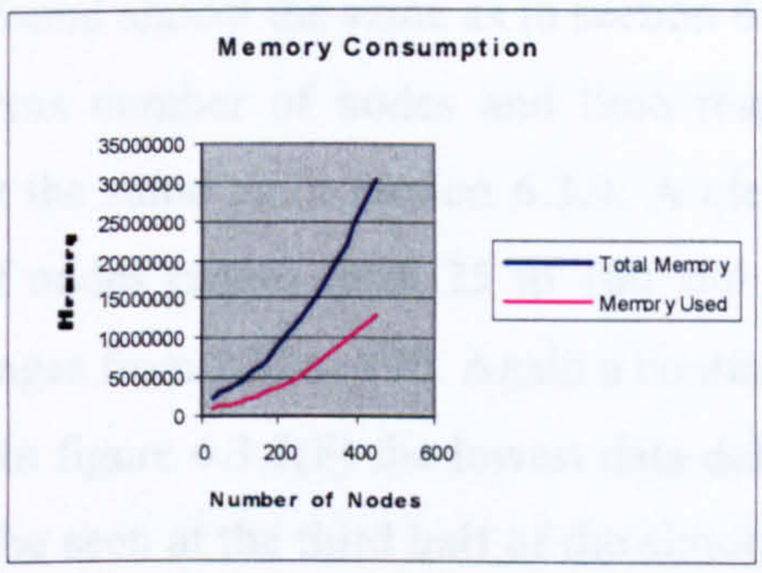
Graph 6.3.4(D) Routes Formed



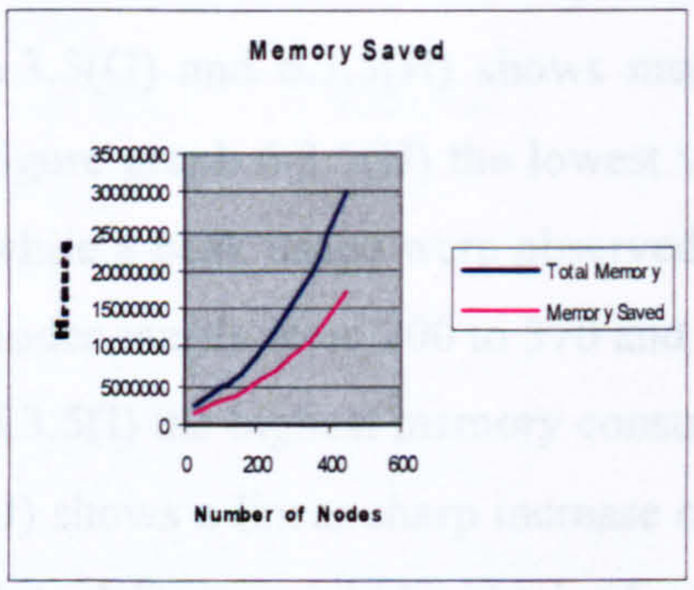


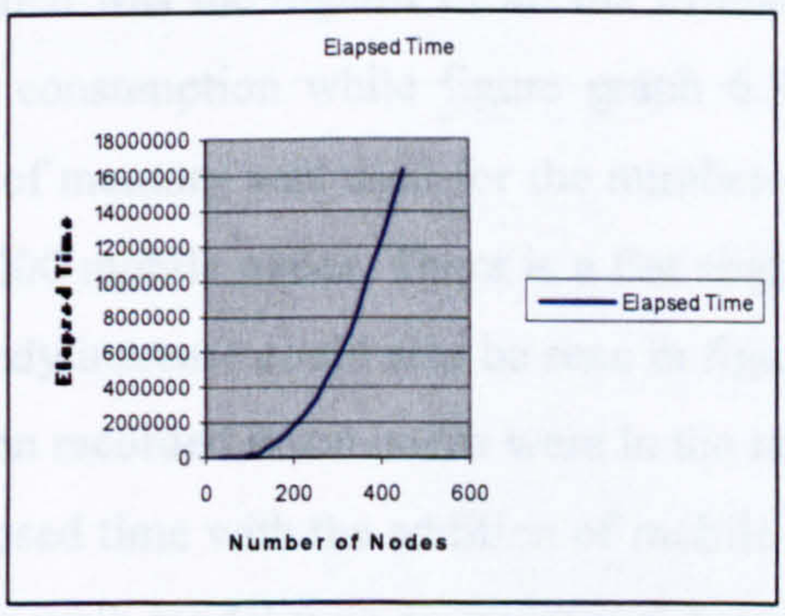
Graph 6.3.4(E) Data delivery Verses Number of Nodes **Graph 6.3.4(F) Data delivery Verses Time**





Graph 6.3.4(G) Memory Consumption (1) **Graph 6.3.4(H) Memory Consumption (2)**





Graph 6.3.4(I) Memory Saved **Graph 6.3.4(J) Elapsed Time**

6.3.5 Waypoint Mobility Model

The Random Way Point model [31] an extension of the Random Walk Model. In it each node at the beginning of its turn first moves to a new position selected at random in the unit square. Simulation started at 10 seconds and ended at 800 seconds with a fixed resolution time of 60 seconds. Similar to 6.3.4 the pause time value was 10 seconds. Mobile nodes were placed in a fixed grid area of 30x30 within a two dimensional fixed field size of 500x500 metres. Various values of mobile nodes in the range of 25 to 450 with a constant data rate of 1 minute were used. Details of all the table fields from experiment number one to experiment

number nine are given in fields table 6.3.5A to table 6.3.5I and screen shots of each of these experiments are included in the appendix.

Results showed smooth message transmission both for broadcast RQDD and ACK messages as shown in figure graph 6.3.5(A) and 6.3.5(B). However, a drop could be observed for RQDD in the middle of the simulation time as shown in figure 6.3.5(C). In this set of experiments ACK messages maintained its fluctuating state as shown in figure graph 6.3.5(B) and 6.3.5(C). The probability of route formation was decreased in comparison with section 6.3.3 and 6.3.4 and increased in comparison with section 6.31 and 6.3.2. A linear increasing graph of route formation against the number of nodes could be seen in figure graph 6.3.5(D). Average data delivery which is 75.04 found almost the same as in section 6.3.5. Figure 6.3.5(E) and 6.3.5(F) shows the graph of data delivery versus number of nodes and time respectively. Average data delivery calculated was 75.06 which are almost the same as in section 6.3.4. A clear drop in data delivery could be seen in figure 6.3.5(E) for number of nodes ranges from 25 to 180 and a steady increased data delivery could be marked for number of node ranges from 220 to 390. Again a continuous drop could be seen for number of nodes ranges from 400 upwards. In figure 6.3.5(F) the lowest data delivery is at the middle of the simulation and the peak data delivery could be seen at the third half of the simulation time. A surprising result is in terms of memory saved which was the highest of all the evaluation experiments. Figure graph 6.3.5(G) and 6.3.5(H) shows memory consumption while figure graph 6.3.5(I) shows memory saved. In figure graph 6.3.5(H) the lowest value of memory was used for the number of nodes ranges from 25 to 180 while a peak usage were observed for 200 mobile nodes. There is a flat slightly below usage of memory for nodes ranges from 200 to 370 and a steady increase could also be seen in figure graph 6.3.5(H). Figure graph 6.3.5(I) the highest memory consumption recorded when nodes were in the range of 180 to 195. Figure 6.3.5 (J) shows a linear sharp increase of elapsed time with the addition of mobile nodes. In conclusion the same data delivery could be obtained with the walk model with an additional benefit of less memory consumption in comparison with random walk model of section 6.3.4. The main drawback is the lesser probability of route formation which could also affect data delivery with the addition of mobile nodes.

Fields tables experiments no 6.3.5

Number of Nodes	25
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table6.3.5A exp no table 6.3.5A

Number of Nodes	50
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table6.3.5B exp no table 6.3.5B

Number of Nodes	75
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table6.3.5C exp no table 6.3.5C

Number of Nodes	100
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table6.3.5D exp no table 6.3.5D

Number of Nodes	125
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table6.3.5E exp no table 6.3.5E

Number of Nodes	150
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table6.3.5F exp no table 6.3.5F

Number of Nodes	250
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table6.3.5G exp no table 6.3.5G

Number of Nodes	350
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table6.3.5H exp no table 6.3.5H

Number of Nodes	450
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table6.3.5I exp no table 6.3.5I

Number of Nodes	**RQDD Sent**	**ACK received**	**Fields**	**Data delivery %cnt**	**Routes Formed**	**Elapsed Time**	**Total** Memory	Memory **used**	Memory **Saved**
25	819	648	500, 500	79.12	82	6536	2482176	1135584	1346592
50	1909	1580	500, 500	82.76	191	33791	3301376	1340704	1960672
75	2796	1928	500, 500	68.95	279	85774	3899392	1582712	2316680
100	3794	2387	500, 500	62.91	379	174244	4657152	1917160	2739992
125	4485	3526	500, 500	78.61	449	379861	5595136	2119304	3475832
150	5919	4031	500, 500	68.10	593	620659	6545408	4772488	1772920
250	8963	7388	500, 500	82.42	897	2852088	12357632	4616762	7740870
350	13541	11153	500, 500	82.36	1356	7945111	19779584	7983256	11796382
450	17204	12074	500, 500	70.181	1720	14073994	30793728	14450312	16343416
	6603.33	4968.33		**Average(Average(75.04)**	660.667				84.43 % saved

Table 6.3.6 Results chart of experiments no 6.3.5

Figures results graphs of experiments no 6.3.5

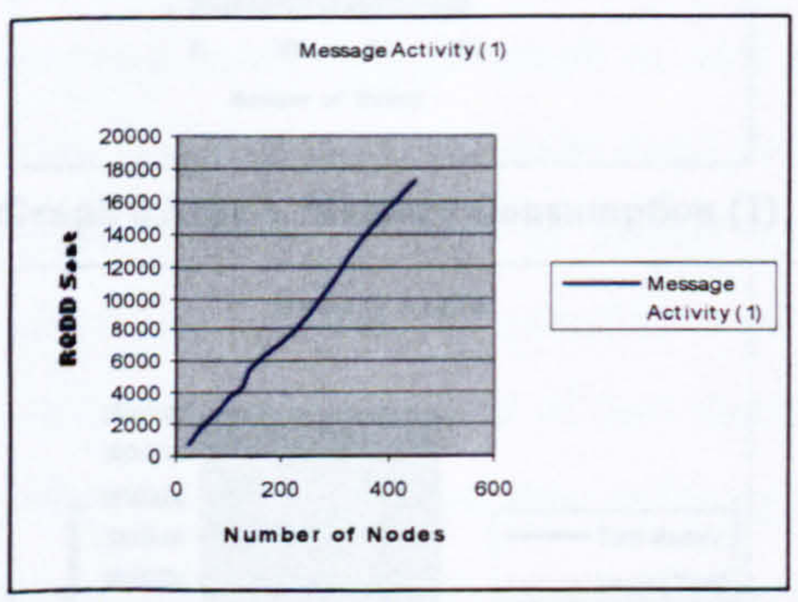


Figure Graph 6.3.5(A) Message Activity (1)

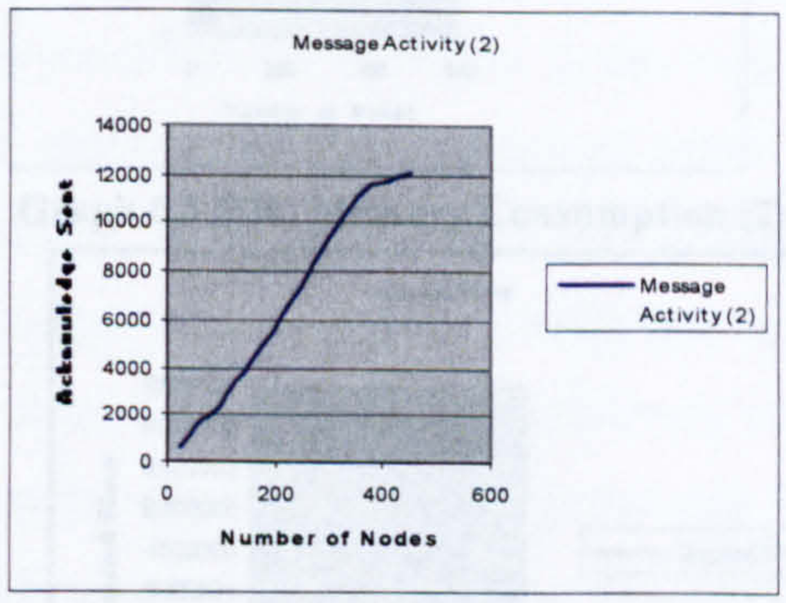
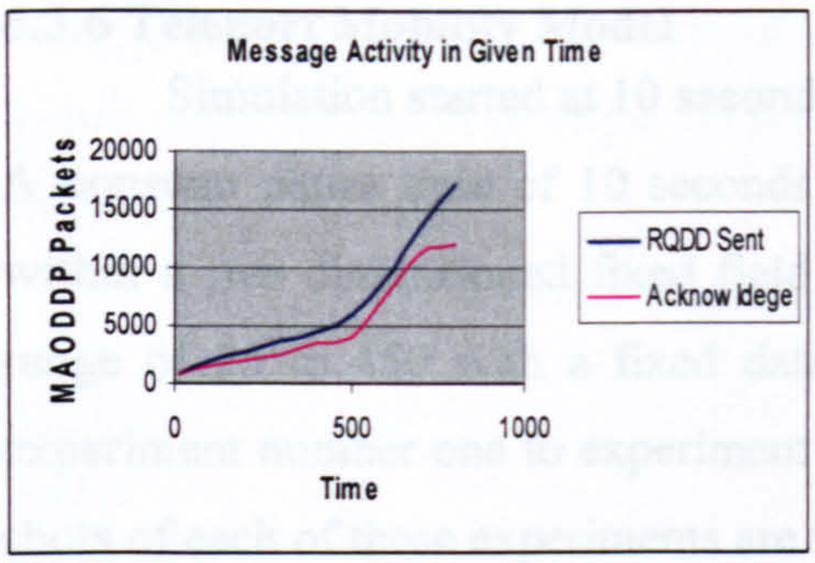
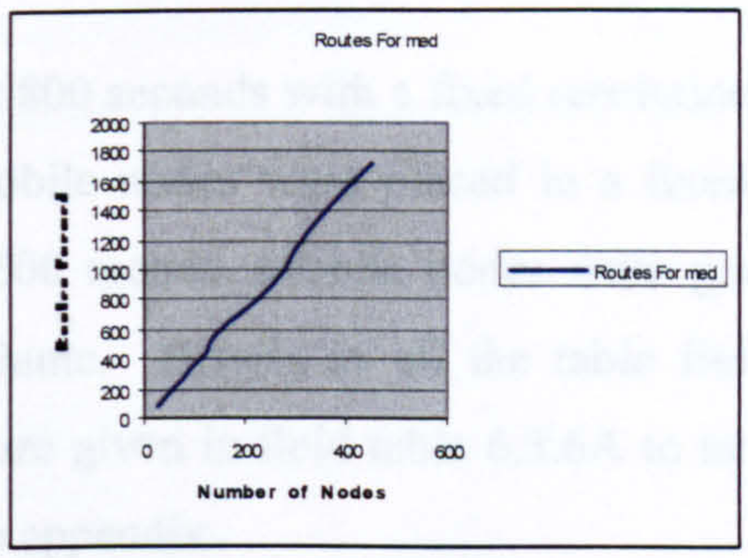


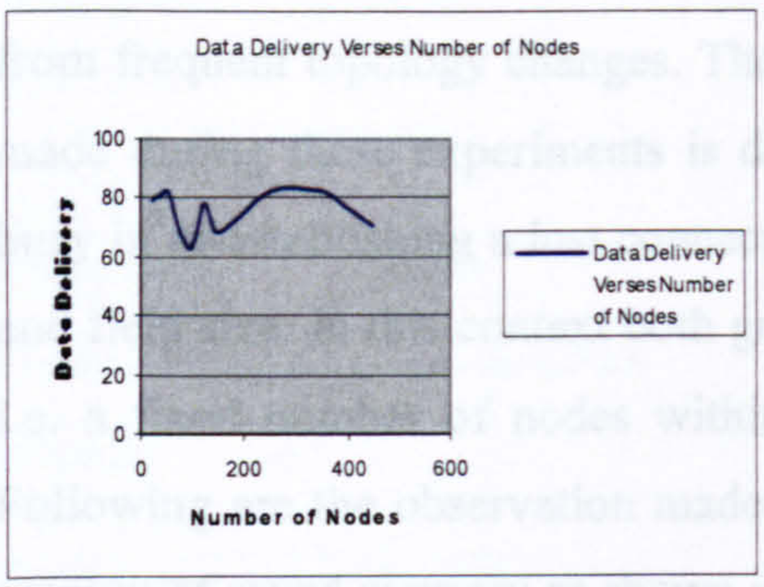
Figure Graph 6.3.5(B) Message Activity (2)



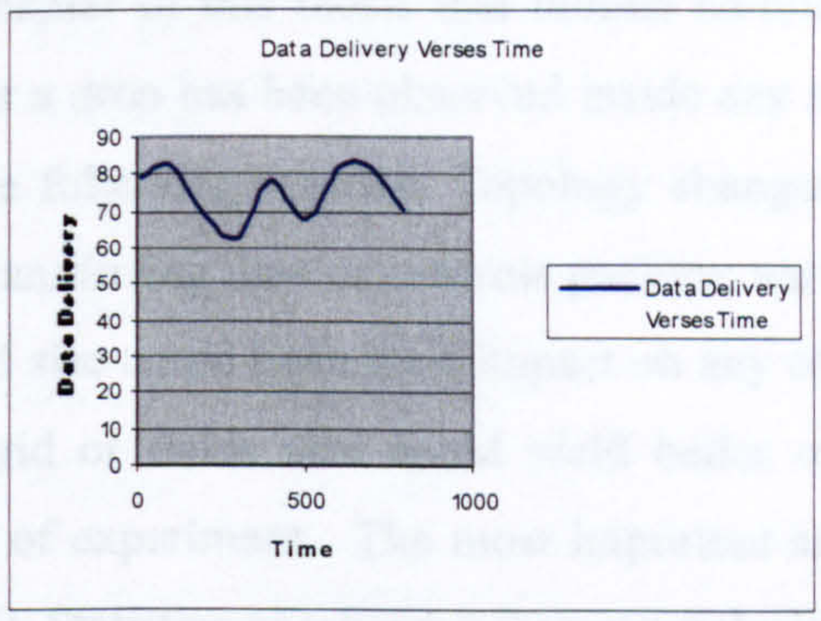
Graph 6.3.5© Message Activity in Given Time



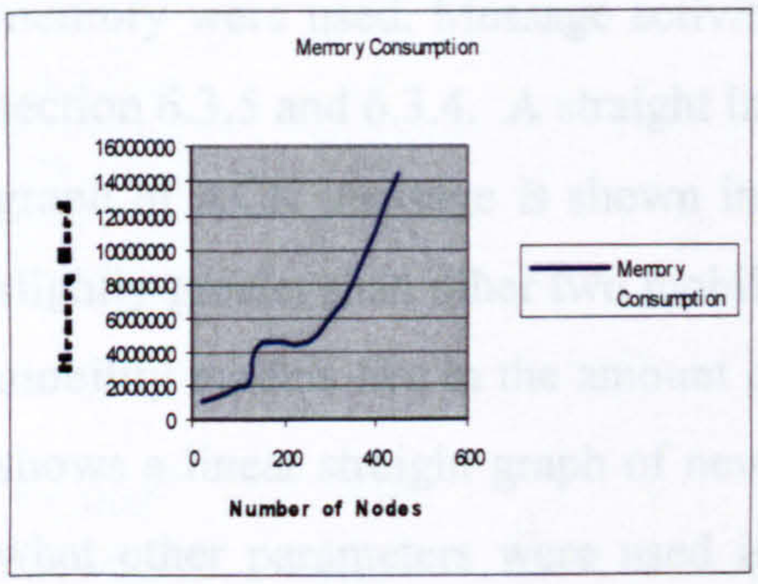
Graph 6.3.5(D) Routes Formed



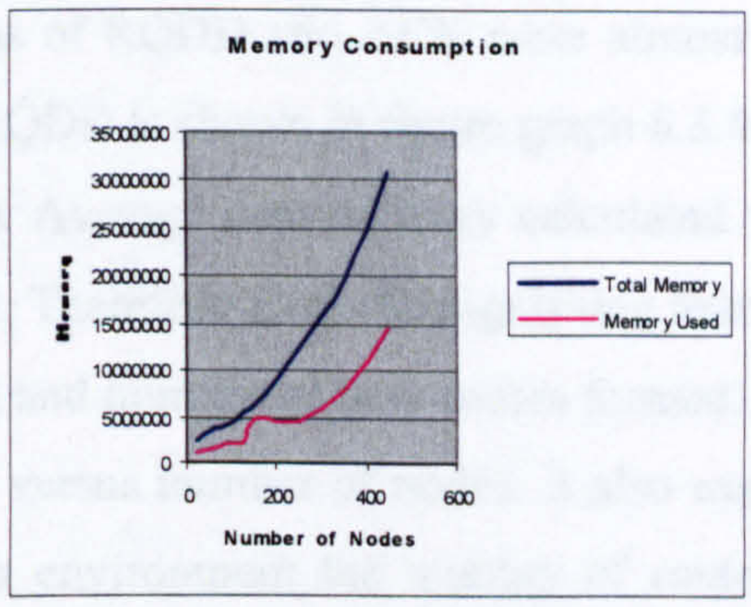
Graph 6.3.5(E) Data delivery Verses Number of Nodes



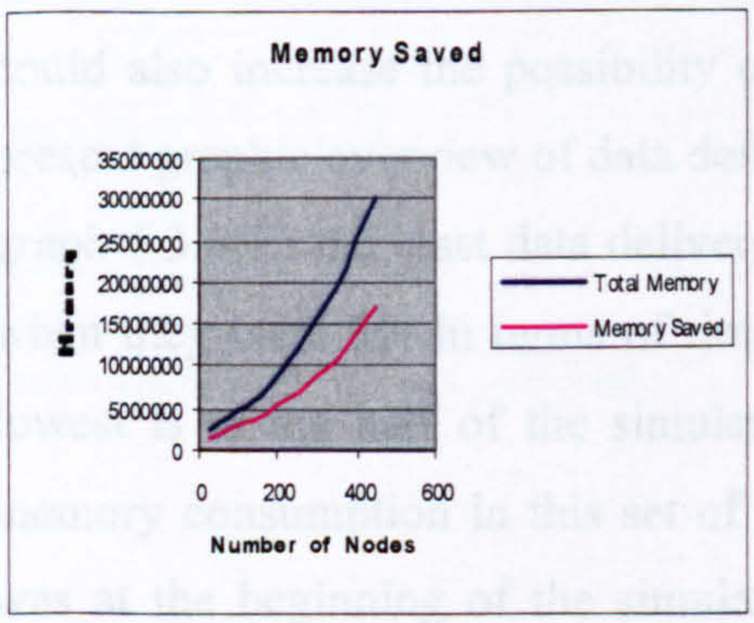
Graph 6.3.5(F) Data delivery Verses Time



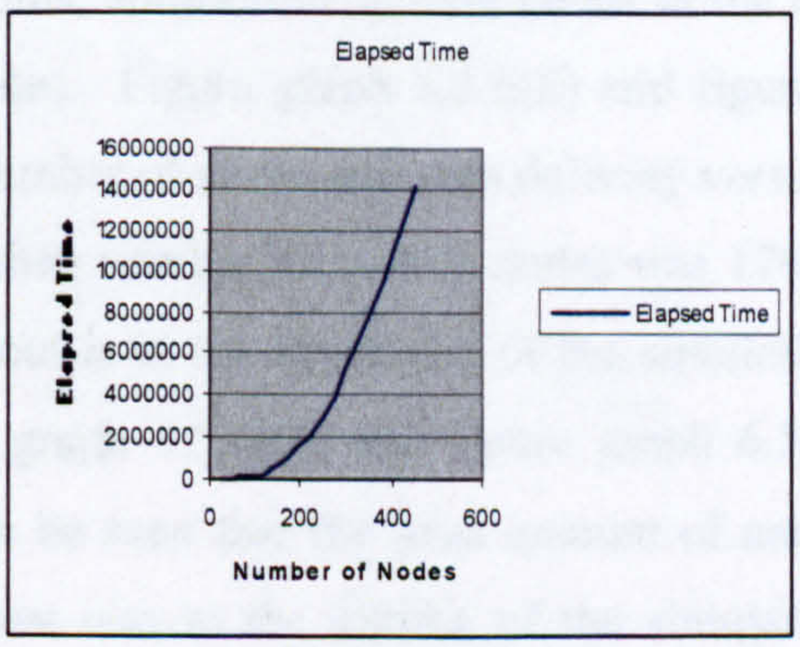
Graph 6.3.5(G), Memory Consumption (1)



Graph 6.3.5(H) Memory Consumption (2)



Graph 6.3.5(I) Memory Saved



Graph 6.3.5(J) Elapsed Time

6.3.6 Teleport Mobility Model

Simulation started at 10 seconds and ended at 800 seconds with a fixed resolution time of 60 seconds. A constant pause time of 10 seconds was used. Mobile nodes were placed in a fixed grid area of 30x30 within a two dimensioned fixed field size of 500x500 metres. Mobile nodes were gradually added in the range of 25 to 450 with a fixed data rate of 1 minute. Details of all the table fields and results from experiment number one to experiment number nine are given in field table 6.3.6A to table 6.3.6I and screen shots of each of these experiments are included in the appendix.

It has been mentioned in the background chapter of this thesis that mobile ad-hoc network suffers from frequent topology changes. Therefore wherever a drop has been observed inside any of the observation made during these experiments is due to one of the following reasons. Topology changes nodes might be busy in re-establishing a lost connection before re-transferring data or controls packets; variation in grid size and field size. In this context both grid size and field size could have their impact on any of the observations i.e. a fixed number of nodes within a specified grid or fields size could yield better or smooth results. Following are the observation made during this set of experiment. The most important achievement is the amount of saved memory as shown in figure 6.3.6 (J). Statistics of table 6.3.7 shows only 3% of the available memory were used. Message activities both in terms of RQDD and ACK were almost the same as were in section 6.3.5 and 6.3.4. A straight linear graph for RQDD is shown in figure graph 6.3.6(A) and a fluctuating graph of ACK message is shown in figure 6.3.6(B). Average data delivery calculated was 66.06 which are slightly greater than other two mobility models used. Therefore clear distinguishing features among the three mobility models lies in the amount of memory used and number of new routes formed. Figure graph 6.3.6D shows a linear straight graph of new routes formed versus number of nodes. It also explains that regardless what other parameters were used in the simulation environment the number of routes formed is directly related with the number of mobile nodes. In other words, addition of mobile nodes in the existing network could also increase the possibility of adding new routes. Figure graph 6.3.6(E) and figure graph 6.3.6(F) present graphic overview of data delivery versus the number of nodes and data delivery versus time. In figure graph 6.3.6(E) the least data delivery was observed when number of mobile nodes was 170 and the highest when they were 30. In terms of time highest throughout is in the beginning of the simulation time and the lowest is at the half of the simulation time. Figure graph 6.3.6(G) and figure graph 6.3.6(F) shows the memory consumption in this set of experiment. It can be seen that the least amount of memory consumed was at the beginning of the simulation and the highest was in the middle of the simulation. In terms of number of nodes the lowest were shown for nodes ranges from 25 to 30 and then there is steady increase with the addition of new nodes. However a drop can also be monitored for mobile nodes ranges from 400 to 450. Elapsed time maintained straight linear graph versus number of nodes as shown in figure graph 6.3.6(J). It is

found that increasing the number of nodes also increased elapsed time. In conclusion teleport model was found more effective in terms of both memory consumption and data delivery with a minor drop in the addition of new routes.

Fields tables experiments no 6.3.6

Number of Nodes	25
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.6A of exp no 6.3.6A

Number of Nodes	50
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.6A of exp no 6.3.6A

Number of Nodes	75
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.6C of exp no 6.3.6C

Number of Nodes	100
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.6D of exp no 6.3.6D

Number of Nodes	125
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.6A of exp no 6.3.6A

Number of Nodes	150
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.6A of exp no 6.3.6A

Number of Nodes	250
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.6A of exp no 6.3.6A

Number of Nodes	350
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.6A of exp no 6.3.6A

Number of Nodes	450
Field Size	500x500
Area grid	30x30
Start time	10
End Time	800
Data Rate	1 min
Pause time	10 secs

Fields table 6.3.6A of exp no 6.3.6A

Number of Nodes	**RQDD Sent**	**ACK received**	**Fields**	**Data delivery %cnt**	**Routes Formed**	**Elapsed Time**	**Total Memory**	**Memory used**	**Memory Saved**
25	820	683	500, 500	83.29	82	2600	70713344	2607784	68105560
50	2083	1771	500, 500	85.02	209	12895	163315712	4759160	158556552
75	2900	2010	500, 500	69.31	290	27155	382074880	9488160	372586720
100	4484	2929	500, 500	65.321	449	64775	178454528	5938440	172516088
125	5300	4362	500, 500	82.30	530	141798	697237504	16942960	680294544
150	6027	4280	500, 500	71.01	605	474023	1233518592	28375160	1205143432
250	9600	7851	500, 500	81.78	960	900131	2724986880	62749784	2662237096
350	13142	10855	500, 500	82.59	1315	2567210	2734227456	68718368	2665509088
450	18409	12777	500, 500	69.40	1842	5936947	2756378624	76300600	2680078024
	6973.889	5279.778		**Average(76.66)**	698				97.47 % Saved

Table 6.3.7 Results chart of experiments no 6.3.6

Figures results graphs of experiments no 6.3.6

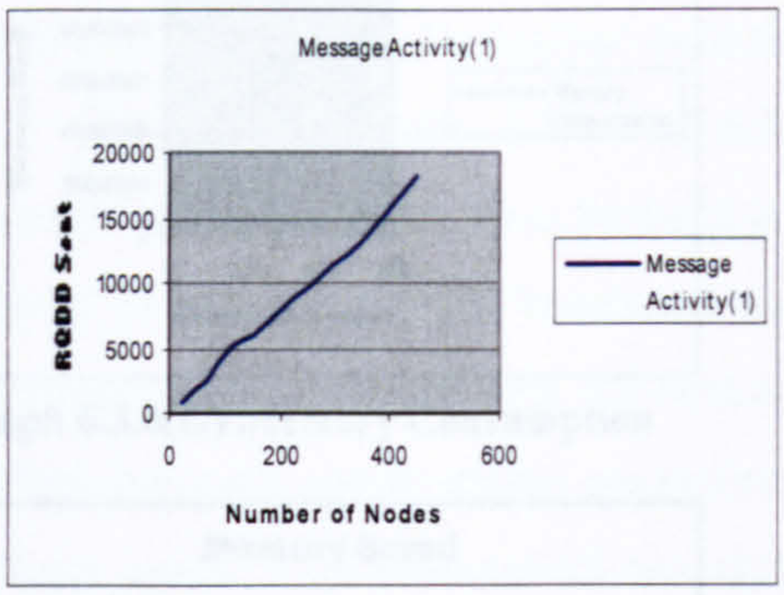


Figure Graph 6.3.6(A) Message Activity (1)

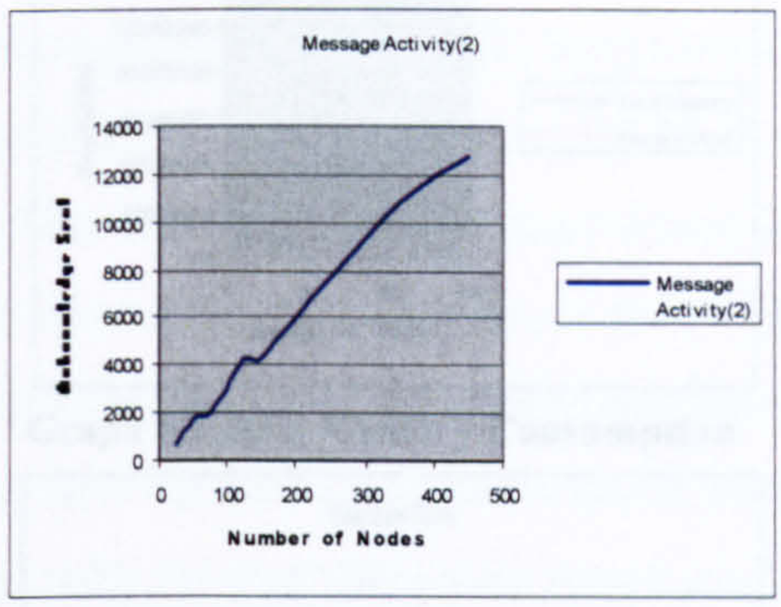
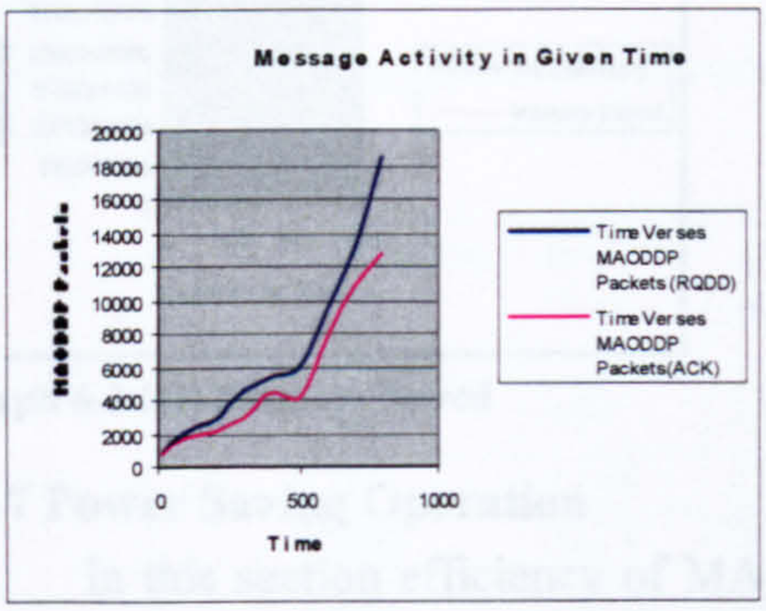
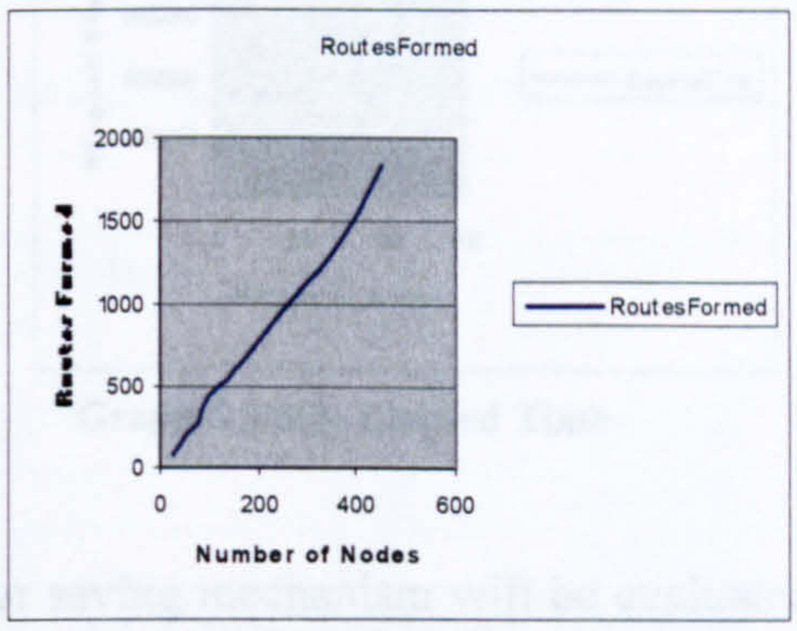


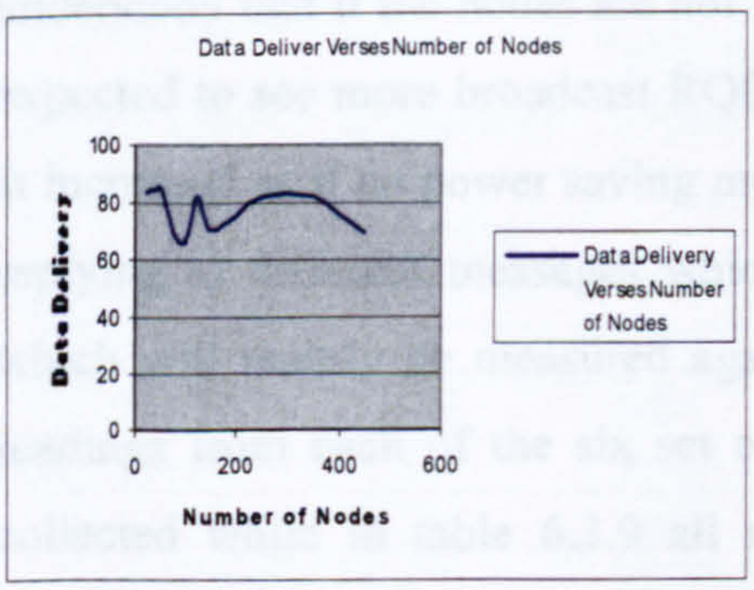
Figure Graph 6.3.6(B) Message Activity (2)



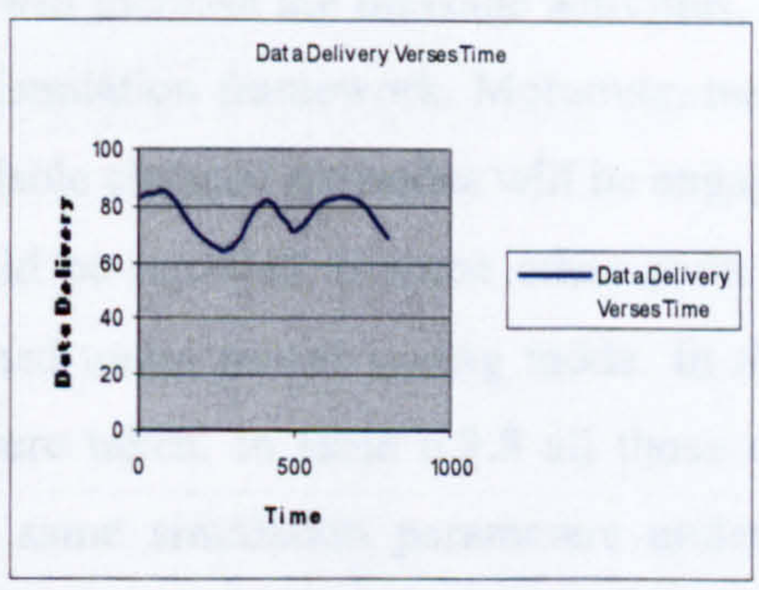
Graph 6.3.6© Message Activity in Given Time



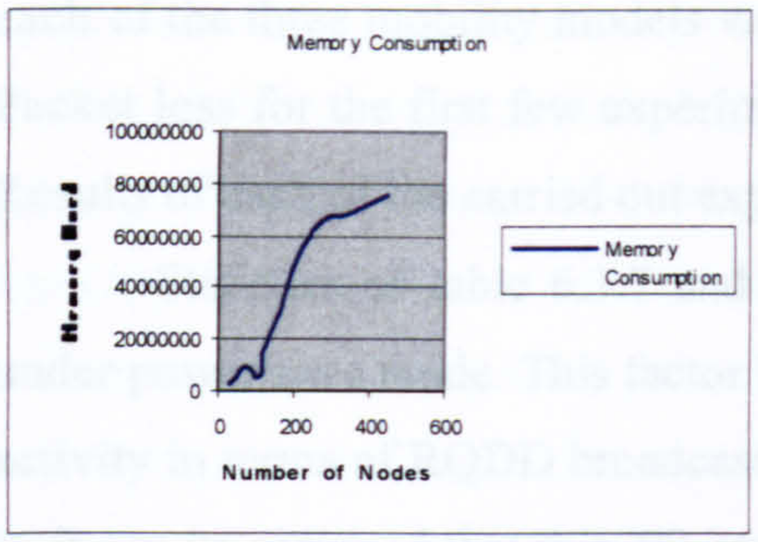
Graph 6.3.6(D) Routes Formed



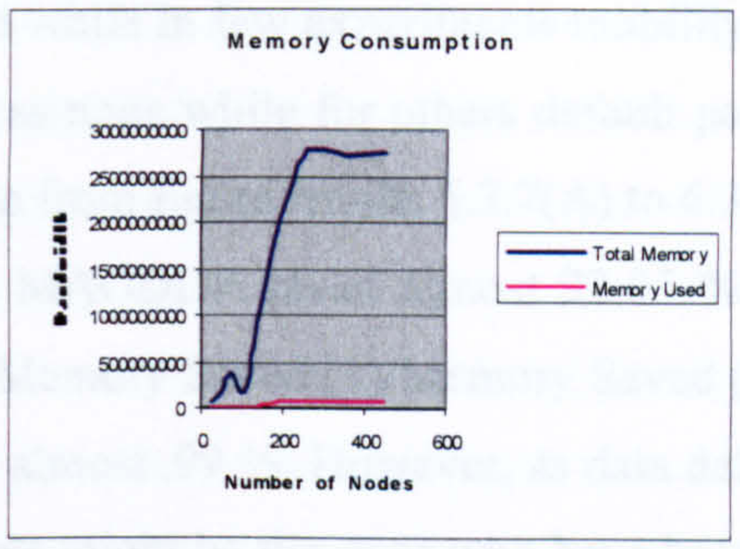
Graph 6.3.6(E) Data delivery Verses Number of Nodes



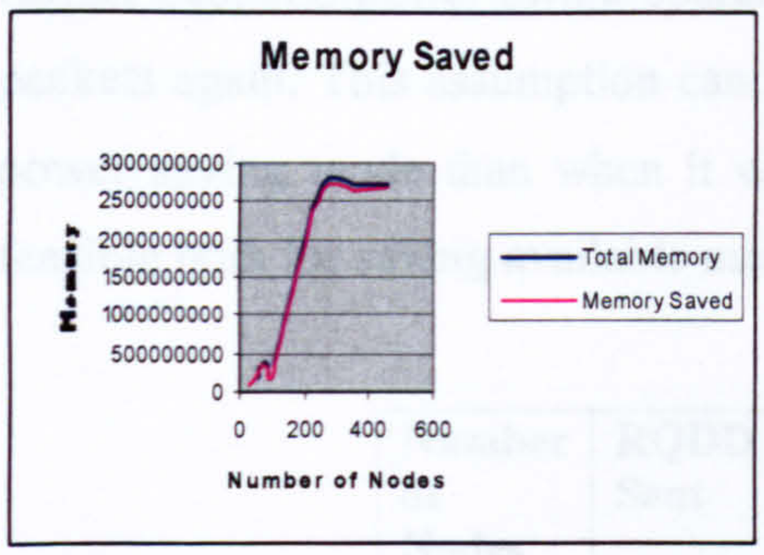
Graph 6.3.6(F) Data delivery Verses Time



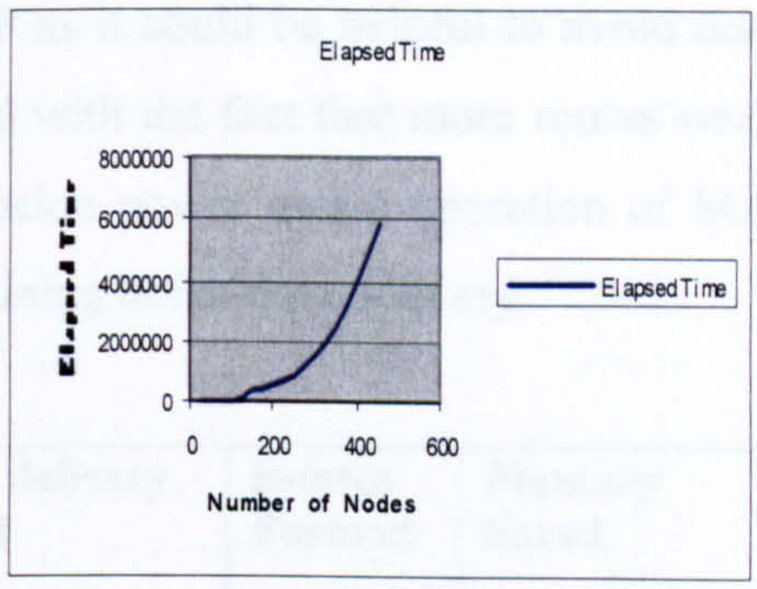
Graph 6.3.6(G) Memory Consumption



Graph 6.3.6(H) Memory Consumption



Graph 6.3.6(I) Memory Saved



Graph 6.3.6(J) Elapsed Time

6.3.7 Power Saving Operation

In this section efficiency of MAODDP power saving mechanism will be evaluated. In SWANS it is not possible to specifically carry out simulation which can highlight direct consumption of battery power of mobile nodes. However some of the understood concepts could be used to make such conclusions. It can be understood that if the nodes are not in sleep states it will increase the message activities. In other words it is expected to see more broadcast RQDD in the same simulation framework. Moreover, memory consumption is increased as if no power saving mechanism is available chances are nodes will be engaged continuously in replying to different messages which otherwise could be rerouted to some other node. These two factors which will mainly be measured against those obtained under power saving mode. In this regard different readings from each of the six set of experiments were taken. In table 6.3.8 all those readings have been collected while in table 6.3.9 all readings for the same simulation parameters under no power saving mechanism are recorded. To obtain such readings a manual procedure was followed. Codes that deal with MAODDP power aware operation were disabled and the whole simulator was recompiled before running simulation for the selected readings.

Six different experiments were carried out with simulation stop time ranges from 600 to 800 seconds with a fixed start and resolution time of 10 and 60 seconds respectively. In some of the experiments nodes were placed in a grid type area ranges from 15x15 to 30x30 while in one of the experiment nodes were placed at random. Two dimensional field size of 500x500 were used in most of the experiments. Tests using

each of the three mobility models were also conducted while in few experiments mobility was kept constant. Packet loss for the first few experiment was declared as none while for others default packet loss was used. Results of each of the carried out experiments are given from figure results 6.3.7(A) to 6.3.7(F).

Statistics of table 6.3.7 and 6.3.8 shows that MAODDP saved almost 22.05 % available memory under power save mode. This factor is calculated via (Memory Saved (1)/Memory Saved (2)) * 100. Message activity in terms of RQDD broadcast has increased by almost .99 %. However, as data delivery dropped by 5 % it can be assumed that this .99 cent broadcast packets might be the ones who have broadcast before. This further supports power aware operation of MAODDP as it could be helpful to avoid dealing with the same packets again. This assumption can also be supported with the fact that more routes were established under power saving mode than when it was off. In conclusion power aware operation of MAODDP was found feasible both for saving available memory and in obtaining better data delivery.

Number of Nodes	RQDD Sent	ACK received	Data delivery %cnt	Routes Formed	Memory Saved
25	460	357	77.60	46	4250336
50	1249	1249	100	144	1455512
100	2435	2374	97.49	349	1933632
250	10553	8796	83.35	1056	7004432
350	13541	11153	82.36	1356	11796382
450	18409	12777	69.40	1842	2680078024
	7774.5	6117.667	**Average(85.03)**	798.83	

Table 6.3.8 Results chart of experiments no 6.3.7(1)

Number of Nodes	RQDD Sent	ACK received	Data delivery %cnt	Routes Formed	Memory Saved
25	517	350	67.698	52	102543040
50	1256	1256	100	126	150975472
100	2496	2402	96.23	251	3973727152
250	10887	8219	75.49	1090	2669826200
350	14015	11024	78.65	1402	2682412184
450	17789	11265	63.32	1767	2690563352
	7826.66	5752.66	**Average(80.23)**	781.33	

Table 6.3.9 Results chart of experiments no 6.3.7(2)

6.4 Discussion

In the light of all the evaluation results it can easily be drawn that MAODDP is well suited in all types of environments. Almost all the operations as defined under MAODDP specification are practically applicable and can produce good results. A variety of different types of tests has been conducted with

MAODDP. These simulation environments were selected in a way which can best reflect the nature of mobile ad-hoc network communication patterns. Normally mobile stations in a mobile ad hoc network do not go beyond twenty nodes. However for the sake of explanation and to consider the extension possibility of MAODDP the number of mobile nodes were selected from the range of 25 to 450. Likewise mobile ad-hoc communication usually takes place in a small area of few hundreds metres. That's justifying the selection of two dimensional fixed field areas of 500x500 meters. For most of the experiments packet lost was chosen as default so that it could reflect the exact amount of packets that are either dropped or loss. A fixed data rate of 1 was chosen to test the protocol under different environments with constant data rate. MAODDP was evaluated both for short and long runs therefore simulation stop were chosen from the range of 600 to 800 seconds.

Exp No	RQDD Sent	ACK received	Data delivery %cnt	Routes Formed	Memory Saved
1	3009.55	2791.77	69.78	421	49.26
2	4113.77	3839.44	91.27	451.77	49.7
3	4532.77	3742.22	86.021	725.333	67.29
4	7258.77	5499.11	75.70	725.33	56.90
5	6603.33	4968.33	75.04	660.667	84.43
6	6973.889	5279.778	76.66	698	97.47
Average	5415.34	4353.44	79.07	613.68	67.50

Table 6.3.9 Results chart of Experiments no 6.3.9

Message activity in terms of broadcasting RQDD was quite impressive however it varies from one to the other simulation environment. Highest broadcast RQDD was recorded in the waypoint model. Results showed higher message activities with reduced data delivery for simulations which take place under a specific mobility model. It might be due to limited and specific communication pattern of mobile nodes. For the first three evaluations with static mobility message activities were low but with higher data delivery. Average collective data delivery of all the evaluation tests is 79.07. This further explains that out of 100 broadcasts RQDD almost 79 packets were successfully delivered to the destination. It was observed

Possibility of adding new routes were quite satisfied, in general scenario this possibility increases with the addition of mobile nodes. The calculated average number of new routes formation is 613. MAODDP performance in terms of saving memory was also impressive. Results showed that almost 67.50 % of available memory was saved. A separate set of experiments was conducted to measure effectiveness of MAODDP power aware mechanisms. Finding shows that almost 22 % of available memory could be saved under power saving mode along with a better data delivery.

6.5 Summary

In this chapter a detailed account of the MAODDP evaluation process was presented. A brief explanation of SWANS with reasons to justify its selection was also discussed. Seven different sets of experiments under various simulation environments were conducted. Each of these sets of experiments along with related field tables, results, graphical views of each result were presented in separate sections. A concluding section discussing results in general has also covered. The effectiveness of MAODDP power aware operation was also evaluated. MAODDP proved to be an effective protocol both in terms of higher message activities, good data delivery and saving battery power. In the next chapter MAODDP will be evaluated against some of the dominant protocols of mobile ad-hoc networks.

CHAPTER 7. MAODDP COMPARISONS WITH OTHER PROTOCOLS

7.1 Introduction

MAODDP was compared against some of the famous routing protocols of mobile ad-hoc networks. The main aim was to monitor MAODDP performance against some selected routing protocols of similar types. Some of the earlier presented studies of similar types are used in this chapter. In this regard, simulation environments were created in the nearest manner as it was in the cited literature [31] [32] [33] [85] [86]. To keep the study clear all the selected protocols evaluated against MAODDP are covered in separate sections. Section 2 of this chapter covers all the comparative studies. Concluding discussions are given in section 3 and a chapter summary is presented in section 4.

7.2 Protocols Studied

DSDV[23], AODV[25] and DSR[28] are some of the famous routing protocols of mobile ad-hoc networks. AODV is classified as a mixture of table driven and on-demand protocols. On the on-demand side, AODV has various operations that are closer to MAODDP than the other two mentioned protocols. Therefore AODV has been selected as the main protocol to evaluate against MAODDP. A fair theoretical comparison of DSDV against MAODDP has also been covered in this section. No implementation of DSR is found under SWANS, therefore the results from a previously conducted study is used to compare DSR against MAODDP.

7.2.1 MAODDP and AODV

MAODDP is a pure on-demand protocol and AODV is a combination of table driven and on-demand protocols. AODV requires that all nodes broadcast periodic updates; however, no such updates are needed in MAODDP. Instead to maintain fresh topology information MAODDP relies on one of four different types of data packets. In AODV the route needs to be established before the data transfer while MAODDP establishes route and deliver simultaneously one after the other.

MAODDP and AODV share several similar features. Examples include support to unicast and multicast routing, security and power saving mechanisms. Based on these protocol specifications various theoretical conclusions could be drawn. It is expected that MAODDP might consume less memory than AODV. Likewise, MAODDP and AODV allow mobile nodes to be in the sleep mode at random intervals of time. In the case of AODV this time interval could decrease due to the additional requirement of update packets. This further explains improved performance of the MAODDP power saving mechanism over

AODV. MAODDP could offer faster data transmission than AODV as nodes do not wait for the establishment of the route before data transfer. In the following sections performance of each of these protocols will be evaluated in a simulation environment. Some of the key factors in evaluation are message activities and memory saved by each of these schemes. Message activities in terms of Route Replies and Acknowledge (ACK) message are of particular importance. Route Replies in AODV are generated if a route could be formed for a broadcast route request (RREQ). MAODDP generated ACK as a sign of successful route establishment and data delivery to the destination.

7.2.1A Simulation Environment and Results

Evaluation experiments were conducted on SWANS under the SuSE Linux 10.1 operating environment. In total five different sets of experiments each comprising three different simulation tests were conducted. Simulation environments were generated by using different parameters. Details of each of these parameters and how they were used is as follows. A fixed set of mobile nodes of 100, 250 and 500 mobile nodes were used in all simulation experiments. Nodes were arranged in one of two patterns. In set-up 1 nodes were placed in a fixed grid area type of 30x30 and in set-up two, nodes were placed randomly in a fixed field range of 500x500. Start time, ending and resolution time was 10 seconds, 800 seconds and 60 seconds respectively. Resolution time defines the time where the simulation ends after nodes stop sending message.

A fixed data of 1 minute was used in all simulation tests. A fixed data rate of 1minute means an average rate of 1.0 data packet per minute. It further explains that some nodes can send 1 packet per minute; some can send 2 packets per minute and vice versa. All the possible parameters of mobility were utilized. For the first and second set of experiments mobility was taken as static and random respectively and for the last three sets each of the three mobility models were used. Packet lost has been left as the default as declaring packet lost none does not mean anything, it is due to the fact that the simulation will have all normal loss. The simulation results of each set of experiments are arranged in order with table 7.1.1MODDP and table 7.1.1AODV being the results of first set of experiments and table 7.1.5MAODDP and 7.1.5AODV representing the results of the last set of evaluation tests. Similarly the results screen shots have been arranged in the same order in the appendix.

Results tables MAODDP and AODV

RQDD	ACK	Routes Added	Memory Saved
2723	2620	273	505393656
10550	5160	525	27158652272
12898	11774	1293	107423704
8723	7108	752	9257156544

Table 7.2.1MODDP with static mobility

RREQ	RREP	Routes Added	Memory Saved
272	272	272	180334752
640	640	640	786307768
1383	1383	1383	2655948328
765	765	765	1207530283

Table 7.2.1AODV with static mobility

RQDD	ACK	Routes added	Memory Saved
2671	2591	268	396444312
6511	6410	653	2657999504
13727	12531	1376	159163904
7636	7177	765	1071202573

Table 7.2.2 MODDP with random placement

RREQ	RREP	Routes Added	Memory Saved
248	248	248	234177336
697	697	697	1159993456
25824	1418	1369	2660627888
8923	787.66	771.33	1351632893

Table 7.2.2 AODV with random placement

RQDD	ACK	Routes added	Memory Saved
3966	2562	396	311962824
10396	8581	1041	1984611064
20493	13452	2047	2686200232
11618	8198	1161	1660924707

Table 7.2.3 MODDP Random Walk

RREQ	RREP	Routes Added	Memory Saved
5562	441	391	164819712
8638	941	910	419091744
17302	1846	1814	2668418384
10500	1076	1038	1084109947

Table 7.2.3 AODV Random Walk

RQDD	ACK	Routes added	Memory Saved
3730	2482	373	161372632
9639	7875	965	2434922496
20799	13632	2078	2686030904
11389	7996	1138	1760775344

Table 7.2.4 MODDP Random Way Point

RQDD	ACK	Routes added	Memory Saved
4000	2569	398	200635560
10399	8586	1041	2663344816
19622	12691	1962	2680800160
11340	7948	1133	1848260179

Table 7.2.5 MODDP Teleport Model

RREQ	RREP	Routes Added	Memory Saved
6085	369	311	176839642
23829	990	900	574434560
17946	1994	1963	2672532360
15953	1117	1058	1141268848

Table 7.2.4 AODV Random Waypoint

RREQ	RREP	Routes Added	Memory Saved
7695	416	341	197586664
22150	958	958	514543040
18903	1955	1921	2671412171
16249	1109	1414	1127847292

Table 7.2.5 AODV Teleport Model

In the first set of experiments the number of nodes was increased gradually with fixed mobility. MAODDP outclassed AODV in terms of message activities as shown in table 7.1A and table 7.1B. MAODDP broadcast 11 times more RQDD than RREQ of AODV and 9 times mores ACK were issued than RREP of AODV. A slightly better result of new route formation was observed in AODV where AODV added 1.01 times more routes than MAODDP. One important observation was about conserved memory. Statistics of table 7.1.1A and 7.1.1B show that MAODDP saved almost 7 times more memory than AODV. Nodes were placed at random in the second set of experiments. AODV shows an improved message activity with 1.16 more broadcast RREQ as shown in table 7.1.2B. However, it showed weak performance in terms of generating route replies. MAODDP broadcast almost 9 times more ACK to the source than RREP of AODV as shown in table 7.1.2A. No changes were observed in route formation capability of AODV and MAODDP. AODV showed a significant improvement in conserving memory. Results showed that AODV saved 1.26 times more memory than MAODDP.

Each of these protocols was monitored under three available mobility models available under SWANS. In the third set of experiments the random walk mobility model was used to generate movements for mobile hosts. MAODDP produced better results than AODV in all the output parameters as shown in table 7.1.3A and 7.1.3B. MAODDP broadcast 1.16 more RQDD with 7.61 times more ACK in comparison

with RREQ and RREP of AODV respectively. Likewise 1.18 times more routes were added by MAODDP and 1.53 times more memory were conserved in comparison with AODV. The random way point is an extension of the random walk model and was used in the forth set of experiments. AODV performed better with 1.40 more RREQ's broadcasted than MAODDP as shown in table 7.1.4B. However, MAODDP outclassed AODV with 7 times more ACK issued than RREP's of AODV as shown in table 7.1.4A. MAODDP also added 1.07 more routes than AODV and saved 1.54 more available memory than AODV. Under the teleport model, AODV produced better results both for broadcasting RREQ and the number of routes that were added. AODV broadcast 1.40 times more RREQ's than RQDD of MAODDP and added 1.24 times more routes than MAODDP. MAODDP outclassed AODV in terms of generating ACK messages and in conserving available memory. Results of table 7.1.5(A) and 7.1.5(B) showed that MAODDP issued 5 times more ACK than RREP's and saved 1.63 times more memory than AODV.

Experiment No	RQDD	ACK	Route Added	Memory Saved
Experiment No1	8723	7108	752	9257156544
Experiment No2	7636	7177	765	1071202573
Experiment No3	11618	8198	1161	1660924707
Experiment No4	11389	7996	1138	1760775344
Experiment No5	11340	7948	1133	1848260179
Average	10141	7685	989	3119663869

Table 7.2.6A Results charts MAODDP

Experiment No	RREQ	RREP	Route Added	Memory Saved
Experiment No1	765	765	765	1207530283
Experiment No2	8923	787	771	1351632893
Experiment No3	10500	1076	1038	1084109947
Experiment No4	15953	1117	1058	1141268848
Experiment No5	16249	1109	1414	1127847292
Average	10478	970	1009	1182477853

Table 7.2.6B Results charts AODV

Tables 7.2.6A and 7.2.6B present average values of each of the output parameters of individual experiments. AODV showed a slightly better performance in terms of broadcast RREQ's shown in figure graph (1). AODV broadcast 1.03 times more RREQ's than RQDD of MAODDP. However, MAODDP outclassed AODV in terms of generating ACK messages. These ACK messages in MAODDP serve two purposes i.e. establishment of route and simultaneous data delivery. Producing more ACK messages implies that more packets are delivered to the destination via suitable routes. Moreover, more ACK message may also mean more active routes. Statistics of table's 7.1.6A and 7.1.6B and figure graph (2) showed that

MAODDP is capable of generating 7.92 times more ACK than AODV RREP. In other words MAODDP offers 7.92 times more effective data communication than AODV. AODV showed better performance by establishing 1.29 times more routes than MAODDP as shown in figure graph (3). AODV uses more controls packets than MAODDP, thus conserves more memory. Statistics of tables 7.1.6(A) and 7.6.1(B) showed MAODDP saved an accumulated average of 2.63 time more memory than AODV. Based on the input parameter SWANS allocates memory for each individual simulation cycle. It is an important factor in measuring the usage of available resources by an individual protocol. Memory saved is measured in Hertz (HZ) as explained within the simulation software supporting document. Figure graph (4) presents graphical comparison of two protocols in terms of memory conservation. MAODDP conserved more memory for mobile nodes ranges from 100 to 300. A drop in conservation could be observed for mobile nodes above 400 where MAODDP could still be seen above AODV.

Figure results MAODDP and AODV

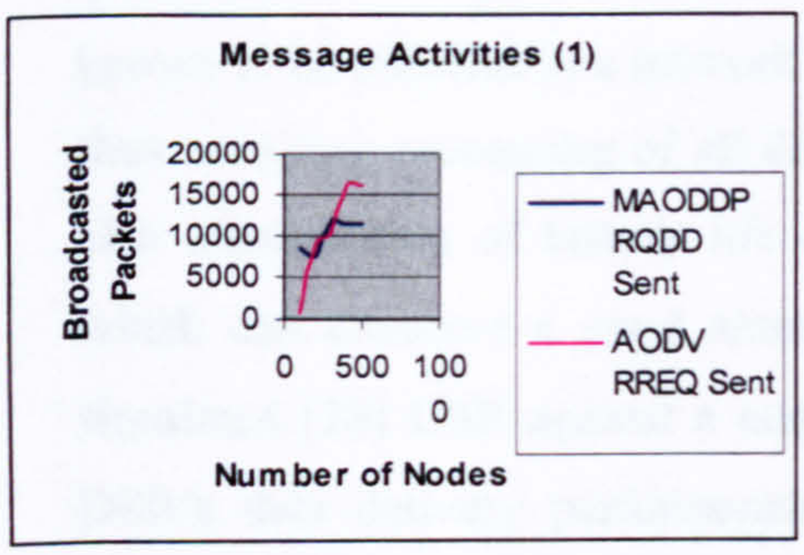


Fig graph (1) Message Activities (1)

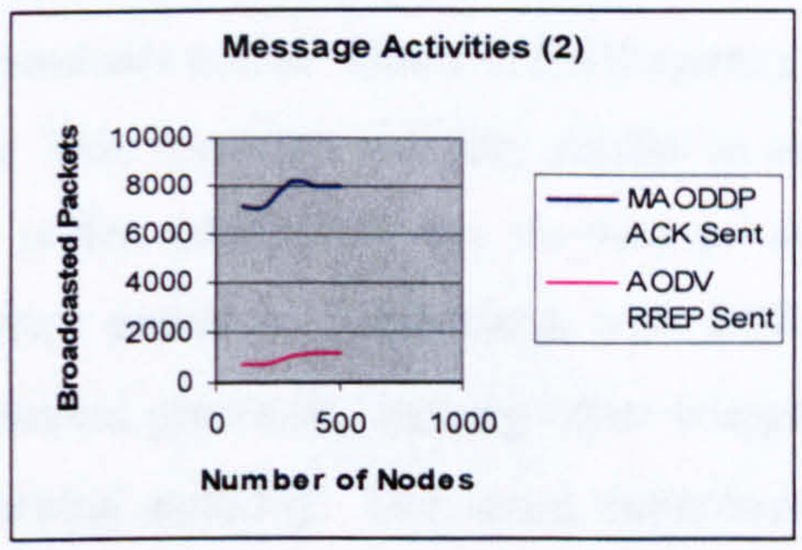


Figure graph (2) Message Activities (2)

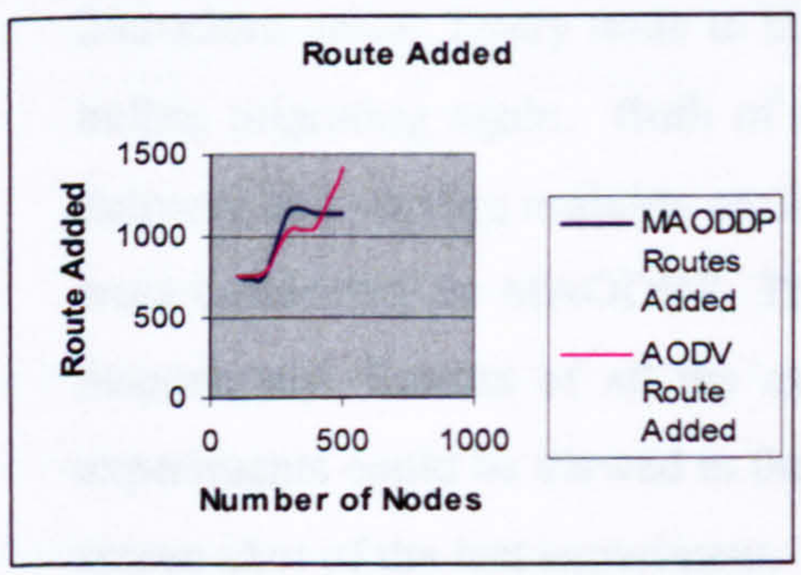


Figure graph (3) Routes Added

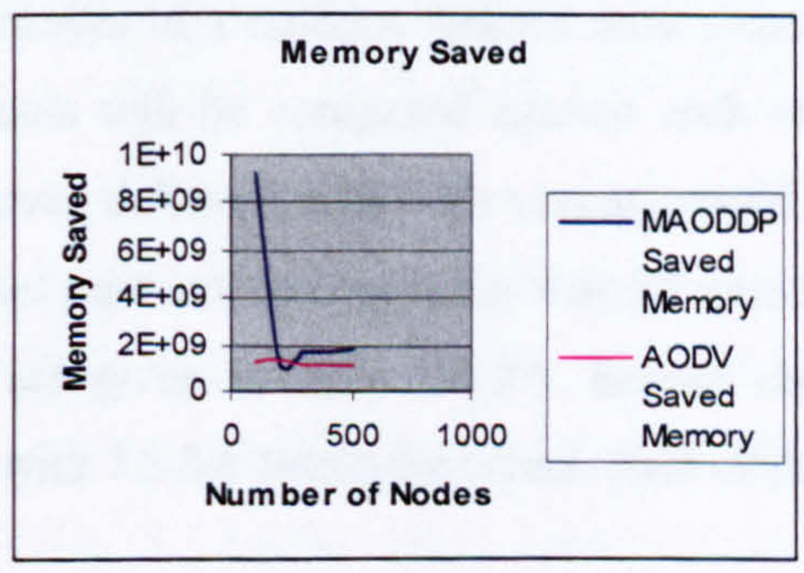


Figure graph (4) Memory Saved

7.2.2 MAODDP and DSDV

DSDV [23] is a table driven protocol which maintains a consistent network view via periodic updates. These periodic updates create an extra network overhead. Unlike DSDV, MAODDP does not require any sort of updates. DSDV is known to perform well in a small network whereas MAODDP can scale to larger networks. Finding optimal values for various reasons is not easy in DSDV .i.e. settling time can cause

network overhead. MAODDP does not involve any such calculation. MAODDP unlike DSDV has its own security mechanism and can support multicasting. DSDV due to its updates requires nodes to be awake most of the time contrary to MAODDP which has its own power saving mechanism. It is at present not possible to compare MAODDP against DSDV in a simulation environment as no implementation of DSDV is found under SWANS.

7.3.3 MAODDP and DSR

Both MAODDP and DSR [28] are on-demand protocols with different operational structures. Unlike MAODDP, DSR is based on a request reply cycle similar to AODV discussed above. It has been mentioned in chapter 3 of this thesis; such an approach could add some extra network overhead. This overhead could possibly be reduced to some extent through the approach adopted in MAODDP. Unlike DSR, MAODDP can support both unicast and multicast routing. Moreover it has its own security mechanisms whereas none is available in DSR. Scalability is not a problem in MAODDP whereas DSR is known to be efficient in a network of a few hundreds nodes. Hosts in DSR operate in promiscuous mode thus requiring processing of all data packets. This approach not only results in additional overhead but also consumption of battery life of mobile nodes. MAODDP has its own power saving mechanisms which can conserve a good amount of battery power in comparison with DSR. Sung-Ju and others simulated [38] DSR against a number of selected protocols. Among other comparisons they measured DSR's data delivery performance under varying mobility. The same experiments were repeated for MAODDP on SWANS. The simulation models the network of 30 mobile nodes migrating within a 20mx20m space. Every node in the network moves in a random fashion with a static time of 5 seconds before migrating again. Both of these protocols will be compared against each other in terms of data delivery in a varying mobility environment. Seven different tests with varying speed of 5 seconds interval were conducted on MAODDP. The simulation start and ending time was 10 seconds and 100 seconds respectively. Results of all the experiments are given in table 7.3.3A. Screen shots of each of these experiments could be viewed in the appendix with 7.3.3A being the screen shot of the first and 7.3.3F the screen shot of the last experiment.

Results tables MAODDP and DSR

Experiment No	RQDD Sent	ACK Sent	Data delivery %
1	17	17	100
2	39	31	79.48
3	48	48	100
4	70	57	81.42
5	81	79	97.53
6	100	100	100

Table 7.3.3A Results chart of Experiment No 7.3.3

Experiment No	Data delivery (approximate)
1	95
2	55
3	45
4	40
5	40
6	40

Table 7.3.3B DSR

Figure results MAODDP and DSR

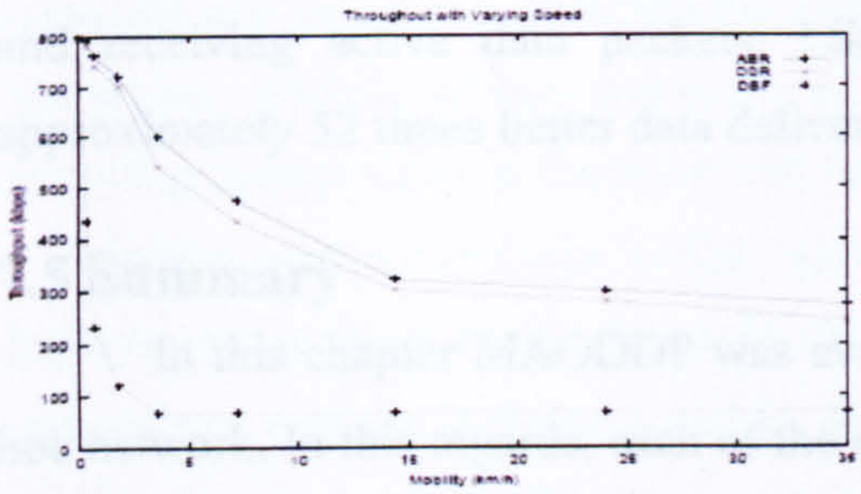


Figure 2: Data Throughput for Different Mobility Speed.

Figure 7.3.3A DSR Data delivery

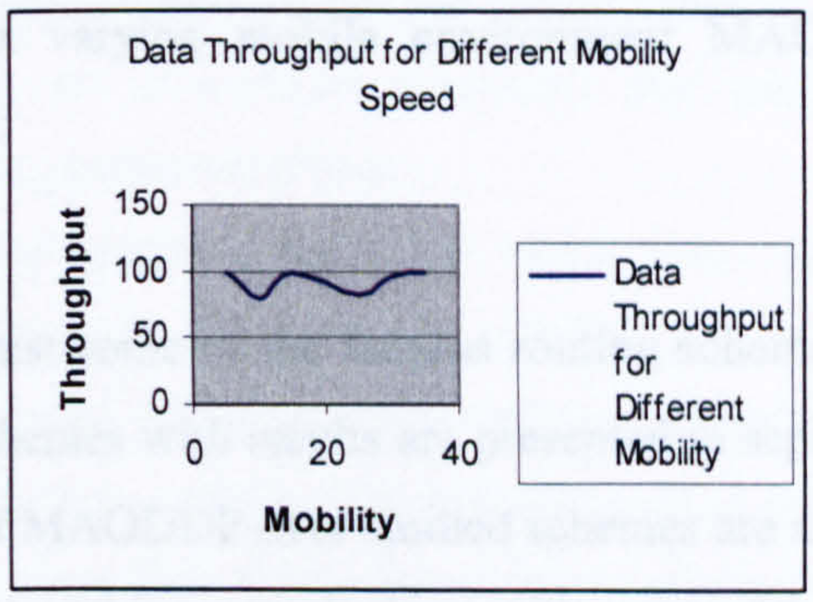


Figure 7.3.3B MAODDP Data delivery

MAODDP throughout the experiments maintains a consistent performance with a minor drop at mobility speeds of 10 and 15 seconds. This drop reflects the nature of mobile ad-hoc network where topology changes happen quite frequently. Therefore it can be explain that this drop might be due to the extra time nodes needed to re-establish the broken connection to deliver data packets. This further proved that MAODDP can perform well in a varied mobile environment. A steady drop can be observed in DSR performance with the increase of mobility speed as shown in the figure obtained from the study[45]. Estimated readings of DSR data delivery obtained from the graph figure 7.3.3A shows that MAODDP performance was 55.87 times better than DSR.

7.4 Discussion

MAODDP offers some additional benefit over schemes studied inside this chapter. In comparison with DSDV, it does not require periodic updates neither it involves complex route calculation. The same is true for AODV, where MAODDP could be used to reduce network overhead which may otherwise happen with both periodic updates and route reply cycle mechanisms of AODV. Moreover, MAODDP has clear advantage over AODV saving mechanisms due to the absent of update control packets. DSR efficiency is limited to the size of the network. The structure of DSR is not very suitable either in a varying mobility environment. Unlike DSR, MAODDP is scalable to all types of network and can perform well in a high mobile environment. Promiscuous operating mode of DSR is another factor which can reduce protocol performance specially in terms of conserving battery power. Therefore, the MAODDP power saving mechanism provides another significant advantage over DSR. Most of these benefits have been compared against each of these schemes in a variety of different patterns. MAODDP outclassed AODV in terms of memory conservation at accumulated average of 2.5 times and found 7 times more efficient in broadcasting and receiving active data packets. Likewise, in a varying mobile environment MAODDP can give approximately 52 times better data delivery than DSR.

7.5 Summary

In this chapter MAODDP was evaluated against some of the famous routing schemes of mobile ad-hoc network. In this regards, each of the evaluated schemes with results are presented in separate sections. A concluding discussion highlighting various benefits of MAODDP over studied schemes are also covered. The next chapter summarizes conclusions and achievement of this research along with some of the possible future work.

CHAPTER 8. CONCLUSIONS AND FUTURE WORK

8.1 Introduction

Wireless cellular networks have been in use since the early eighties. In future wireless systems, there will be a need for the rapid deployment of mobile users. It's relatively hard to have such instant network formation within the existing frame of a single hop of fixed wireless system. This fact has motivated the wireless community to review the old theme of the "on-the-fly" for mobile ad-hoc networks. The idea of mobile ad-hoc networking goes back to the DARPA (Defence Advanced Research Projects Agency) packet radio network, which was in used in the 1970's. A mobile ad-hoc network is a collection of mobile devices establishing a short lived or temporary network in the absence of a supporting structure. Mobile ad-hoc networks can be used in establishing efficient, dynamic communication for rescue, emergency, and military operations. Mobile ad-hoc networks offer a unique art of network formation and could be helpful in providing connectivity in the situations where it is not possible otherwise.

Mobile ad-hoc networks suffer with a number of different problems which limit their capability of deployment at large scale. Some of the main problems in this area are memory and power constraints; security and routing. Routing could be defined as a process of information exchange in between two hosts in a network. Absence of a fixed infrastructure makes routing a challenging and demanding issue in mobile ad-hoc networks. Taking routing as one of the main issue this thesis investigated routing mechanisms for mobile ad-hoc networks. Our research findings concluded that most of the existing schemes lack with various issues. Therefore this thesis concluded that there is a definite need for developing some new routing algorithm. From the beginning till the end of our research investigation a set pattern was followed. Starting from the background research and analyzing existing schemes this thesis ended at the successful delivery of a novel and effective routing protocol for mobile ad-hoc networks.

8.2 Contributions

Our research contributed some new ideas in this field. Not limited to the primary focus of our research, we have forwarded some interesting ideas to the research community. Details of the main contributions of this thesis are as follows.

- **Literature Surveys:** A critical evaluation of thirty routing protocols was conducted as a part of our research investigation. In most of the existing literature selected protocols were evaluated. Therefore evaluation of some well known protocols is one of the achievements of our research project. We have also attempted to classify mobile ad-hoc networks which were not done before. On the basis of our background research findings, we have classified mobile ad-hoc networks into one of three generations. These findings have been published in Computing Unplugged Magazine [181]. This thesis also identified some of the routing related issues. This concept was not so clearly defined in the reported research. Our investigation concluded that some of those issues are interrelated with routing mechanisms in the context of mobile ad-hoc networks. Our research[182] suggests that these issues should be dealt with alongside routing to deliver an effective routing mechanism. The literature survey enabled us to identify some of the key characteristics of a routing protocol for mobile ad-hoc networks. These findings were used to define some of the primary objectives of our research.
- **Novel Routing Algorithm for Mobile Ad-hoc Network:** This thesis proposed a novel routing algorithm "Mobile ad-hoc on Demand Data Delivery protocol (MAODDP)" as a routing solution for mobile ad-hoc networks. Based on a unique integrated structure, some of the key features of this protocol are
    - Integrated approach: MAODDP adopts an integrated approach and deals with routing related issues along with routing. In most of the existing routing techniques[23,25] these factors have not been addressed fully. Research show[3] weak performance of existing techniques in addressing these issues.
    - Route establishment and data delivery: MAODDP focuses on route establishment and data delivery one after the other. In a table driven continuous network updates could result in unnecessary network overhead, thereby resulting in reduction of network efficiency[40]. Likewise too many query packets prior to data delivery could yield the same effect[34]. MAODDP neither requires any regular updates nor does it depend on route establishment prior to data delivery.

- Power aware operation: MAODDP conserves battery power of mobile nodes with its own power saving mechanisms. Such an approach is absent from table driven protocols however some of the on-demand protocols like AODV adopt power saving mechanisms. However, AODV requires mobile nodes to broadcast at regular intervals of time therefore the sleep time of mobile nodes become less than in MAODDP where no such periodic broadcast is needed.
- Security: Some of the famous protocols like DSDV and AODV are silent on security issues. MAODDP has its own security mechanisms and offers secure routing in mobile ad-hoc networks.
- Selection of Shortest route: The selection of shortest path is an important aspect in mobile ad-hoc network communication. MAODDP adopts a totally new strategy for selecting the shortest and the best path from a source to a destination. MAODDP discovers the shortest path via additional counters known as hop-counters. A hop-counter is useful for mobile nodes to discover the shortest path from them to the destination of interest.
- Guaranteed Data Delivery. MAODDP introduced the acknowledge packet as a sign of successful data delivery. No such mechanism is found in some of the famous existing schemes.
- Support of real time application: Support to real time communication is possible via MAODDP as mobile nodes do not need to wait for route establishment prior to data transfer.
- Bandwidth conservation: Bandwidth conservation is important in mobile ad-hoc networks. In comparison with MAODDP existing routing techniques[13, 33, 51] shows no specific procedure to save unnecessary bandwidth consumption. Table driven protocols due to their specific routing approach consume more bandwidth. Similarly on-demand protocols with too many query packets, the route replies could result in the same. Moreover issues like network congestion and network speed are related with the available bandwidth. A network with higher available bandwidth could result in better performance and less chance of network reduction[54, 55]. MAODDP addressed most of these deficiencies in the available schemes.

- **Implementation and Evaluation:** Developing the functional model based on the MAODDP specification was an essential requirement both for the present and future development of the protocol. This implementation has later been evaluated in a simulation framework to verify various concepts in a more practical environment. Evaluation experiments revealed MAODDP's effectiveness in terms of message activities. MAODDP accumulated average data delivery calculated as 79.07% with conservation of an accumulated average memory of 67.50%. The

calculated average number of new route formation is 613. A separate set of experiments was conducted to measure the effectiveness of the MAODDP power aware mechanisms. Findings show that almost 22 % of available memory could be saved under power saving mode along with a better data delivery. MAODDP was compared against AODV and DSR, MAODDP performed well in active route construction and in conserving available memory. AODV showed slightly better performance in terms of broadcast route requests and adding new routes in simulation runs. Results showed that MAODDP saved an accumulated average of 2.6 times more memory than AODV. MAODDP data delivery in a varying mobile environment found 55.87 times better than DSR.

8.3 Future Work

This thesis has reviewed all important technologies associated with mobile ad-hoc networks in the context of routing mechanisms. One of the tasks was to monitor transformation of routing mechanisms from conventional to fixed wireless and to mobile ad-hoc networks. Our investigation besides the invention of a novel routing algorithm highlighted various connections between protocols for different network types. However, we feel that our research findings could be extended further into the following directions

- **Support of Multimedia Applications:** Multimedia applications have their due importance in all advanced applications. Developing a robust multimedia streaming with an efficient lossy redundant packet recovery technique for mobile ad-hoc networks is an important research topic. However, it was out of the scope of our project. It will be interesting if a study comprising protocol performance to support such application could be carried out.
- **Sensor Networks:** Mobile ad-hoc sensor networks to some extent share some similar characteristics as mobile ad-hoc networks. A mobile ad-hoc sensor or hybrid ad-hoc network consists of a number of sensors spread in a geographical area. In order to support routed communications between two mobile nodes, the routing protocol determines the node connectivity and routes packets accordingly. Routing in ad-hoc sensor and ad-hoc networks suffers with some of the similar constraints. Therefore it may be useful if some of the research findings could further be extended to apply in ad-hoc sensor networks.
- **Measuring quality of services.** Mobile ad-hoc networks are expected to play an important role in the deployment of future wireless communication systems. Therefore, it is extremely important that these networks should be able to provide efficient quality of service that can meet the vendor requirements. To provide efficient quality of service in mobile ad-hoc networks, there is a solid need to establish new architectures and services for routine network controls. Besides architectures,

protocols to handle various network controls are needed to achieve high quality of service. It would be interesting if MAODDP performance could be measured in such scenarios.

- **Data transportation in an Internet Based Mobile Ad-hoc Network.** Connectivity of a mobile ad-hoc network with a larger network such as the Internet is a growing research topic. It would be particularly important if in future protocols could be applied to support such applications.
- **Simulation in different networking environment.** Depending on user requirements the network environment could vary from place to place. It is therefore preferable if the protocol performance could be measured in various network environments. Testing the protocol performance under greater traffic loads and comparing results with those for existing protocols would be useful information. It can also be helpful in further development of the proposed protocol in order to support various ad-hoc networking scenarios.

Absence of standard solutions for problems of mobile ad-hoc networks is one of the main reasons in their frequent deployment. We are confident that we have proposed a solution fully capable of fulfilling the routing requirement of mobile ad-hoc networks. We understand after some of the minor work as described above MAODDP could be adopted as one of the standard routing protocols for mobile ad-hoc networks.

REFERENCES

[1]. Matthias Grossglauser and David Tse. Mobility Increases the Capacity of Ad-hoc Wireless Networks. In IEEE Conference on Computer Communications (INFOCOM). 2001.

[2]. H.Bakht, M.Merabti, and R.Askwith. Mobile Ad-Hoc on Demand Data Delivery Protocol. In the proceedings of PGNET 2002 , Liverpool John Moores University, Liverpool, UK, 17th - 18th June 2002.

[3]. H.Bakht. A study of routing protocols for mobile ad hoc networks. In 1st International Computer Engineering Conference. December 2004. Cairo, Egypt.

[4]. H.Bakht, Some applications of mobile ad-hoc network, in Computing Unplugged Magazine. September 2004. p. 1.

[5]. H.Bakht. Our future home environment and mobile ad-hoc networks, in Computing Unplugged Magazine. October 2005. p. 1.

[6]. H.Bakht. Sensor networks and ad hoc networking, in Computing Unplugged. Oct 2004. p. 1.

[7]. H.Bakht. Group communication in combat operations using mobile ad-hoc networks, in Computing Unplugged Magazine. October 2005. p. 2.

[8]. H.Bakht. Some characteristics of mobile ad-hoc networks, in Computing Unplugged Magazine. July 2004. p. 2.

[9]. B.Prabhakar. Some problems in ad hoc wireless networking. Stanford Networking Research Center Project workshop, June 7, 2001.

[10]. M.Papdopouli and H.Schulzrinned. Connection Sharing in an Ad-Hoc Wireless Network among Collaborating Hosts. in Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), (Basking Ridge, New Jersey), June 1999: p. pp. 169--185.

[11]. J.J. Garcia-Luna-Aceves and M. Spohn. Bandwidth-Efficient Link-State Routing in Wireless Networks, ed. E. Ad Hoc Routing (C. Perkins, Chapter 10. 2001: Addison-Wesley Longman.

[12]. D. Kim, et al., Routing Mechanisms for Mobile Ad Hoc Networks based on the Energy Drain Rate. IEEE Transactions on Mobile Computing, April-June 2003. 2(2): p. 161-173.

[13]. A.Kesselman, D.Kowalski, and M.Segal. Energy efficient communication in ad hoc networks from user's and designer's perspective. ACM SIGMOBILE Mobile Computing and Communications Review, January, 2005. 9(1): p. 15 - 26.

[14]. B. Chen. et al. Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. In Proceedings of the 7th ACM International Conference on Mobile Computing and Networking. July, 2001. Rome, Italy.

[15]. T. Wan, E. Kranakis, and P. Van Oorschot. Securing the Destination Sequenced Distance Vector Routing Protocol (S-DSDV). In 6th International Conference on Information and Communications Security. Oct. 27-29, 2004. Malaga, Spain: Springer Verlag.

[16]. H. Bakht. Critical ad-hoc networking features, in Computing Unplugged Magazine. March 2005. p. 3.

[17]. C-C.Chiang, et al. Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel. In Proceedings of the IEEE Singapore International Conference on Networks. April, 1997. Singapore.

[18]. Jinyang Li, et al. Capacity of Ad Hoc Wireless Networks. In 7th ACM International Conference on Mobile Computing and Networking. July 2001. Rome,Italy.

[19]. H.Bakht. Service awareness in mobile ad-hoc networks, in Computing Unplugged Magazine. December 2004.

[20]. L.Ewerton and J.J.Garcia-Luna-Aceves. Multicasting Along Meshes in Ad-Hoc Networks. In Proceedings of the IEEE International Conference on Communications (ICC). June,1999. Vancouver, Canada.

[21]. G. Holland and N.Vaidya. Analysis of TCP Performance over Mobile Ad Hoc Networks. In Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom). 1999.
[22]. B.S. Manoj and C. S.R.Murthy. TCP over Ad Hoc Wireless Networks. Transport Layer and Security Protocols for Ad Hoc Wireless Networks. Jan 21, 2005.
[23]. S.Kannan, John E Mellor, and D.D.Kouvatsos. Investigation of routing in DSDV. 4th Annual Post-Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting, Liverpool UK, 2003.
[24]. G. Pei, M. Gerla, and T.-W. Chen. Fisheye State Routing in Mobile Ad Hoc Networks. Proceedings of Workshop on Wireless Networks and Mobile Computing, 2000.
[25]. C.E. Perkins and Elizabeth M. Royer, Ad-hoc On-Demand Distance Vector Routing. Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, February 1999: p. pp. 90-100.
[26] E.M.Royer and C.E. Perkins. Multicast Operation of the Ad-hoc On-Demand Distance Vector Routing Protocol. Proceedings of MobiCom, 1999.
[27]. H. Bakht, M. Merabti, and R. Askwith. Centralized frame for routing in mobile ad-hoc networks. In International Conference on Computer Communication. September, 2004. Beijing, China.
[28]. D. B. Johnson and D.A. Maltz, Dynamic Source Routing in Ad hoc wireless Networks. Mobile Computing, 1996. 353.
[29]. A. Aaron and J. WengA. Performance Comparison of Ad-Hoc Routing Protocols for Networks with Node Energy Constraints. June, 2001, EE 360 Class Project, Stanford University.
[30]. I. D. Aron and S. K.S.Gupta. On the Scalability of On-Demand Routing Protocols for Mobile Ad-hoc Networks: An Analytical Study. Journal of Interconnection Networks, January, 2001.
[31]. J. Broch, et al., A Performance Comparison of Multi-Hop Wireless Ad-Hoc Network Routing Protocols. In Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom), 1998.
[32]. S. R. Das, et al. Comparative performance evaluation of routing protocols for mobile ad hoc networks. In 7th International Conference on Computer Communications and Networks (IC3N), October,1998. Lafayette, LA University of Texas, San Antonio.
[33]. S. R. Das, R.Castañeda, and J.Yan. Simulation Based Performance Evaluation of Mobile Ad Hoc Network Routing Protocols. ACM/Baltzer Mobile Networks and Applications (MONET), 2000: p. 179-189.
[34]. M.Asim and A.J. Pullin. Comparison analysis of MAODDP with some other prominent wireless ad hoc routing protocols, in IBITE Computing. 2005, Liverpool Hope University: Liverpool. 2000.
[35]. E. Royer and C.Toh. A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks. IEEE Personal Communications Magazine, April 1999: p. 46-55.
[36]. S. Sesay, et al. Simulation Comparison of Four Wireless Ad hoc Routing Protocols. Information Technology, 2004. 3(3): p. 219-226.
[37]. C.K. Toh, S.J.Lee and M.Gerla. A simulation study of table driven and on-demand routing protocols for mobile ad hoc networks. IEEE Network, 1999. 13: p. 48-54.
[38]. W. Wang, Y. Lu, and B. Bhargava. On Security Study of Two Distance Vector Routing Protocols for Mobile Ad Hoc Networks. In IEEE International Conference on Pervasive Computing and Communications. March 2003. Dallas-Fort Worth, Texas, USA.
[39]. F.Stajano and R.Anderson. The Resurrecting Duckling: Security Issues for Ad-Hoc Wireless Network. In 7th International Workshop on Security Protocols. April, 1999. Cambridge, UK.
[40]. H.Bakht. A comparison based overview of destination distance sequence vector routing (DSDV) and mobile ad hoc on demand data delivery protocol "(MAODDP)",. In International Workshop on Wireless Ad-hoc Networks. May, 2005. London, United Kingdom.

[41]. S.-J. Lee, M. Gerla, and C.-C. Chiang. On-Demand Multicast Routing Protocol. In Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC99). Sep. 1999. New Orleans, LA.
[42]. S.Toumpis and A.Goldsmith. Some Capacity Results for Ad Hoc Networks. Allerton Conference on Communication, Control and Computing, 2000.
[43]. A. Mukhija and R.Bose. RRP - A Bandwidth-efficient Routing Protocol for Mobile Ad-hoc Networks. In 2nd International Information and Telecommunication Technologies Symposium. November, 2003. Florianopolis, Brazil.
[44]. H.Bakht. Routing Protocols for mobile ad hoc networks. In GERI Annual Research Symposium 2005. June 2005. Liverpool, United Kingdom.
[45]. E. M. Royer, S.Lee, and C.E. Perkins. The Effects of MAC Protocols on Ad hoc Network Communication. In IEEE WCNC. September, 2000.
[46]. S.Lee and C.Kim. A new Wireless ad hoc multicast routing protocol. Computer Networks, June 2001(38): p. 121-135.
[47]. S. Basagni. I. Chlamtac, and V. R. Syrotiuk. A Distance Routing Effect Algorithm for Mobility (DREAM). in ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom). October,1998. Dallas, TX, USA
[48]. N.Nikaein, H.Labiod, and C.Bonnet. Distributed Dynamic Routing Algorithm (DDR) for Mobile Ad Hoc Networks. In Proceedings of theACM Symposium on Mobile Ad Hoc Networking Computing (MOBIHOC2000). 2000.
[49]. K. Bur and C. Ersoy. Multicast Routing for Ad Hoc Networks with a Multiclass Scheme or Quality of Service. In ISCIS 2004. October, 2004. Antalya.
[50]. N.Nikaein and C.Bonnet. Hybrid Ad Hoc Routing Protocol. In International Symposium on Telecommunications. 2001.
[51]. C-Y. Chang and C-T. Chang, Hierarchical Cellular-Based Management for Mobile Hosts in Ad-Hoc Wireless Networks. Computer Communication, 2001(24): p. 554-1567.
[52]. Z.J. Haas and M.R. Pearlman. The Performance of the Zone Routing Protocol in Reconfigurable Wireless Networks. IEEE JSAC Issue on Ad-Hoc Networks, June, 1999.
[53]. Z.J Hass and M.R. Pearlman. The Performance of Query Control Schemes for the Zone Routing Protocol. ACM/IEEE Transactions on Networking, August, 2001. 9(4): p. 427-438.
[54]. A. Helmy. Contact-extended Zone-based Routing for Transactions in Ad Hoc Networks. IEEE Transactions on Vehicular Technology, July, 2003.
[55]. P.Sinha, S.V.Krishnamurthy and S.Dao. Scalable Unidirectional Routing with Zone Routing Protocol (ZRP) Extensions for Mobile Ad-Hoc Networks. In Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC2000). 2000. Chicago, USA.
[56]. J. J. Garcia-Luna-Aceves. Flow-oriented Protocols for Scalable Wireless Networks. In Proceedings of the 5th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems. 2002. Atlanta, Georgia, USA.
[57]. J.J Garcia-Luna-Aceves and M. Spohn. Source tree routing in wireless networks. In Proceeding of IEEE International Conference on Network Protocols. Oct 1999.
[58]. C.N.Sekharant, et al. DST a routing protocol for ad hoc networks using distributed spanning trees. In IEEE Wireless Communications and Networking Conferences. 1999.
[59]. B. Bellur and R. G. Ogier. A reliable efficient topology broadcast protocol for dynamic networks. In IEEE Conference on Computer Communications (INFOCOM). 1999.
[60]. M. Goyal, K.K. Ramakrishnan, and W. Feng. Achieving Faster Failure Detection in OSPF Networks. In Proceedings of the IEEE International Conference on Communications (ICC). 2003.

[61]. I. D. Aron and S. K. S. Gupta. A Witness-Aided Routing Protocol for Mobile Ad-Hoc Networks with Unidirectional Links. In Proceedings of the First International Conference on Mobile Data Acces. 1999: Springer-Verlag, London, UK.
[62]. H.Zimmermann, The ISO Model of Architecture for Open Systems Interconnection. IEEE Transactions on Communications, April, 1980. **28**(4): p. 425-432.
[63]. Mark a Sportack. RIP's stability features. in IP Routing Fundamentals. 1999: Indianapolis.
[64]. C.K-Toh. Associatively Based Routing for Ad-hoc Mobile Networks. Wireless Personal Communications, 1997. **4**.
[65]. T-Wei.Chen and M.Gerla. Global State Routing: A New Routing Scheme for Ad-hoc Wireless Networks. In Proceedings of the IEEE International Conference on Communications (ICC). 1998.
[66]. M. S. Corson and A. Ephremides. A Distributed Routing Algorithm for Mobile Wireless Networks. ACM/Baltzer J. Wireless Network, February, 1995. **1**(1): p. 61-82.
[67]. L. Ji and M.S. Corson. Lightweight Adaptive Multicast protocol (LAM). In IEEE GLOBECOM. 1998. Sydney, Australia.
[68]. R. Ogier, F. Templin, and M. Lewis. Topology Dissemination Based on Reverse-Path Forwarding (TBRPF). 2004.
[69]. S.Murthy and J. J. Garcia-Luna Aceves. An Efficient Routing Protocol for Wireless Networks. ACM/Baltzer on Mobile Networks and Applications, October, 1996. **1**(2): p. 183-197.
[70]. G.Pei, M.Gerla, and X.Hong. LANMAR: Landmark Routing for Large Scale Wireless Ad Hoc Networks with Group Mobility. In Proceedings of the ACM Symposium on Mobile Ad Hoc Networking Computing. August, 2000. Boston, MA, USA.
[71]. Philippe Jacquet, et al., Optimized Link State Routing Protocol. Internet Draft, draft-ietf-manet-olsr-04.txt, work in progress, June 2001.
[72]. G. Pei, M. Gerla, and X Hong. A Wireless Hierarchical Routing Protocol with Group Mobility. 1999. Boston, MA.
[73]. M. Jiang, J. Li, and Y. C. Tay. Cluster Based Routing Protocol (CBRP) Functional Specification. June, 1999.
[74]. M.Gerla, X.Hong, and G.Pei. Landmark Routing for Large Ad Hoc Wireless Networks. In IEEE GLOBECOM. November, 2000. SanFrancisco, CA, USA.
[75]. W-H Liao, J-P Sheu and Y-C. Tseng. GeoGRID & Geographical GRID: A Fully Location-Aware Routing Protocol for Mobile Ad Hoc Networks,. Telecommunication Systems, 2001. **18**(1-3): p. 37-60.
[76]. X.Lin and I.Stojmenivoc. GPS based distributed routing algorithms for wireless networks, in Technical Report Computer Science SITE, University of Ottawa. 2000: Ottawa, Ontario,Canada.
[77]. Brad Karp and H.T.Kung. (Gpsr) Greedy perimeter stateless routing for wireless networks. In 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking. August 2000.
[78]. Y-B.Ko and N.H. Vaidya. Location - Aided Routing (LAR) in Mobile Ad Hoc Networks. In Mobile Computing and Networking. 1998. Dallas, Texas, USA.
[79]. I. Stojmenovic and X. Lin. GEDIR: Loop-free location based routing in wireless networks. In International Conference on Parallel and Distributing Computing System. Nov,1999.
[80]. Vi. D. Park and M.S. Corson. A Performance Comparison of the Temporally-Ordered Routing Algorithm and Ideal Link-State Routing. In Proceeding of the IEEE Symposium on Computers and Communication, June
1998.
[81]. R.Sivakumar, P.Sinha, and V.Bharghavan. CEDAR: Core Extraction Distributed Ad hoc Routing. IEEE Journal on Selected Areas in Communication, Special Issue on Ad hoc Networks, 1999. 17(8).
[82]. R.Dube, et al., Signal Stability based adaptive routing (SSR alt SSA) for ad hoc mobile networks. IEEE Personal Communication, Feb. 1997.

[83]. R.V.Boppanan and S.Konduru. An Adaptive Distance Vector Routing Algorithm for Mobile Ad-hoc Networks. In Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. 2001.
[84]. C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. Computer Communications Review, 94. 24(4): p. 234- 244.
[85]. S.Palchaudhuri and S.Lavu. A Performance Comparison of Ad Hoc Network Routing Protocols.
[86]. Per Johansson, et al. Scenario based performance analysis of routing protocols for mobile ad hoc networks. In proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking MobiCom99. August 1999.
[87]. Xiaoyan Hong, Kaixin Xu, and Mario Gerla. Scalable Routing Protocols for Mobile Ad Hoc Networks. IEEE Network,, July/August 2002. 16(4): p. 11-21.
[88]. Y-C. Hu, A.Perrig, and D.B. Johnson. Ariadne: A Secure On-Demand Routing Protocol for Ad-hoc Networks. In Proceedings of the Mobile Ad-Hoc Networking and Computing. September.2002. Atlanta, Georgia, USA.
[89]. David B. Johnson, David A. Maltz, and Josh Broch. DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. Ad Hoc Networking, 2001.
[90]. Mahesh K. Marina and Samir R. Das. Performance of Route Caching Strategies in Dynamic Source Routing. In Proceedings of the 21st International Conference on Distributed Computing Systems. April 16 - 19, 2001. Mesa, Arizona.
[91]. J. Raju and J.J. Garcia-Luna-Aceves. Scenario-based comparison of Source-Tracing and Dynamic Source Routing Protocols for Ad-Hoc Networks. In IEEE ICC 2001. June 2001. Helsinki, Finland.
[92]. Basagni S., I. Chlamtac and V. R Syrotiuk. Dynamic Source Routing for Ad Hoc Networks Using the Global Positioning System. In Proceedings of the IEEE Wireless Communications and Networking Conference 1999 (WCNC'99). September 21-24, 1999. New Orleans, LA,.
[93]. J. Wu. An Extended Dynamic Source Routing Scheme in Ad Hoc Wireless Networks. In Proceedings of the 35th Annual Hawaii International Conference on System Sciences. January 07 - 10, 2002. Big Island, Hawaii.
[94]. G.Chakrabarti and S.S. Kulkarni. A Modified Approach to Dynamic Source Routing in Mobile Ad-Hoc Networks. ADHOC-NOW, 2002.
[95]. I.D. Aron and Sandeep K.S.Gupta. On the scalability of on-demand routing protocols for mobile ad hoc networks: An analytical study. Journal of Interconnection Networks, Jan 2001.
[96]. V. Park and S. Corson. Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In IEEE Conference on Computer Communications. 1997.
[97]. V. Park and S. Corson. Temporally-Ordered Routing Algorithm (TORA) Version 1 Functional Specification. Internet Draft, draft-ietf-manet-tora-spec-01.txt,, August 7, 1998.
[98]. V.D. Park and M.S.Corson. A Performance Comparison of Temporally-Ordered Routing Algorithm. Published in the Proceedings of ISCC.1998, 30 June.
[99]. C.E.Perkins. Link Reversal routing chapter 8, in Ad Hoc Networking. 2001, Addison Wesley 2001.
[100]. Chai-Keong Toh. Associativity-based routing for ad-hoc mobile networks Technical Report. 1996, University of Cambridge Computer Laboratory Cambridge CB2 3QH: Cambridge.
[101]. R.K Padmanaban. et al.R.K Padmanaban, et al., Optimized associativity-based threshold routing for mobile ad hoc networks. 2001, College of Engineering, Anna University: Guindy.
[102]. S.J. Lee, M. Gerla, and C.K. Toh. A Simulation Study of Table-Driven and On-Demand Routing Protocols for Mobile Ad Hoc Networks. IEEE Network, July 1999. **13**(4): p. 48-54.
[103]. G.Karumanchi, S.Muralidharan, and R.Prakash. Information Dissemination in Partitionable Mobile Ad Hoc Networks. 18th IEEE Symposium on Reliable Distributed Systems Lausanne, Switzerland, 18 - 21 October, 1999.

[104]. Daniel Lang. TECHREPORT A comprehensive overview about selected Ad Hoc Networking Routing Protocols. March 2003.
[105]. M.Gerla, et al. Fisheye state routing protocol for ad hoc networks. Internet Draftthttp://www.ieft/internet-drafts/draft-ietf-manet-fsr00.txt.
[106]. J.C.Navas and T.Imielinski. Geographic Addressing and Routing. In Proceedings of the Third ACM/IEEE International Conference on Mobile Computing and Networking. September 1997. Budapest, Hungary.
[107]. T.D. Dyer and R.V.Boppana. A comparison of tcp performance over three routing protocols for mobile ad hoc networks. In proceedings of the ACM Symposium on Mobile Ad Hoc Networking Computing (MOBIHOC). 2001.
[108]. R.Sivakumar, P.Sinha, and V.Bharghavan. Core Extraction Distributed Ad hoc Routing (CEDAR) Specification. Internet Draft, draft-ietf-manet-cedar-spec-00.txt.
[109]. W.Su and M.Gerla. IPV6 flow handoff in ad-hoc wireless networks using mobility perdication. In Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM). December 1999.
[110]. J.Hubaux, et al. Toward mobile ad-hoc wans: Termindoes. In Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'2000). September 2000. Chicago Ecole Polytechnique Federale de Lausanne.
[111]. L.Blazevic, S.Glordano, and J.Y.L.Boundec. Self Organize wide area routing. Tech Rep EPFL-Communication Systemms Department (DSC), Ecole Polytechnique Federale de Lausanne CH-1015 Lausanne Switzerland 2000.
[112]. B.R.Smith, S.Murthy, and J.J.Garcia. Securing distance vector routing protocol. In Symposium on Network and Distributed System Security. 1997.
[113]. A. Patwardhan, et al. Secure Routing and Intrusion Detection in Ad Hoc Networks. In Third IEEE International Conference on Pervasive Computing and Communications. March 8-12, 2005. Kauaii Island, Hawaii.
[114]. S.Murthy and J.J. Carcia-Luna-Aceves. A Routing Protocol for Packet Radio Networks. 1st ACM International Conference on Mobile Computing and Networking (Mobicom). 1995: p. 86-95.
[115]. C.W. Wu and Y.C. Tay. Ad hoc Multicast Routing protocol utilizing Increasing id-numbers (AMRIS). In Proceedings of IEEE MILCOM'99, Atlantic City, NJ, 1999.
[116]. C.-K Toh, G. Guichal, and S. Bunchua. On-demand associativity-based multicast routing for ad hoc mobile networks (ABAM). Vehicular Technology Conference IEEE VTS Fall VTC 2000, 2000.
[117]. G.Barua and M.Agarwal. Caching of routes in ad hoc on-demand distance vector routing for mobile ad hoc networks. In Proceedings of the 15th international conference on Computer communication. 2002. Mumbai, Maharashtra, India.
[118]. I.D. Chakeres and E.M. Belding-Royer. AODV Routing Protocol Implementation Design. In Proceedings of the 24th International Conference on Distributed Computing Systems Workshops - W7: EC (ICDCSW'04). March 23 - 24, 2004. Hachioji, Tokyo, Japan.
[119]. Yih-Chun.Hu and D.B.Johnson. Caching Strategies in On Demand Routing Protocols for Wireless Ad Hoc Networks. In Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking, August 6-11,2000, Boston, MA, USA., 2000.
[120]. D.A.Maltz, et al. The Effects of On-Demand Behaviour in Routing Protocols for Multi-Hop Wireless Ad Hoc Networks. IEEE Journal on Selected Areas in Communication. August,1999. 8(17): p. 1439-1453.
[121]. D. B. Johnson. Routing in Ad Hoc Networks of Mobile Hosts. In IEEE Workshop on Mobile Computing Systems and Applications. December, 1994.
[122]. Y.Hu, A.Perrig, and D.B. Johnson. SEAD: Secure Efficient Distance Vector Routing for Mobile Wireless Ad Hoc Networks. Ad Hoc Networks, July, 2003. 1(1): p. 175-192.

[123]. H. Bakht, et al. Multicasting in mobile ad-hoc networks. In 9th CDMA International Conference. October,2004. Seoul, Korea.
[124]. H. Rangarajan and J.J. Garcia-Luna-Aceves. Achieving Loop-Free Incremental Routing in Ad Hoc Networks. In 9th IEEE Symposium on Computers and Communications (ISCC'2004),. June 29-July 1 2004. Alexandria, Egypt.
[125]. Y.Hu, D. B. Johnson, and D. A. Maltz. Flow State in the Dynamic Source Routing Protocol. Internet Draft, draft-ietf-manet-dsrflow-00.txt, 2001.
[126]. H.Bakht. The history of mobile ad-hoc networks. In Computing Unplugged. August 2005. p. 2.

APPENDIX

Figure results of experiments no 6.3.1 to 6.3.7

```
cmtbeth@holly:~/jist-manx-1.0.6> manx driver.manetpsim -n 25 -s grid:3x5 -f 500x500 -t 10,600,60 -a 1.0 -m static
 -l none
------------
Packet stats:
------------
Rqst packets sent = 460
Rqst packets recv = 9635
Ack packets sent = 357
Ack packets recv = 384
Total manetp packets sent = 817
Total manetp packets recv = 10019
------------
Overall stats:
------------
Message to deliver = 250
Route Query Data Delivery in process   = 46
Total Acknowledge Generated      = 357
Total Routes added       = 46

Free Bandwidth: 4251752
Max Bandwidth Available:   9227466888
Total Bandwidth: 9150464
Bandwidth used:     4901120
start time  : Fri Jul 07 22:27:13 BST 2006
end time    : Fri Jul 07 22:27:19 BST 2006
elapsed time: 6372
25     466    9635   357    384    0      0      817    10019  4906120 6372
```

Figure 6.3.1A Results of experiment no 6.3.1A

```
cmtbeth@holly:~/jist-manx-1.0.6/results> manx driver.manetpsim -n 50 -s grid:10x10 -f 500x500 -t 10,600,60 -a 1.0
 -m static -l none
------------
Packet stats:
------------
Rqst packets sent = 1073
Rqst packets recv = 43581
Ack packets sent = 831
Ack packets recv = 985
Total manetp packets sent = 1904
Total manetp packets recv = 44466
-- -- -------
Overall stats:
-- ----------
Message to deliver = 500
Route Query Data Delivery in process   = 108
Total Acknowledge Generated      = 831
Total Routes added       = 108

Free Bandwidth: 1438776
Max Bandwidth Available:   9227466888
Total Bandwidth: 2748416
Bandwidth used:     1311056
start time  : Fri Jul 07 22:31:50 BST 2006
end time    : Fri Jul 07 22:32:16 BST 2006
elapsed time: 16136
50     1073   43581  831    985    0      0      1904   44466  1311056 16136
```

Figure 7.3.1B Results of experiment no 7.3.1B

```
cmtbeth@holly:~/jist-manx-1.0.6/results> manx driver.manetpsim -n 75 -s grid:15x15 -f 500x500 -t 15,600,60 -a 1.0
 -m static -l none
------------
Packet stats:
------------
Rqst packets sent = 1533
Rqst packets recv = 86433
Ack packets sent = 987
Ack packets recv = 1018
Total manetp packets sent = 2520
Total manetp packets recv = 87451
------------
Overall stats:
------------
Message to deliver = 756
Route Query Data Delivery in process   = 156
Total Acknowledge Generated      = 987
Total Routes added       = 153

Free Bandwidth: 1751672
Max Bandwidth Available:   9227466888
Total Bandwidth: 3150016
Bandwidth used:     1487760
start time  : Fri Jul 07 22:47:07 BST 2006
end time    : Fri Jul 07 22:47:46 BST 2006
elapsed time: 38861
75     1533   86433  987    1018   0      0      2520   87451  1487760 38861
```

Figure 6.3.1C Results of experiment no 6.3.1C

```
cmtbeth@holly:~/jist-manx-1.0.6/results> manx driver.manetpsim -n 100 -s grid:10x10 -f 500x500 -t 10,600,60 -a 1.
0 -m static -l none
------------
Packet stats:
------------
Rqst packets sent = 2268
Rqst packets recv = 147842
Ack packets sent = 1306
Ack packets recv = 1356
Total manetp packets sent = 3566
Total manetp packets recv = 149198
------------
Overall stats:
------------
Message to deliver = 1008
Route Query Data Delivery in process   = 227
Total Acknowledge Generated      = 1306
Total Routes added       = 223

Free Bandwidth: 1065816
Max Bandwidth Available:   9227466888
Total Bandwidth: 3449536
Bandwidth used:     1785136
start time  : Fri Jul 07 22:49:43 BST 2006
end time    : Fri Jul 07 23:51:11 BST 2006
elapsed time: 87694
100    2268   147842 1306   1356   0      0      3566   149198 1785136 87694
```

Figure 6.3.1D Results of experiment no 6.3.1D

```
[redacted]:~/jist-swans-1.0.6/results/Experiments/Experiment3> swans driver.aoddpsim -n 121 -s grid:11x11 -f 37
5x375 -t 10,600,60 -g 1.0 -m static -l none
 ------------
Packet stats:
------------
Rqdd packets sent = 2544
Rqdd packets recv = 300671
Ack packets sent = 2235
Ack packets recv = 2409
Total aoddp packets sent = 5179
Total aoddp packets recv = 303080
------------
Overall stats:
 ------------
Message to deliver = 1250
Route Query Data Delivery in process   = 295
Total Acknowledge Generated      = 2235
Total Routes added        = 295

Free Bandwidth:  2038672
Max Bandwidth Available:   922746880
Total Bandwidth: 4255744
Bandwidth used:     2190480
start time  : Tue Jul 11 21:17:39 BST 2006
end time    : Tue Jul 11 21:21:15 BST 2006
elapsed time: 215810
121    2544    300671  2235    2409    0       0       5179    303080  2190480 215810
```

Figure 6.3.1E results of exp no 6.3.1E

```
[redacted]:~/jist-swans-1.0.6> swans driver.aoddpsim -n 150 -s grid:15x15 -f 503x500 -t 10,600,60 -g 1.0 -m stat
ic -l none
 ------------
Packet stats:
------------
Rqdd packets sent = 3766
Rqdd packets recv = 373515
Ack packets sent = 2270
Ack packets recv = 2347
Total aoddp packets sent = 6044
Total aoddp packets recv = 375966
 ------------
Overall stats:
 ------------
Message to deliver = 1500
Route Query Data Delivery in process   = 379
Total Acknowledge Generated      = 2270
Total Routes added        = 378

Free Bandwidth:  2318696
Max Bandwidth Available:   922746880
Total Bandwidth: 4935680
Bandwidth used:     2590280
start time  : Sat Jul 08 13:50:00 BST 2006
end time    : Sat Jul 08 13:55:21 BST 2006
elapsed time: 321056
150    3766    373515  2270    2347    0       0       6044    375966  2590280 321056
```

Figure 6.3.1F Results of experiment no 6.3.1F

```
[redacted]:~/jist-swans-1.0.6/results/Experiments/Experiment3> swans driver.aoddpsim -n 250 -s grid:20x20 -f 95
0x356 -t 10,600,60 -g 1.0 -m static -l none
 ------------
Packet stats:
 ------------
Rqdd packets sent = 5569
Rqdd packets recv = 1144849
Ack packets sent = 4268
Ack packets recv = 4509
Total aoddp packets sent = 9777
Total aoddp packets recv = 1149438
 ------------
Overall stats:
 ------------
Message to deliver = 2500
Route Query Data Delivery in process   = 553
Total Acknowledge Generated      = 4268
Total Routes added        = 553

Free Bandwidth:  6407488
Max Bandwidth Available:   922746880
Total Bandwidth: 9412608
Bandwidth used:     4926616
start time  : Tue Jul 11 22:04:10 BST 2006
end time    : Tue Jul 11 22:28:21 BST 2006
elapsed time: 1451020
250    5569    1144849 4268    4509    0       0       9777    1149438 4926616 1451020
```

Figure 6.3.1G Results of experiment no 6.3.1G

```
[redacted]:~/jist-swans-1.0.6/results> swans driver.aoddpsim -n 350 -s grid:20x20 -f 500x500 -t 10,600,60 -g 1.0
 -m static -l none
 ------------
Packet stats:
 ------------
Rqdd packets sent = 8485
Rqdd packets recv = 2426965
Ack packets sent = 6452
Ack packets recv = 6895
Total aoddp packets sent = 14937
Total aoddp packets recv = 2433860
 ------------
Overall stats:
 ------------
Message to deliver = 3500
Route Query Data Delivery in process   = 852
Total Acknowledge Generated      = 6452
Total Routes added        = 851

Free Bandwidth:  7393568
Max Bandwidth Available:   922746880
Total Bandwidth: 13513648
Bandwidth used:     6121496
start time  : Sun Jul 09 13:01:26 BST 2006
end time    : Sun Jul 09 14:16:21 BST 2006
elapsed time: 4137072
350    8485    2426965 6452    6895    0       0       14937   2433860 6121496 4137072
```

Figure 6.3.1H Results of experiment no 6.3.1H

```
[redacted]:~/jist-manu-1.0.6/results/7H> manu driver.manetpain -n 450 -a grid:21x25 -f 500x500 -t 10,000,00 -s 1.0
 -m static -l manu
------------
Packet stats:
------------
Rqst packets sent = 105.6
Rqst packets recv = 1205030
Act packets sent = 6412
Act packets recv = 6422
Total manetp packets sent = 16920
Total manetp packets recv = 1213422
 ------------
Overall stats:
 ------------
Message to deliver = 4500
Route Query Data Delivery in process  = 1056
Total Acknowledge Generated     = 6413
Total Routes added       = 1055

Free Bandwidth  12813648
Max Bandwidth Available:  923744880
Total Bandwidth: 25112576
Bandwidth used:    12298944
Start time  : Sun Jul 09 18:14:07 BST 2006
End time    : Sun Jul 09 18:05:39 BST 2006
Elapsed time: 6691931
450     10516   1205030 6412    6422    0       0       16920   1213422 12298944        6691933
```

Figure 6.3.1I Results of experiment no 6.3.1I

RESULTS FIGURES OF EXPERIMENTS NO 6.3.2

```
[redacted]:~/jist-manu-1.0.6/results/732> manu driver.manetpain -n 25 -a gr.6:5x5 -f 500x500 -t 10,000,00, -s 1
.0 -m static
 ------------
Packet stats:
------------
Rqst packets sent = 550
Rqst packets recv = 13950
Act packets sent = 550
Act packets recv = 614
Total manetp packets sent = 1116
Total manetp packets recv = 14564
-------------
Overall stats:
-------------
Message to deliver = 133
Route Query Data Delivery in process  = 26
Total Acknowledge Generated     = 550
Total Routes added       = 26

Free Bandwidth:  1356186
Max Bandwidth Available:  922746880
Total Bandwidth: 2698560
Bandwidth used:    1145016
Start time  : Thu Jul 13 15:34:41 BST 2006
End time    : Thu Jul 13 15:34:46 BST 2006
Elapsed time: 5144
25      550     13950   550     614     0       0       1116    14564   1145016 5144
```

Figure 6.3.2A Results of experiment no 6.3.2A

```
[redacted]:~/jist-manu-1.0.6/results/732> manu driver.manetpain -n 50 -a grid:10x10 -f 500x500 -t 10,000,00, -s
 1.0 -m static
------------
Packet stats:
------------
Rqst packets sent = 1249
Rqst packets recv = 62450
Act packets sent = 1249
Act packets recv = 1374
Total manetp packets sent = 2498
Total manetp packets recv = 63824
 -- -- - --
Overall stats:
- -- ---  --
Message to deliver = 666
Route Query Data Delivery in process  = 125
Total Acknowledge Generated     = 1249
Total Routes added       = 125

Free Bandwidth:  1458028
Max Bandwidth Available:  922746880
Total Bandwidth: 2760096
Bandwidth used:    1113384
Start time  : Thu Jul 13 15:37:17 BST 2006
End time    : Thu Jul 13 15:37:41 BST 2006
Elapsed time: 24399
50      1249    62450   1249    1374    0       0       2498    63824   1113384 24399
```

Figure 6.3.2B Results of experiment no 6.3.2B

```
[redacted]:~/jist-manu-1.0.6/results/732> manu driver.manetpain -n 75 -a grid:10x10 -f 500x500 -t 10,000,00, -s
 1.0 -m static
 ------------
Packet stats:
 ------------
Rqst packets sent = 2039
Rqst packets recv = 106948
Act packets sent = 1247
Act packets recv = 1268
Total manetp packets sent = 3100
Total manetp packets recv = 108216
 - ----------
Overall stats:
-- ----------
Message to deliver = 1000
Route Query Data Delivery in process  = 204
Total Acknowledge Generated     = 1247
Total Routes added       = 204

Free Bandwidth:  1794712
Max Bandwidth Available:  922746880
Total Bandwidth: 3215369
Bandwidth used:    1418064
Start time  : Thu Jul 13 15:38:05 BST 2006
End time    : Thu Jul 13 15:38:58 BST 2006
Elapsed time: 52972
75      2039    106948  1247    1268    0       0       3306    108216  1418064 52972
```

Figure 6.3.2C Results of experiment no 6.3.2C

```
[redacted]:~/jist-manu-1.0.6/results/732> manu driver.manetpain -n 100 -a grid:10x10 -f 500x500 -t 10,000,00, -
s 1.0 -m static
 ... ---- ...
Packet stats:
 ----------
Rqst packets sent = 3533
Rqst packets recv = 249538
Act packets sent = 2478
Act packets recv = 2661
Total manetp packets sent = 5011
Total manetp packets recv = 251201
- ----------
Overall stats:
-------------
Message to deliver = 1333
Route Query Data Delivery in process  = 254
Total Acknowledge Generated     = 2478
Total Routes added       = 254

Free Bandwidth:  1949056
Max Bandwidth Available:  922746880
Total Bandwidth: 3600384
Bandwidth used:    1652744
Start time  : Thu Jul 13 15:39:38 BST 2006
End time    : Thu Jul 13 15:42:11 BST 2006
Elapsed time: 151909
100     3533    249538  2478    2661    0       0       5011    251201  1652744 151909
```

Figure 6.3.2D Results of experiment no 6.3.2D

```
[redacted]:~/jist-manet-1.0.6/results/732> manet driver.manetpsim -n 125 -a grid:12x12 -f 900x500 -t 10,800,60, -s 1.0 -m static
------------
Packet stats:
------------
Rqst packets sent = 3430
Rqst packets recv = 425000
Ack packets sent = 3400
Ack packets recv = 3741
Total manetp packets sent = 6831
Total manetp packets recv = 428741
-------------
Overall stats:
-------------
Message to deliver = 3691
Route Query Data Delivery in process = 341
Total Acknowledge Generated = 3400
Total Routes added = 341

Free Bandwidth: 2127946
Max Bandwidth Available: 922746880
Total Bandwidth: 4330472
Bandwidth used: 2212934
start time : Thu Jul 13 15:48:30 BST 2006
end time : Thu Jul 13 15:54:43 BST 2006
elapsed time: 373728
125 3430 425000 3430 3741 0 0 6831 428741 2202804 373728
```

Figure 6.3.2E Results of experiment no 6.3.2E

```
[redacted]:~/jist-manet-1.0.6/results/732> manet driver.manetpsim -n 150 -a grid:15x13 -f 900x500 -t 10,800,60, -s 1.0 -m static
------------
Packet stats:
------------
Rqst packets sent = 4066
Rqst packets recv = 426924
Ack packets sent = 2584
Ack packets recv = 2687
Total manetp packets sent = 6650
Total manetp packets recv = 429611
-------------
Overall stats:
-------------
Message to deliver = 3000
Route Query Data Delivery in process = 408
Total Acknowledge Generated = 2584
Total Routes added = 408

Free Bandwidth: 2306416
Max Bandwidth Available: 922746880
Total Bandwidth: 4984832
Bandwidth used: 2599832
start time : Thu Jul 13 15:58:26 BST 2006
end time : Thu Jul 13 16:04:41 BST 2006
elapsed time: 380797
150 4066 426924 2584 2687 0 0 6650 429611 2599832 380797
```

Figure 6.3.2F Results of experiment no 6.3.2F

```
[redacted]:~/jist-manet-1.0.6/results/732> manet driver.manetpsim -n 200 -a grid:25x25 -f 900x500 -t 10,800,60 -s 1.0 -m static -l none
------------
Packet stats:
------------
Rqst packets sent = 6495
Rqst packets recv = 1597736
Ack packets sent = 6361
Ack packets recv = 6957
Total manetp packets sent = 12810
Total manetp packets recv = 1604693
-------------
Overall stats:
-------- ---
Message to deliver = 3333
Route Query Data Delivery in process = 647
Total Acknowledge Generated = 6361
Total Routes added = 647

Free Bandwidth: 5081464
Max Bandwidth Available: 922746880
Total Bandwidth: 9666560
Bandwidth used: 4586512
start time : Thu Jul 13 19:15:20 BST 2006
end time : Thu Jul 13 19:50:56 BST 2006
elapsed time: 2126693
200 6495 1597736 6361 6957 0 0 12810 1604693 4586512 2126693
```

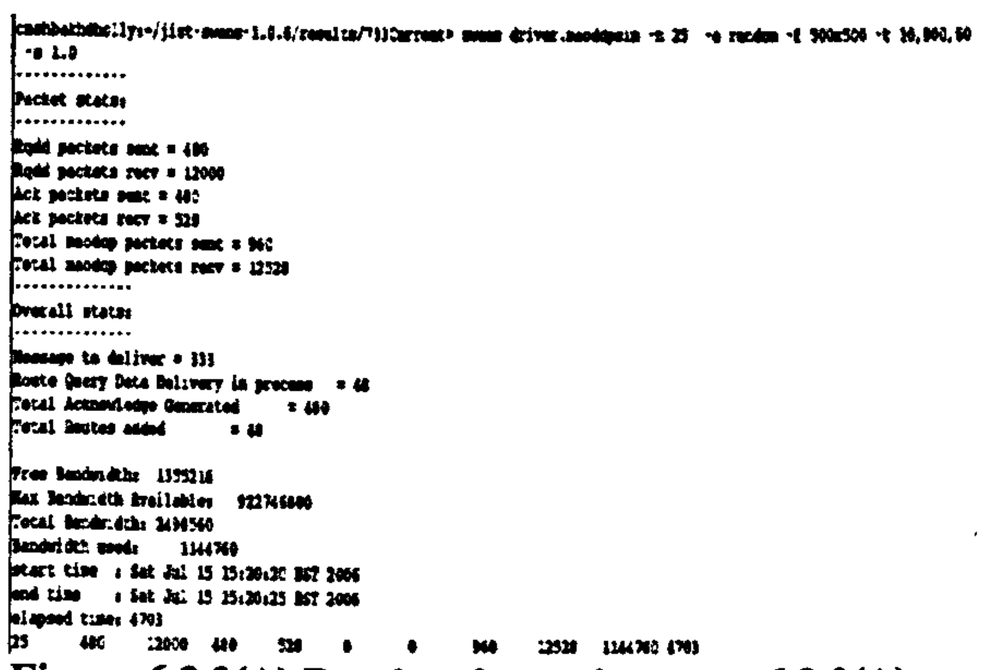
Figure 6.3.2G Results of experiment no 6.3.2G

```
[redacted]:~/jist-manet-1.0.6/results/732> manet driver.manetpsim -n 450 -a grid:25x25 -f 900x500 -t 10,800,60 -s 1.0 -m static -l none
------------
Packet stats:
------------
Rqst packets sent = 11563
Rqst packets recv = 3618310
Ack packets sent = 7250
Ack packets recv = 7226
Total manetp packets sent = 18821
Total manetp packets recv = 3623036
-------------
Overall stats:
-------------
Message to deliver = 6000
Route Query Data Delivery in process = 1150
Total Acknowledge Generated = 7250
Total Routes added = 1150

Free Bandwidth: 11726736
Max Bandwidth Available: 922746880
Total Bandwidth: 255.3984
Bandwidth used: 11728664
start time : Fri Jul 14 12:56:14 BST 2006
end time : Fri Jul 14 14:16:14 BST 2006
elapsed time: 7794792
450 1.563 3618310 7250 7226 0 0 18821 3623036 11728664 7794792
```

Figure 6.3.2I Results of experiment no 6.3.2I

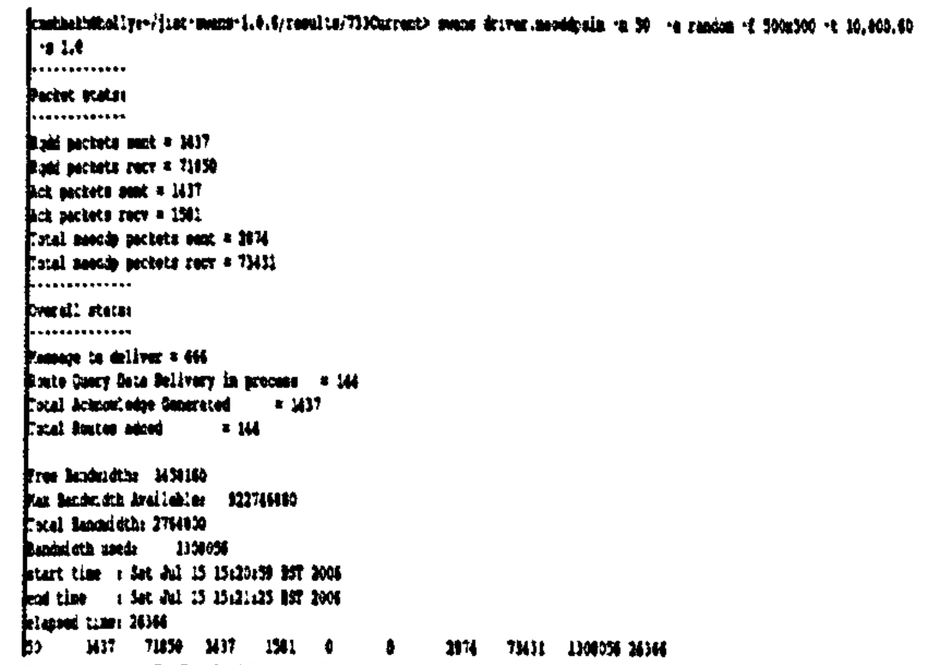
FIGURES RESULTS OF EXPERIMENTS NO 6.3.3



```
-----------
Packet stats:
-----------
Rqdd packets sent = 480
Rqdd packets recv = 12000
Ack packets sent = 480
Ack packets recv = 528
Total manetp packets sent = 960
Total manetp packets recv = 12528
-----------
Overall stats:
-----------
Message to deliver = 333
Route Query Data Delivery in process  = 48
Total Acknowledge Generated     = 480
Total Routes added       = 48

Free Bandwidth:  1395216
Max Bandwidth Available:  922746880
Total Bandwidth: 2490560
Bandwidth used:     1144760
start time  : Sat Jul 15 15:20:20 BST 2006
end time    : Sat Jul 15 15:20:25 BST 2006
elapsed time: 4703
25     480    12000  480    528    0      0      960    12528  1144760 4703
```

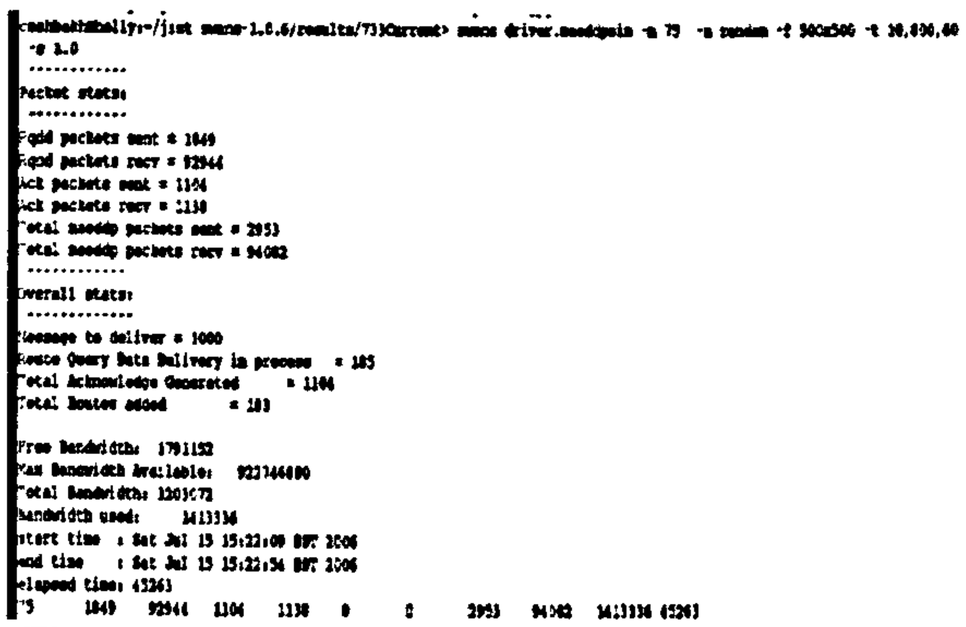
Figure 6.3.3(A) Results of experiment no 6.3.3(A)



```
-----------
Packet stats:
-----------
Rqdd packets sent = 1437
Rqdd packets recv = 71850
Ack packets sent = 1437
Ack packets recv = 1581
Total manetp packets sent = 2874
Total manetp packets recv = 73431
-----------
Overall stats:
-----------
Message to deliver = 666
Route Query Data Delivery in process  = 144
Total Acknowledge Generated     = 1437
Total Routes added       = 144

Free Bandwidth:  1458160
Max Bandwidth Available:  922746880
Total Bandwidth: 2764800
Bandwidth used:     1338656
start time  : Sat Jul 15 15:20:59 BST 2006
end time    : Sat Jul 15 15:21:25 BST 2006
elapsed time: 26366
50     1437   71850  1437   1581   0      0      2874   73431  1338656 26366
```

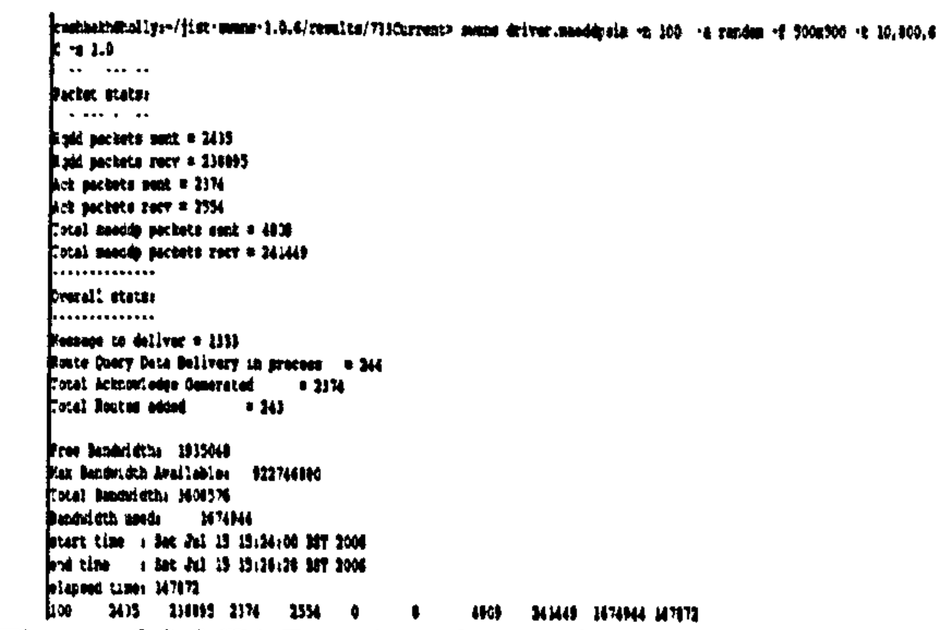
Figure 6.3.3(B) Results of experiment no 6.3.3(B)



```
-----------
Packet stats:
-----------
Rqdd packets sent = 1649
Rqdd packets recv = 92944
Ack packets sent = 1104
Ack packets recv = 1138
Total manetp packets sent = 2953
Total manetp packets recv = 94082
-----------
Overall stats:
-----------
Message to deliver = 1000
Route Query Data Delivery in process  = 185
Total Acknowledge Generated     = 1104
Total Routes added       = 181

Free Bandwidth:  1791152
Max Bandwidth Available:  922746880
Total Bandwidth: 1203072
Bandwidth used:     1411336
start time  : Sat Jul 15 15:22:09 BST 2006
end time    : Sat Jul 15 15:22:54 BST 2006
elapsed time: 45261
75     1649   92944  1104   1138   0      0      2953   94082  1411336 45261
```

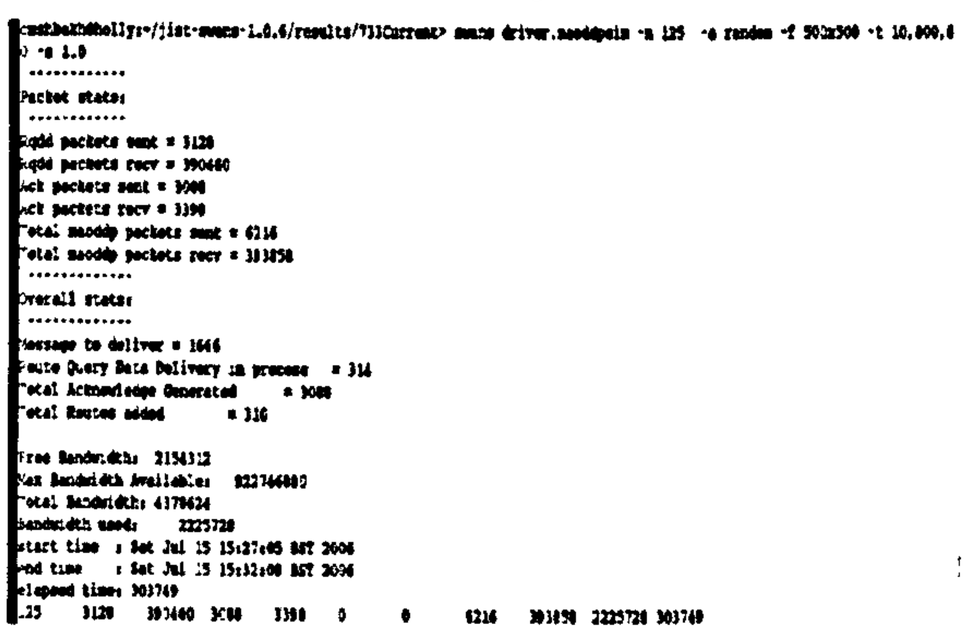
Figure 6.3.3© Results of experiment no 7.3.3©



```
-----------
Packet stats:
-----------
Rqdd packets sent = 2435
Rqdd packets recv = 138895
Ack packets sent = 2174
Ack packets recv = 2554
Total manetp packets sent = 4828
Total manetp packets recv = 141449
-----------
Overall stats:
-----------
Message to deliver = 1333
Route Query Data Delivery in process  = 244
Total Acknowledge Generated     = 2174
Total Routes added       = 243

Free Bandwidth:  1915040
Max Bandwidth Available:  922746880
Total Bandwidth: 2600576
Bandwidth used:     1674944
start time  : Sat Jul 15 15:24:00 BST 2006
end time    : Sat Jul 15 15:26:28 BST 2006
elapsed time: 147872
100    2435   138895 2174   2554   0      0      4809   141449 1674944 147872
```

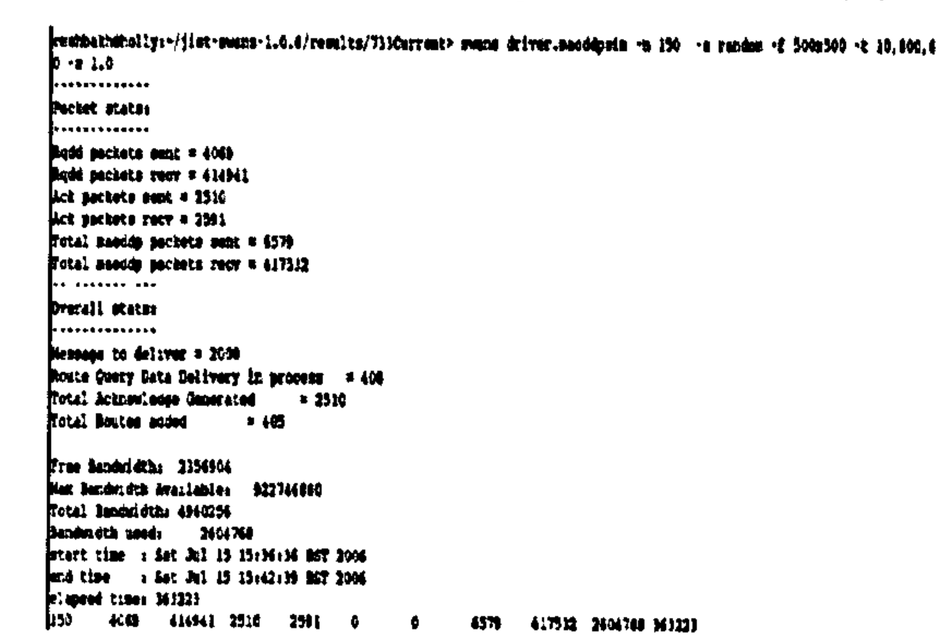
Figure 6.3.3(D) Results of experiment no 6.3.3(D)



```
-----------
Packet stats:
-----------
Rqdd packets sent = 3120
Rqdd packets recv = 390480
Ack packets sent = 3000
Ack packets recv = 3390
Total manetp packets sent = 6216
Total manetp packets recv = 393858
-----------
Overall stats:
-----------
Message to deliver = 1666
Route Query Data Delivery in process  = 314
Total Acknowledge Generated     = 3000
Total Routes added       = 316

Free Bandwidth:  2154312
Max Bandwidth Available:  922746880
Total Bandwidth: 4178624
Bandwidth used:     2225728
start time  : Sat Jul 15 15:27:05 BST 2006
end time    : Sat Jul 15 15:32:08 BST 2006
elapsed time: 303749
125    3120   390460 3000   3390   0      0      6216   393858 2225728 303749
```

Figure 6.3.3(E) Results of experiment no 6.3.3(E)



```
-----------
Packet stats:
-----------
Rqdd packets sent = 4060
Rqdd packets recv = 414941
Ack packets sent = 2510
Ack packets recv = 2981
Total manetp packets sent = 6579
Total manetp packets recv = 417312
-----------
Overall stats:
-----------
Message to deliver = 2030
Route Query Data Delivery in process  = 408
Total Acknowledge Generated     = 2510
Total Routes added       = 405

Free Bandwidth:  2356904
Max Bandwidth Available:  922746880
Total Bandwidth: 4940256
Bandwidth used:     2604768
start time  : Sat Jul 15 15:36:36 BST 2006
end time    : Sat Jul 15 15:42:39 BST 2006
elapsed time: 361221
150    4060   414941 2510   2981   0      0      6579   417312 2604768 361221
```

Figure 6.3.3(F) Results of experiment no 6.3.3(F)

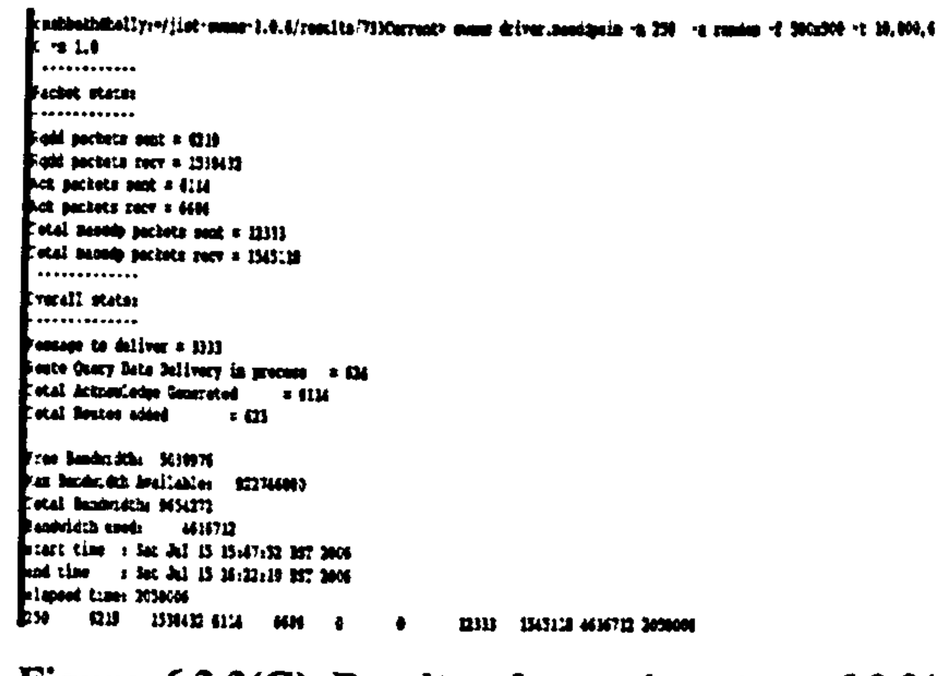


Figure 6.3.3(G) Results of experiment no 6.3.3(G)

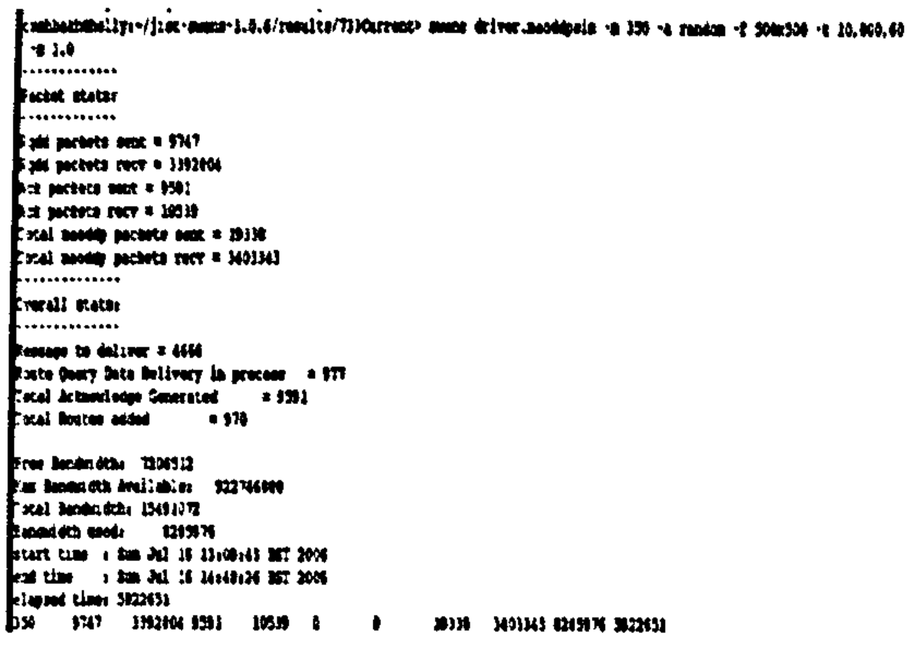


Figure 6.3.3(H) Results of experiment no 6.3.3(H)

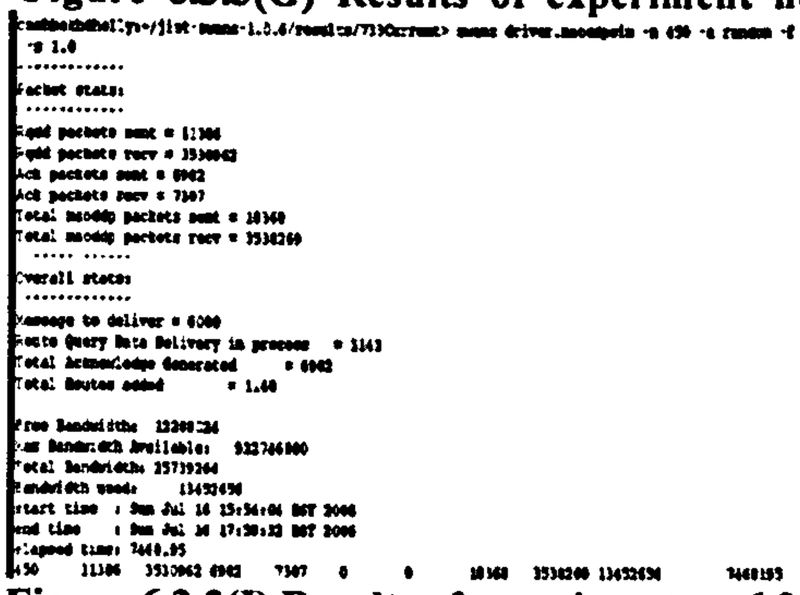


Figure 6.3.3(I) Results of experiment no 6.3.3(I)

FIGURES RESULTS OF EXPERIMENTS NO 6.3.4

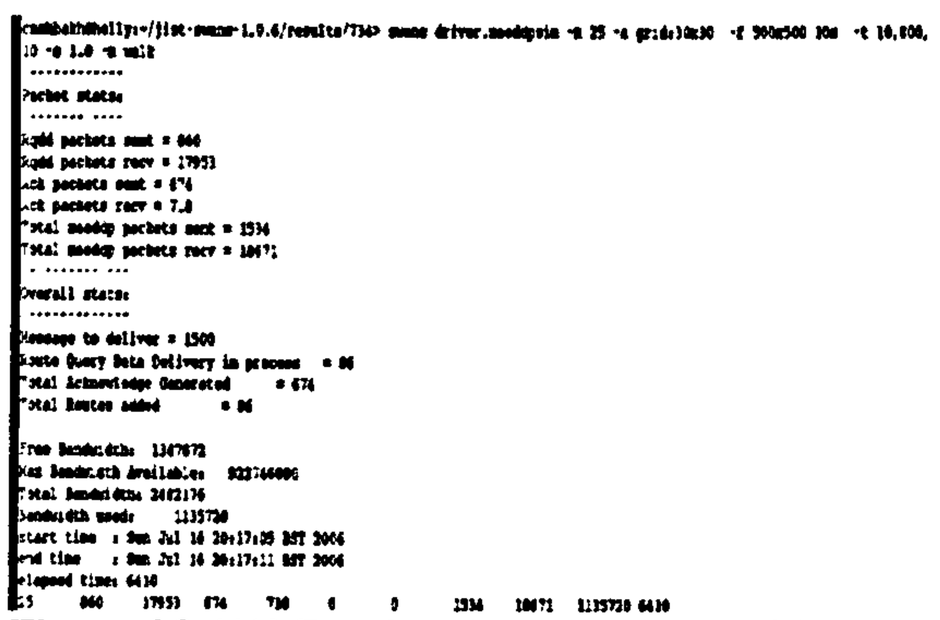


Figure 6.3.4(A) Results of experiment no 6.3.4(A)

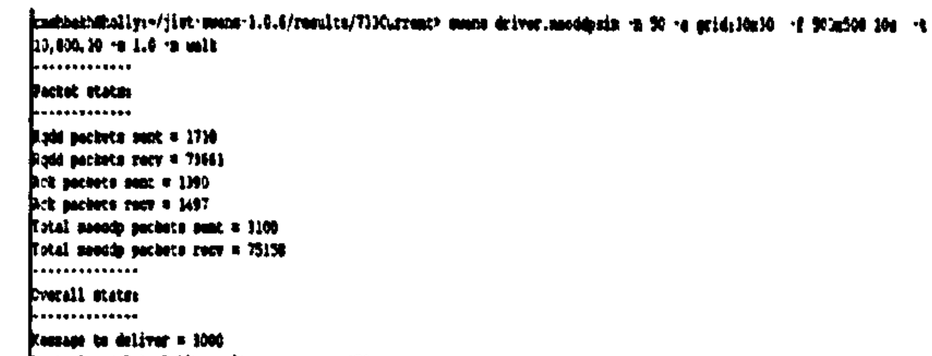


Figure 6.3.4(B) Results of experiment no 6.3.4(B)

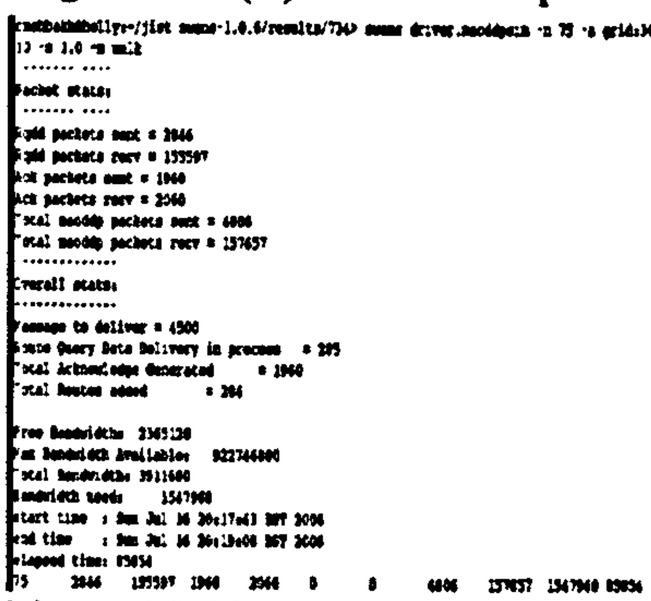


Figure 6.3.4© Results of experiment no 6.3.4©

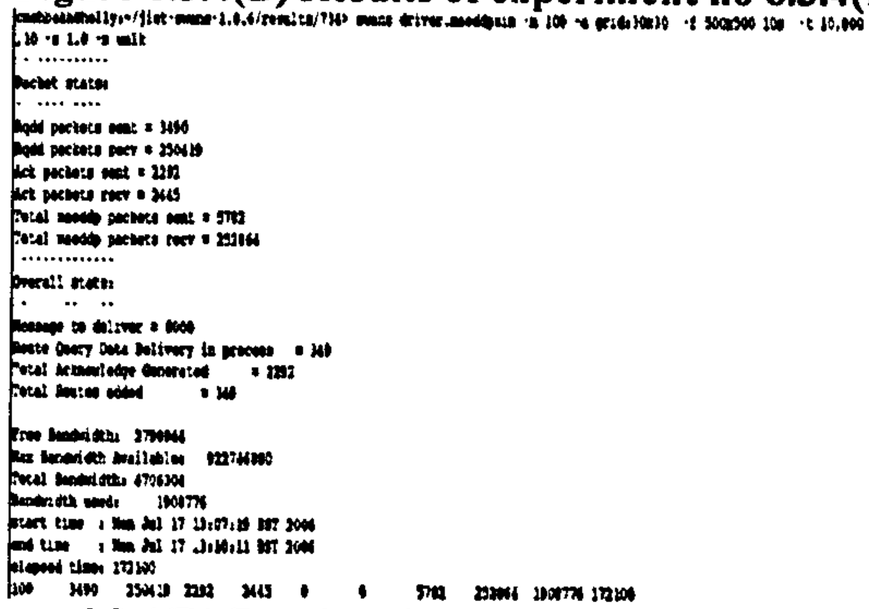


Figure 6.3.4(D) Results of experiment no 6.3.4(D)

```
[mabbott@bally:~/jist-manet-1.0.6/results/734> manet driver.manetpcs -n 125 -s gri
.10 -r 1.0 -m walk
- ----------
Packet stats:
-------------
Rqst packets sent = 4130
Rqst packets recv = 464496
Ack packets sent = 3567
Ack packets recv = 3830
Total manetp packets sent = 7906
Total manetp packets recv = 472129
-------------
Overall stats:
-------------
Message to deliver = 7500
Route Query Data Delivery in process  = 434
Total Acknowledge Generated     = 3567
Total Routes added      = 434

Free Bandwidth: 3257088
Max Bandwidth Available:  922746880
Total Bandwidth: 5570752
Bandwidth used:    2123008
start time  : Mon Jul 17 14:20:35 BST 2006
end time    : Mon Jul 17 14:27:01 BST 2006
elapsed time: 385961
125    4130   464496 3567   3830   0      0      7906   472128 2123008 385961
```

Figure 6.3.4(E) Results of experiment no 6.3.4(E)

```
[mabbott@bally:~/jist-manet-1.0.6/results/734> manet driver.manetpcs -n 250 -s gri
.10 -r 1.0 -m walk
 ----- -----
Packet stats:
-------------
Rqst packets sent = 10551
Rqst packets recv = 2287247
Ack packets sent = 8796
Ack packets recv = 9456
Total manetp packets sent = 19349
Total manetp packets recv = 2296708
 -------------
Overall stats:
-------------
Message to deliver = 15000
Route Query Data Delivery in process  = 1056
Total Acknowledge Generated     = 8796
Total Routes added      = 1056

Free Bandwidth:  7005848
Max Bandwidth Available:  922746880
Total Bandwidth: 12294192
Bandwidth used:    5291760
start time  : Mon Jul 17 15:14:47 BST 2006
end time    : Mon Jul 17 16:09:19 BST 2006
elapsed time: 3271415
250    10553  2287247 8796   9456   0      0      19349  2296795 5291760 3271415
```

Figure 6.3.4(G) Results of experiment no 6.3.4(G)

```
[mabbott@bally:~/jist-manet-1.0.6/results/734> manet driver.manetpcs -n 450 -s gri
.0 -r 1.0 -m walk
 ----------
Packet stats:
 -------------
Rqst packets sent = 20098
Rqst packets recv = 6675.66
Ack packets sent = 13947
Ack packets recv = 14741
Total manetp packets sent = 34045
Total manetp packets recv = 6689887
 ------- -----
Overall stats:
 -------------
Message to deliver = 27000
Route Query Data Delivery in process  = 2011
Total Acknowledge Generated     = 13947
Total Routes added      = 2003

Free Bandwidth:  17196456
Max Bandwidth Available:  922746880
Total Bandwidth: 30212096
Bandwidth used:    13057056
start time  : Sun Jul 16 20:19:31 BST 2006
end time    : Mon Jul 17 00:51:18 BST 2006
elapsed time: 16300617
450    20098  6675166 13947  14741  0      0      34045  6689887 13057056       16330617
```

Figure 6.3.4(I) Results of experiment no 6.3.4(I)

```
[mabbott@bally:~/jist-manet-1.0.6/results/734> manet driver.manetpcs -n 150 -s gri
.10 -r 1.0 -m walk
 -----------
Packet stats:
 -------------
Rqst packets sent = 6307
Rqst packets recv = 703933
Ack packets sent = 4455
Ack packets recv = 4709
Total manetp packets sent = 10862
Total manetp packets recv = 708642
 --- --  ----
Overall stats:
 -- ------ --
Message to deliver = 9000
Route Query Data Delivery in process  = 639
Total Acknowledge Generated     = 4455
Total Routes added      = 638

Free Bandwidth:  3756624
Max Bandwidth Available:  922746880
Total Bandwidth: 6496256
Bandwidth used:    2741040
start time  : Mon Jul 17 14:48:23 BST 2006
end time    : Mon Jul 17 14:58:37 BST 2006
elapsed time: 674209
150    6307   703933 4455   4709   0      0      10862  708642 2741040 674209
```

Figure 6.3.4(F) Results of experiment no 6.3.4(F)

```
[mabbott@bally:~/jist-manet-1.0.6/results/734> manet driver.manetpcs -n 350 -s gri
 10 -r 1.0 -m walk
 ----- -----
Packet stats:
 -------------
Rqst packets sent = 15046
Rqst packets recv = 4530816
Ack packets sent = 12411
Ack packets recv = 13177
Total manetp packets sent = 27457
Total manetp packets recv = 4544193
 -------------
Overall stats:
 ---  --------
Message to deliver = 21000
Route Query Data Delivery in process  = 1507
Total Acknowledge Generated     = 12411
Total Routes added      = 1507

Free Bandwidth:  10761000
Max Bandwidth Available:  922746880
Total Bandwidth: 19607552
Bandwidth used:    8847960
start time  : Mon Jul 17 16:46:11 BST 2006
end time    : Mon Jul 17 19:12:50 BST 2006
elapsed time: 8799934
350    15046  4530816 12411  13177  0      0      27457  4544193 8847960 8799934
```

Figure 6.3.4(H) Results of experiment no 6.3.4(H)

FIGURES RESULTS OF EXPERIMENTS NO 7.3.5

```
-o 1.0 -m waypoint
... --------
Packet stats:
------------
Rqdd packets sent = 819
Rqdd packets recv = 16887
Ack packets sent = 648
Ack packets recv = 684
Total manetdp packets sent = 1467
Total manetdp packets recv = 17571
------------
Overall stats:
------------
Message to deliver = 1500
Route Query Data Delivery in process  = 82
Total Acknowledge Generated     = 648
Total Routes added       = 82

Free Bandwidth:  1348030
Max Bandwidth Available:   922746880
Total Bandwidth: 2402176
Bandwidth used:     1135584
Start time  : Mon Jul 17 21:13:36 BST 2006
End time    : Mon Jul 17 21:13:41 BST 2006
Elapsed time: 4536
25      818     16887   648     684     0       0       1467    17571   1135584 4536
```

Figure 6.3.5(A) Results of experiment no 6.3.5(A)

```
-o 1.0 -m waypoint
------------
Packet stats:
------------
Rqdd packets sent = 1909
Rqdd packets recv = 83030
Ack packets sent = 1580
Ack packets recv = 1698
Total manetdp packets sent = 3489
Total manetdp packets recv = 81728
-------------
Overall stats:
-------------
Message to deliver = 3000
Route Query Data Delivery in process   = 191
Total Acknowledge Generated     = 1580
Total Routes added       = 191

Free Bandwidth:  1962088
Max Bandwidth Available:   922746880
Total Bandwidth: 3301376
Bandwidth used:     1340704
Start time  : Mon Jul 17 21:14:16 BST 2006
End time    : Mon Jul 17 21:14:50 BST 2006
Elapsed time: 33791
50      1909    83030   1580    1698    0       0       3489    81728   1340704 33791
```

Figure 6.3.5(B) Results of experiment no 6.3.5(B)

```
-o 1.0 -m waypoint
-----------
Packet stats:
-----------
Rqdd packets sent = 3796
Rqdd packets recv = 156913
Ack packets sent = 2928
Ack packets recv = 2063
Total manetdp packets sent = 4726
Total manetdp packets recv = 158976
------------
Overall stats:
------------
Message to deliver = 4500
Route Query Data Delivery in process   = 289
Total Acknowledge Generated     = 1928
Total Routes added       = 279

Free Bandwidth:  2110096
Max Bandwidth Available:   922746880
Total Bandwidth: 3099392
Bandwidth used:     1582712
Start time  : Mon Jul 17 21:43:32 BST 2006
End time    : Mon Jul 17 21:44:58 BST 2006
Elapsed time: 85774
75      3796    156913  1928    2063    0       0       4726    158976  1582712 85774
```

Figure 6.3.5© Results of experiment no 6.3.5©

```
-o 1.0 -m waypoint
-----------
Packet stats:
-----------
Rqdd packets sent = 7594
Rqdd packets recv = 260164
Ack packets sent = 2387
Ack packets recv = 2494
Total manetdp packets sent = 6181
Total manetdp packets recv = 262658
------------
Overall stats:
------------
Message to deliver = 6000
Route Query Data Delivery in process   = 380
Total Acknowledge Generated     = 2387
Total Routes added       = 378

Free Bandwidth:  1741408
Max Bandwidth Available:   922746880
Total Bandwidth: 4657152
Bandwidth used:     1917160
Start time  : Mon Jul 17 22:21:01 BST 2006
End time    : Mon Jul 17 22:24:35 BST 2006
Elapsed time: 174344
100     3794    260164  2387    2494    0       0       6181    262658  1917160 174344
```

Figure 6.3.5(D) Results of experiment no 6.3.5(D)

```
-o 1.0 -m waypoint
-----------
Packet stats:
-----------
Rqdd packets sent = 4485
Rqdd packets recv = 469478
Ack packets sent = 3526
Ack packets recv = 3706
Total manetdp packets sent = 8011
Total manetdp packets recv = 473264
-------------
Overall stats:
-------------
Message to deliver = 7500
Route Query Data Delivery in process   = 449
Total Acknowledge Generated     = 3526
Total Routes added       = 449

Free Bandwidth:  3477248
Max Bandwidth Available:   922746880
Total Bandwidth: 5595136
Bandwidth used:     2119304
Start time  : Mon Jul 17 22:02:51 BST 2006
End time    : Mon Jul 17 22:09:11 BST 2006
Elapsed time: 379861
125     4485    469478  3526    3706    0       0       8011    473264  2119304 379861
```

Figure 6.3.5(E) Results of experiment no 6.3.5(E)

```
-o 1.0 -m waypoint
------------
Packet stats:
------------
Rqdd packets sent = 5919
Rqdd packets recv = 649979
Ack packets sent = 4031
Ack packets recv = 4260
Total manetdp packets sent = 9950
Total manetdp packets recv = 654239
-------------
Overall stats:
-------------
Message to deliver = 9000
Route Query Data Delivery in process   = 593
Total Acknowledge Generated     = 4031
Total Routes added       = 593

Free Bandwidth:  4269872
Max Bandwidth Available:   922746880
Total Bandwidth: 6545600
Bandwidth used:     2476952
Start time  : Mon Jul 17 23:57:41 BST 2006
End time    : Mon Jul 17 23:08:02 BST 2006
Elapsed time: 620439
150     5919    649979  4031    4260    0       0       9950    654239  2476952 620439
```

Figure 6.3.5(F) Results of experiment no 6.3.5(F)

```
mathaikh@holly:~/jist-swans-1.0.6/results/735> swans driver.nacdpsim -n 250 -s grid:30x30 -f 500x500 10s -t 10,800,6
0 -s 1.0 -m waypoint
------------
Packet stats:
------------
Rqdd packets sent = 8963
Rqdd packets recv = 1940554
Ack packets sent = 7388
Ack packets recv = 7967
Total nacdp packets sent = 16351
Total nacdp packets recv = 1948521
------------
Overall stats:
------------
Message to deliver = 15000
Route Query Data Delivery in process  = 897
Total Acknowledge Generated     = 7388
Total Routes added       = 897

Free Bandwidth: 7586560
Max Bandwidth Available:  922746880
Total Bandwidth: 12357632
Bandwidth used:    4772480
start time  : Tue Jul 18 12:16:01 BST 2006
end time    : Tue Jul 18 13:03:33 BST 2006
elapsed time: 2852048
250    8963   1940554 7388   7967   0      0      16351  1948521 4772480 2852048
```

Figure 6.3.5(G) Results of experiment no 6.3.5(G)

```
mathaikh@holly:~/jist-swans-1.0.6/results/735> swans driver.nacdpsim -n 150 -s grid:30x30 -f 500x500 10s -t 10,800,6
0 -s 1.0 -m waypoint
------------
Packet stats:
------------
Rqdd packets sent = 11541
Rqdd packets recv = 4091215
Ack packets sent = 11153
Ack packets recv = 12028
Total nacdp packets sent = 24694
Total nacdp packets recv = 4103243
------------
Overall stats:
------------
Message to deliver = 21000
Route Query Data Delivery in process  = 1356
Total Acknowledge Generated     = 11153
Total Routes added       = 1356

Free Bandwidth: 11787744
Max Bandwidth Available:  922746880
Total Bandwidth: 19779584
Bandwidth used:    7991256
start time  : Tue Jul 18 17:13:55 BST 2006
end time    : Tue Jul 18 19:26:20 BST 2006
elapsed time: 7945111
150    11541  4091215 11153  12028  0      0      24694  4103243 7991256 7945111
```

Figure 6.3.5(H) Results of experiment no 6.3.5(H)

```
mathaikh@holly:~/jist-swans-1.0.6/results/735> swans driver.nacdpsim -n 450 -s grid:30x30 -f 500x500 10s -t 10,800,6
0 -s 1.0 -m waypoint
------------
Packet stats:
------------
Rqdd packets sent = 17206
Rqdd packets recv = 3004641
Ack packets sent = 12074
Ack packets recv = 12772
Total nacdp packets sent = 29278
Total nacdp packets recv = 5017413
------------
Overall stats:
------------
Message to deliver = 27000
Route Query Data Delivery in process  = 1722
Total Acknowledge Generated     = 12074
Total Routes added       = 1720

Free Bandwidth: 16344832
Max Bandwidth Available:  922746880
Total Bandwidth: 10791728
Bandwidth used:    14450312
start time  : Mon Jul 17 23:30:06 BST 2006
end time    : Tue Jul 18 03:24:40 BST 2006
elapsed time: 14073954
450    17206  5034641 12074  12772  0      0      29278  5017413 14450312       14073954
```

Figure 6.3.5(I) Results of experiment no 6.3.5(I)

FIGURES RESULTS OF EXPERIMENTS NO 6.3.6

```
mathaikh@holly:~/jist-swans-1.0.6/results/736> swans driver.nacdpsim -n 25 -s grid:30x30 -f 500x500 10s -t 10
,800,60 -s 1.0 -m teleport
------------
Packet stats:
------------
Rqdd packets sent = 820
Rqdd packets recv = 17613
Ack packets sent = 683
Ack packets recv = 727
Total nacdp packets sent = 1503
Total nacdp packets recv = 18340
------------
Overall stats:
------------
Message to deliver = 1500
Route Query Data Delivery in process  = 82
Total Acknowledge Generated     = 683
Total Routes added       = 82

Free Bandwidth: 68105560
Max Bandwidth Available:  7214202880
Total Bandwidth: 70713344
Bandwidth used:    2607784
start time  : Tue Jul 25 17:35:46 BST 2006
end time    : Tue Jul 25 17:35:48 BST 2006
elapsed time: 2600
25     820    17613  683    727    0      0      1503   18340  2607784 2600
```

Figure 6.3.6() Results of experiment no 6.3.6(A)

```
mathaikh@holly:~/jist-swans-1.0.6/results/736> swans driver.nacdpsim -n 50 -s grid:30x30 -f 500x500 10s -t 10
,800,60 -s 1.0 -m teleport
------------
Packet stats:
------------
Rqdd packets sent = 2086
Rqdd packets recv = 91019
Ack packets sent = 1771
Ack packets recv = 1893
Total nacdp packets sent = 3857
Total nacdp packets recv = 92912
------------
Overall stats:
------------
Message to deliver = 3000
Route Query Data Delivery in process  = 209
Total Acknowledge Generated     = 1771
Total Routes added       = 209

Free Bandwidth: 158556552
Max Bandwidth Available:  7214202880
Total Bandwidth: 163315712
Bandwidth used:    4759160
start time  : Tue Jul 25 17:36:32 BST 2006
end time    : Tue Jul 25 17:36:45 BST 2006
elapsed time: 12895
50     2086   91019  1771   1893   0      0      3857   92912  4759160 12895
```

Figure 6.3.6() Results of experiment no 6.3.6(B)

```
cashbakh@holly:"/jist-swans-1.0.6/results/736) swans driver.macddpsim -n 75 -s grid:30x30 -f 500x500 10s -t 10
,800,60 -s 1.0 -m teleport
-----------
Packet stats:
-----------
Rqdd packets sent = 2900
Rqdd packets recv = 162152
Ack packets sent = 2010
Ack packets recv = 2121
Total macddp packets sent = 4910
Total macddp packets recv = 164273
-----------
Overall stats:
-----------
Message to deliver = 4500
Route Query Data Delivery in process  = 290
Total Acknowledge Generated     = 2010
Total Routes added       = 290

Free Bandwidth: 372586720
Max Bandwidth Available:  7214202880
Total Bandwidth: 382274880
Bandwidth used:    9488160
start time  : Tue Jul 25 17:37:42 BST 2006
end time   : Tue Jul 25 17:39:09 BST 2006
elapsed time: 27155
75    2900   162152 2010   2121   0     0     4910   164273 9488160 27155
```

Figure 6.3.6© Results of experiment no 6.3.6©

```
cashbakh@holly:"/jist-swans-1.0.6/results/736) swans driver.macddpsim -n 100 -s grid:30x30 -f 500x500 10s -t 1
0,800,60 -s 1.0 -m teleport
-----------
Packet stats:
-----------
Rqdd packets sent = 4484
Rqdd packets recv = 313704
Ack packets sent = 2929
Ack packets recv = 3112
Total macddp packets sent = 7413
Total macddp packets recv = 316816
-----------
Overall stats:
-----------
Message to deliver = 6000
Route Query Data Delivery in process  = 449
Total Acknowledge Generated     = 2929
Total Routes added       = 449

Free Bandwidth: 172516088
Max Bandwidth Available:  7214202880
Total Bandwidth: 178454528
Bandwidth used:    5938440
start time  : Tue Jul 25 17:38:34 BST 2006
end time   : Tue Jul 25 17:39:38 BST 2006
elapsed time: 64775
100   4484   313704 2929   3112   0     0     7413   316816 5938440 64775
```

Figure 6.3.6(D) Results of experiment no 6.3.6(D)

```
cashbakh@holly:"/jist-swans-1.0.6/results/736) swans driver.macddpsim -n 125 -s grid:30x30 -f 500x500 10s -t 1
0,800,60 -s 1.0 -m teleport
-----------
Packet stats:
-----------
Rqdd packets sent = 5300
Rqdd packets recv = 565815
Ack packets sent = 4362
Ack packets recv = 4719
Total macddp packets sent = 9662
Total macddp packets recv = 570534
-----------
Overall stats:
-----------
Message to deliver = 7500
Route Query Data Delivery in process  = 530
Total Acknowledge Generated     = 4362
Total Routes added       = 530

Free Bandwidth: 680294544
Max Bandwidth Available:  7214202880
Total Bandwidth: 697237504
Bandwidth used:    16942960
start time  : Tue Jul 25 17:44:41 BST 2006
end time   : Tue Jul 25 17:47:03 BST 2006
elapsed time: 141798
125   5300   565815 4362   4719   0     0     9662   570534 16942960       141798
```

Figure 6.3.6(E) Results of experiment no 6.3.6(E)

```
cashbakh@holly:"/jist-swans-1.0.6/results/736) swans driver.macddpsim -n 150 -s grid:30x30 -f 500x500 10s -t 1
0,800,60 -s 1.0 -m teleport
-----------
Packet stats:
-----------
Rqdd packets sent = 6027
Rqdd packets recv = 687297
Ack packets sent = 4280
Ack packets recv = 4518
Total macddp packets sent = 10307
Total macddp packets recv = 691815
-----------
Overall stats:
-----------
Message to deliver = 9000
Route Query Data Delivery in process  = 605
Total Acknowledge Generated     = 4280
Total Routes added       = 605

Free Bandwidth: 1205143432
Max Bandwidth Available:  7214202880
Total Bandwidth: 1233518592
Bandwidth used:    28375160
start time  : Tue Jul 25 18:17:04 BST 2006
end time   : Tue Jul 25 18:24:58 BST 2006
elapsed time: 474023
150   6027   687297 4280   4518   0     0     10307  691815 28375160       474023
```

Figure 6.3.6(F) Results of experiment no 6.3.6(F)

```
cashbakh@holly:"/jist-swans-1.0.6/results/736) swans driver.macddpsim -n 250 -s grid:30x30 -f 500x500 10s -t 1
0,800,60 -s 1.0 -m teleport
-----------
Packet stats:
-----------
Rqdd packets sent = 9600
Rqdd packets recv = 2040626
Ack packets sent = 7851
Ack packets recv = 8410
Total macddp packets sent = 17451
Total macddp packets recv = 2049036
-----------
Overall stats:
-----------
Message to deliver = 15000
Route Query Data Delivery in process  = 960
Total Acknowledge Generated     = 7851
Total Routes added       = 960

Free Bandwidth: 2662237696
Max Bandwidth Available:  7214202880
Total Bandwidth: 2724986880
Bandwidth used:    62749784
start time  : Tue Jul 25 18:56:29 BST 2006
end time   : Tue Jul 25 19:11:29 BST 2006
elapsed time: 900131
250   9600   2040626 7851   8410   0     0     17451  2049036 62749784       900131
```

Figure 6.3.6(G) Results of experiment no 6.3.6(G)

```
cashbakh@holly:"/jist-swans-1.0.6/results/736) swans driver.macddpsim -n 350 -s grid:30x30 -f 500x500 10s -t 1
0,800,60 -s 1.0 -m teleport
-----------
Packet stats:
-----------
Rqdd packets sent = 13142
Rqdd packets recv = 3971551
Ack packets sent = 10855
Ack packets recv = 11699
Total macddp packets sent = 23997
Total macddp packets recv = 3983250
-----------
Overall stats:
-----------
Message to deliver = 21000
Route Query Data Delivery in process  = 1315
Total Acknowledge Generated     = 10855
Total Routes added       = 1315

Free Bandwidth: 2665509088
Max Bandwidth Available:  7214202880
Total Bandwidth: 2734227456
Bandwidth used:    68718368
start time  : Tue Jul 25 19:18:12 BST 2006
end time   : Tue Jul 25 20:00:59 BST 2006
elapsed time: 2567210
350   13142  3971551 10855  11699  0     0     23997  3983250 68718368       2567210
```

Figure 6.3.6(H) Results of experiment no 6.3.6(H)

```
cnethbakh@holly:"/jist-swans-1.0.6/results/736) swans driver.aacdcpsim -n 450 -a grid:30x30 -f 500x500 10s -t 1
0,800,60 -s 1.0 -m teleport
-----------
Packet stats:
-----------
Rqdd packets sent = 18409
Rqdd packets recv = 6146659
Ack packets sent = 12777
Ack packets recv = 13574
Total aacddp packets sent = 31186
Total aacddp packets recv = 6160233
-----------
Overall stats:
-----------
Message to deliver = 27000
Route Query Data Delivery in process  = 1843
Total Acknowledge Generated     = 12777
Total Routes added       = 1842

Free Bandwidth: 2680078024
Max Bandwidth Available:  7214202880
Total Bandwidth: 2756378624
Bandwidth used:    76300600
start time  : Tue Jul 25 20:07:42 BST 2006
end time   : Tue Jul 25 21:46:39 BST 2006
elapsed time: 5936947
450    18409   6146659 12777   13574   0     0     31186   6160233 76300600      5936947
```

Figure 6.3.6(I) Results of experiment no 6.3.6(I)

FIGURES RESULTS OF EXPERIMENTS NO 7.3.7

```
cnethbakh@holly:"/jist-swans-1.0.6/results/737) swans driver.aacddpsim -n 25 -a grid:5x5 -f 500x500 -t 10,600,6
0 -s 1.0 -m static -l none
-----------
Packet stats:
-----------
Rqdd packets sent = 517
Rqdd packets recv = 9860
Ack packets sent = 350
Ack packets recv = 386
Total aacddp packets sent = 867
Total aacddp packets recv = 10246
-----------
Overall stats:
-----------
Message to deliver = 250
Route Query Data Delivery in process  = 52
Total Acknowledge Generated     = 350
Total Routes added       = 52

Free Bandwidth: 102543040
Max Bandwidth Available:  7214202880
Total Bandwidth: 105775104
Bandwidth used:    3232064
start time  : Wed Jul 26 15:19:38 BST 2006
end time   : Wed Jul 26 15:19:40 BST 2006
elapsed time: 1697
25     517    9860   350    386    0     0     867    10246  3232064 1697
```

Figure 6.3.7(A) Results of experiment no 6.3.7(A)

```
cnethbakh@holly:"/jist-swans-1.0.6/results/737) swans driver.aacddpsim -n 100 -a random -f 500x500 -t 10,800,6
0 -s 1.0
-----------
Packet stats:
-----------
Rqdd packets sent = 2496
Rqdd packets recv = 242703
Ack packets sent = 2402
Ack packets recv = 2653
Total aacddp packets sent = 4898
Total aacddp packets recv = 245356
-----------
Overall stats:
-----------
Message to deliver = 1333
Route Query Data Delivery in process  = 251
Total Acknowledge Generated     = 2402
Total Routes added       = 251

Free Bandwidth: 383727512
Max Bandwidth Available:  7214202880
Total Bandwidth: 393805824
Bandwidth used:    10078312
start time  : Wed Jul 26 15:26:18 BST 2006
end time   : Wed Jul 26 15:27:01 BST 2006
elapsed time: 43257
100    2496    242703 2402    2653   0     0     4898    245356 10078312      43257
```

Figure 6.3.7© Results of experiment no 6.3.7©

```
cnethbakh@holly:"/jist-swans-1.0.6/results/737) swans driver.aacddpsim -n 50 -a grid:10x10 -f 500x500 -t 10,800
,60 -s 1.0 -m static -l none
-----------
Packet stats:
-----------
Rqdd packets sent = 1256
Rqdd packets recv = 62800
Ack packets sent = 1256
Ack packets recv = 1382
Total aacddp packets sent = 2512
Total aacddp packets recv = 64182
-----------
Overall stats:
-----------
Message to deliver = 666
Route Query Data Delivery in process  = 126
Total Acknowledge Generated     = 1256
Total Routes added       = 126

Free Bandwidth: 150975472
Max Bandwidth Available:  7214202880
Total Bandwidth: 155516928
Bandwidth used:    4541456
start time  : Wed Jul 26 15:20:52 BST 2006
end time   : Wed Jul 26 15:20:59 BST 2006
elapsed time: 7718
50     1256   62800  1256   1382   0     0     2512    64182  4541456 7718
```

Figure 6.3.7(B) Results of experiment no 6.3.7(B)

```
cnethbakh@holly:"/jist-swans-1.0.6/results/737) swans driver.aacddpsim -n 250 -a grid:30x30 -f 500x500 10s -s
1.0 -m walk
-----------
Packet stats:
-----------
Rqdd packets sent = 10887
Rqdd packets recv = 2173822
Ack packets sent = 8219
Ack packets recv = 9272
Total aacddp packets sent = 19106
Total aacddp packets recv = 2183094
-----------
Overall stats:
-----------
Message to deliver = 15000
Route Query Data Delivery in process  = 1090
Total Acknowledge Generated     = 8219
Total Routes added       = 1090

Free Bandwidth: 2669826200
Max Bandwidth Available:  7214202880
Total Bandwidth: 2732785664
Bandwidth used:    62959464
start time  : Fri Aug 04 17:31:38 BST 2006
end time   : Fri Aug 04 17:47:22 BST 2006
elapsed time: 943271
250    10887  2173822 8219   9272   0     0     19106   2183094 62959464      943271
```

Figure 6.3.7(D) Results of experiment no 6.3.7(D)

```
cashbaki@holly:"/jist-swans-1.0.6/results/737) swans driver.maoddpsim -n 350 -s grid:30x30  -f 500x500 10s  -t
 10,800,60 -m 1.0 -m waypoint
-----------

Packet stats:
-----------

Rapid packets sent = 14015
Rapid packets recv = 4135166
Ack packets sent = 11024
Ack packets recv = 12375
Total maoddp packets sent = 25039
Total maoddp packets recv = 4147541
-----------

Overall stats:
-----------

Message to deliver = 21000
Route Query Data Delivery in process  = 1403
Total Acknowledge Generated     = 11024
Total Routes added       = 1402

Free Bandwidth:  2682412184
Max Bandwidth Available:   7214202880
Total Bandwidth: 2749956096
Bandwidth used:     67543912
start time  : Wed Jul 26 16:03:30 BST 2006
end time    : Wed Jul 26 16:43:03 BST 2006
elapsed time: 2373264
350     14015   4135166 11024   12375   0       0       25039   4147541 67543912        2373264
```

Figure 6.3.7(E) Results of experiment no 6.3.7(E)

```
cashbaki@holly:"/jist-swans-1.0.6/results/737) swans driver.maoddpsim -n 450 -s grid:30x30  -f 500x500 10s  -t
 10,800,60 -m 1.0 -m teleport
-----------

Packet stats:
-----------

Rapid packets sent = 17789
Rapid packets recv = 5742325
Ack packets sent = 11265
Ack packets recv = 13040
Total maoddp packets sent = 29054
Total maoddp packets recv = 5755365
-----------

Overall stats:
-----------

Message to deliver = 27000
Route Query Data Delivery in process  = 1780
Total Acknowledge Generated     = 11265
Total Routes added       = 1767

Free Bandwidth:  2690563352
Max Bandwidth Available:   7214202880
Total Bandwidth: 2769879040
Bandwidth used:     79315688
start time  : Wed Jul 26 19:30:25 BST 2006
end time    : Wed Jul 26 20:41:18 BST 2006
elapsed time: 4252444
450     17789   5742325 11265   13040   0       0       29054   5755365 79315688        4252444
```

Figure 6.3.7(F) Results of experiment no 6.3.7(F)

Figure results experiment 7.2.1 to 7.2.3

```
cmahbaki@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aacddpsim -n 100 -s grid:30x30 -f 500x500 -t 10,80
0,60 -s 1.0 -m static -l none
------------
Packet stats:
------------
Rqdd packets sent = 2723
Rqdd packets recv = 264388
Ack packets sent = 2620
Ack packets recv = 2693
Total aacddp packets sent = 5343
Total aacddp packets recv = 267281
------------
Overall stats:
------------
Message to deliver = 1333
Route Query Data Delivery in process  = 273
Total Acknowledge Generated     = 2620
Total Routes added       = 273

Free Memory:  505393656
Max Memory Available:   7214202880
Total Memory: 522125312
Memory used:     16731656
start time  : Mon Aug 14 15:22:25 BST 2006
end time    : Mon Aug 14 15:23:09 BST 2006
elapsed time: 44124
100    2723   264388 2620   2693   0      0      5343   267281 16731656       44124
```

Figure Results 7.2.1A

```
cmahbaki@holly.~/jist-swans-1.0.6/results/8.1) swans driver.aodvsim -n 100 -s grid.30x30 -f 500x500 -t 10,800,
60 -s 1.0 -m static -l none
------------
Packet stats:
------------
Rreq packets sent = 272
Rreq packets recv = 26928
Rrep packets sent = 272
Rrep packets recv = 272
Rerr packets sent = 828
Rerr packets recv = 84
Hello packets sent = 1263
Hello packets recv = 65
Total aodv packets sent = 2635
Total aodv packets recv = 27349
Non-hello packets sent = 1372
Non-hello packets recv = 27284
------------
Overall stats:
------------
Messages to deliver = 1333
Route requests      = 272
Route replies       = 272
Routes added        = 272

Freemem:  180334792
maxmem:   7214202880
totalmem: 186187776
Used      5853024
start time  : Mon Aug 14 15:01:26 BST 2006
end time    : Mon Aug 14 15:01:33 BST 2006
Elapsed time: 6080
100    272    26928  272    272    828    84     1263   65     2635   27349  1372   27284  946302
0      6080
```

Figure Results 7.2.1B

```
cmahbaki@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aacddpsim -n 250 -s grid:30x30 -f 500x500 -t 10,800,60 -s 1.0
 -m static -l none
------------
Packet stats:
------------
Rqdd packets sent = 10550
Rqdd packets recv = 2076886
Ack packets sent = 5160
Ack packets recv = 5634
Total aacddp packets sent = 15710
Total aacddp packets recv = 2082520
------------
Overall stats:
------------
Message to deliver = 3333
Route Query Data Delivery in process  = 1061
Total Acknowledge Generated     = 5160
Total Routes added       = 525

Free Memory:  2658652272
Max Memory Available:   7214202880
Total Memory: 2719219712
Memory used:     60567440
start time  : Mon Aug 14 22:05:54 BST 2006
end time    : Mon Aug 14 22:18:32 BST 2006
elapsed time: 757935
250    10550  2076886 5160   5634   0      0      15710  2082520 60567440      757935
```

Figure Results 7.2.2A

```
cmahbaki@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aodvsim -n 250 -s grid:30x30 -f 500x500 -t 10,800,
60 -s 1.0 -m static -l none
------------
Packet stats:
------------
Rreq packets sent = 640
Rreq packets recv = 159360
Rrep packets sent = 640
Rrep packets recv = 640
Rerr packets sent = 1888
Rerr packets recv = 86
Hello packets sent = 2964
Hello packets recv = 72
Total aodv packets sent = 6132
Total aodv packets recv = 160158
Non-hello packets sent = 3168
Non-hello packets recv = 160086
------------
Overall stats:
------------
Messages to deliver = 3333
Route requests      = 640
Route replies       = 640
Routes added        = 640

Freemem:  786307768
maxmem:   7214202880
totalmem: 813629440
Used:     27321672
start time  : Mon Aug 14 15:24:23 BST 2006
end time    : Mon Aug 14 15:25:23 BST 2006
Elapsed time: 60262
250    640    159360 640    640    1888   86     2964   72     6132   160158 3168   160086 273216
72     60262
```

Figure Results 7.2.2B

```
cmahbaki@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aacddpsim -n 500 -s grid:30x30 -f 500x500 -t 10,80
0,60 -s 1.0 -m static
------------
Packet stats:
------------
Rqdd packets sent = 12898
Rqdd packets recv = 6056608
Ack packets sent = 11774
Ack packets recv = 12498
Total aacddp packets sent = 24672
Total aacddp packets recv = 6069106
------------
Overall stats:
------------
Message to deliver = 6666
Route Query Data Delivery in process  = 1293
Total Acknowledge Generated     = 11774
Total Routes added       = 1293

Free Memory:  107423704
Max Memory Available:   7214202880
Total Memory: 136511488
Memory used:     29087784
start time  : Tue Aug 15 20:02:49 BST 2006
end time    : Tue Aug 15 21:15:31 BST 2006
elapsed time: 4362154
500    12898  6056608 11774  12498  0      0      24672  6069106 29087784      4362154
```

Figure Results 7.2.3A

```
cmahbaki@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aodvsim -n 500 -s grid.30x30 -f 500x500 -t 10,800,
60 -s 1.0 -m static -l none
------------
Packet stats:
------------
Rreq packets sent = 1383
Rreq packets recv = 690117
Rrep packets sent = 1383
Rrep packets recv = 1383
Rerr packets sent = 4349
Rerr packets recv = 89
Hello packets sent = 6171
Hello packets recv = 74
Total aodv packets sent = 13286
Total aodv packets recv = 691663
Non-hello packets sent = 7115
Non-hello packets recv = 691589
------------
Overall stats:
------------
Messages to deliver = 6666
Route requests      = 1383
Route replies       = 1383
Routes added        = 1383

Freemem:  2659948328
maxmem:   7214202880
totalmem: 2736586752
Used:     80638424
start time  : Mon Aug 14 15:47:17 BST 2006
end time    : Mon Aug 14 15:55:16 BST 2006
Elapsed time: 478648
500    1383   690117 1383   1383   4349   89     6171   74     13286  691663 7115   691589 806384
24     478648
```

Figure Results 7.2.3B

```
[cashbaik@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aaodpsim -n 100 -s random -f 500x500 -t 10,800,6
0 -s 1.0
------------
Packet stats:
------------
Rqst packets sent = 2671
Rqst packets recv = 261986
Ack packets sent = 2591
Ack packets recv = 2785
Total aaodp packets sent = 5262
Total aaodp packets recv = 264771
------------
Overall stats:
------------
Message to deliver = 1333
Route Query Data Delivery in process  = 258
Total Acknowledge Generated     = 2591
Total Routes added       = 258

Free Memory:  396444312
Max Memory Available:   7214202880
Total Memory: 406781952
Memory used:     10337640
start time  : Wed Aug 16 12:29:31 BST 2006
end time    : Wed Aug 16 12:30:19 BST 2006
elapsed time: 48491
100    2671   261986 2591   2785   0     0     5262   264771 10337640       48491
```

Figure Results 7.2.4A

```
[cashbaik@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aaodpsim -n 100 -s random -f 500x500 -t 10,800,6
0 -s 1.0
------------
Packet stats:
------------
Rqst packets sent = 248
Rqst packets recv = 24536
Rrep packets sent = 248
Rrep packets recv = 248
Rerr packets sent = 698
Rerr packets recv = 60
Hello packets sent = 1176
Hello packets recv = 60
Total aodv packets sent = 2370
Total aodv packets recv = 24904
Non-hello packets sent = 1194
Non-hello packets recv = 24844
------------
Overall stats:
------------
Messages to deliver = 1333
Route requests      = 248
Route replies       = 248
Routes added        = 248

Freemem:  234177336
maxmem:   7214202880
totalmem: 241172480
used:     6995144
start time  : Mon Aug 14 18:18:30 BST 2006
end time    : Mon Aug 14 18:18:36 BST 2006
elapsed time: 6037
100    248    24536  248    248    698    60     1176   60     2370   24904  1194   24844  699514
0      6037
```

Figure Results 7.2. 4B

```
[cashbaik@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aaodpsim -n 250 -s random -f 500x500 -t 10,800,6
0 -s 1.0
------------
Packet stats:
------------
Rqst packets sent = 6511
Rqst packets recv = 1609047
Ack packets sent = 6410
Ack packets recv = 7006
Total aaodp packets sent = 12921
Total aaodp packets recv = 1616053
------------
Overall stats:
------------
Message to deliver = 3333
Route Query Data Delivery in process  = 653
Total Acknowledge Generated     = 6410
Total Routes added       = 653

Free Memory:  2657999504
Max Memory Available:   7214202880
Total Memory: 2719350784
Memory used:     61351280
start time  : Wed Aug 16 12:33:47 BST 2006
end time    : Wed Aug 16 12:44:02 BST 2006
elapsed time: 615488
250    6511   1609047 6410   7006   0     0     12921  1616053 61351280      615488
```

Figure Results 7.2.5A

```
[cashbaik@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aaodpsim -n 250 -s random -f 500x500 -t 10,800,6
0 -s 1.0
------------
Packet stats:
------------
Rqst packets sent = 697
Rqst packets recv = 173541
Rrep packets sent = 697
Rrep packets recv = 697
Rerr packets sent = 2275
Rerr packets recv = 127
Hello packets sent = 3136
Hello packets recv = 67
Total aodv packets sent = 6805
Total aodv packets recv = 174432
Non-hello packets sent = 3669
Non-hello packets recv = 174365
------------
Overall stats:
------------
Messages to deliver = 3333
Route requests      = 697
Route replies       = 697
Routes added        = 697

Freemem:  1159993456
maxmem:   7214202880
totalmem: 1191116800
used:     31123344
start time  : Mon Aug 14 18:26:50 BST 2006
end time    : Mon Aug 14 18:27:57 BST 2006
elapsed time: 67015
250    697    173541 697    697    2275   127    3136   67     6805   174432 3669   174365 311233
44     67015
```

Figure Results 7.2. 5B

```
[cashbaik@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aaodpsim -n 500 -s random -f 500x500 -t 10,800,6
0 -s 1.0
------------
Packet stats:
------------
Rqst packets sent = 13727
Rqst packets recv = 6465790
Ack packets sent = 12531
Ack packets recv = 13292
Total aaodp packets sent = 26258
Total aaodp packets recv = 6479082
------------
Overall stats:
------------
Message to deliver = 6666
Route Query Data Delivery in process  = 1376
Total Acknowledge Generated     = 12531
Total Routes added       = 1376

Free Memory:  159163904
Max Memory Available:   7214202880
Total Memory: 189595648
Memory used:     30431744
start time  : Wed Aug 16 16:58:38 BST 2006
end time    : Wed Aug 16 18:23:30 BST 2006
elapsed time: 5091585
500    13727  6465790 12531  13292  0     0     26258  6479082 30431744      5091585
```

Figure Results 7.2.6A

```
[cashbaik@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aaodpsim -n 500 -s random -f 500x500 -t 10,800,6
0 -s 1.0
------------
Packet stats:
------------
Rqst packets sent = 25824
Rqst packets recv = 786009
Rrep packets sent = 1418
Rrep packets recv = 1374
Rerr packets sent = 4585
Rerr packets recv = 131
Hello packets sent = 3137
Hello packets recv = 45
Total aodv packets sent = 34964
Total aodv packets recv = 787559
Non-hello packets sent = 31827
Non-hello packets recv = 787514
------------
Overall stats:
------------
Messages to deliver = 6666
Route requests      = 1372
Route replies       = 1413
Routes added        = 1369

Freemem:  2560627888
maxmem:   7214202880
totalmem: 2741567488
used:     80939600
start time  : Mon Aug 14 19:47:07 BST 2006
end time    : Mon Aug 14 19:55:34 BST 2006
elapsed time: 506416
500    25824  786009 1418   1374   4585   131    3137   45     34964  787559 31827  787514 809398
00     506416
```

Figure Results 7.2. 6B

```
[cashbaik@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aaodpsim -n 100 -s grid:30x30 -f 900x500 10 -t
 10,800,60 -m walk -s 1.0
------------
Packet stats:
------------
Rqst packets sent = 3960
Rqst packets recv = 279021
Ack packets sent = 2562
Ack packets recv = 2706
Total aaodp packets sent = 6522
Total aaodp packets recv = 281727
------------
Overall stats:
------------
Message to deliver = 6000
Route Query Data Delivery in process  = 396
Total Acknowledge Generated     = 2562
Total Routes added       = 396

Free Memory:  311962824
Max Memory Available:   7214202880
Total Memory: 320798720
Memory used:     8835896
start time  : Wed Aug 16 18:26:58 BST 2006
end time    : Wed Aug 16 18:27:53 BST 2006
elapsed time: 55370
100    3960   279021 2562   2706   0     0     6522   281727 8835896 55370
```

Figure Results 7.2.7A

```
[cashbaik@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aaodpsim -n 100 -s grid:30x30 -f 900x500 10s -t 1
0,800,60 -m 1.0 -m walk
------------
Packet stats:
------------
Rqst packets sent = 5562
Rqst packets recv = 58528
Rrep packets sent = 441
Rrep packets recv = 393
Rerr packets sent = 1005
Rerr packets recv = 161
Hello packets sent = 1622
Hello packets recv = 193
Total aodv packets sent = 8630
Total aodv packets recv = 59279
Non-hello packets sent = 7008
Non-hello packets recv = 59082
------------
Overall stats:
------------
Messages to deliver = 6000
Route requests      = 399
Route replies       = 441
Routes added        = 393

Freemem:  164819712
maxmem:   7214202880
totalmem: 170524672
used:     5704960
start time  : Mon Aug 14 21:44:29 BST 2006
end time    : Mon Aug 14 21:44:41 BST 2006
elapsed time: 12499
100    5562   58528  441    393    1005   161    1622   193    8630   59279  7008   59082  870496
0      12499
```

Figure Results 7.2. 7B

```
csmbakr@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aaodpsim -n 250  -s grid:30x30  -f 500x500 10  -t
 10,800,60 -m walk -s 1.0
------------
Packet stats:
------------
Rodd packets sent = 10396
Rodd packets recv = 2247260
Ack packets sent = 8581
Ack packets recv = 9245
Total aaodp packets sent = 18977
Total aaodp packets recv = 2256505
------------
Overall stats:
------------
Message to deliver = 15000
Route Query Data Delivery in process  = 1041
Total Acknowledge Generated     = 8581
Total Routes added       = 1041

Free Memory:  1984611064
Max Memory Available:   7214202880
Total Memory: 2033254400
Memory used:     48643336
start time  : Wed Aug 16 20:17:36 BST 2006
end time    : Wed Aug 16 20:33:47 BST 2006
elapsed time: 970816
250     10396   2247260 8581    9245    0       0       18977   2256505 48643336        970816
```

Figure Results 7.2.8A

```
csmbakr@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aodvsim -n 250  -s grid:30x30  -f 500x500 10  -t 1
0,800,60 -m 1.0  -s walk
------------
Packet stats:
------------
Rreq packets sent = 8638
Rreq packets recv = 257189
Rrep packets sent = 941
Rrep packets recv = 910
Rerr packets sent = 2854
Rerr packets recv = 203
Hello packets sent = 2835
Hello packets recv = 214
Total aodv packets sent = 14968
Total aodv packets recv = 258516
Non-hello packets sent = 12133
Non-hello packets recv = 258302
------------
Overall stats:
------------
Messages to deliver = 15000
Route requests      = 914
Route replies       = 941
Routes added        = 910

freemem:  419091744
maxmem:   7214202880
totalmem: 435093504
used:     16001760
start time  : Thu Aug 17 19:55:09 BST 2006
end time    . Thu Aug 17 19.57:40 BST 2006
elapsed time: 150641
250     8638    257189  941     910     2854    203     2835    214     14968   258516  12133   258302  160017
60      150641
```

Figure Results 7.2. 8B

```
csmbakr@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aaodpsim -n 500  -s grid:30x30  -f 500x500 10  -t
 10,800,60 -m walk -s 1.0
------------
Packet stats:
------------
Rodd packets sent = 20493
Rodd packets recv = 7145196
Ack packets sent = 13452
Ack packets recv = 14187
Total aaodp packets sent = 33945
Total aaodp packets recv = 7159383
------------
Overall stats:
------------
Message to deliver = 30000
Route Query Data Delivery in process  = 2052
Total Acknowledge Generated     = 13452
Total Routes added       = 2047

Free Memory:  2686200232
Max Memory Available:   7214202880
Total Memory: 2769682432
Memory used:     83482200
start time  : Wed Aug 16 18.28.31 BST 2006
end time    : Wed Aug 16 20:09:55 BST 2006
elapsed time: 6083574
500     20493   7145196 13452   14187   0       0       33945   7159383 83482200        6083574
```

Figure Results 7.2.9A

```
csmbakr@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aaodpsim -n 500  -s grid:30x30  -f 500x500 10  -t 1
0,800,60 -m 1.0  -s walk
------------
Packet stats:
------------
Rreq packets sent = 17302
Rreq packets recv = 967519
Rrep packets sent = 1846
Rrep packets recv = 1814
Rerr packets sent = 5285
Rerr packets recv = 362
Hello packets sent = 6087
Hello packets recv = 217
Total aodv packets sent = 30520
Total aodv packets recv = 969912
Non-hello packets sent = 24433
Non-hello packets recv = 969695
------------
Overall stats:
------------
Messages to deliver = 30000
Route requests      = 1819
Route replies       = 1846
Routes added        = 1814

freemem:  2668418384
maxmem:   7214202880
totalmem: 2748776448
used:     80358064
start time  : Thu Aug 17 20 09:30 BST 2006
end time    : Thu Aug 17 20:22:31 BST 2006
elapsed time: 773220
500     17302   967519  1846    1814    5285    362     6087    217     30520   969912  24433   969695  803580
64      773220
```

Figure Results 7.2. 9B

```
csmbakr@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aaodpsim -n 100  -s grid:30x30  -f 500x500 10  -t
 10,800,60 -m waypoint -s 1.0
------------
Packet stats:
------------
Rodd packets sent = 3730
Rodd packets recv = 265858
Ack packets sent = 2482
Ack packets recv = 2614
Total aaodp packets sent = 6212
Total aaodp packets recv = 268472
------------
Overall stats:
------------
Message to deliver = 6000
Route Query Data Delivery in process  = 373
Total Acknowledge Generated     = 2482
Total Routes added       = 373

Free Memory:  161372632
Max Memory Available:   7214202880
Total Memory: 166920192
Memory used:     5547560
start time  : Wed Aug 16 20:46:43 BST 2006
end time    : Wed Aug 16 20:47:38 BST 2006
elapsed time: 54701
100     3730    265858  2482    2614    0       0       6212    268472  5547560 54701
```

Figure Results 7.2.10A

```
csmbakr@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aodvsim -n 100  -s grid:30x30  -f 500x500 10  -t 1
0,800,60 -m waypoint -s 1.0
------------
Packet stats:
------------
Rreq packets sent = 6085
Rreq packets recv = 53445
Rrep packets sent = 369
Rrep packets recv = 311
Rerr packets sent = 770
Rerr packets recv = 113
Hello packets sent = 1114
Hello packets recv = 169
Total aodv packets sent = 8338
Total aodv packets recv = 54038
Non-hello packets sent = 7224
Non-hello packets recv = 53869
------------
Overall stats:
------------
Messages to deliver = 6000
Route requests      = 320
Route replies       = 369
Routes added        = 311

freemem:  175839624
maxmem:   7214202880
totalmem: 186974208
used:     10134584
start time  : Wed Aug 16 20.45:10 BST 2006
end time    . Wed Aug 16 20.46.22 BST 2006
elapsed time: 11314
100     6085    53445   369     311     770     113     1114    169     8338    54038   7224    53869   101345
64      11314
```

Figure Results 7.2. 10B

```
cmehbakh@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aacdpsim -n 250 -s grid:30x30 -f 500x500 10 -t
 10,800,60 -m waypoint -s 1.0
------------
Packet stats:
------------
Rqdd packets sent = 9639
Rqdd packets recv = 2073531
Ack packets sent = 7875
Ack packets recv = 8498
Total aacddp packets sent = 17514
Total aacddp packets recv = 2082029
------------
Overall stats:
------------
Message to deliver = 15000
Route Query Data Delivery in process   = 965
Total Acknowledge Generated      = 7875
Total Routes added        = 965

Free Memory:  2434922496
Max Memory Available:   7214202880
Total Memory: 2492661760
Memory used:     57739264
start time  : Wed Aug 16 20:50:22 BST 2006
end time    : Wed Aug 16 21:05:24 BST 2006
elapsed time: 902184
250     9639    2073531 7875    8498    0       0       17514   2082029 57739264        902184
```

Figure Results 7.2.11A

```
cmehbakh@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aacdpsim -n 500 -s grid:30x30 -f 500x500 10 -t
 10,800,60 -m waypoint -s 1.0
------------
Packet stats:
------------
Rqdd packets sent = 20799
Rqdd packets recv = 7311669
Ack packets sent = 13632
Ack packets recv = 14388
Total aacddp packets sent = 34431
Total aacddp packets recv = 7326057
------------
Overall stats:
------------
Message to deliver = 30000
Route Query Data Delivery in process   = 2083
Total Acknowledge Generated      = 13632
Total Routes added        = 2078

Free Memory:  2686030904
Max Memory Available:   7214202880
Total Memory: 2768830464
Memory used:     82799560
start time  : Wed Aug 16 21:45:19 BST 2006
end time    : Wed Aug 16 23:29:22 BST 2006
elapsed time: 6243177
500     20799   7311669 13632   14388   0       0       34431   7326057 82799560        6243177
```

Figure Results 7.2.12A

```
cmehbakh@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aacdpsim -n 100 -s grid:30x30 -f 500x500 10 -t
 10,800,60 -m teleport -s 1.0
------------
Packet stats:
------------
Rqdd packets sent = 4000
Rqdd packets recv = 279660
Ack packets sent = 2569
Ack packets recv = 2726
Total aacddp packets sent = 6569
Total aacddp packets recv = 282386
------------
Overall stats:
------------
Message to deliver = 6000
Route Query Data Delivery in process   = 400
Total Acknowledge Generated      = 2569
Total Routes added        = 398

Free Memory:  20063560
Max Memory Available:   7214202880
Total Memory: 207028224
Memory used:     6392664
start time  : Thu Aug 17 11:38:30 BST 2006
end time    : Thu Aug 17 11:39:27 BST 2006
elapsed time: 57227
100     4000    279660  2569    2726    0       0       6569    282386  6392664 57227
```

Figure Results 7.2.13A

```
cmehbakh@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aodvsim -n 250 -s grid:30x30 -f 500x500 10 -t 1
0,800,60 -m waypoint -s 1.0
------------
Packet stats:
------------
Rreq packets sent = 23829
Rreq packets recv = 314902
Rrep packets sent = 990
Rrep packets recv = 899
Rerr packets sent = 2409
Rerr packets recv = 263
Hello packets sent = 2592
Hello packets recv = 204
Total aodv packets sent = 29820
Total aodv packets recv = 316268
Non-hello packets sent = 27228
Non-hello packets recv = 316064
------------
Overall stats:
------------
Messages to deliver = 15000
Route requests      = 910
Route replies       = 990
Routes added        = 900

Freemem:  574434560
maxmem:   7214202880
totalmem: 993959552
used:     19124992
start time  : Wed Aug 16 21:16:30 BST 2006
end time    : Wed Aug 16 21:18:48 BST 2006
elapsed time: 129661
250     23829   314902  990     899     2409    263     2592    204     29820   316268  27228   316064  191249
32      129661
```

Figure Results 7.2. 11B

```
cmehbakh@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aodvsim -n 500 -s grid:30x30 -f 500x500 10 -t 1
0,800,60 -m waypoint -s 1.0
------------
Packet stats:
------------
Rreq packets sent = 17946
Rreq packets recv = 1042410
Rrep packets sent = 1994
Rrep packets recv = 1963
Rerr packets sent = 5724
Rerr packets recv = 365
Hello packets sent = 6392
Hello packets recv = 276
Total aodv packets sent = 32056
Total aodv packets recv = 1045014
Non-hello packets sent = 25664
Non-hello packets recv = 1044738
------------
Overall stats:
------------
Messages to deliver = 30000
Route requests      = 1967
Route replies       = 1994
Routes added        = 1963

Freemem:  2672532360
maxmem    7214202880
totalmem: 2755067904
used:     825535544
start time  : Thu Aug 17 11:07:20 BST 2006
end time    : Thu Aug 17 11:21:59 BST 2006
elapsed time: 879116
500     17946   1042410 1994    1963    5724    365     6392    276     32056   1045014 25664   1044738 825535
44      879116
```

Figure Results 7.2. 12B

```
cmehbakh@holly:~/jist-swans-1.0.6/results/8.1) swans driver.aodvsim -n 100 -s grid:30x30 -f 500x500 10 -t 1
0,800,60 -m teleport -s 1.0
------------
Packet stats:
------------
Rreq packets sent = 7695
Rreq packets recv = 62953
Rrep packets sent = 416
Rrep packets recv = 341
Rerr packets sent = 887
Rerr packets recv = 160
Hello packets sent = 1293
Hello packets recv = 159
Total aodv packets sent = 10291
Total aodv packets recv = 63213
Non-hello packets sent = 8998
Non-hello packets recv = 63054
------------
Overall stats:
------------
Messages to deliver = 6000
Route requests      = 350
Route replies       = 416
Routes added        = 341

Freemem:  197586664
maxmem:   7214202880
totalmem: 204013568
used:     6426904
start time  : Thu Aug 17 15:28:33 BST 2006
end time    : Thu Aug 17 15:28:48 BST 2006
elapsed time: 14657
100     7695    62953   416     341     887     160     1293    159     10291   63213   8998    63054   642690
4       14657
```

Figure Results 7.2. 13B

```
[redacted]:~/jist-swans-1.0.6/results/8.1) swans driver.aodvsim -n 250  -s grid:30x30  -f 900x900 10  -t 1
0,800,60 -m teleport -s 1.0
-----------
Packet stats:
-----------
Rreq packets sent = 22150
Rreq packets recv = 322647
Rrep packets sent = 1042
Rrep packets recv = 958
Rerr packets sent = 2812
Rerr packets recv = 281
Hello packets sent = 2767
Hello packets recv = 189
Total aodv packets sent = 28771
Total aodv packets recv = 324075
Non-hello packets sent = 26004
Non-hello packets recv = 323886
-----------
Overall stats:
-----------
Messages to deliver = 15000
Route requests      = 969
Route replies       = 1042
Routes added        = 958

Freemem:  514543040
Maxmem:   7214202880
Totalmem: 532480000
Used:     17936960
start time    Thu Aug 17 16:42:08 BST 2006
end time      Thu Aug 17 16:44:18 BST 2006
Elapsed time: 130950
250     22150  322647 1042   958    2812   281    2767   189    28771  324075 26004  323886 179360
50      130950
```

Figure Results 7.2.14A

Figure Results 7.2. 14B

```
[redacted]:~/jist-swans-1.0.6/results/8.1) swans driver.aodvsim -n 300  -s grid:30x30  -f 900x900 10  -t 1
0,800,60 -m teleport -s 1.0
-----------
Packet stats:
-----------
Rreq packets sent = 19522
Rreq packets recv = 661970
Rrep packets sent = 12691
Rrep packets recv = 13407
Rerr packets sent = 3213
Rerr packets recv = 6827377
-----------
Overall stats:
-----------
Message to deliver = 30000
Route Query Data Delivery in process  = 1964
Total Acknowledge Generated     = 12691
Total Routes added      = 1962

Free Memory:  2560800160
Max Memory Available:  7214202880
Total Memory: 2764046336
Memory used:    8324616
start time  : Thu Aug 17 17:54:24 BST 2006
end time    : Thu Aug 17 19:40:23 BST 2006
Elapsed time: 6358147
300     19522  661970 12691  13407  0      0      32213  6827377 8324616       6358147
```

Figure Results 7.2.15A

```
[redacted]:~/jist-swans-1.0.6/results/8.1) swans driver.aodvsim -n 300  -s grid:30x30  -f 900x900 10  -t 1
0,800,60 -m teleport -s 1 0
-----------
Packet stats:
-----------
Rreq packets sent = 18903
Rreq packets recv = 1026390
Rrep packets sent = 1955
Rrep packets recv = 1921
Rerr packets sent = 5692
Rerr packets recv = 341
Hello packets sent = 6416
Hello packets recv = 258
Total aodv packets sent = 32966
Total aodv packets recv = 1028910
Non-hello packets sent = 26550
Non-hello packets recv = 1028652
-----------
Overall stats:
-----------
Messages to deliver = 30000
Route requests      = 1926
Route replies       = 1955
Routes added        = 1921

Freemem:  2671412176
Maxmem:   7214202880
Totalmem: 2754150400
Used:     82738224
start time  : Thu Aug 17 16:49:35 BST 2006
end time    : Thu Aug 17 17:03:23 BST 2006
Elapsed time: 828643
300     18903  1026390 1955   1921   5692   341    6416   258    32966  1028910 26550  1028652 827382
24      828643
```

Figure Results 7.2. 15B

Figure results experiment 7.2.3

```
[redacted]:"/jist-swans-1.0.6/results/8.3.3) swans driver.aodvpsim -n 30 -s random  -f 20x20 -t 10,100,10
 -s 0.05
-----------
Packet stats:
-----------
Rqdd packets sent = 17
Rqdd packets recv = 510
Rck packets sent = 17
Rck packets recv = 19
Total maodvp packets sent = 34
Total maodvp packets recv = 529
-----------
Overall stats:
-----------
Message to deliver = 2
Route Query Data Delivery in process  = 2
Total Acknowledge Generated     = 17
Total Routes added      = 2

Free Memory:  13692512
Max Memory Available:  7214202880
Total Memory: 14942208
Memory used:    1249696
start time  : Sun Aug 20 19:59:31 BST 2006
end time    : Sun Aug 20 19:59:31 BST 2006
Elapsed time: 525
30      17     510    17     19     0      0      34     529    1249696 525
```

Figure Results 7.2.3A

```
[redacted]:"/jist-swans-1.0.6/results/8.3.3) swans driver.aodvpsim -n 30 -s random  -f 20x20 -t 10,100,10
 -s 0.10
-----------
Packet stats:
-----------
Rqdd packets sent = 39
Rqdd packets recv = 930
Rck packets sent = 31
Rck packets recv = 34
Total maodvp packets sent = 70
Total maodvp packets recv = 964
-----------
Overall stats:
-----------
Message to deliver = 5
Route Query Data Delivery in process  = 5
Total Acknowledge Generated     = 31
Total Routes added      = 5

Free Memory:  13626936
Max Memory Available:  7214202880
Total Memory: 14942208
Memory used:    1315272
start time  : Sun Aug 20 20:00:18 BST 2006
end time    : Sun Aug 20 20:00:18 BST 2006
Elapsed time: 756
30      39     930    31     34     0      0      70     964    1315272 756
```

Figure Results 7.2.3B

```
cmshbakh@holly:~/jist-swans-1.0.6/results/8.3.3> swans driver.maoddpsim -n 30 -a random  -f 20x20 -t 10,100,10
 -s 0.15
------------
Packet stats:
------------
Rqdd packets sent = 48
Rqdd packets recv = 1440
Ack packets sent = 48
Ack packets recv = 54
Total maoddp packets sent = 96
Total maoddp packets recv = 1494
------------
Overall stats:
------------
Message to deliver = 7
Route Query Data Delivery in process   = 6
Total Acknowledge Generated      = 48
Total Routes added       = 6

Free Memory:  13986960
Max Memory Available:   7214202880
Total Memory: 15335424
Memory used:     1348464
start time  : Sun Aug 20 20:00:40 BST 2006
end time    : Sun Aug 20 20:00:41 BST 2006
elapsed time: 871
30      48      1440    48      54      0       0       96      1494    1348464 871
```

Figure Results 7.2.3C

```
cmshbakh@holly:~/jist-swans-1.0.6/results/8.3.3> swans driver.maoddpsim -n 30 -a random  -f 20x20 -t 10,100,10
 -s 0.20
------------
Packet stats:
------------
Rqdd packets sent = 70
Rqdd packets recv = 1769
Ack packets sent = 57
Ack packets recv = 62
Total maoddp packets sent = 127
Total maoddp packets recv = 1831
------------
Overall stats:
------------
Message to deliver = 10
Route Query Data Delivery in process   = 9
Total Acknowledge Generated      = 57
Total Routes added       = 9

Free Memory:  13989760
Max Memory Available:   7214202880
Total Memory: 15335424
Memory used:     1345664
start time  : Sun Aug 20 20:01:02 BST 2006
end time    : Sun Aug 20 20:01:02 BST 2006
elapsed time: 830
30      70      1769    57      62      0       0       127     1831    1345664 830
```

Figure Results 7.2.3D

```
cmshbakh@holly:~/jist-swans-1.0.6/results/8.3.3> swans driver.maoddpsim -n 30 -a random  -f 20x20 -t 10,100,10
 -s 0.25
------------
Packet stats:
------------
Rqdd packets sent = 81
Rqdd packets recv = 2370
Ack packets sent = 79
Ack packets recv = 90
Total maoddp packets sent = 160
Total maoddp packets recv = 2460
------------
Overall stats:
------------
Message to deliver = 12
Route Query Data Delivery in process   = 11
Total Acknowledge Generated      = 79
Total Routes added       = 11

Free Memory:  26405408
Max Memory Available:   7214202880
Total Memory: 27983872
Memory used:     1578464
start time  : Sun Aug 20 20:01:21 BST 2006
end time    : Sun Aug 20 20:01:21 BST 2006
elapsed time: 917
30      81      2370    79      90      0       0       160     2460    1578464 917
```

Figure Results 7.2.3E

```
cmshbakh@holly:~/jist-swans-1.0.6/results/8.3.3> swans driver.maoddpsim -n 30 -a random  -f 20x20 -t 10,100,10
 -s 0.35
------------
Packet stats:
------------
Rqdd packets sent = 100
Rqdd packets recv = 3000
Ack packets sent = 100
Ack packets recv = 114
Total maoddp packets sent = 200
Total maoddp packets recv = 3114
------------
Overall stats:
------------
Message to deliver = 17
Route Query Data Delivery in process   = 14
Total Acknowledge Generated      = 100
Total Routes added       = 14

Free Memory:  26361968
Max Memory Available:   7214202880
Total Memory: 27983872
Memory used:     1621904
start time  : Sun Aug 20 20:01:44 BST 2006
end time    : Sun Aug 20 20:01:46 BST 2006
elapsed time: 1494
30      100     3000    100     114     0       0       200     3114    1621904 1494
```

Figure Results 7.2.3F

Implementation codes

```
package jist.swans.route;

import jist.swans.Constants;
import jist.swans.mac.MacAddress;
import jist.swans.misc.Message;
import jist.swans.misc.Util;
import jist.swans.net.NetAddress;
import jist.swans.net.NetInterface;
import jist.swans.net.NetMessage;

import jist.runtime.JistAPI;

import java.util.Iterator;
import java.util.Map;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Set;

/**
 * Mobile Ad-hoc On-Demand Data Delivery Protocol Implementation

 */
public class RouteMaoddp implements RouteInterface.Maoddp
```

```
{
  /** debug mode. */
  public static final boolean DEBUG_MODE = false;

  /** Starting value for node sequence numbers. */
  public static final int SEQUENCE_NUMBER_START = 0;
  /** Starting value for RQDD ID sequence numbers. */
  public static final int RQDD_ID_SEQUENCE_NUMBER_START = 0;
  /** The maximum duration of time a RQDD buffer entry can remain in the RQDD buffer.
*/
  public static final long RQDD_BUFFER_EXPIRE_TIME = 5*Constants.SECOND;
  /** The maximum number of entries allowed in the RQDD buffer. */
  public static final int MAX_RQDD_BUFFER_SIZE = 10;

/** Period of time after which the MAODDP timeout event gets called. */
public static final long MAODDP_TIMEOUT = 30 * Constants.SECOND;

  /** The initial TTL value for any Route Request instance. */
public static final byte TTL_START = 1;
/** The amount added to current TTL upon successive broadcasts of a RQDD message. */
public static final byte TTL_INCREMENT = 2;
/** The maximum TTL for any RQDD message. */
public static final byte TTL_THRESHOLD = 19;

/** Constant term of the RQDD Timeout duration. */
public static final long RQDD_TIMEOUT_BASE = 2 * Constants.SECOND;
/** Variable term of the RQDD Timeout duration, dependant on the RQDD's TTL. */
public static final long RQDD_TIMEOUT_PER_TTL = 1 * Constants.SECOND;

/** The maximum amount of jitter before sending a packet. */
public static final long TRANSMISSION_JITTER = 1 * Constants.MILLI_SECOND;

//////////////////////////////////////////////////////
// messages
//

/**
 * Represents a Route Request (RQDD) message.
 */
private static class RouteQueryDataDeliveryMessage implements Message
{
  /** RQDD message size in bytes. */
  private static final int MESSAGE_SIZE = 24;
  /** Route Request identification number. */
  private int rqddId;
  /** Destination node IP address. */
  private NetAddress destIp;
  /** Originator node IP address. */
  private NetAddress origIp;
  /** Latest known destination node sequence number. */
  private int destSeqNum;
/** Originator node sequence number. */
private int origSeqNum;
/** Hop count from originator node. */
private int hopCount;   //note: actually an 8-bit field in spec
/** Flag which indicates an unknown destination node sequence number. */
private boolean unknownDestSeqNum;
/**
```

```
     * Constructs a new RQDD Message object.
     *
     * @param rqddId RRQD message identification number
     * @param destIp Destination node net address
     * @param origIp Originator node net address
     * @param destSeqNum Destination node sequence number
     * @param origSeqNum Originator node sequence number
     * @param unknownDestSeqNum Flag indicating an unknown destination node sequence
number
     * @param hopCount hop count
     */
    public RouteQueryDataDeliveryMessage(int rqddId, NetAddress destIp, NetAddress
origIp, int destSeqNum, int origSeqNum, boolean unknownDestSeqNum, int hopCount)
    {
      this.rqddId = rqddId;
      this.destIp = destIp;
      this.origIp = origIp;
      this.destSeqNum = destSeqNum;
      this.origSeqNum = origSeqNum;
      this.unknownDestSeqNum = unknownDestSeqNum;
      this.hopCount = hopCount;
    }

    /**
     * Constructs a copy of an existing RRQD message object.
     *
     * @param rqdd An existing RQDD message
     */
    public RouteQueryDataDeliveryMessage(RouteQueryDataDeliveryMessage rqdd)
    {
      this(rqdd.getRqddId(), rqdd.getDestIp(), rqdd.getOrigIp(), rqdd.getDestSeqNum(),
rqdd.getOrigSeqNum(), rqdd.getUnknownDestSeqNum(), rqdd.getHopCount());
    }

    /**
     * Returns RRQD id.
     *
     * @return RQDD id
     */
    public int getRqddId()
    {
      return rqddId;
    }
    /**
     * Returns destination net address.
     *
     * @return Destination net address
     */
    public NetAddress getDestIp()
    {
      return destIp;
    }
    /**
     * Returns originator net address.
     *
     * @return Originator node net address
     */
    public NetAddress getOrigIp()
    {
```

```
    return origIp;
  }
  /**
   * Returns destination sequence number.
   *
   * @return Destination node sequence number
   */
  public int getDestSeqNum()
  {
    return destSeqNum;
  }
  /**
   * Returns originator sequence number.
   *
   * @return Originator sequence number
   */
  public int getOrigSeqNum()
  {
    return origSeqNum;
  }
  /**
   * Returns hop count.
   *
   * @return hop count
   */
  public int getHopCount()
  {
    return hopCount;
  }
  /**
   * Returns unknown destination sequence number flag.
   *
   * @return unknown destination sequence number flag
   */
  public boolean getUnknownDestSeqNum()
  {
    return unknownDestSeqNum;
  }

  /**
   * Increment hop count for this message.
   */
  public void incHopCount()
  {
    hopCount++;
  }

  /**
   * Sets the destination sequence number.
   *
   * @param dsn destination sequence number
   */
  public void setDestSeqNum(int dsn)
  {
    destSeqNum = dsn;
  }

  /**
   * Sets the unknown destination sequence number flag.
```

```
     *
     * @param flag unknown destination sequence number flag
     */
    public void setUnknownDestSeqNum(boolean flag)
    {
      unknownDestSeqNum = flag;
    }

    /**
     * Return packet size.
     *
     * @return packet size
     */
    public int getSize()
    {
      return MESSAGE_SIZE;
    }

    /**
     * Store packet into byte array.
     *
     * @param msg destination byte array
     * @param offset byte array starting offset
     */
    public void getBytes(byte[] msg, int offset)
    {
      /*
      byte[] rqddIdBytes = intToByteArray(this.rqddId);
      byte[] destIpBytes = destIp.getIP().getAddress();
      byte[] srcIpBytes = srcIp.getIP().getAddress();


      //int totalSize = 0;
      //totalSize += rqddIdBytes.length;
      //totalSize += destIpBytes.length;
      //totalSize += srcIpBytes.length;
      //msg = new byte[totalSize];

      for (int i=0; i<rqddIdBytes.length; i++)
      {
        msg[offset++] = rqddIdBytes[i];
      }
      for (int i=0; i<destIpBytes.length; i++)
      {
        msg[offset++] = destIpBytes[i];
      }
      for (int i=0; i<srcIpBytes.length; i++)
      {
        msg[offset++] = srcIpBytes[i];
      }
      */
        throw new RuntimeException("RouteQueryDataDeliveryMessage.getBytes() not
implememented.");
      }

    }

    /**
     * Represents a Route Reply (ACK) message.
```

```
 */
private static class AcknowledgeMessage implements Message
{
  /** ACK Message size in bytes. */
  private static final int MESSAGE_SIZE = 20;

  /** ACK message destination IP address field. */
  private NetAddress destIp;
  /** ACK message destination sequence number field. */
  private int destSeqNum;
  /** ACK message originator sequence number field. */
  private NetAddress origIp;
  /** ACK message hop count field. */
  private int hopCount;

  /**
   * Constructs a new ACK message object.
   *
   * @param destIp ACK message destination node net address
   * @param destSeqNum ACK message destination node sequence number
   * @param origIp ACK message originator node net address
   * @param hopCount ACK message hopcount
   */
  public AcknowledgeMessage(NetAddress destIp, int destSeqNum, NetAddress origIp, int
hopCount)
  {
    this.destIp = destIp;
    this.destSeqNum = destSeqNum;
    this.origIp = origIp;
    this.hopCount = hopCount;
  }

/**
 * Returns destination ip address.
 *
 * @return destination ip address
 */
public NetAddress getDestIp()
{
  return destIp;
}
/**
 * Returns destination sequence number.
 *
 * @return destination sequence number
 */
public int getDestSeqNum()
{
  return destSeqNum;
}
/**
 * Returns originator sequence number.
 *
 * @return originator sequence number
 */
public NetAddress getOrigIp()
{

  return origIp;
```

```
    }
    /**
     * Returns hop count.
     *
     * @return hop count
     */
    public int getHopCount()
    {
      return hopCount;
    }
    /**
     * Increments hop count.
     */
    public void incHopCount()
    {
      hopCount++;
    }
    /**
     * Returns packet size.
     *
     * @return packet size
     */

    public int getSize()
    {
      return MESSAGE_SIZE;
    }
    /**
     * Store packet into byte array.
     *
     * @param msg destination byte array
     * @param offset byte array starting offset
     */
    public void getBytes(byte[] msg, int offset)
    {
      throw new RuntimeException("AcknowledgeMessage.getBytes() not implemented.");
    }
  }

  /**
   * Represents a Route Error (RERR) message class.
   */
  private static class RouteErrorMessage implements Message

  {
    /** RERR Message size in bytes. */
    private static final int MESSAGE_SIZE = 20;

    /** List of net addresses for destinations that have become unreachable. */
    private LinkedList unreachableList;

    /**
     * Constructs a new Route Error (RERR) Message object with an empty unreachable
list.
     */
    public RouteErrorMessage()
    {
      this(new LinkedList());
    }
```

```
    /**
     * Constructs a new Route Error (RERR) Message object with a given unreachable
list.
     *
     * @param list List of net addresses for destinations that have become unreachable
     */
    public RouteErrorMessage(LinkedList list)
    {
      this.unreachableList = list;
    }
    /**
     * Returns the unreachable list.
     *
     * @return linked list of unreachable node net addresses
     */
    public LinkedList getUnreachableList()
    {
      return this.unreachableList;
    }

    /**
     * Add an unreachable node.
     *
     * @param node netAddress of node to be added
     */
    public void addUnreachable(NetAddress node)
    {
      this.unreachableList.add(node);
    }
    /**
     * Return packet size.
     *
     * @return packet size
     */
    public int getSize()
    {
      return MESSAGE_SIZE;

    }
    /**
     * Store packet into byte array.
     *
     * @param msg destination byte array
     * @param offset byte array starting offset
     */
    public void getBytes(byte[] msg, int offset)
    {
      throw new RuntimeException("RouteReplyMessage.getBytes() not implemented.");
    }
  }
  /**
   * Data structure to collect MAODDP statistics.
   */
  public static class MaoddpStats
  {
    /** sent packets. */
    public final MaoddpPacketStats send = new MaoddpPacketStats();
    /** received packets. */
```

```
public final MaoddpPacketStats recv = new MaoddpPacketStats();
/** messages sent by transport layer. */
public long netMsgs;
/** number of total route query data delivery(excluding retransmissions). */
public long rqddOrig;
/** number of acknowledged generated. */
public long ackOrig;
/** number of new routes formed. */
public long rqddSucc;

/** Reset statistics. */
public void clear()
{
  send.clear();
  recv.clear();
  netMsgs = 0;
  rqddOrig = 0;
  ackOrig = 0;
  rqddSucc = 0;
 }
}

/** Packet stats. */
public static class MaoddpPacketStats
{
  /** Sum of RQDD, ACK, RERR, and HELLO packets. */
  public long maoddpPackets;
  /** RQDD packets. */
  public long rqddPackets;
  /** ACK packets. */
  public long ackPackets;
  /** RERR packets. */
  public long rerrPackets;

  /** Reset statistics. */
  public void clear()
  {
    maoddpPackets = 0;
    rqddPackets = 0;
    ackPackets = 0;
    rerrPackets = 0;
  }
 }

 /**
  * Represents a RQDD for a route by a node.  A single route query data delivery can
repeatedly
  * broadcast RQDD messages, with increasing TTL value, until destionation has been
found.
  * TTL values start at TTL_START, increment by TTL_INCREMENT, but do not exceed
  * TTL_THRESHOLD.
  */
private static class RouteQueryDataDelivery
{
  /** Net address of node for which we seek a route. */
  private NetAddress destIp;

  /** Route query data delivery identifier. */
  private int rqddId;
```

```
/** Time to live. */
private byte ttl;

/**
 * Indicates whether this RQDD has been satisfied .
 * Once set to true, this entry can be removed from the rqdd list.
 */
private boolean routeFound = false;

/**
 * Reference to the encapsulating RouteMaoddp instance.
 */
private RouteMaoddp thisNode;

/**
 * Constructs a new RQDD object.
 *
 * @param destIp net address of node for which we seek a route
 * @param thisNode reference to this RouteMaoddp instance
 */
public RouteQueryDataDelivery(NetAddress destIp, RouteMaoddp thisNode)
{
  this.thisNode = thisNode;
  this.destIp = destIp;
  this.ttl = TTL_START;
  this.obtainNewRqddId();
}

/**
 * Return RQDD id.
 *
 * @return RQDD id
 */
public int getRqddId()
{
  return rqddId;
}
/**
 * Returns destination net address.
 *
 * @return destination net address
 */
public NetAddress getDest()
{
  return destIp;
}
/**
 * Returns TTL.
 *
 * @return TTL value
 */
public byte getTtl()
{
  return ttl;
}

/**
 * Assigns a fresh RQDD id number, and updates the node's global RQDD id counter.
```

```
 */
public void obtainNewRqddId()
{
  rqddId = thisNode.rqddIdSeqNum++;
}

/**
 * Increments the TTL value by TTL_INCREMENT, but ensures that it does not
 * exceed TTL_THRESHOLD.
 *
 */
public void incTtl()
{
  ttl = (byte)Math.min(ttl+TTL_INCREMENT, TTL_THRESHOLD);
}

/**
 * Sets the Route Found flag.
 *
 * @param b value to assign to flag
 */
public void setRouteFound(boolean b)
{
  routeFound = b;
}

/**
 * Creates and broadcasts a RQDD message.
 *
 * Also, updates RQDD buffer and routing table.
 */
public void broadcast()
{
  //save RQDD info in buffer (so it knows not to forward if it receives it again)
  thisNode.rqddBuffer.addEntry(new RqddBufferEntry(rqddId, thisNode.netAddr));

  // determine destination sequence number to use, by checking routing table entry
  int destSeqNum=0;
  boolean unknownDestSeqNum=false;
  RouteTableEntry routeEntry =
(RouteTableEntry)thisNode.routeTable.lookup(this.destIp);
  if (routeEntry == null)
  {
    unknownDestSeqNum=true;
  }
  else
  {
    unknownDestSeqNum=false;
    destSeqNum=routeEntry.getDestSeqNum();
  }

// increment node's own sequence number before broadcasting RQDD
thisNode.seqNum++;

// update route-to-self with new SN
RouteTableEntry selfEntry = thisNode.routeTable.lookup(thisNode.netAddr);
selfEntry.setDestSeqNum(thisNode.seqNum);
thisNode.routeTable.printTable();
```

```
        // create RQDD message
        RouteQueryDataDeliveryMessage rqddMsg =
          new RouteQueryDataDeliveryMessage(
            rqddId,
            destIp,
            thisNode.netAddr,
            destSeqNum,
            thisNode.seqNum,
            unknownDestSeqNum,
            0);

        // create IP message containing the RQDD message
        NetMessage.Ip rqddMsgIp =
          new NetMessage.Ip(
            rqddMsg,
            thisNode.netAddr,
            NetAddress.ANY,
            Constants.NET_PROTOCOL_MAODDP,
            Constants.NET_PRIORITY_NORMAL,
            this.ttl);

        // broadcast the IP message
        printlnDebug("Broadcasting RQDD message with rqddId="+rqddId+", ttl="+ttl,
thisNode.netAddr);
        thisNode.self.sendIpMsg(rqddMsgIp, MacAddress.ANY);

        //stats
        if (thisNode.stats != null)
        {
          thisNode.stats.send.rqddPackets++;
          thisNode.stats.send.maoddpPackets++;
        }
      }
      /**
       * Buffer for keeping track of recently sent RQDD messages (so they are not resent).
       * Also keeps track of recent RQDD messages that were answered with an Acknowledge.
       */
      private static class RqddBuffer
      {
        /** List of RqddBufferEntry objects. */
        private LinkedList list;
        /** Local net address. */
        private NetAddress localAddr;
        /**
         * Constructs a Route Query Data Delivery  Buffer object.
         *
         * @param netAddr local net address
         */
        public RqddBuffer(NetAddress netAddr)
        {
          list = new LinkedList();
          localAddr = netAddr;
        }

        /**
         * Adds an entry to the RQDD buffer.
         * If RQDD buffer is full, oldest entries are removed to make room.
         * Also, any expired entries are removed.
```

```
 *
 * @param entry RqddBufferEntry to be added
 */
public void addEntry(RqddBufferEntry entry)
{
  clearExpiredEntries();    // clear expired entries

  //if list is full, remove oldest entry
  if (list.size() == MAX_RQDD_BUFFER_SIZE)
  {
    list.removeLast();
  }
  else if (list.size() > MAX_RQDD_BUFFER_SIZE)
  {
    throw new RuntimeException("RQDD Buffer is larger than allowed size!");
  }

  list.addFirst(entry);
}

/**
 * Checks if a given RouteBufferEntry exists in the RQDD Buffer.
 *
 * @param entry the RouteBufferEntry
 * @return True, if the RQDD buffer contains the specified entry
 */
public boolean contains(RqddBufferEntry entry)
{
  return list.contains(entry);
}

/**
 * Remove all expired entries.
 */
public void clearExpiredEntries()
{
  // Removes entries starting from end of list, since entries are in order
  while (!list.isEmpty() && JistAPI.getTime() >
((RqddBufferEntry)list.getLast()).getTimeSent() + RQDD_BUFFER_EXPIRE_TIME)
  {
    printlnDebug("Removing Entry from RqddBuffer", localAddr);
    list.removeLast();
  }
}

}

/**
 * A single entry of the RQDD Buffer.
 */
private static class RqddBufferEntry
{
/** RQDD id of RQDD message sent. */
private int rqddId;
/** Net address of node that originating the RQDD message. */
private NetAddress originIp;
/** Time (simulation time) that this entry was created. */
private long timeSent;
```

```
/**
 * Constructs a RQDD Buffer Entry object.
 *
 * @param rqddId RQDD id of RQDD message
 * @param originIp Net address of node that originated the RQDD message
 */
public RqddBufferEntry(int rqddId, NetAddress originIp)
{
  this.rqddId = rqddId;
  this.originIp = originIp;
  this.timeSent = JistAPI.getTime();
}

/**
 * Returns timestamp for creation of this entry.
 *
 * @return time that entry was created
 */
public long getTimeSent()
{
  return timeSent;
}

/**
 * Checks whether given RqddBufferEntry is equal to this one.
 * Two entries are equal if the RQDD id and origin IP's are the same.
 *
 * @param o An object of type RqddBufferEntry
 * @return True, if objects are equal
 */
public boolean equals(Object o)
{
  if (o == null || (!(o instanceof RqddBufferEntry)))
  {
    return false;
  }
  RqddBufferEntry entry = (RqddBufferEntry)o;
  return (this.rqddId == entry.rqddId && this.originIp.equals(entry.originIp));
}

/**
 * Returns a hash code.
 *
 * @return hash code
 */
public int hashCode()
{
  return this.rqddId;
}
}

/**
 * A routing table contains a hash map, consisting of NetAddress->RouteTableEntry
mappings.
 */
private static class RouteTable
{
  /** The routing table. */
  private HashMap table;
```

```
/** Address of local node. */
private NetAddress localAddr;

/**
 * Constructs a RouteTable object.
 *
 * @param netAddr local address of this node
 */
public RouteTable(NetAddress netAddr)
{
  table = new HashMap();
  localAddr = netAddr;
}

/**
 * Adds a new entry to the routing table.  Also adds next hop to outgoing table.
 *
 * @param key destination address
 * @param value routing information for this destination
 */
public void add(NetAddress key, RouteTableEntry value)
{
  //add entry to routing table
  table.put(key, value);
}


/**
 * Removes entry with given key from routing table.
 *
 * @param key destination net address
 * @return true, if entry existed and not null; false, otherwise
 */
private boolean remove(NetAddress key)
{
  Object obj = table.remove(key);
  if (obj == null) return false;
  printlnDebug("Removing destination "+key+" from routing table", localAddr);
  return true;
}

/**
 * Look up routing information for a given destination address.
 *
 * @param key destination address
 * @return routing information for this destination
 */
public RouteTableEntry lookup(NetAddress key)
{
  return (RouteTableEntry)table.get(key);
}

/**
 * Remove all route table entries whose destination is specified in a given list.
 *
 * @param list List of destinations (of type NetAddress)
 * @return true, if at least one entry was removed from the table
 */
public boolean removeList(LinkedList list)
```

```
{
  boolean atLeastOneRemoved = false;
  Iterator itr = list.iterator();
  while (itr.hasNext())
  {
    NetAddress addr = (NetAddress)itr.next();
    if (this.remove(addr))
    {
      atLeastOneRemoved = true;
    }
  }
  return atLeastOneRemoved;
}

/**
 * Remove all routing table entries with a given next hop address.
 *
 * @param nextHop the next hop address
 */
public void removeNextHop(MacAddress nextHop)
{
  printlnDebug("Removing all route table entries through "+nextHop, localAddr);
  Iterator itr = table.values().iterator();
  while (itr.hasNext())
  {
    RouteTableEntry entry = (RouteTableEntry)itr.next();
    if (entry.getNextHop().equals(nextHop))
    {
      //remove entry from routing table
      itr.remove();
    }
  }
}

/**
 * Returns all destinations through a given next hop.
 *
 * @param hop Next hop address
 * @return list of destinations (of type NetAddress)
 */
public LinkedList destsViaHop(MacAddress hop)
{
  LinkedList list = new LinkedList();
  Iterator itr = table.entrySet().iterator();
  while (itr.hasNext())
  {
    Map.Entry kvpair = (Map.Entry)itr.next();
    RouteTableEntry rtentry = (RouteTableEntry)kvpair.getValue();
    NetAddress dest = (NetAddress)kvpair.getKey();
    if (rtentry.getNextHop().equals(hop))
    {
      list.add(dest);
    }
  }
  return list;
}

/**
 * Print contents of routing table, for debugging purposes.
```

```
     */
    public void printTable()
    {
      Iterator itr = table.entrySet().iterator();
      while (itr.hasNext())
      {
        Map.Entry mapEntry = (Map.Entry)itr.next();
        NetAddress dest = (NetAddress)mapEntry.getKey();
        RouteTableEntry route = (RouteTableEntry)mapEntry.getValue();
        printDebug("route_table: ["+dest+": (", localAddr);
        if (route != null)
        {
          printlnDebug_plain("nextHop="+route.nextHop+" DSN="+route.destSeqNum+"
hopCnt="+route.hopCount+")]");
        }
        else
        {
          printlnDebug_plain("null)]");
        }
      }
    }

    /**
     * Information to be stored for each destination in routing table.
     */
    private static class RouteTableEntry
    {
      /** Next hop address. */
      private MacAddress nextHop;
      /** Latest known sequence number for destination node. */
      private int destSeqNum;
      /** Hop count for known route to destination. */
      private int hopCount;

      /**
       * Constructs a RouteTableEntry object.
       *
       * @param nextHop next hop address
       * @param destSeqNum latest known sequence number of destination node
       * @param hopCount hop count
       */
      public RouteTableEntry(MacAddress nextHop, int destSeqNum, int hopCount)
      {
        this.nextHop = nextHop;
        this.destSeqNum = destSeqNum;
        this.hopCount = hopCount;
      }

      /**
       * Returns next hop address.
       * @return next hop address.
       */
      public MacAddress getNextHop()
      {
        return nextHop;
      }
```

```
    /**
     * Returns latest known sequence number for destination.
     *
     * @return sequence number
     */
    public int getDestSeqNum()
    {
      return destSeqNum;
    }

    /**
     * Returns hop count for route.
     *
     * @return hop count
     */
    public int getHopCount()
    {
      return hopCount;
    }

    /**
     * Sets a new latest known sequence number for destination node.
     *
     * @param dsn sequence number
     */
    public void setDestSeqNum(int dsn)
    {
      destSeqNum = dsn;
    }
  }

  /**
   * A MessageQueue object temporarily stores transport-layer messages while routes are
   * being determined.  When route information becomes available, the messages are then
   * sent along the routes.
   */
  private static class MessageQueue
  {
    /** list of IP messages (with type NetMessage.Ip). */
    private LinkedList list;
    /** reference to this RouteMaoddp instance. */
    private RouteMaoddp thisNode;

    /**
     * Constructs a MessageQueue object, with an empty list.
     *
     * @param thisNode reference to this RouteMaoddp instance
     */
    public MessageQueue(RouteMaoddp thisNode)
    {
      list = new LinkedList();
      this.thisNode = thisNode;
    }

    /**
     * Adds a NetMessage.Ip to the queue.
     *
     * @param msg message to add to queue
     */
```

```
    public void add(NetMessage.Ip msg)
    {
      list.addLast(msg);
    }

    /**
     * Sends all messages in queue destined for a given destination via a given next
hop.
     *
     * @param dest destination address
     * @param nextHop next hop address
     */
    public void dequeueAndSend(NetAddress dest, MacAddress nextHop)
    {
      for (int i=0; i<list.size(); i++)
      {
        if (((NetMessage.Ip)list.get(i)).getDst().equals(dest))
        {
          NetMessage.Ip msg = (NetMessage.Ip)list.remove(i);
          printlnDebug("Routing IP message to "+nextHop, thisNode.netAddr);
          thisNode.self.sendIpMsg(msg, nextHop);
        }
      }
    }

    /**
     * Removes all messages bound for a given destination.
     *
     * @param dest destination net address
     */
    public void removeMsgsForDest(NetAddress dest)
    {
      Iterator itr = list.listIterator(0);
      while (itr.hasNext())
      {
        NetMessage.Ip msg = (NetMessage.Ip)itr.next();
        if (msg.getDst().equals(dest))
        {
          itr.remove();
        }
      }
    }
  private static class ErrorList
    {
      /** Data structure for storing the precursor set. */
      private Map map = new HashMap();

      /** Reference to this RouteMaoddp instance. */
     private RouteMaoddp thisNode;

      public void sendRERR(LinkedList nodes, byte ttl)
      {
        //define RERR message
        RouteErrorMessage rerrMsg = new RouteErrorMessage(nodes);
        //wrap RERR message in IP message
        NetMessage.Ip ipMsg = new NetMessage.Ip(rerrMsg, thisNode.netAddr,
NetAddress.ANY,
```

```
    Constants.NET_PROTOCOL_MAODDP, Constants.NET_PRIORITY_NORMAL, ttl);
  //send the IP message to each precursor
  Iterator itr = map.keySet().iterator();
  while (itr.hasNext())
  {
    MacAddress macAddr = (MacAddress)itr.next();
    printlnDebug("Sending RERR to nodes "+macAddr, thisNode.netAddr);
    thisNode.self.sendIpMsg(ipMsg, macAddr);
    if (thisNode.stats != null)
    {
      thisNode.stats.send.rerrPackets++;
      thisNode.stats.send.maoddpPackets++;
    }
  }
  /**
   * Represents the set of neighboring nodes through which this node routes messages.
   *
   * The node expects to periodically receive messages from each
   * neighbor in its outgoing set.  If it does not receive any messages from a
   * particular neighbor over a certain period of time, it can assume that neighbor
   * is no longer within range.
   *
   * Each node in this set is mapped to a corresponding OutgoingInfo object.
   */
private static class OutgoingSet
 {
   /** Data structure for the outgoing node set. */
   private Map map = new HashMap();

   /** Local net address. */
   private NetAddress localAddr;
 /**
  * Constructs a new outgoingSet object.req
  *
  * @param netAddr local net address
  */
 public OutgoingSet(NetAddress netAddr)
 {
   localAddr = netAddr;
 }

 /**
  * Returns an iterator for this outgoing set.
  *
  * @return the iterator
  */
 public Iterator iterator()
 {
   return map.entrySet().iterator();
 }

 /**
  * Adds an entry to the outgoing node set.
  *
  * @param m mac address of node to add
  */
 public void add(MacAddress m)
```

```
{
  printlnDebug("Adding "+m+" to outgoing set", localAddr);
  map.put(m, new OutgoingInfo());
}

/**
 * Returns the outgoing node info for a given MAC address.
 *
 * @param m the given MAC addressreq
 * @return the corresponding outgoing node info
 */
public OutgoingInfo getInfo(MacAddress m)
{
    return (OutgoingInfo)map.get(m);
  }
}

/**
 * Information for each node in the outgoing node set.
 */
private static class OutgoingInfo
{

}
////////////////////////////////////////////////////
// RouteMaoddp locals
//

/** Network entity. */
private NetInterface netEntity;
/** Self-referencing proxy entity. */
private RouteInterface.Maoddp self;

/** local network address. */
private NetAddress netAddr;
/** node sequence number. */
private int seqNum;
/** sequence number for RQDD id's. */
private int rqddIdSeqNum;
/** routing table. */
private RouteTable routeTable;
/** list of pending RQDD (originated by this node). */
private LinkedList rqddList;
/** buffer for storing info about previously sent RQDD messages. */
private RqddBuffer rqddBuffer;
/** buffer for storing messages that need routes. */
private MessageQueue msgQueue;
/** set of nodes that route through this node. */
/** set of nodes that this node routes through. */
private OutgoingSet outgoingSet;

// statistics
/** statistics accumulator. */
private MaoddpStats stats;
private ErrorList  EL ;
/**
 * Constructs new RouteMaoddp instance.
 *
 * @param addr node's network address
```

```
   */
  public RouteMaoddp(NetAddress addr)
  {
    this.netAddr = addr;
    this.seqNum = SEQUENCE_NUMBER_START;
    this.rqddIdSeqNum = RQDD_ID_SEQUENCE_NUMBER_START;

    // proxy entity
    this.self = (RouteInterface.Maoddp)JistAPI.proxy(this,
RouteInterface.Maoddp.class);

    //instantiate rqdd list
    this.rqddList = new LinkedList();

    //instantiate RQDD buffer
    this.rqddBuffer = new RqddBuffer(addr);

    //instantiate message queue
    this.msgQueue = new MessageQueue(this);

    //outgoing set
    this.outgoingSet = new OutgoingSet(addr);
    //instantiate routing table
    this.routeTable = new RouteTable(addr);
    //route to self
    routeTable.add(this.netAddr, new RouteTableEntry(MacAddress.NULL, this.seqNum, 0));
    routeTable.printTable();

  }

  /**
   * This event is called periodically after a RQDD is originated, until
   * a route has been found.
   *
   * Each time it is called, it rebroadcasts the RQDD message with a new
   * rqdd id and incremented TTL.
   *
   * @param rqddObj RouteQuery Data Delivery object
   */
  public void RQDDtimeout(Object rqddObj)
  {
    RouteQueryDataDelivery rqdd = (RouteQueryDataDelivery)rqddObj;
    if(!rqdd.routeFound)
    {
      printlnDebug("RQDD timeout event at "+JistAPI.getTime());
      if (rqdd.getTtl() < TTL_THRESHOLD)
      {
        //broadcast new RQDD message with new RQDD ID and incremented TTL
        rqdd.obtainNewRqddId();
        rqdd.incTtl();
        rqdd.broadcast();
        JistAPI.sleep(computeRQDDTimeout(rqdd.getTtl()));
        self.RQDDtimeout(rqddObj);
      }
      else
      {
        //throw out queued packets
        msgQueue.removeMsgsForDest(rqdd.getDest());
```

```
        //remove RQDD from rqddList
        rqddList.remove(rqddObj);
      }
    }
  }

  /**
   * MAODDP Timeout event, which gets called periodically at fixed intervals.
   *
   * Clears expired RQDD buffer entries.rqdd
   *
   *
   */
  public void timeout()
  {
    printlnDebug("Timeout at "+JistAPI.getTime());

    //clear expired entries in RQDD buffer
    rqddBuffer.clearExpiredEntries();
      JistAPI.sleep(MAODDP_TIMEOUT);

  }


  /**
   * Sends IP message after transmission delay
   *
   * @param ipMsg IP message to send
   * @param destMacAddr next hop mac address
   */
  public void sendIpMsg(NetMessage.Ip ipMsg, MacAddress destMacAddr)
  {
    //transmission delay
   randomSleep(TRANSMISSION_JITTER);
    //send message
    netEntity.send(ipMsg, Constants.NET_INTERFACE_DEFAULT, destMacAddr);
     }


  /** {@inheritDoc} */
  public void start()
  {
  printlnDebug("Timeout at "+JistAPI.getTime());
    //clear expired entries in RQDD buffer
    rqddBuffer.clearExpiredEntries();
    JistAPI.sleep(MAODDP_TIMEOUT);
    //self.timeout();
  }
  /**
   * Called by the network layer for every incoming packet. A routing
   * implementation may wish to look at these packets for informational
   * purposes, but should not change their contents.
   *
   * @param msg incoming packet
   * @param lastHop last link-level hop of incoming packet
   */
  public void peek(NetMessage msg, MacAddress lastHop)
  {
```

```
OutgoingInfo lastHopInfo = outgoingSet.getInfo(lastHop);
printlnDebug("Calling peek()");
printlnDebug("peek: lastHop = " + lastHop.hashCode());
NetMessage.Ip ipMsg = null;
if (msg instanceof NetMessage.Ip)
{
  ipMsg = (NetMessage.Ip)msg;
  printlnDebug("peek: src=" + ipMsg.getSrc());
  printlnDebug("peek: dst=" + ipMsg.getDst());
  printlnDebug("peek: payload=" + ipMsg.getPayload());
  printlnDebug("peek: priority=" + ipMsg.getPriority());
  printlnDebug("peek: protocol=" + ipMsg.getProtocol());
  printlnDebug("peek: TTL=" + ipMsg.getTTL());
  printlnDebug("peek: msg size = " + ipMsg.getSize());
 }

}

/**
 * Called by the network layer to request transmission of a packet that
 * requires routing. It is the responsibility of the routing layer to provide
 * a best-effort transmission of this packet to an appropriate next hop by
 * calling the network slayer sending routines once this routing information
 * becomes available.
 *
 * @param msg outgoing packet
 */
public void send(NetMessage msg)
{
  NetMessage.Ip ipMsg = (NetMessage.Ip)msg;
  if (ipMsg.getDst().equals(netAddr))
  {
    throw new RuntimeException("Message is already at destination.  Why is
RouteMaoddp.send() being called?");
  }

  printlnDebug("Attempting to route from " + netAddr + " to " + ipMsg.getDst());

  printlnDebug("src="+ipMsg.getSrc()+" dst="+ipMsg.getDst()+"
prot="+ipMsg.getProtocol()+" ttl="+ipMsg.getTTL()+" getMsg="+ipMsg.getPayload());

  //Look up next hop address for this destination IP in routing table
  NetAddress destNetAddr = ipMsg.getDst();
  RouteTableEntry routeEntry = routeTable.lookup(destNetAddr);
  MacAddress nextHopMacAddr = routeEntry == null ? null : routeEntry.getNextHop();

  //If next hop address found in routing table, forward message
  if (nextHopMacAddr != null)
  {
    printlnDebug("Attempting to route from " + netAddr
      + " to " + ipMsg.getDst()
      + " via " + nextHopMacAddr);
printlnDebug("sent ipMsg:"
      + " src=" + ipMsg.getSrc()
      + " dst=" + ipMsg.getDst()
      + " TTL=" + ipMsg.getTTL());

//Forward message to next hop
```

```
      self.sendIpMsg(ipMsg, nextHopMacAddr);
    }
    //Otherwise, save message in queue; broadcast RQDD message
    else
    {
      // save message in queue
      msgQueue.add(ipMsg);

      RouteQueryDataDelivery rqdd = new RouteQueryDataDelivery(destNetAddr, this);
      println Debug("Adding rqdd id "+rqdd.getRqddId()+" to rqdd list");
      rqddList.add(rqdd);
      rqdd.broadcast();
      //stats
      if (stats != null)
      {
        stats.rqddOrig++;
      }
      JistAPI.sleep(computeRQDDTimeout(rqdd.getTtl()));
      self.RQDDtimeout(rqdd);
    }
  }

  /**
   * Receive a message from network layer.
   *
   * @param msg message received
   * @param src source network address
   * @param lastHop source link address
   * @param macId mac identifier
   * @param dst destination network address
   * @param priority packet priority
   * @param ttl packet time-to-live
   */
  public void receive(
    Message msg,
    NetAddress src,
    MacAddress lastHop,
    byte macId,
    NetAddress dst,
    byte priority,
    byte ttl)
  {
    RouteQueryDataDeliveryMessage rqddMsg = null;
    printlnDebug("receive: src=" + src + " lastHop=" + lastHop
                 + " dst=" + dst + " ttl=" + ttl);

   if (msg instanceof RouteQueryDataDeliveryMessage)
    {
      receiveRouteQueryDataDeliveryMessage((RouteQueryDataDeliveryMessage)msg, src,
lastHop, dst, priority, ttl);
      if (stats != null)
      {
        stats.recv.maoddpPackets++;
        stats.recv.rqddPackets++;
      }
    }
  else if (msg instanceof AcknowledgeMessage)
  {
```

```
      receiveAcknowledgeMessage((AcknowledgeMessage)msg, src, lastHop, dst, priority,
ttl);
      if (stats != null)
      {
        stats.recv.maoddpPackets++;
        stats.recv.ackPackets++;
      }
    }
    else if (msg instanceof RouteErrorMessage)
    {
      receiveRouteErrorMessage((RouteErrorMessage)msg, lastHop, ttl);
      if (stats != null)
      {
        stats.recv.maoddpPackets++;
        stats.recv.rerrPackets++;
      }
    }
    else
    {
      throw new RuntimeException("RouteMaoddp.receive() does not know how to handle
message of type"+ msg);
    }
  }

  /**
   * Process an incoming RQDD message.
   *
   * @param rqddMsg incoming route query data delivery message
   * @param src source of message
   * @param lastHop last hop of message
   * @param dst destination of message
   * @param priority message priority
   * @param ttl message TTL
   */
  private void receiveRouteQueryDataDeliveryMessage(
    RouteQueryDataDeliveryMessage rqddMsg,
    NetAddress src,
    MacAddress lastHop,
    NetAddress dst,
    byte priority,
    byte ttl)

  {

    if (DEBUG_MODE)
    {
      printlnDebug("Receiving RQDD:"
          + " rqddId=" + rqddMsg.getRqddId()
          + " destIp=" + rqddMsg.getDestIp()
          + " origIp=" + rqddMsg.getOrigIp()
          + " destSN=" + rqddMsg.getDestSeqNum()
          + " origSN=" + rqddMsg.getOrigSeqNum( )
          + " unkDSN=" + rqddMsg.getUnknownDestSeqNum()
          + " hopCnt=" + rqddMsg.getHopCount());
    }

    //If necessary, add/update route from this node to the RQDD originator through
previous hop
    RouteTableEntry origRouteEntry = routeTable.lookup(rqddMsg.getOrigIp());
```

```
    boolean updateRoute = shouldUpdateRouteToOrigin(rqddMsg, origRouteEntry);
    if (updateRoute)
    {
      //add/update route to RQDD originator through previous hop
      routeTable.add(rqddMsg.getOrigIp(), new RouteTableEntry(lastHop,
rqddMsg.getOrigSeqNum(), rqddMsg.getHopCount()+1));
      routeTable.printTable();
    }

    //Check if this node is dest, or has a route to dest with higher SN
    //If so, send back a ACK; otherwise, forward RQDD to neighbors.
    boolean isDest = rqddMsg.destIp.equals(this.netAddr);
    RouteTableEntry destRouteEntry = routeTable.lookup(rqddMsg.destIp);
    boolean routeToDestExists = (destRouteEntry != null && destRouteEntry.nextHop !=
null);
    boolean hasFreshRoute = routeToDestExists && !rqddMsg.getUnknownDestSeqNum() &&
destRouteEntry.getDestSeqNum() > rqddMsg.getDestSeqNum();
    boolean inRqddBuffer = rqddBuffer.contains(new RqddBufferEntry(rqddMsg.getRqddId(),
rqddMsg.getOrigIp()));

 if (hasFreshRoute)
    {
      if (!inRqddBuffer||updateRoute)
      {   //check buffer first to see if this node has already sent this RQDD before
        byte newTtl = (byte) (ttl - 1); // decrement ttl
         if (newTtl > 0)
           printlnDebug("Forwarding RQDD");
          forwardRouteQueryDataDeliveryMessage(rqddMsg, newTtl, destRouteEntry);
            }
    }

else if (isDest)
          {
        generateAcknowledgeMessage(rqddMsg, isDest, destRouteEntry);
        stats.recv.maoddpPackets++;
        stats.recv.rqddPackets++;
      }


}
 /**
  * Process an incoming ACK message.
  *
  * @param ackMsg incoming acknowledge message
  * @param src source of message
  * @param lastHop last hop of message
  * @param dst destination of message
  * @param priority message priority
  * @param ttl message TTL value
  */
private void receiveAcknowledgeMessage(
  AcknowledgeMessage ackMsg,
  NetAddress src,
  MacAddress lastHop,
  NetAddress dst,
  byte priority,
  byte ttl)
{
```

```
        printlnDebug("handling ACK:"
            + " destIp=" + ackMsg.getDestIp()
            + " destSN=" + ackMsg.getDestSeqNum()
            + " origIp=" + ackMsg.getOrigIp()
            + " hopCnt=" + ackMsg.getHopCount());

        //Add route to routing table if:
        // - no entry currently exists, OR
        // - the DSN of the ACK msg > the DSN of the existing route, OR
        // - the DSN's are equal, but (hopCount of the ACK msg < existing hopCount).
        RouteTableEntry entry = routeTable.lookup(ackMsg.getDestIp());
        if (entry == null ||
           (ackMsg.getDestSeqNum()>entry.getDestSeqNum()) ||
           (ackMsg.getDestSeqNum()==entry.getDestSeqNum() &&
ackMsg.hopCount<entry.getHopCount()))
        {
          routeTable.add(
            ackMsg.getDestIp(),
            new RouteTableEntry(
              lastHop,
              ackMsg.getDestSeqNum(),
              ackMsg.getHopCount()+1));
          routeTable.printTable();
          outgoingSet.add(lastHop);

        }

        //Case 1: This node is the originator of the RQDD
        if (this.netAddr.equals(ackMsg.getOrigIp()))
        {
          //go through rqddlist, setting routeFound=true, and removing them
          Iterator itr = rqddList.iterator();
          while (itr.hasNext())
          {
            RouteQueryDataDelivery rqdd = (RouteQueryDataDelivery)itr.next();
            if (rqdd.getDest().equals(ackMsg.getDestIp()))
            {
              //printlnDebug("Removing rqdd from rqddlist");
              //rqdd.setRouteFound(true);
              //stats
              if (stats != null)
              {
                stats.recv.maoddpPackets++;
                stats.recv.ackPackets++;
                stats.rqddSucc++;   //indicate RQDD was successfully satisfied
              }
             itr.remove();
            }
          }

          //send out queued messages
          msgQueue.dequeueAndSend(ackMsg.getDestIp(), lastHop);
        }
        //Case 2: this node is not the originator of the RQDD
        else
        {
          RouteTableEntry origRouteEntry = routeTable.lookup(ackMsg.getOrigIp());
          if (origRouteEntry != null && (ttl > 0))
          {
```

```
MacAddress nextHop = origRouteEntry.getNextHop();
ackMsg.incHopCount();  //increment ACK's hop count
NetMessage.Ip ackMsgIp =
  new NetMessage.Ip(
    ackMsg,
    src,
    dst,
    Constants.NET_PROTOCOL_MAODDP,
    Constants.NET_PRIORITY_NORMAL,
    (byte)(ttl-1));
printlnDebug("Forwarding ACK message to node "+nextHop);
self.sendIpMsg(ackMsgIp, nextHop);
//stats
      if (stats != null)
    {
      stats.recv.maoddpPackets++;
      stats.recv.ackPackets++;
     // stats.rqddSucc++;   //indicate RQDD was successfully satisfied
    }
   /* if (stats != null)
    {
      stats.send.ackPackets++;
      stats.send.maoddpPackets++;
    } */

    this.outgoingSet.add(nextHop);
  }
 }
}

 /**
  * Process an incoming RERR message.
  *
  * @param rerrMsg incoming route error message
  * @param lastHop last hop of message
  * @param ttl message TTL value
  */
 private void receiveRouteErrorMessage(RouteErrorMessage rerrMsg, MacAddress lastHop,
byte ttl)
 {
   printlnDebug("Receiving RERR message from "+lastHop);
   //Remove from route table entries for all destinations indicated by RERR msg
   boolean somethingRemoved = routeTable.removeList(rerrMsg.getUnreachableList());

 }

  /**
  * Forwards a RQDD message to all neighbors.
  *
  * @param rqddMsg incoming RQDD message
  * @param newTtl  TTL value for the (new) RQDD message to be forwarded
  * @param destRouteEntry route table entry for destination node
  */
 private void forwardRouteQueryDataDeliveryMessage(RouteQueryDataDeliveryMessage
rqddMsg, byte newTtl, RouteTableEntry destRouteEntry)
 {
   //save RQDD info in buffer
   rqddBuffer.addEntry(new RqddBufferEntry(rqddMsg.getRqddId(), rqddMsg.getOrigIp()));
```

```
      // make copy of RQDD message for sending (this may not be necessary)
      RouteQueryDataDeliveryMessage newRqddMsg = new
 RouteQueryDataDeliveryMessage(rqddMsg);

      // increment hop count of this message (should i be making copy of message first?)
      newRqddMsg.incHopCount();

      // set RQDD's DSN to max of RQDD's existing DSN and this node's DSN
      if (destRouteEntry != null)
      {
        boolean condition = newRqddMsg.getUnknownDestSeqNum() ||
destRouteEntry.getDestSeqNum() > newRqddMsg.getDestSeqNum();
        if (condition)
        {
          newRqddMsg.setDestSeqNum(destRouteEntry.getDestSeqNum());
          newRqddMsg.setUnknownDestSeqNum(false);
        }
/*else {
          EL.sendRERR(routeTable.destsViaHop(destRouteEntry), Constants.TTL_DEFAULT); }
*/
      }
      NetMessage.Ip rqddMsgIp =
        new NetMessage.Ip(
          newRqddMsg,
          this.netAddr,
          NetAddress.ANY,
          Constants.NET_PROTOCOL_MAODDP,
          Constants.NET_PRIORITY_NORMAL,
          newTtl);

       //send RQDD message
       self.sendIpMsg(rqddMsgIp, MacAddress.ANY);
        stats.recv.maoddpPackets++;
           stats.recv.rqddPackets++;
       //stats
      /* if (stats != null)
       {
        // stats.send.rqddPackets++;
        // stats.send.maoddpPackets++;
       } */
    }


    /**
     * Generates and sends a ACK message.
     *
     * @param  RQDD message that this ACK message is responding to
     * @param isDest true if this node is the destination of the RQDD message
     * @param destRouteEntry route table entry for the destination node, if this is not
dest. (otherwise, null)
     */
    private void generateAcknowledgeMessage(RouteQueryDataDeliveryMessage rqddMsg,
boolean isDest, RouteTableEntry destRouteEntry)
    {
      //add rqdddMsg to rqdd buffer (if not there), so we do not send this same ACK again
      /* RqddBufferEntry newEntry = new RqddBufferEntry(rqddMsg.rqddId, rqddMsg.origIp);
       if (!rqddBuffer.contains(newEntry))
       {
```

```
      rqddBuffer.addEntry(newEntry);
    }

    // set initial hop count, based on whether this node is destination or intermediate
node
    int initialHopCount=0;
    if (!isDest)
    {
      initialHopCount = destRouteEntry.getHopCount();
    }

    //update this node's SN if destSeqNum in packet is greater than this node's SN
    if (!rqddMsg.getUnknownDestSeqNum() && rqddMsg.getDestSeqNum() > this.seqNum)
    {
      this.seqNum = rqddMsg.getDestSeqNum();  // update SN

      //update route-to-self with updated SN
      RouteTableEntry selfRoute = routeTable.lookup(this.netAddr);
      selfRoute.setDestSeqNum(this.seqNum);
      routeTable.printTable();
    }  */

    //create Acknowledge Message
   int initialHopCount=0;
    AcknowledgeMessage ackMsg = new
AcknowledgeMessage(rqddMsg.getDestIp(),this.seqNum,rqddMsg.getOrigIp(),initialHopCount)
;

    //get next hop from routing table
    MacAddress nextHop = routeTable.lookup(rqddMsg.getOrigIp()).getNextHop();

    // create IP message containing the ACK message
    NetMessage.Ip ackMsgIp =
      new NetMessage.Ip(
        ackMsg,
        this.netAddr,
        NetAddress.ANY,   // <-- is this ok?
        Constants.NET_PROTOCOL_MAODDP,
        Constants.NET_PRIORITY_NORMAL,
        Constants.TTL_DEFAULT);

    printlnDebug("Sending ACK to "+nextHop);

    // send message to next hop
    self.sendIpMsg(ackMsgIp, nextHop);

    //stats
    if (stats != null)
    {
      stats.send.ackPackets++;
      stats.send.maoddpPackets++;
      stats.ackOrig++;
    }

    outgoingSet.add(nextHop);
  }
```

```
  /**
   * Decides whether a node receiving a RQDD message should update its route to the
   * RQDD originator.
   *
   * @param rqddMsg incoming route query data delivery message
   * @param origRouteEntry existing routing table entry for   the RQDD-originating node
   * @return true, if node should update its routing table entry to RQDD-originating
node
   */
  private boolean shouldUpdateRouteToOrigin(RouteQueryDataDeliveryMessage rqddMsg,
RouteTableEntry origRouteEntry)
  {
    // if routing table does not contain a route to the originator
    if (origRouteEntry == null || origRouteEntry.getNextHop() == null)
      return true;

    // if originator SN in RQDD message is greater than existing SN in routing table,
return true
    if (rqddMsg.getOrigSeqNum() > origRouteEntry.getDestSeqNum())
      return true;

    //if SN's are equal, but hop count is better, return true
    boolean equalSeqNum = rqddMsg.getOrigSeqNum() == origRouteEntry.getDestSeqNum();
    boolean hopCountBetter = rqddMsg.getHopCount()+1 < origRouteEntry.getHopCount();
    if (equalSeqNum && hopCountBetter)
    {
      return true;
    }

    return false;
  }

/**
 * Computes the RQDD Timeout period, given a TTL value.
 *
 * @param ttl TTL value
 * @return timeout period
 */
private long computeRQDDTimeout(byte ttl)
{
  return RQDD_TIMEOUT_BASE + RQDD_TIMEOUT_PER_TTL * ttl;
}

/**
 * Sleep for a random time.
 *
 * @param time max sleep time
 */
private static void randomSleep(long time)
{
  JistAPI.sleep(Util.randomTime(time));
}

/**
  * Sets maoddp statistics object.
  *
  * @param stats maoddp statistics object
  */
public void setStats(MaoddpStats stats)
```

```
  {
    this.stats = stats;
  }

  /**
   * Gets node's local address.
   *
   * @return local address
   */
  public NetAddress getLocalAddr()
  {
    return this.netAddr;
  }

  /**
   * Returns self-referencing proxy entity.
   *
   * @return self-referencing proxy entity
   */
  public RouteInterface.Maoddp getProxy()
  {
    return this.self;
  }

  /**
   * Sets network entity.
   *
   * @param netEntity network entity
   */
  public void setNetEntity(NetInterface netEntity)
  {
    this.netEntity = netEntity;
  }


  ////////////////////////////////////////////////
  // DEBUG CODE
  //

  /**
   * Println given string with JiST time and local net address, if debug mode on.
   *
   * @param s string to print
   */
  private void printlnDebug(String s)
  {
    if (RouteMaoddp.DEBUG_MODE)
    {
      System.out.println(JistAPI.getTime()+"\t"+netAddr+": "+s);
    }
  }
  /**
   * Print given string with JiST time and local net address, if debug mode on.
   *
   * @param s string to print
   */
  private void printDebug(String s)
  {
```

```
    if (RouteMaoddp.DEBUG_MODE)
    {
      System.out.print(JistAPI.getTime()+"\t"+netAddr+": "+s);
    }
  }

  /**
   * Println given string with JiST time and given net address, if debug mode on.
   *
   * @param s string to print
   * @param addr node address
   */
  private static void printlnDebug(String s, NetAddress addr)
  {
    if (RouteMaoddp.DEBUG_MODE)
    {
      System.out.println(JistAPI.getTime()+"\t"+addr+": "+s);
    }
  }

  /**
   * Print given string with JiST time and given net address, if debug mode on.
   *
   * @param s string to print)
   * @param addr node address
   */
  private static void printDebug(String s, NetAddress addr)
  {
    if (RouteMaoddp.DEBUG_MODE)
    {
      System.out.print(JistAPI.getTime()+"\t"+addr+": "+s);
    }
  }

  /**
   * Println given string only if debug mode on.
   *
   * @param s string to print
   */
  private static void printlnDebug_plain(String s)
  {
    if (RouteMaoddp.DEBUG_MODE)
    {
      System.out.println(s);
    }
  }

  /**
   * Print given string only if debug mode on.
   * @param s string to print
   */
  private static void printDebug_plain(String s)
  {
    if (RouteMaoddp.DEBUG_MODE)
    {
      System.out.print(s);
    }
  }
```

```
  /**
   * Prints the node's outgoing set.
   */
  public void printOutgoing()
  {
    Set keySet = outgoingSet.map.keySet();
    Iterator itr = keySet.iterator();
    System.out.print(netAddr+": outg: ");
    while (itr.hasNext())
    {
      MacAddress mac = (MacAddress)itr.next();
      System.out.print(mac+", ");
    }
      System.out.println();
    }

  }
```

MAODDP driver for swans

```
package driver;

import jist.swans.field.Field;
import jist.swans.field.Mobility;
import jist.swans.field.Placement;
import jist.swans.field.Fading;
import jist.swans.field.Spatial;
import jist.swans.field.PathLoss;
import jist.swans.radio.RadioNoise;
import jist.swans.radio.RadioNoiseIndep;
import jist.swans.radio.RadioInfo;
import jist.swans.mac.MacAddress;
import jist.swans.mac.MacDumb;
import jist.swans.net.NetAddress;
import jist.swans.net.NetMessage;
import jist.swans.net.NetIp;
import jist.swans.net.PacketLoss;
import jist.swans.trans.TransUdp;
import jist.swans.route.RouteInterface;
import jist.swans.route.RouteMaoddp;
import jist.swans.misc.Util;
import jist.swans.misc.Mapper;
import jist.swans.misc.Location;
import jist.swans.misc.Message;
import jist.swans.Constants;
import java.util.Iterator;

import jist.runtime.JistAPI;

import jargs.gnu.*;

import java.util.Date;
import java.util.Random;
import java.util.Vector;
```

```
/**
 * MAODDP simulation.
 *
 *
 *
 */
public class maoddpsim
{

  /** Default port number to send and receive packets. */
  private static final int PORT = 3001;

  ////////////////////////////////////////////////////
  // command-line options
  //

  /** Simulation parameters with default values. */
  private static class CommandLineOptions
  {
    /** Whether to print a usage statement. */
    private boolean help = false;
    ///** Time to end simulation. */
    //private int endTime = -1;
    /** Routing protocol to use. */
    private int protocol = Constants.NET_PROTOCOL_MAODDP;
    /** Number of nodes. */
    private int nodes = 1000;
    /** Field dimensions (in meters). */
    private Location.Location2D field = new Location.Location2D(1000, 1000);
    /** Field wrap-around. */
    private boolean wrapField = false;
    /** Node placement model. */
    private int placement = Constants.PLACEMENT_RANDOM;
    /** Node placement options. */
    private String placementOpts = null;
    /** Node mobility model. */
    private int mobility = Constants.MOBILITY_STATIC;
    /** Node mobility options. */
    private String mobilityOpts = null;
    /** Packet loss model. */
    private int loss = Constants.NET_LOSS_NONE;
    /** Packet loss options. */
    private String lossOpts = "";
    /** Number of messages sent per minute per node. */
    private double sendRate = 1.0;
    /** Start of sending (seconds). */
    private int startTime = 60;
    /** Number of seconds to send messages. */
    private int duration = 3600;
    /** Number of seconds after messages stop sending to end simulation. */
    private int resolutionTime = 30;
    /** Random seed. */
    private int seed = 0;
    /** binning mode. */
    public int spatial_mode = Constants.SPATIAL_HIER;
    /** binning degree. */
    public int spatial_div = 5;
  } // class: CommandLineOptions
```

```
  /** Prints a usage statement. */
  private static void showUsage()
  {
    System.out.println("Usage: java driver.maoddpsim [options]");
    System.out.println();
    System.out.println("  -h, --help              print this message");
    System.out.println("  -n, --nodes             number of nodes: n [100] ");
    System.out.println("  -f, --field             field dimensions: x,y [100,100]");
    System.out.println("  -a, --arrange           placement model: [random],grid:ixj");
    System.out.println("  -m, --mobility          mobility:
[static],waypoint:opts,teleport:p,walk:opts");
    System.out.println("  -l, --loss              packet loss model: [none],uniform:p");
    System.out.println("  -s, --send rate         send rate per-minute: [1.0]");
    System.out.println("  -t, --timing            node activity timing:
start,duration,resolution [60,3600,30]");
    System.out.println("  -r, --randomseed        random seed: [0]");
    System.out.println();
    System.out.println("e.g.");
    System.out.println("  swans driver.maoddpsim -n 25 -f 2000x2000 -a grid:5x5 -t
10,600,60");
    System.out.println();
  }

  /**
   * Parses command-line arguments.
   *
   * @param args command-line arguments
   * @return parsed command-line options
   * @throws CmdLineParser.OptionException if the command-line arguments are not well-
formed.
   */
  private static CommandLineOptions parseCommandLineOptions(String[] args)
  throws CmdLineParser.OptionException
  {
    if(args.length==0)
    {
      args = new String[] { "-h" };
    }
    CmdLineParser parser = new CmdLineParser();
    CmdLineParser.Option opt_help = parser.addBooleanOption('h', "help");
    CmdLineParser.Option opt_nodes = parser.addIntegerOption('n', "nodes");
    CmdLineParser.Option opt_field = parser.addStringOption('f', "field");
    CmdLineParser.Option opt_placement = parser.addStringOption('a', "arrange");
    CmdLineParser.Option opt_mobility = parser.addStringOption('m', "mobility");
    CmdLineParser.Option opt_loss = parser.addStringOption('l', "loss");
    CmdLineParser.Option opt_rate = parser.addDoubleOption('s', "send rate");
    CmdLineParser.Option opt_timing = parser.addStringOption('t', "timing");
    CmdLineParser.Option opt_randseed = parser.addIntegerOption('r', "randomseed");
    parser.parse(args);

    CommandLineOptions cmdOpts = new CommandLineOptions();
    // help
    if(parser.getOptionValue(opt_help) != null)
    {
      cmdOpts.help = true;
    }
    //// endat
    //if (parser.getOptionValue(opt_endat) != null)
```

```
    //{
    //  cmdOpts.endTime = ((Integer)parser.getOptionValue(opt_endat)).intValue();
    //}
    // nodes
    if(parser.getOptionValue(opt_nodes) != null)
    {
      cmdOpts.nodes = ((Integer)parser.getOptionValue(opt_nodes)).intValue();
    }
    // field
    if(parser.getOptionValue(opt_field) != null)
    {
      cmdOpts.field =
(Location.Location2D)Location.parse((String)parser.getOptionValue(opt_field));
    }
    // placement
    if(parser.getOptionValue(opt_placement) != null)
    {
      String placementString =
((String)parser.getOptionValue(opt_placement)).split(":")[0];
      if(placementString!=null)
      {
        if(placementString.equalsIgnoreCase("random"))
        {
          cmdOpts.placement = Constants.PLACEMENT_RANDOM;
        }
        else if(placementString.equalsIgnoreCase("grid"))
        {
          cmdOpts.placement = Constants.PLACEMENT_GRID;
        }
        else
        {
          throw new CmdLineParser.IllegalOptionValueException(opt_placement,
"unrecognized placement model");
        }
      }
      cmdOpts.placementOpts =
Util.stringJoin((String[])Util.rest(((String)parser.getOptionValue(opt_placement)).spli
t(":")), ":");
    }
    // mobility
    if(parser.getOptionValue(opt_mobility)!=null)
    {
      String mobilityString =
((String)parser.getOptionValue(opt_mobility)).split(":")[0];
      if(mobilityString!=null)
      {
        if(mobilityString.equalsIgnoreCase("static"))
        {
          cmdOpts.mobility = Constants.MOBILITY_STATIC;
        }
        else if(mobilityString.equalsIgnoreCase("waypoint"))
        {
          cmdOpts.mobility = Constants.MOBILITY_WAYPOINT;
        }
        else if(mobilityString.equalsIgnoreCase("teleport"))
        {
          cmdOpts.mobility = Constants.MOBILITY_TELEPORT;
        }
        else if(mobilityString.equalsIgnoreCase("walk"))
```

```
        {
          cmdOpts.mobility = Constants.MOBILITY_WALK;
        }
        else
        {
          throw new CmdLineParser.IllegalOptionValueException(opt_mobility,
"unrecognized mobility model");
        }
      }
      cmdOpts.mobilityOpts =
Util.stringJoin((String[])Util.rest(((String)parser.getOptionValue(opt_mobility)).split
(":")), ":");
    }
    // loss
    if(parser.getOptionValue(opt_loss)!=null)
    {
      String lossString = ((String)parser.getOptionValue(opt_loss)).split(":")[0];
      if(lossString!=null)
      {
        if(lossString.equalsIgnoreCase("none"))
        {
          cmdOpts.loss = Constants.NET_LOSS_NONE;
        }
        else if(lossString.equalsIgnoreCase("uniform"))
        {
          cmdOpts.loss = Constants.NET_LOSS_UNIFORM;
        }
        else
        {
          throw new CmdLineParser.IllegalOptionValueException(opt_loss, "unrecognized
mobility model");
        }
        cmdOpts.lossOpts =
Util.stringJoin((String[])Util.rest(((String)parser.getOptionValue(opt_loss)).split(":"
)), ":");
      }

      //// bordercast
      //if(parser.getOptionValue(opt_bordercast) != null)
      //{
      //  String[] data = ((String)parser.getOptionValue(opt_bordercast)).split(",");
      //  if(data.length!=3) throw new
CmdLineParser.IllegalOptionValueException(opt_bordercast, "bad format:
num,start,delay");
      //  cmdOpts.bordercasts = Integer.parseInt(data[0]);
      //  cmdOpts.bordercastStart = Integer.parseInt(data[1]);
      //  cmdOpts.bordercastDelay = Integer.parseInt(data[2]);
      //}

      //send rate
      if (parser.getOptionValue(opt_rate) != null)
      {
        cmdOpts.sendRate = ((Double)parser.getOptionValue(opt_rate)).doubleValue();
      }

      //timing parameters
      if (parser.getOptionValue(opt_timing) != null)
      {
```

```
      String[] data = ((String)parser.getOptionValue(opt_timing)).split(",");
      if(data.length!=3) throw new
CmdLineParser.IllegalOptionValueException(opt_timing, "bad format:
start,duration,resolution");
      cmdOpts.startTime = Integer.parseInt(data[0]);
      cmdOpts.duration = Integer.parseInt(data[1]);
      cmdOpts.resolutionTime = Integer.parseInt(data[2]);
    }

    // random seed
    if (parser.getOptionValue(opt_randseed) != null)
    {
      cmdOpts.seed = ((Integer)parser.getOptionValue(opt_randseed)).intValue();
    }

    return cmdOpts;

  } // parseCommandLineOptions


  ////////////////////////////////////////////////
  // simulation setup
  //

  /**
   * Add node to the field and start it.
   *
   * @param opts command-line options
   * @param i node number, which also serves as its address
   * @param routers list of zrp entities to be appended to
   * @param stats statistics collector
   * @param field simulation field
   * @param place node placement model
   * @param radioInfo shared radio information
   * @param protMap registered protocol map
   * @param inLoss packet incoming loss model
   * @param outLoss packet outgoing loss model
   */
  public static void addNode(CommandLineOptions opts, int i, Vector routers,
RouteMaoddp.MaoddpStats stats,
      Field field, Placement place, RadioInfo.RadioInfoShared radioInfo, Mapper
protMap,
      PacketLoss inLoss, PacketLoss outLoss)
  {
    // radio
    RadioNoise radio = new RadioNoiseIndep(i, radioInfo);

    // mac
    MacDumb mac = new MacDumb(new MacAddress(i), radio.getRadioInfo());

    // network
    final NetAddress address = new NetAddress(i);
    NetIp net = new NetIp(address, protMap, inLoss, outLoss);

    // routing
    RouteInterface route = null;
    switch(opts.protocol)
    {
      case Constants.NET_PROTOCOL_MAODDP:
```

```
      RouteMaoddp maoddp = new RouteMaoddp(address);
      maoddp.setNetEntity(net.getProxy());
      maoddp.getProxy().start();
      route = maoddp.getProxy();
      routers.add(maoddp);
      // statistics
      maoddp.setStats(stats);
      break;
    default:
      throw new RuntimeException("invalid routing protocol");
  }

  // transport
    TransUdp udp = new TransUdp();

    // placement
    Location location = place.getNextLocation();
    field.addRadio(radio.getRadioInfo(), radio.getProxy(), location);
    field.startMobility(radio.getRadioInfo().getUnique().getID());

    // node entity hookup
    radio.setFieldEntity(field.getProxy());
    radio.setMacEntity(mac.getProxy());
    byte intId = net.addInterface(mac.getProxy());
    net.setRouting(route);
    mac.setRadioEntity(radio.getProxy());
    mac.setNetEntity(net.getProxy(), intId);
    udp.setNetEntity(net.getProxy());
    net.setProtocolHandler(Constants.NET_PROTOCOL_UDP, udp.getProxy());
    net.setProtocolHandler(opts.protocol, route);
  }  //method: addNode

  /**
   * Constructs field and nodes with given command-line options, establishes
   * client/server pairs and starts them.
   *
   * @param opts command-line parameters
   * @param routers vectors to place zrp objects into
   * @param stats zrp statistics collection object
   */
  private static void buildField(CommandLineOptions opts, final Vector routers, final
RouteMaoddp.MaoddpStats stats)
  {
    // initialize node mobility model
    Mobility mobility = null;
    switch(opts.mobility)
    {
      case Constants.MOBILITY_STATIC:
        mobility = new Mobility.Static();
        break;
      case Constants.MOBILITY_WAYPOINT:
        mobility = new Mobility.RandomWaypoint(opts.field, opts.mobilityOpts);
        break;
      case Constants.MOBILITY_TELEPORT:
        mobility = new Mobility.Teleport(opts.field,
Long.parseLong(opts.mobilityOpts));
        break;
      case Constants.MOBILITY_WALK:
        mobility = new Mobility.RandomWalk(opts.field, opts.mobilityOpts);
```

```
      break;
    default:
      throw new RuntimeException("unknown node mobility model");
  }
  // initialize spatial binning
  Spatial spatial = null;
  switch(opts.spatial_mode)
  {
    case Constants.SPATIAL_LINEAR:
      spatial = new Spatial.LinearList(opts.field);
      break;
    case Constants.SPATIAL_GRID:
      spatial = new Spatial.Grid(opts.field, opts.spatial_div);
      break;
    case Constants.SPATIAL_HIER:
      spatial = new Spatial.HierGrid(opts.field, opts.spatial_div);
      break;
    default:
      throw new RuntimeException("unknown spatial binning model");
  }
  if(opts.wrapField) spatial = new Spatial.TiledWraparound(spatial);
  // initialize field
  Field field = new Field(spatial, new Fading.None(), new PathLoss.FreeSpace(),
    mobility, Constants.PROPAGATION_LIMIT_DEFAULT);
  // initialize shared radio information
  RadioInfo.RadioInfoShared radioInfo = RadioInfo.createShared(
    Constants.FREQUENCY_DEFAULT,
    Constants.BANDWIDTH_DEFAULT,
    Constants.TRANSMIT_DEFAULT,
    Constants.GAIN_DEFAULT,
    Util.fromDB(Constants.SENSITIVITY_DEFAULT),
    Util.fromDB(Constants.THRESHOLD_DEFAULT),
    Constants.TEMPERATURE_DEFAULT,
    Constants.TEMPERATURE_FACTOR_DEFAULT,
    Constants.AMBIENT_NOISE_DEFAULT);
  // initialize shared protocol mapper
  Mapper protMap = new Mapper(new int[] { Constants.NET_PROTOCOL_UDP, opts.protocol,
});
  // initialize packet loss models
  PacketLoss outLoss = new PacketLoss.Zero();
  PacketLoss inLoss = null;
  switch(opts.loss)
  {
    case Constants.NET_LOSS_NONE:
      inLoss = new PacketLoss.Zero();
      break;
    case Constants.NET_LOSS_UNIFORM:
      inLoss = new PacketLoss.Uniform(Double.parseDouble(opts.lossOpts));
    default:
      throw new RuntimeException("unknown packet loss model");
  }
  // initialize node placement model
  Placement place = null;
  switch(opts.placement)
  {
    case Constants.PLACEMENT_RANDOM:
      place = new Placement.Random(opts.field);
      break;
    case Constants.PLACEMENT_GRID:
```

```
        place = new Placement.Grid(opts.field, opts.placementOpts);
        break;
      default:
        throw new RuntimeException("unknown node placement model");
    }
    // create each node
    for (int i=1; i<=opts.nodes; i++)
    {
      addNode(opts, i, routers, stats, field, place, radioInfo, protMap, inLoss,
outLoss);
    }

    // set up message sending events
    JistAPI.sleep(opts.startTime*Constants.SECOND);
    //System.out.println("clear stats at t="+JistAPI.getTimeString());
    stats.clear();
    int numTotalMessages = (int)Math.floor(((double)opts.sendRate/60) * opts.nodes *
opts.duration);
    long delayInterval = (long)Math.ceil((double)opts.duration *
(double)Constants.SECOND / (double)numTotalMessages);
    for(int i=0; i<numTotalMessages; i++)
    {
      //pick random send node
      int srcIdx = Constants.random.nextInt(routers.size());
      int destIdx;
      do
      {
        //pick random dest node
        destIdx = Constants.random.nextInt(routers.size());
      } while (destIdx == srcIdx);
      RouteMaoddp srcMaoddp = (RouteMaoddp)routers.elementAt(srcIdx);
      RouteMaoddp destMaoddp = (RouteMaoddp)routers.elementAt(destIdx);
      TransUdp.UdpMessage udpMsg = new TransUdp.UdpMessage(PORT, PORT, Message.NULL);
      NetMessage msg = new NetMessage.Ip(udpMsg, srcMaoddp.getLocalAddr(),
destMaoddp.getLocalAddr(),
          Constants.NET_PROTOCOL_UDP, Constants.NET_PRIORITY_NORMAL,
Constants.TTL_DEFAULT);
      srcMaoddp.getProxy().send(msg);
      //stats
      if (stats != null)
      {
        stats.netMsgs++;
      }
      JistAPI.sleep(delayInterval);
    }

  } // buildField


  /**
   * Display statistics at end of simulation.
   *
   * @param stats statistics collection object
   */
  public static void showStats(Vector routers, RouteMaoddp.MaoddpStats stats,
CommandLineOptions opt, Date startTime)
  {
    Date endTime = new Date();
    long elapsedTime = endTime.getTime() - startTime.getTime();
```

```
System.err.println("-------------");
System.err.println("Packet stats:");
System.err.println("-------------");

System.err.println("Rqdd packets sent = "+stats.send.rqddPackets);
System.err.println("Rqdd packets recv = "+stats.recv.rqddPackets);

System.err.println("Ack packets sent = "+stats.send.ackPackets);
System.err.println("Ack packets recv = "+stats.recv.ackPackets);

/*System.err.println("Rerr packets sent = "+stats.send.rerrPackets);
System.err.println("Rerr packets recv = "+stats.recv.rerrPackets);

System.err.println("Hello packets sent = "+stats.send.helloPackets);
System.err.println("Hello packets recv = "+stats.recv.helloPackets);*/

System.err.println("Total maoddp packets sent = "+stats.send.maoddpPackets);
System.err.println("Total maoddp packets recv = "+stats.recv.maoddpPackets);

System.err.println("--------------");
System.err.println("Overall stats:");
System.err.println("--------------");
System.err.println("Message to deliver = "+stats.netMsgs);
System.err.println("Route Query Data Delivery in process   = "+stats.rqddOrig);
System.err.println("Total Acknowledge Generated      = "+stats.ackOrig);
System.err.println("Total Routes added         = "+stats.rqddSucc);

System.err.println();
System.gc();
System.err.println("Free Memory:  "+Runtime.getRuntime().freeMemory());
System.err.println("Max Memory Available:   "+Runtime.getRuntime().maxMemory());
System.err.println("Total Memory: "+Runtime.getRuntime().totalMemory());
    long usedMem = Runtime.getRuntime().totalMemory()-
Runtime.getRuntime().freeMemory();
    System.err.println("Memory used:     "+usedMem);

    System.err.println("start time  : "+startTime);
    System.err.println("end time    : "+endTime);
    System.err.println("elapsed time: "+elapsedTime);
    System.err.flush();

    System.out.println(opt.nodes+"\t"
      +stats.send.rqddPackets+"\t"
      +stats.recv.rqddPackets+"\t"
      +stats.send.ackPackets+"\t"
      +stats.recv.ackPackets+"\t"
      +stats.send.rerrPackets+"\t"
      +stats.recv.rerrPackets+"\t"

      +stats.send.maoddpPackets+"\t"
      +stats.recv.maoddpPackets+"\t"

      +usedMem+"\t"
      +elapsedTime);

//clear memory
routers = null;
stats = null;
```

```
}

/**
 * Main entry point.
 *
 * @param args command-line arguments
 */
public static void main(String[] args)
{
  try
  {
    final CommandLineOptions options = parseCommandLineOptions(args);
    if(options.help)
    {
      showUsage();
      return;
    }
    long endTime = options.startTime+options.duration+options.resolutionTime;
    if(endTime>0)
    {
      JistAPI.endAt(endTime*Constants.SECOND);
    }
    //Constants.random = new Random(options.seed);
    Constants.random = new Random();
    final Vector routers = new Vector();
    final RouteMaoddp.MaoddpStats stats = new RouteMaoddp.MaoddpStats();
    final Date startTime = new Date();
    buildField(options, routers, stats);
    JistAPI.runAt(new Runnable()
        {
          public void run()
          {
            showStats(routers, stats, options, startTime);
          }
        }, JistAPI.END);

     //added
    /*
     JistAPI.runAt(new Runnable()
         {
           public void run()
           {
             Iterator itr = routers.iterator();
             while (itr.hasNext())
             {
               RouteMaoddp maoddp = (RouteMaoddp)itr.next();

               maoddp.printOutgoing();
             }
           }
         }, JistAPI.END); */

    }
    catch(CmdLineParser.OptionException e)
    {
      System.out.println(e.getMessage());
    }
}} }}
```