Performance Study of Real-World Wireless Mobile Ad Hoc Networks

Yağız Özen

Submitted to the
Institute of Graduate Studies and Research
in partial fulfilment of the requirements for the Degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
June 2010
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yılmaz
Director (a)

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Prof. Dr. Hasan Kömürcügil
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Asst. Prof. Dr. Gürcü Öz
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Işık Aybay
2. Assoc. Prof. Dr. Muhammed Salamah
3. Asst. Prof. Dr. Gürcü Öz

ABSTRACT

Wireless ad hoc network is one of the most popular network types these days. The reason for this is the advantages that wireless ad hoc networks provide for users or group of users. The most important characteristic of wireless ad hoc networks that make them more popular when compared with any other network type is that they do not need any infrastructure to be setup in advance. This characteristic of wireless ad hoc networks make the research on this topic more valuable due to increasing number of people using wireless ad hoc networks. The fact that no fixed router is used in the network ensures that network nodes are adaptable to the topology changes in a mobile wireless ad hoc network. This advantage makes wireless ad hoc networks useful in battlefield areas where there is need for networks that have a dynamic working strategy, which does not increase the complexity of setting a network. Other possible application areas of wireless ad hoc networks are disaster areas, rescue emergency operations and in vehicles that satisfies the required mobility and fast deployment network need.

This thesis provides extensive real-world experimental investigation of wireless ad hoc networks with mobile and stationary nodes in different outdoor environments. The performance of wireless ad hoc networks is measured under various scenarios. For the experimental investigations, more than one network configuration and different parameters were used in real-world outdoor environment. Conducting such experiments and gathering information regarding the results of these experiments will yield very valuable information since investigation of such networks requires two aspects to take into account. One of them is the simulation and modeling of these networks and the other is the conducting of real-world experiments by using testbed programs.

The most popular performance metrics for wireless ad hoc networks, delivery ratio, average round trip time or average end to end delay, and average number of hops were investigated in this study. It is seen that delivery ratio decreases with the distance between the nodes. The average round trip time is not affected by the distance; hence it increases with respect to application data size and the number of intermediate nodes in the network. The average number of hops changes if the distance between the source and the destination decreases since there will be no need for intermediate nodes for forwarding the packets.

Keywords: Wireless Ad Hoc Networks, Outdoor Experimental Study, Performance Evaluation, Multithreaded Programs, Wireless Ad Hoc Protocols.

ÖZ

Son günlerde, kablosuz ve altyapısız ağ bağlantıları en popular ağ bağlantılarının birtanesi olmayı başarmıştır. Bunun sebebi ise kablosuz altyapısız ağ bağlantılarının kullanıcılara sağladığı avantajlardır. Kablosuz ve alt yapısız ağ bağlantılarının popular olmasını sağlayan en önemli etken onların en önemli özelliklerinden biri olan, hiçbir alt yapıya dayalı olmamasıdır. Bu özellik sayesinde, bu ağların kullanım alanları günden güne artmakta ve bu ise bu konu altında yapılan araştırmalarda ulaşılan sonuçların çok değerli olmasına sebeb olmaktadır. Bu tür ağ bağlantılarında herhangi bir yönlendiricinin kullanımına ihtiyaç duyulmaması, bu ağ bağlantılarının kullanıcıların hareketli olduğu ortamlarda kullanılmasını mümkün kılmıştır. Kablosuz ve alt yapısız ağ bağlantılarının hareketli ortamlara kolay uyum sağlamasının getirdiği avantajla, bu tür ağ bağlantılarının savaş alanlarında kullanılabileceği akla geliyor. Diğer kullanım alanları ise, acil kurtarma operasyonları, felaket alanları, araçlar arası kullanım ve daha bir çok alan listelenebilir.

Bu tez çalışmasında, geniş kapsamlı gerçek dünyada yapılan deneysel çalışmalar sunuluyor. Bu çalışmalarda, bina dışında hareketsiz veya kullanıcılar tarafından taşınarak hareketli hale getirilmiş bilgisayarlar kullanılmıştır. Bu deneysel çalışmaların amaçı, kablosuz ve altyapısız ağ bağlantılarının birçok senaryo altındaki performansını ölçmektir. Birden fazla ağ konfigurasyonu ve parametreleri kullanılmıştır. Bu tür ağ bağlantılarının performans değerlendirilmesi iki farklı açı ile ele alınmalıdır. Birincisi simulasyon ve modelleme yapılmasıdır, ikincisi ise, önceden tasarlanmış program yardımı ile deneysel çalışmaların yapılmasıdır.

Bu tez çalışmasında en önemli ve en çok kullanılan ölçü birimleri ele alınmıştır. Bunlar ise, ortalama paket teslim oranı, ortalama sekme sayısı, göreçeli trafik ve bir paketin hedefine ulaşmak için harcadığı süredir.

Anahtar kelimeler: Kablosuz ve Altyapısız Ağ Bağlantıları, Deneysel Çalışma, Performans Değerlendirmesi, Çoklu İşlemli Programlar, Kablosuz ve Altyapısız Ağ Bağlantı Protokolleri.

TABLE OF CONTENTS

ABSTRACT iii

ÖZ… v

LIST OF TABLES ix

LIST OF FIGURES xii

1 INTRODUCTION 1

2 SURVEY OF ROUTING PROTOCOLS 4

2.1 Unicast Routing Protocols 7

2.2 Multicast Routing Protocols 9

2.3 Anycast Routing Protocols 11

3 SURVEY OF EXISTING EXPERIMENTAL STUDIES 14

3.1 Main Direction to Investigate Wireless Mobile Ad Hoc Networks 14

3.2 Experimental Study in Wireless Ad Hoc Networks 14

3.3 Challenges in Real-World Experimental Studies 27

4 TEST-BED PROGRAM 32

4.1 Purpose of the Program 32

4.2 The Structure of the Program 32

4.3 Collected Information 48

5 ORGANIZATION OF EXPERIMENTS 50

5.1 Experiments with Two Nodes 51

5.2 Experiments with more than Two Nodes 53

5.3 Experiments with One Source Node and Three Destination Nodes 57

6 EXPERIMENTAL RESULTS AND THEIR ANALYSIS 59

6.1 Performance Metrics 59

6.2 Results of Experiments 60

6.3 Discussion of the Experimental Results 86

6.4 Average Values and Confidence Intervals of the Investigated Performance Metrics 92

7 CONCLUSION 94

8 REFERENCES 97

APPENDICES 103

Appendix A: The source text of the Application-Layer Program 104

Appendix B: Raw Results of the Experiments 121

LIST OF TABLES

Table 2.1:Classification of wireless ad hoc routing protocols. 6

Table 3.1: Summarized information about the real-world experiments. 30

Table 3.2: Summarized information about the real-world experiments. (Continue) . 30

Table 6.1: The average values of round trip time which varies with distance under different application data sizes with using two nodes... 61

Table 6.2: The average values of delivery ratio which varies with respect to distance under different application data sizes with using two laptops. 62

Table 6.3: The average delivery ratio with respect to the inter arrival packet time under different application data sizes with using two laptops. 63

Table 6.4: The average round trip time values with respect to distance under different application data sizes with using two nodes. 64

Table 6.5: The average delivery ratio values with respect to distance under different application data sizes using two nodes and an obstacle between them in scenario one. 65

Table 6.6: The average round trip time values with respect to distance under different application data sizes using two nodes and an obstacle between them in scenario one. 66

Table 6.7: The average delivery ratio values with respect to distance under different application data sizes in scenario two. 67

Table 6.8: The average round trip time values with respect to distance under different application data sizes in scenario two. 68

Table 6.9: The average delivery ratio values with respect to distance under different application data sizes in scenario three. 69

Table 6.10: The average round trip time values with respect to distance under different application data sizes in scenario three. 70

Table 6.11: The average delivery ratio values with respect to number of intermediate nodes between the source and destination nodes, under different application data sizes. 71

Table 6.12: The average round trip time values with respect to number of intermediate nodes under different application data sizes. 72

Table 6.13: The average round trip time values with respect to data size under two different scenarios. 73

Table 6.14: The delivery ratio with respect to data size under two different scenarios. 74

Table 6.15: The average number of hop values with respect to application data size in two different scenarios. 75

Table 6.16: The average delivery ratio values with respect to distance and the application data size is fixed to 50 bytes. 77

Table 6.17: Average round trip time values with respect to distance when the application data size is 50 bytes. 78

Table 6.18: Average delivery ratio values with respect to distance when the application data size is 800 bytes. 79

Table 6.19: The average round trip time values with respect to distance when the application data size is 800 bytes. 80

Table 6.20: The average delivery ratio values with respect to distance when the application data size is 4000 bytes. 81

Table 6.21: The average round trip time values with respect to distance when the application data size is 4000 bytes. 82

Table 6.22: The overall round trip time values with respect to inter-node distance under different application data sizes. 83

Table 6.23: The overall delivery ratio values with respect to distance under different application data sizes. 84

Table 6.24: The average delivery ratio values with respect to data size under different height values from the ground. 85

Table 6.25: Average values and 95% confidence intervals of the performance metrics for application data size = 100 bytes. 92

Table 6.26: Average values and 95% confidence intervals of the performance metrics for application data size = 1000 bytes. 92

Table 6.27: Average values and 95% confidence intervals of the performance metrics for application data size = 2000 bytes. 93

Table 6.28: Average values and 95% confidence intervals of the performance metrics for application data size = 4000 bytes. 93

Table 6.29: Average values and 95% confidence intervals of the performance metrics for application data size = 8000 bytes. 93

LIST OF FIGURES

Figure 3.1: Positioning of the laptops during the experiment 18

Figure 4.1: Multicast mode for wireless network architecture. 33

Figure 4.2: Structure of the program. 35

Figure 4.3: A scenario of message passing in the wireless network of three nodes. 36

Figure 4.4: A simple wireless ad hoc network with three nodes. 37

Figure 4.5: The algorithm of the originating thread in the program. 38

Figure 4.6: A 128 bytes request datagram with data types (a), A 128 bytes reply datagram with data types (b). 39

Figure 4.7: Data structure of a request and a reply message counter. 40

Figure 4.8: The algorithm of the relaying thread in the test-bed program. 42

Figure 4.9: Algorithm for the relaying thread at the originator node after handling a reply message from the network. 43

Figure 4.10: Algorithm for the destination node after handling a request message from network. 44

Figure 4.11: Algorithm for the intermediate node after handling a message (request or reply) from the network. 45

Figure 4.12:   Received messages at the intermediate node within a sliding window consisting of 20 cells. 46

Figure 4.13: A scenario of messaging in the wireless network of four nodes. 47

Figure 5.1 : Configuration of the experiments with using two nodes. 51

Figure 5.2: A configuration of a wireless ad hoc network consisting of source node and destination node with a building, scenario three. 53

Figure 5.3: A configuration of a wireless ad hoc network consisting of five nodes.. 54
Figure 5.4: A configuration of a wireless ad hoc network consisting of five nodes in open area. 55
Figure 5.5: A configuration of a wireless ad hoc network consisting of ten nodes... 56
Figure 5.6: The position of source and destination nodes in the network. 57
Figure 6.1: Average round trip time versus distance with different data sizes. 61
Figure 6.2: The delivery ratio versus inter-node distance with different data sizes. . 62
Figure 6.3: The delivery ratio versus inter-packet transmission time with different application data sizes.. 63
Figure 6.4: The average round trip time versus inter-packet transmission time with different application data sizes. 64
Figure 6.5: The delivery ratio versus inter-node distance, for different application data sizes in scenario one. 65
Figure 6.6: The average round trip time versus inter-node distance, for different application data sizes in scenario one. 66
Figure 6.7: The delivery ratio versus inter-node distance, for different application data sizes in scenario two. 67
Figure 6.8: The average round trip time versus inter-node distance, for different application data sizes in scenario two. 68
Figure 6.9: The delivery ratio versus inter-node distance, for different application data sizes in scenario three. 69
Figure 6.10: The average round trip time versus inter-node distance, for different application data sizes in scenario three. 70
Figure 6.11: The delivery ratio versus the number of intermediate nodes between the source and destination nodes, for different application data sizes. 71

Figure 6.12: The average round trip time versus the number of intermediate nodes between the source and destination nodes, for different application data sizes......... 72

Figure 6.13: The average round trip time versus application data sizes between the source node and the destination node in an open area with different number of fixed nodes. 73

Figure 6.14: The delivery ratio versus application data sizes between the source node and the destination node in an open area with different number of fixed nodes. 74

Figure 6.15: The average number of hop versus application data size with different number of fixed nodes.. 75

Figure 6.16: The delivery ratio versus inter-node distance, for different directions with application data size is 50 bytes.. 77

Figure 6.17: The average round trip time versus inter-node distance, for different directions with application data size = 50 bytes. 78

Figure 6.18: The delivery ratio versus inter-node distance, for different directions with application data size = 800 bytes. 79

Figure 6.19: The average round trip time versus inter-node distance, for different directions with application data size = 800 bytes. 80

Figure 6.20: The delivery ratio versus inter-node distance, for different directions with application data size = 4000 bytes. 81

Figure 6.21: The average round trip time versus inter-node distance, for different directions with application data size = 4000 bytes. 82

Figure 6.22: The average round trip time (overall) versus inter-node distance for different application data sizes... 83

Figure 6.23: The delivery ratio (overall) versus inter-node distance, for different application data sizes.. 84

Figure 6.24: The average delivery ratio versus application data size under different height of the laptops to the ground level. 85

Chapter 1

INTRODUCTION

Nowadays, wireless networks are one of the most popular computer networks which use radio frequency channels to communicate between the nodes in the network without using any wire. One of the most important benefit of wireless networks is that they do not require any wire to connect the nodes to each other. Computers in home or anywhere else can be connected easily by means of wireless cards. There are two types of components used in many kinds of wireless networks: wireless routers and access points.

Ad hoc wireless network is one of the wireless networks that enables the users of the network to directly communicate with each other. This means that ad hoc wireless networks do not need any routers or access points to be used in the network. Since there is no wire and no fixed router in this kind of networks, it is not difficult to enable the mobility in the network because the nodes will arbitrarily arrange themselves with respect to the topology changes. The transmission area of each node is limited. Hence, in order to reach a node that is out of a node's transmission area, another node should be used as intermediate node in order to forward the needed information. Since there is no any router or access point, every node inside the network can work as a router and fulfill the responsibility of forwarding information. This means that there will be a multi-hop wireless link between the sender and the receiver.

The earliest mobile ad hoc networks (MANETs) were called “packet radio” networks, and they were sponsored by DARPA in the early 1970s[1]. SURAN (Survivable Adaptive Network) was proposed by DARPA in 1983 to support a larger scale network [1]. The idea of multi-hop links in ad hoc networks dates back to 500 B.C., Darius I who was the king of Persia and inventor of multihop communication system. For sending messages and news, he yelled to his men who were located at tall structures in each remote province of his empire. This new communication system was 25 times faster than the regular messaging system of his time.

Since each node in wireless ad hoc networks can play the role of being source, destination, and a router, each node in the network needs to be intelligent. This intelligence is figured out by a routing protocol that is used for packet transmission between the network nodes. If the routing protocol of a network is well configured, it will increase the efficiency of the network. Since the wireless ad hoc networks have limited bandwidth, power consumption problem and mobility [2], the routing protocol should be simple, power conserving and capable of handling fast topology changes in the network configuration.

Easy and fast deployment of wireless ad hoc networks and the decreased dependence on infrastructure makes this type of networks preferable in some areas. Besides being used as cell phones and for gaming purposes, wireless ad hoc networks can also be used in disaster areas or in search and rescue emergency operations. In our daily life, using wireless ad hoc network in taxis, stadiums and aircrafts is also possible as for military reasons, these networks can be deployed on battlefield areas because they are good at mobility, it is fast and easy to setup.

The purpose of this thesis is to investigate the characteristics of wireless ad hoc networks under different conditions with the use of some performance metrics. In order to be able to investigate the characteristics, we carried out a series of experiments in outdoor real-world network environment by the use of the developed program.

The rest of the thesis is organized as follows. Chapter 2 presents a classification of the routing protocols in wireless ad hoc networks to explain the behaviors of routing protocols in various conditions. In Chapter 3, information is given about what was done about conducting real-world experiments by using wireless ad hoc networks. In Chapter 4, detailed information is given about the testbed program which was used in the experiments. Chapter 5 describes the experiments that were conducted by our research group. Chapter 6 presents the results of the conducted experiments, a discussion of these results and presents information about the parameters and the performance metrics. In Chapter 7, the study is concluded.

Chapter 2

SURVEY OF ROUTING PROTOCOLS

Wireless ad hoc network is a collection of nodes that communicate with each other without requiring a hardware component such as a router for centralized control. Any node in a wireless network can be a source node, an intermediate node which acts as a router, and a destination node. The main characteristics of a wireless ad hoc network can change with respect to the selected routing protocol.

There are lots of routing protocols in the literature and we can mainly classify them as unicast, multicast and anycast. Unicast means sending the packets to a single destination host from a source node in a specific network. There is a one to one relationship between the source and the destination node. On the other hand, multicast means sending a packet from a source node to a group of nodes in that network. In this kind of network, each node has a multicast address and more than one node can have the same multicast address in the same network. Thus, when a packet is sent to a specific multicast address, a group of nodes receive this packet if they belong to that address. Multicast has a one-to-many association between network addresses and network endpoints. Lastly, anycast means sending a packet from a source node to the nearest server or to the best localized server in the network. In anycast mechanism, there is one or more server(s) in the network and the aim is to send the packet to the best server among all the other servers (if there is more than one). And the word “best” can vary with respect to the anycast protocol that is used.

It can be the nearest node, the least traffic involving server or any other thing depending on what system you are using.

These three categories can also be divided into three subcategories in themselves as reactive protocols, proactive protocols and hybrid protocols. Reactive protocols can also be named as on demand protocols which have a mechanism of finding a route from a source node to a destination node(s) when a source node want to send a packet. This means that generally a route discovery mechanism is activated before sending the original data to the destination to find out the route that is going to be used for sending data. Moreover, there are two kinds of reactive protocols. The first one works by combining the entire route address with the original data after finding the best route and sending the whole packet. The intermediate nodes do not need to care about to which node they need to forward the packet since that information will be provided inside the packet with the data that we aim to send from the source node. The second type of reactive protocols works by setting a routing table inside each intermediate node. And each time a packet goes to an intermediate node, the current node will decide where to forward the packet by looking at the table inside it. The difference of this mechanism derives from putting the next hop address in the packet instead of putting the entire route information.

In proactive protocol, each node maintains the routing information for every node in the network. Depending on which protocol is being used, the number of tables required for keeping the routing information can vary. The important thing is that, the tables are kept updated periodically even if there is no need for a data transmission from a source node to any destination node.

Table 2.1:Classification of wireless ad hoc routing protocols.

	Unicast	**Multicast**	**Anycast**
Reactive Protocols	AODV DSR TORA LAR ABR	MAODV ODMRP ABAM	A-AODV ARDSR
Proactive Protocols	DSDV WRP LANMAR OLSR STAR APRL	AMRIS AMROUTE CAMP MOLSR	Route-Count Based Anycast Routing Protocol
Hybrid Protocols	ZRP HARP ZHLS	ZMAODV ZODMRP MZR	Hybrid Anycast Routing Protocol

The combination of reactive and proactive protocols forms hybrid protocols. In most hybrid protocols, a zone-based mechanism is used for dividing the network into zones. The node(s) that are close to the destination node work like a proactive protocol and periodically send information to the neighbor nodes for keeping their routing tables up to date. The nodes that are far enough, work like a reactive protocol by sending route discovery messages to the network. As a result, the route discovery process takes less time and less overheads with respect to the other two types of protocols. More information can be found in [2][3] about reactive, proactive and hybrid protocols, their comparisons and some classifications.

In Table 2.1, the classification of some routing protocols with respect to their transmission type and working mechanism is given. At least one protocol from each part of the table will be explained below.

2.1 Unicast Routing Protocols

Ad hoc on-demand distance vector (AODV) [4] is a routing protocol which establishes a route to a destination node only when it is necessary. This means that AODV is a reactive protocol. It is based on DSDV and DSR [5] algorithms. It uses a route discovery mechanism for finding a route to a destination whenever it is needed and also it uses sequence numbering procedure. Once a route is found to a destination, this route is used for future data sending. In the route discovery procedure, the source node sends a route discovery message to the network and whenever the destination node receives this message which is flooded by the source node, it sends a reply back to the source node with the same path.

Dynamic Source Routing (DSR) protocol is also a reactive protocol in unicast class. It uses a route discovery mechanism since it is reactive, and the main difference between DSR and AODV is that, while sending the data packets from a source node to a destination node, the packets in DSR carry the complete address of the route which they will use for travelling until their destination. However, AODV only carries the destination address in each of the data packet that is sent out by the source node. This provides some advantages to AODV protocol in high dynamic networks with large number of nodes since the routing overheads of AODV protocol are less than the DSR protocol’s routing overheads. On the other hand, DSR has an important advantage in saving more than one route to the source’s cache and whenever the network switches from listening mode to transmitting mode, it checks the cache of the sender to find a valid route to a destination instead of directly initializing a route discovery procedure.

Optimized Link State Routing (OLSR) protocol [6] has a unicast and a proactive working mechanism. Each node in the network exchanges Hello and Topology Control (TC) messages between them in order to keep the topology information up to date periodically. Since it is a proactive protocol, even though there is no need for a transmission in the network, the nodes will know where to send a packet in case of a need for transmission at any time. One of the features of OLSR protocol is that, it manages to send the control packets in such a way that the packets will not be retransmitted after a predefined value which is called Multipoint Replaying (MPR) strategy. Only the predefined set of nodes can retransmit the TC packet in the network but other nodes cannot.

Source-Tree Adaptive Routing (STAR) protocol [7] also works in one-to-one manner and it is a proactive protocol. It is similar to OLSR, but in STAR, Least Overhead Routing Approach (LORA) is used to exchange routing information. The aim of the LORA approach is to reduce the amount of routing overhead used in the network. Normally, the control packets are periodically exchanged in the network to see the topology changes in the network but in this case they will be exchanged depending on some conditions. In [3], it is stated that STAR can have a large amount of memory and processing overheads in large and highly dynamic networks. This is because each node needs to create/update a partial topology graph of the network.

The working mechanism of APRL [8] is proactive. Each node in the network knows about a route to any of the other node in the network. And this is handled by broadcasting routing beacons to network by every single node. This beacon contains the exact copy of the sender’s routing table. The nodes that receive any beacon use the information inside them for renewing their current routing table and then propagate the renewed information to network. Nodes use the route that they learn about first, without considering the length or quality of the route. Any route in the table will be timeout if within a specific time that node does not receive any beacon. Also APRL records some predefined number of alternative route as soon as a primary route times out.

The Zone Routing Protocol (ZRP) [9] is a hybrid type protocol. The network is divided into routing zones in this protocol. The nodes that are at maximum “d” distance from the node “N”, belong to the same routing zone of “N”. Since hybrid protocols are the combination of reactive and proactive protocols, the mechanism of proactive is used inside the routing zones and the reactive mechanism is used for the communication of different routing zones. Route discovery process of ZRP is very similar to DSR protocol. The aim of this hybrid protocol is to reduce the control overheads of proactive protocols and the time required for finding an optimal path to the destination.

The routing protocols that are described above belong to unicast class. Some developers and researchers modified some of these protocols and created a new routing protocol that belongs to Multicast and Anycast classes. Some of the multicast and anycast protocols will be briefly described below.

2.2 Multicast Routing Protocols

The Multicast of Ad hoc On-Demand Distance Vector (MAODV) protocols [10] is one of the modified unicast protocols to fit in the multicast class. It is a reactive routing protocol. AODV protocol which is a unicast protocol is extended and MAODV is formed. MAODV provides the advantage of dynamic and multihop routing between mobile nodes. In MAODV, each node has three tables. The first one is Routing Table (RT) which works exactly in the same way with AODV. The second one is Multicast Routing Table (MRT) which contains the information about the multicast group addresses and the hop counts to the multicast group leader and to other multicast group members. The third table is the request table which provides required information for the optimization. This protocol shares many common features with AODV.

The On-Demand Multicast Routing Protocol (ODMRP) [11] is also a reactive multicast protocol that creates routes on demand. For multicast packet transmission, forwarding group mechanism is used. Each multicast group is related with a forwarding group and the nodes in that forwarding group are responsible for forwarding multicast packets of the multicast group. Protocol has two main phases like in unicast reactive protocols which are the request phase and the reply phase. If there is no route known for transmission of a packet, Join Request packet is delivered to the entire network. More information can be derived from [11].

Multicast Optimized Link State Routing (MOLSR) protocol [12] is the extended version of the OLSR protocol. It is a proactive multicast protocol using mainly two methods while delivering the data to a group of destination nodes, two methods are mainly used. These are tree-based or mesh-based methods. MOLSR involves tree-based method. Multicast trees are built with the use of the exchanging of topology control messages which are used in OLSR to see how the topology changes with respect to time. These trees are updated whenever a topology change is detected in the network. With the use of these trees, the shortest path to the necessary destination(s) is found by the MOLSR.

Multicast routing protocol is based on Zone Routing (MZR) [13] protocol which is a multicast protocol and at the same time it has a hybrid mechanism. MZR is a source initiated on-demand protocol. With the use of the zone routing mechanism, it creates a source based multicast delivery tree. It means that whenever a data need to be sent to a multicast group, the creation of tree is triggered by the request for sending data. The creation and maintenance of tree mechanisms in ZRP is used in MZR. The reactive mechanism of ZRP is for the creation of source based tree and the proactive mechanism is for keeping the zone routing table up to date by sending advertisement messages periodically. The zones are created in the network depending on the hop distance of a node. Having a pure proactive mechanism can corrupt the network in terms of bandwidth. For this reason, instead of a pure proactive mechanism, a combination of proactive and reactive mechanism can be used to prevent the occurrence of bandwidth problem.

2.3 Anycast Routing Protocols

Anycast Ad hoc On Demand Distance Vector routing (A-AODV) [14] protocol is based on AODV protocol. AODV protocol is extended to enable A-AODV support anycast function. A-AODV discovers routes only when it is needed. It is a reactive protocol. Routing tables and RREQ packets in AODV are modified for A-AODV. Anycast Group ID is added to the routing table entry. If a route is needed for transmission and if there is a route to any anycast server, it will be used. If there is more than one route available, it will choose the route with the smallest hop counts that is nearest to the server. If there is not more than one route available, RREQ message is generated and it is proceeded to route discovery process.

Anycast Routing based Dynamic Source Routing (ARDSR) [15] protocol is the extension of DSR protocol. The protocol is a reactive protocol like A-AODV and it is an extension for the anycast networks. The routes are created only when they are needed for data transmission. ARDSR has two phases which are route discovery and route maintenance. When a data is needed to be sent, the source first checks its cache if there is any route. If no route is found, ANYREQ is flooded to the neighbors. At one point, when the destination receives this message, ANYREP message will be replied by the anycast server. Moreover, these routes that will be stored on caches need to be maintained since the network can be mobile. There will be some link breakages because of the mobility. In this kind of a situation, RRER message is sent to the source to tell that the link is broken. There are some references that compare the performance of A-AODV and ARDSR in the literature. [16] is one of them.

In [17], an anycast proactive routing protocol is proposed. The proposed protocol works in a proactive manner. Routing tables are recorded at every node and hop count. Route count and lifetime are recorded for each node. Each node periodically sends control messages to the neighbor nodes since it is a proactive protocol. With the use of these control messages, nodes find out the shortest distance to one of the anycast group members and also at the same distance, they count how many different routes there are. When a node needs to forward packet, the packet is forwarded to the shortest distance anycast member. If there are two anycast members at the same distance, as second criteria, the packets are forwarded to the anycast group member with the larger “number of routes” variable. This gives the advantage of having more stable routes to the anycast server.

A Hybrid Anycast Routing protocol is proposed in [18] for load balancing in heterogeneous access networks. The proposed protocol is based on AODV protocol. Some important modifications are made on AODV to support anycast routing. The modification involves the combination of reactive and proactive mechanisms. The protocol consists of two regions. The first one is the proactive region and the second one is the reactive region. Proactive region surrounds the nodes that are “m” hop away from the anycast server and all the other nodes that do not belong to a proactive region but belong to a reactive region. The working mechanism of the protocol consists of five phases, Hello message transmission, Route discovery for proactive region, Route discovery for reactive region, Route selection and Route maintenance. Hello message transmission is done by access points (anycast servers) to make the nodes aware they belong to a proactive region. Only the nodes that are inside the “m” hops of distance can receive this message and distinguish themselves as a member of a proactive region. The process of receiving the hello messages and setting them as a member of a proactive region is the second phase (Route Discovery for proactive region). Route Discovery for reactive region works in a similar way with AODV route discovery but this one has more fields inside the RREQ and RREP packets. Route selection phase is done for choosing the best route to forward the packets. A cost metric is used in the protocol to make a healthy decision and the Route maintenance is same as the AODV’s route maintenance. In [18], some experiments are done to show the performance of the proposed protocol.

Chapter 3

SURVEY OF EXISTING EXPERIMENTAL STUDIES

3.1 Main Direction to Investigate Wireless Mobile Ad Hoc Networks

In order to investigate a wireless ad hoc network's performance, two aspects need to be considered. One of them is the real-world experimental part and the other one is the simulation modeling. It will not be enough to make only the simulations for the performance measuring of wireless ad hoc networks. The reason for this is that; the environmental effects cannot be applied in the simulations exactly in the same way as in the real-world's environmental conditions. The results of real-world experimental studies can be very important for understanding the wireless ad hoc network's performance. The real-world experimental investigations require the use of a large number of computers, good test-bed software on these computers and most importantly man power to control each computer. However, finding the necessary people for deploying such an experiment may be difficult. The next difficulty in real-world experiments is that when repetition is needed for a conducted experiment, you may not find the same environmental conditions since the environmental conditions cannot be controlled by the experimenter.

3.2 Experimental Study in Wireless Ad Hoc Networks

Some assumptions are made in simulations and these assumptions can sometimes lead to incorrect results. In [19], it was stated that some of the assumptions that are made in simulations are not always correct in the real-world experiments. In the literature some real-world experiments were conducted in order to prove that some assumptions are not true when the real world environment is considered.

In [19] a group of outdoor experiments were conducted with 33 laptops and each laptop had its own GPS device to receive signals from the other nodes containing the coordinates of the node itself. In order to examine the axioms, extensive log files which keep related information for nodes' positions were created. The first axiom claims that "world is flat". In some simulation models, it is assumed that the world is flat; but it cannot be true. In the real-world, there are hills and buildings and these can be counted as an obstacle which considerably affects the radio signal propagation. The second axiom is that "A radio's transmission area is circular". In theoretical analysis, it is assumed that the radio signal's transmission area is circular and it is not exactly the same in the real-world. In the paper [19], it was stated that the angle between the wireless cards on a laptop to another laptop's wireless card affects the transmission area. Another axiom is "Signal strength is a simple function of distance". They took into consideration only received beacons and recipient's signal log to obtain the signal strength associated with that beacon. When the signal strength of individual beacons was investigated, it was noticed that there is not any simple function that will predict the signal strength of an individual beacon based on the distance alone. In [19], the simulation results were compared with the outdoor results that were derived from the outdoor experiments.

In [20], some experiments were conducted for understanding the capacity of the radio medium, the asymmetry of the used cards and the effect of broadcast on unicast flows and the interfering range. Linux operating system was used on every laptop and UDP packets were sent with the implementation of CBR (Constant Bit Rate). A toolbox software was developed to deploy different scenarios. In this group of experiments no routing protocol was used. Without using any routing protocol 802.11 performance was measured. Each node has a predefined table consisting information about the nodes that will send the packets. Furthermore, the developed software monitored many parameters during the experiments such as the time of the packet that was sent with the information by which station it was sent, the time of received packet and by which station it was received with which power and the noise level information of that time being, packet flow ID and sequence number within the flow and last-hop identificator. With the use of the software, things that were not considered in simulations were investigated their importance was highlighted in the real-world by conducting some experiments.

A group of experiments was conducted which helped us to understand certain issues that need to be considered in real world experiments. One group of experiments was about the effect of positioning the laptops in the network. For this group of experiments, two laptops were used and no forwarding mechanism was used since there would be only one hop in the transmission. They examined four different positioning of those two laptops to see the effect of throughput. Each stage lasted 130 seconds. During each stage, they changed the packet sizes as 200, 500, 1000, 1400 and each was sent for 20 seconds. The remaining time was used to increase the distance between the communications to 15 meters. The configuration of the laptops can be seen in Figure 3.1. In position 1, as it can be seen, the wireless cards of the laptops were facing the same direction. In position 2, the cards were set in opposite directions. In position 3, the cards were facing each other and in the last position, as it can be seen the cards were facing the opposite direction with one laptop’s LCD facing directly back of the other laptop’s LCD.

The throughput was measured and the best throughput in terms of bits/second was seen at position 3 where the cards were looking at each other. The worst throughput was observed at position 2 in which the cards were facing completely the opposite sides. In simulations, these kinds of things are not generally taken into account since it needs to be careful while conducting experiments in real-world.

The other group of experiments was done in the paper [20], which is about sharing the medium. The experiments were done with two stations which tried to send data to two different stations that acted as a receiver. Then, the network was set in a way that the two sender stations sent the packets to the same destination. After that only one station sent data to only one destination. The results of these experiments were investigated to see how it affected the communication.

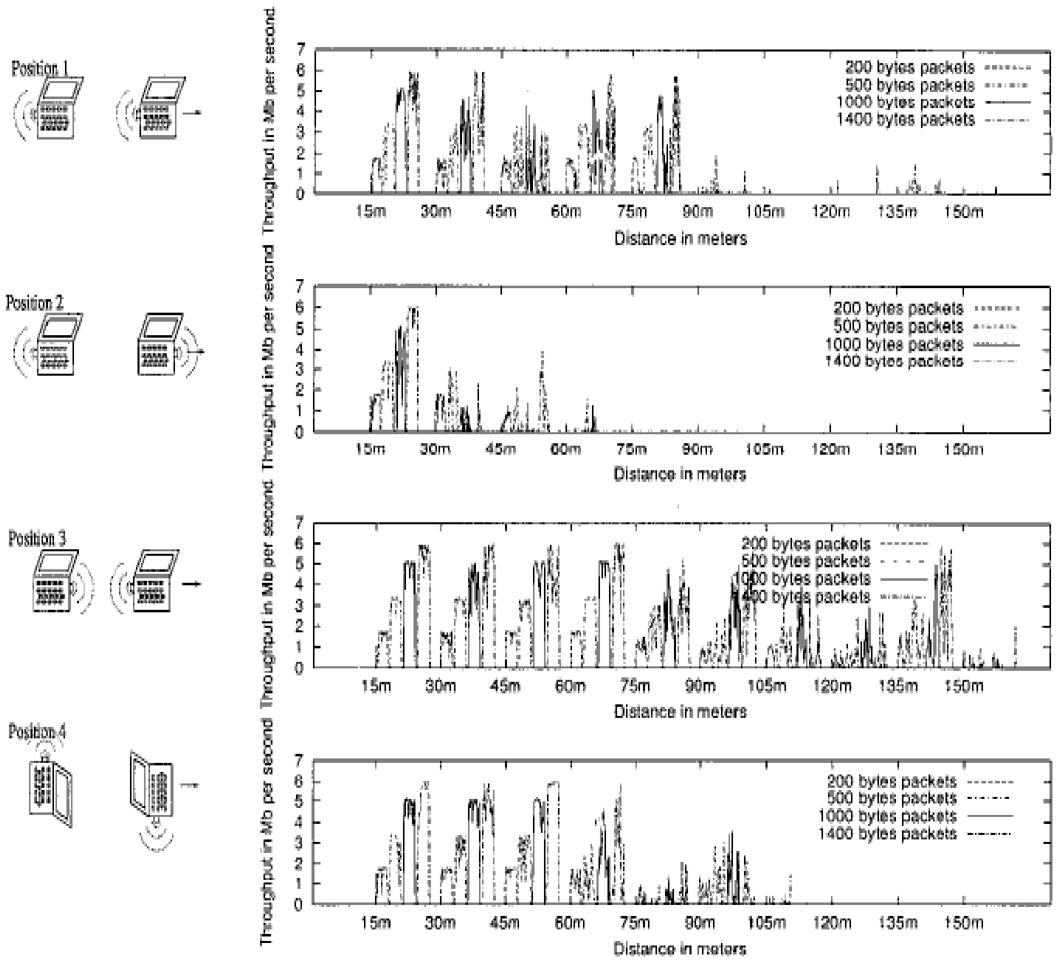


Figure 3.1: Positioning of the laptops during the experiment [20].

Another group of experiments was done to see how the throughput varied when the number of stations that transmit data in the network was increased. The experiments were started with one transmitter and one receiver station, and the number of transmitter was increased step-by-step until seven transmitters. At the end of the experiments, it was noticed that the throughput increases with respect to the number of transmitter. The reason of this can be explained by thinking the idle time of the receiver during the experiment. When receiver receives one packet of information, it will be inactive until the arrival of the next packet. However, if the number of transmitter is increased, this idle period will decrease since the packets that arrive at the receiver increase per unit time. It is called parallel decrease of back offs which leads to achieve better throughput when the number of transmitter increases. It was seen that when two nodes were communicating at max range (189 meters), the bandwidth was fully used by the monitored destination and when they were close to each other the bandwidth was not fully used.

They compared the simulation results with the outdoor results that they achieved. Therefore, from the [19] [20], it should be understood that before conducting any experiment in the real-world, the assumptions that are used in simulations shouldn't be used.

Other researchers stated in [21], that the common assumptions of route symmetry in simulations of ad hoc networks are not true in real-world experiments. They found that when the number of hops increases, the chance of any route to be symmetric decreases. In [21], they used 16 laptops equipped with IEEE 802.11b and 802.11g network interface cards. Four of these laptops were used to generate real-time and non-real time UDP traffic to all other nodes in the network. The protocol that they used was Optimized Link State Routing (OLSR). They used the default parameter values of OLSR which can be found in [23]. They were doing the experiments for winning the Mobile Ad hoc Network Interoperability and Cooperation Challenge 2007 (MANIAC) which is a multi-institution competition. Two of the laptops were used for monitoring the network traffic and the topology. They used a monitoring tool for this purpose which will be explained later on. They also designed an application which was used for making the dynamic changes in routing and forwarding decision by the people who played role in conducting experiments easier and in collecting traffic and routing data at each node more efficiently and easily. The application gives the ability to drop, forward or redirect traffic. Each team analyzed each packet and decided what to do with it. The options are; forwarding it according to the routing table, dropping the packet or redirecting the packet to a different next hop other than the specified entry in the table. Each team controlled two nodes during the experiments. Moreover, the program stored some information about the routing table, number of packets accepted, dropped and forwarded by each node. The paper focused on topology and routing subjects. The result that they achieved showed that a high degree of topology and route changes occur, even when there is low mobility. From the results, it is understood that routing proactively in a real ad hoc network is extremely difficult, because when the route is more than one hop, it is asymmetric.

In [21], information was given about a monitoring tool that was used in the experiments. In [22], they developed this monitoring tool and used it in some experiments. As it is mentioned, this Monitor for Mobile Ad hoc Networks (MMAN) tool was used to gather information about the network for constructing partial network views. Moreover, the good thing of this tool is that, it does not generate any additional traffic in the MANET which it monitors. And it doesn’t require much storage and processing resources. This tool can be used for network management, security assessment and anomaly detection. OLSR protocol is used in this tool. A number of monitoring units was distributed in the network and the units were equipped with two network interface cards. One of them was used for MANET packets and the other one was used for the communication of the packets between the other Monitoring Units (MUs). These MUs collected information about the network topology, link changes and delivered this information to the management nodes. The advantage of using two network interface cards is that no additional traffic was generated in the MANET. They conducted experiments in two settings.

One of them which was with 10 nodes MANET was deployed across a large house while some of the nodes were inside the house, some of them were outside. And in the second setting, 10 nodes of MANET were deployed in an office building. Some of the node's operating system were different. All of them were Linux but not the same version. One node had Fedora Core 5, four nodes had Fedora Core 4 and five nodes had Slack ware Linux 10.2. The performance of MANET was investigated in two scenarios. In one of the scenarios only one MU was used in the MANET whose coverage area was partial. In the second scenario two MUs were used in MANET covering the 80%-90% of the MANET. In the environment, there were obstacles such as walls, other electrical devices as well as the wireless networks, shadowing and interference. Experiments were run for 6 periods and each of the periods took 30 minutes. They tested the performance of MMAN under different networking conditions such as; with different network densities, partial and complete coverage of the MANET, node's cooperation levels and different traffic rates in a real world environment. They concluded that MMAN had been successful for all the scenarios. More information can be found about this monitoring tool in [22].

In [24], outdoor experiments were conducted for comparing four different routing protocols. These were APRL, AODV, ODMRP and STARA. They used 33 802.11-enabled laptops moving randomly in a field. In addition to this, they compared the outdoor results with both indoor and simulation results for all four algorithms. For brief information about these four algorithms, please refer to Chapter 2.

Computers used in the experiments had 10GB Hard Disk, 128MB of main memory and a 500 MHz Intel Pentium3 CPU with 256 KB of cache. They all ran Linux kernel version 2.2.19 with PCMCIA card manager version 3.2.4 and had Lucent

(Orinoco) Wavelan Turbo Gold 802.11b wireless card. There were some common parts in each of these four algorithms. All four algorithms were implemented to application layer through the use of a tunnel device. They were using UDP for the traffic between a specific neighbor and multicast IP for traffic to reach every neighbor. All four algorithms were implemented in C++ and shared a core set of classes.

They implemented a traffic generator to each node in the network. By using this traffic generator, a sequence of packet streams was sent to a randomly selected node in the network. For determining the destination node, a uniform distribution was used. For the time between the streams and packets, exponential distribution was used. And for determining the number of packets and the sizes, Gaussian distribution was used. The traffic generator on each laptop generated packet streams with a mean packet size of 1200 bytes and the approximate value of the mean of the packets per stream was 5.5. The mean delay between streams and packets was approximately 15 seconds and 3 seconds respectively.

Outdoor experiments were done in a rectangular area of 365 meters long by 225 meters wide. The area of the experiments was far enough from the campus wireless to prevent any interference. They used GPS service on each laptop which recorded the current position once per second and synchronized the laptop clock with the GPS clock for time synchronization. Every 3 seconds, GPS service on each laptop broadcast a beacon containing its own coordinate and any other coordinates that it knew about the other nodes. The parameters were set according to the published simulation studies which gave effective results. APRL broadcast its beacon every 6 seconds and any route which had not been refreshed by a beacon within the last 12 seconds expired. And STARA broadcast a NP every 2 seconds. If a path was not explored for 6 seconds, it sent a dummy data packet. If NP_ACK didn't come twice from a neighbor, it was removed from the list. AODV broadcast each RREQ twice and a route expired if it is not used for 12 seconds. Hello packets were sent every 6 seconds and if two successive hello packets were not received by a neighbor, they were removed from the neighbor set. The movements of the laptops were handled by dividing the field into 4 parts. Experimenters chose a position randomly between the parts that they were not currently in, and walked to that position and repeated the same steps after reaching there. Message delivery ratio, communication efficiency, hops count and end-to-end latency were used as performance metrics.

There are many routing protocols for mobile ad hoc networks, but there are not many protocols which also consider the secure routing in MANET. In [25], they modified the existing AODV protocol and proposed SAODV (Secure-AODV). Since AODV protocol does not concern any security system, it is vulnerable to some types of attacks. In this reference, they introduced a "malicious node" and stated whether a node is an attacker node without having enough information about its type. On the other hand, if the node has enough information about its type, it is counted as a legal node. There are mainly three different ways of attacking a network according to this paper.

The first one is "Message Tampering Attack". Attacker can change the content of routing messages and forward them with wrong information. For instance, one aim of the attacker can be analyzing the communication between the source node and a destination node. The only way of analyzing the communication between the source node and the destination node during the whole data flow process is to make sure that the information that is being sent passes this specific route that the attacker can analyze. If the attacker decreases the hop count information, it will increase the chance that the packers will flow on that specific path. Moreover, the destination sequence number can be increased by an attacker in order to make the other nodes believe that this is a “fresher” route.

The second type of attack is “Message Dropping Attack”. The attacker nodes are set to drop some or all data information that is passing through them. As it is known, in ad hoc networks each node can play the role of end hosts and routers, so dropping the packets can paralyze the network with respect to the number of message dropped.

The third type of attack is the “Message Reply (or Wormhole) Attack”. Attackers can retransmit secretly listened messages again later in a different place. Wormhole attack is one of the reply attacks. Wormhole attacker can send the RREQ message directly to the destination node to prevent any other routes from being discovered.

There are some security requirements in the protocol. Source authentication is one of them and its aim is to verify that the node is the one that it says to be. The other one is the Neighbor Authentication and its aim is to ensure that the receiver should check the identity of the sender and be sure that the sender really tells the truth about itself. The other one is the Message Integrity which is used to verify that the data which is routed has not changed during the routing process. The last one is the Access Control which checks the rights of the nodes that are trying to access the network. The proposed SAODV protocol uses digital signatures to verify whether the information that does not change in the packets is true or false. Also hop count is being checked in RREQ and RREP messages.

Some experiments were done to see the performance difference between AODV and SAODV. The experiments were done in indoor environment with some parameters. For instance, bit rate for 802.11b MAC is 11Mb/s. For AODV and SAODV, HELLO packets are sent every 1 second. Link will be counted as broken if HELLO packet is not received within 2 seconds. For SAODV, additional size for RREQ, RREP and RERR are 448,448,404 bytes respectively. 448 bytes include signature, top hash, hash, certificate, other header info. For 404 bytes includes signature, certificate and other header info are included. The laptops that were used for experiments had Intel Pentium M 1.6 GHz CPU with 1024 KB cache more than 60 GB Hard disk and 512 MB RAM. Totally 6 laptops were used and each of them equipped with an internal 11 a/b/g wireless LAN mini PCI adapter. The operating system was Windows XP version 2.0. The indoor room had 17mx7m area and the laptops were placed in the same lab. The speed of the mobility was 0.5 m/s and each session took 15 minutes. Data rate was 11 Mb/s with auto-rate function disabled. Minimum transmission power mode was used and the transmission range was 50m. Each user held the laptops and walked randomly in the room. During the experiments the amount of control overheads (RREQ, RREP, RERR) that was generated was collected. When each time a control packet was forwarded, it was counted as one transmission. For TCP traffic the average throughput was used. Average TCP throughput for AODV-withAttack, withoutAttack and SAODV-withAttack, withoutAttack are the performance metrics.

In conclusion, SAODV is effective in preventing control message tampering and data dropping attacks under TCP traffic. All the information that is written about the security system of a protocol will be future work. And it will be extended to support more types of attacks.

There are some routing protocols that look for the shortest path by checking the delay of packets such as; AODV and DSR. Moreover, some of them check the signal strength. In [26], a new criterion was introduced for choosing a better route. Joint route hop count, node stability and route traffic load balance were the criteria for choosing the best route among all other routes. In [26], AODV and SAR protocols were compared and the performance metrics used in the paper are, delivery ratio, end-to-end delay, control cost, hop counts and they are all versus traffic load. An overview of SAR is as follows.

In SAR, when there is more than one route, it selects the best one with its union selection parameter W, which jointly considers, hop count, stability of the route and traffic load of the route.

For the experiments, two laptops were used for measuring the transmitting capacity of single node. Two nodes were placed very close to each other and one of them was set to send packets to the other one without any routing. On the computers wireless LAN card was used, based on IEEE 802.11b standards and the WEP function was disabled on the cards. Packet length was fixed at 1024 bytes. End-to-End delay versus Traffic Load performance metric was used for this experiment.

The other experiment was done in indoor environment. The four laptops were placed in 8mx8m office and the source, one intermediate and the destination node were not moving. Only one intermediate node was moving between the source and destination. Since they couldn’t change the transmission power of the laptops they did the mobility in that way. For outdoor experiments that they conducted, they didn’t use any mobility. Four nodes were placed 20m-30m away from each other and the source node and the destination node were selected randomly among those four laptops. For indoor experiments, they used delivery ratio, end-to-end delay, control cost and end-to-end delay jitter performance metrics that were measured with respect to the system traffic load. For outdoor experiments, instead of end-to-end delay jitter, they measured hop count performance metrics with respect to system traffic load. They compared the results that they found with the AODV protocol results. By looking at the outputs, it was understood that SAR has more efficient results.

3.3 Challenges in Real-World Experimental Studies

Dealing with real-world experiments can be really challenging. The most challenging factor in the experiments is the environmental conditions on the transmission of the wireless signals. Since the experiments are conducted outdoors, the buildings, cars, people walking around and even the electricity poles can be counted as environmental effects that dramatically affect the propagation of the wireless signal. The presence or absence of these obstacles is a crucial factor for choosing the environment where experiments are to be conducted. It should be away from any building to prevent the risk of interference. Also there should not be car traffic around the experiment environment since they affect the propagation of wireless signals. The electricity poles create a huge magnetic area which can affect the results of the experiments in a bad way. Furthermore, there should not be another wireless service around the experiment area for preventing the inference that they can generate. One of the most challenging factors in conducting real-world outdoor experiments is finding an enough large area that satisfies the criterias required for achieving results that show the pure behavior of wireless ad hoc networks. After finding such an area, the rest is not very simple. Conserving the battery life of the laptops is also a challenging factor in the real-world experiments since there is no power supply in the fields to recharge the batteries of the laptops. Just before starting the testbed that is developed, the laptops should be connected to the same wireless server which is created by any of the laptops. The laptops should be connected to the wireless server one by one since the laptops that are far away need to connect to the network after connecting the ones that are closer to the wireless server. During this connection period if any of the laptops in the middle disconnects from the network by mistake, the laptops that are more distant to the wireless server than the disconnected ones, also quit the network. Those laptops needed to be reconnected to the network and this whole process will consume the battery life of all the laptops.

Another challenging thing while conducting experiments is the weather conditions. The experiments are tried to be conducted within the same time interval since it is guessed that the temperature and humidity will not be very different than the temperature and humidity in other days. The wind, rain or even the cloudiness of the weather cannot be predicted precisely. Even the weather forecasts cannot be very clear when a specific time interval is considered for the experiments. If the weather is windy, the wireless signals will not be received or sent to longer distances as in sunny and calm weather. So bad weather will make the network setup process harder and longer, which will consume the battery power of laptops early when the testbed is started.

In the previous section, the effects of the positioning of the laptops were stated [20]. Also during the experiments, the positions of the laptops are arranged with respect to the results of the experiments that are done in [20], since the positioning may affect the results in a bad way.

As a result, in order to conduct any real-world experiments in outdoor environment, the environmental conditions should be similar every day you conduct the experiment. Every time a small problem happens in the network, battery power will have to be spent to fix this problem. When the time passes and the weather conditions change and the experiments will not yield fully accurate results. All these things should be taken into consideration before and during a real-world experiment.

The summarized information about real-world experiments in ad hoc networks can be found in Tables 3.1 and 3.2. Most of the papers listed in the Tables 3.1 and 3.2 were explained in Section 3.2 and also there are some additional papers that were not explained. The routing protocols used in the experiments, information about the maximum number of the nodes, the mobility, and environment of the experiment, performance metrics and the purpose of the experiment can be found in Tables 3.1 and 3.2.

Table 3.1: Summarized information about the real-world experiments.

Paper	Routing Protocol(s)	Max # of nodes	Mobility	Environment	Performance Metrics	Purpose
[19]	APRL AODV ODMRP STARA	33	**YES**	-Outdoor -Indoor	• Beacon Reception Ratio vs. Distance • Packet Delivery Ratio vs. Avg. Interarrival time	Explaining the assumptions that are done in simulations is not always true in real-world.
[20]	No Routing Protocol	8	**NO**	?	• Throughput vs. distance • # of packets vs. transmission time • SNR vs. time	To understand the effect of capacity of the radio medium, asymmetry of the used cards, the effect of broadcast on unicast flows and interferencing range
[24]	APRL AODV ODMRP STARA	33	**YES**	- Outdoor -Indoor	• Message Delivery Ratio • Communication Efficiency • Hop Count • End-to-End latency	Comparison of four different protocols.
[25]	SAODV and AODV	6	**YES**	-Indoor	• Throughput • Routing Packets • Control Overheads	To implement security mechanism to the AODV protocol.
[26]	SAR and AODV	4	**YES**	-Indoor -Outdoor	• Delivery Ratio • End-to-End Delay • End-to-End Delay Jitter • Control Cost • Hop Count	Introduce new criteria to choose a better route among the others.

Table 3.2: Summarized information about the real-world experiments.(Continued)

Paper	Routing Protocol(s)	Max # of nodes	Mobility	Environment	Performance Metrics	Purpose
[34]	MQOLSR And OLSR	10	**NO**	-Ring Topology -Fully Connected Topology	• Average Control Message Overhead versus number of nodes.	Purpose of MQOLSR is to reduce delay jitter and increase network throughput.
[35]	Modified AODV6	8	**YES**	-Indoor	• Respond time versus Number of nodes • Success Rate versus number of nodes.	To analyze the performance of IPv6 based mobile ad hoc networks by conducting real-world experiments.
[21]	OLSR	16	**YES**	-Indoor	• Percentage of time versus Percentage of nodes forming the largest connected component • Percentage of time versus Percentage of Symmetric nodes	Describe the collected data from a heterogeneous ad hoc network created during the MANIAC challenge competition.
[22]	OLSR	10	**YES**	-Indoor and outdoor	• Performance of Partial Coverage versus complete coverage • Traffic Load and Cooperation	Providing solution to the challenges of monitoring MANETs by introducing MMAN.

Chapter 4

TEST-BED PROGRAM

4.1 Purpose of the Program

In order to investigate the performance of wireless ad hoc networks some experiments were conducted in real-world environment. The network nodes that were involved in experiments ran a testbed program which was developed by our research group. This application layer program was developed based on the simulation model and presented in [27]. The purpose of this program is to monitor the network during the experiment and produce statistics. For instance, the number of packet received from a link can be different than the number of packets sent to a link. The program collects some statistical information and computes information that helps us to understand the performance of the wireless ad hoc networks.

4.2 The Structure of the Program

The program was implemented as a multithreaded C program under windows OS. In the program, flooding scheme was used for data dissemination [28]. In this scheme, a node transmits each message to all its neighbors. The neighbors, in their turn, rely each received data packet to their neighbors, and so on until the message propagates to the entire network.

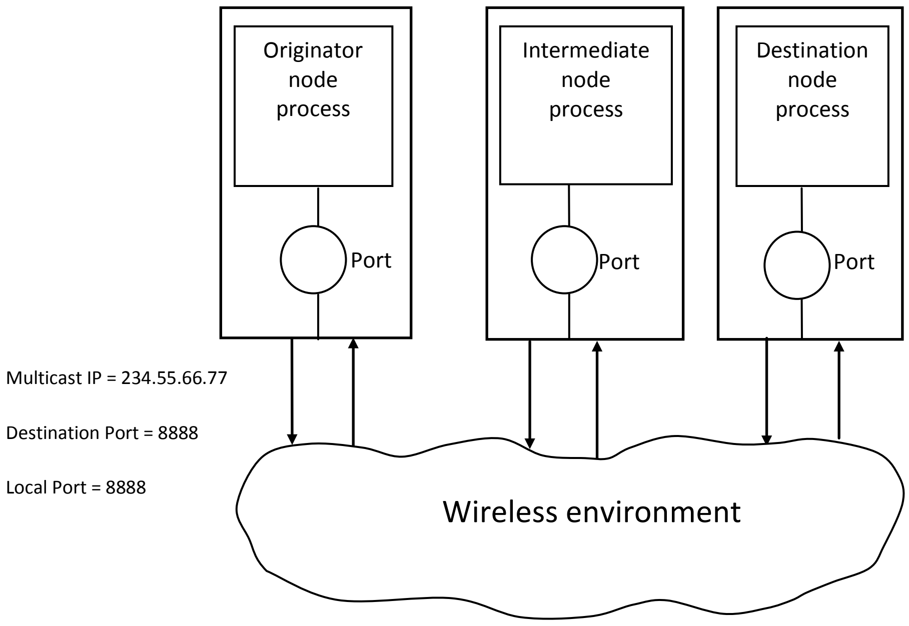


Figure 4.1: Multicast mode for wireless network architecture.

In a wireless ad hoc network under consideration, any node wishing to transmit a message broadcasts one or more packets to the network. Area-restricted multicast mode of transmission mechanism is used to send each packet to the destination node. The multicast mode here represents a limited broadcast form. Each multicast packet is received by a group of hosts whose network interfaces have been configured to receive multicast packets, as shown in Figure 4.1.

To multicast packets, the socket mechanism was used with the UDP transport protocol. IP and CSMA/CA protocols were also used at the network layer and MAC layer, respectively. The MAC layer performs the collusion detection by expecting the reception of an acknowledgment to any transmitted frame except multicast frames [29]. According to [30, 31], multicast packets are not acknowledged.

In the experimental investigation same program ran on all laptop computers in the ad hoc network configuration. There are two threads in the program - the originating thread and the relaying thread. The simplified structure of the multithreaded program, as it works in different nodes, is shown in Figure 4.2.

The originating thread is active only on the source node and is used to send data packets to the destination node in the multicast mode. If the destination node is in the coverage area of the source node the packet will be delivered directly. Otherwise it will be sent through one or more intermediate nodes.

The relaying thread is active on all nodes that have a function of receiving multicast messages from the network. Sending multicast messages is also performed by the relaying thread from the intermediate and the destination nodes. The flow of messages between the threads in the program on different nodes in wireless ad hoc network environment is also shown in the Figure 4.2.

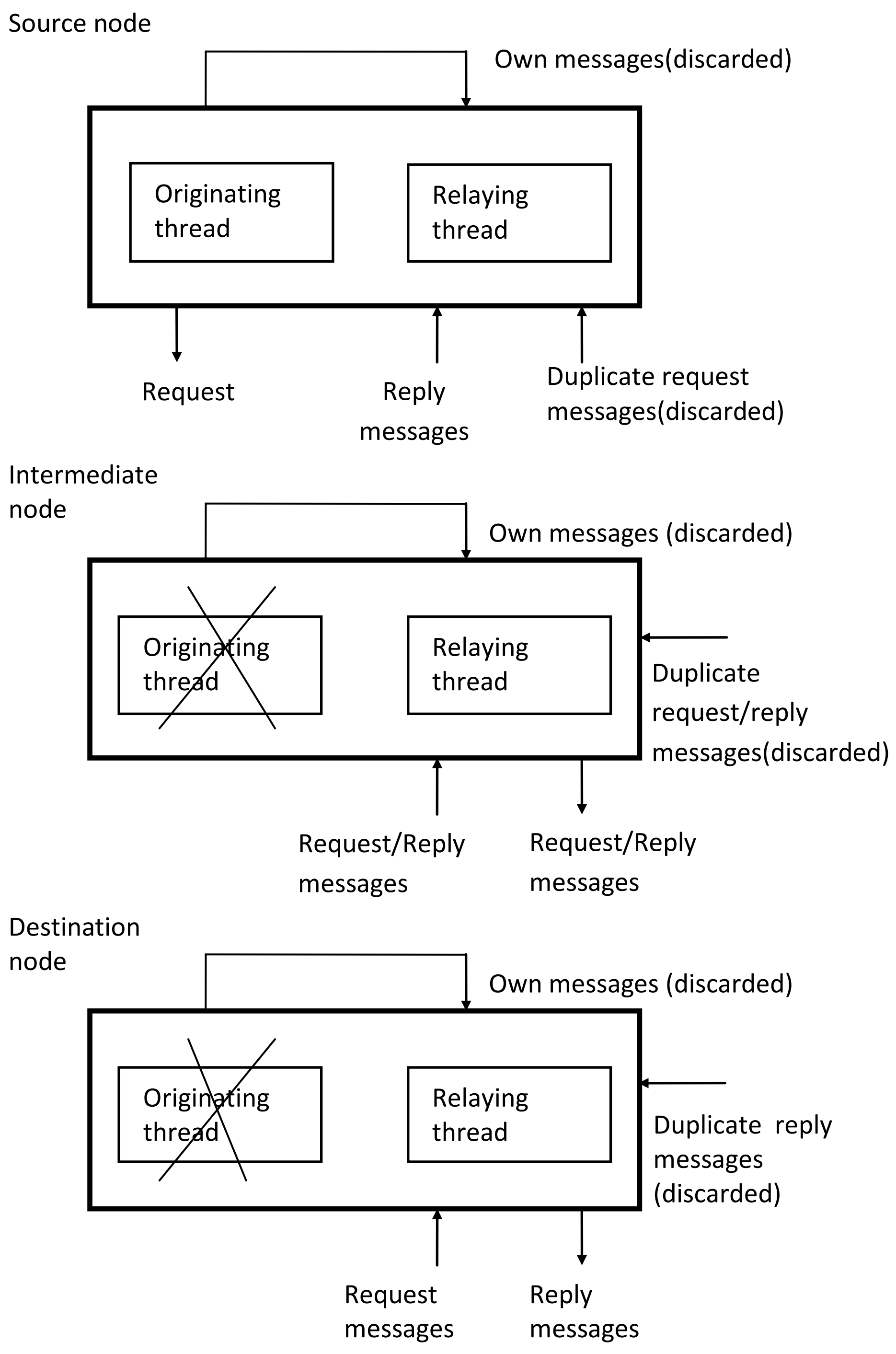


Figure 4.2: Structure of the program.

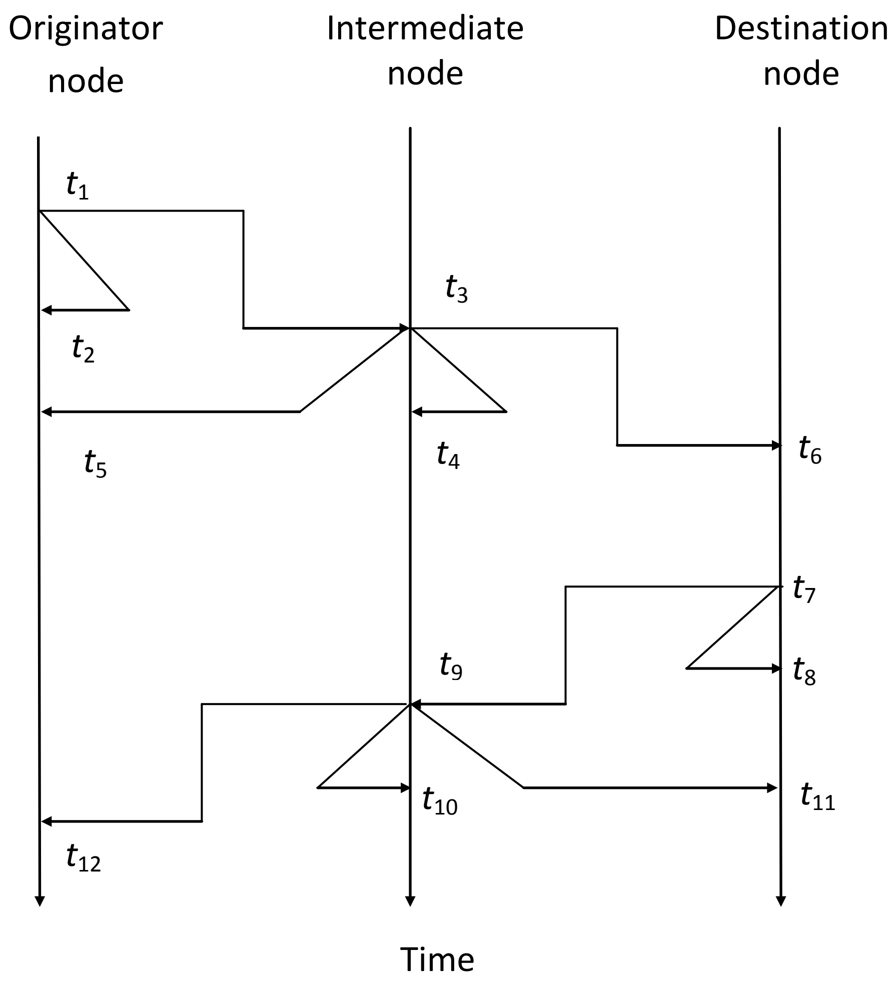


Figure 4.3: A scenario of message passing in the wireless network of three nodes.

Figure 4.3 shows corresponding timing diagram for three nodes in the wireless ad hoc network. In this configuration the destination node is not in the coverage area of the originator node and the intermediate node is in the coverage area of both the originator node and the destination node as shown in Figure 4.4.

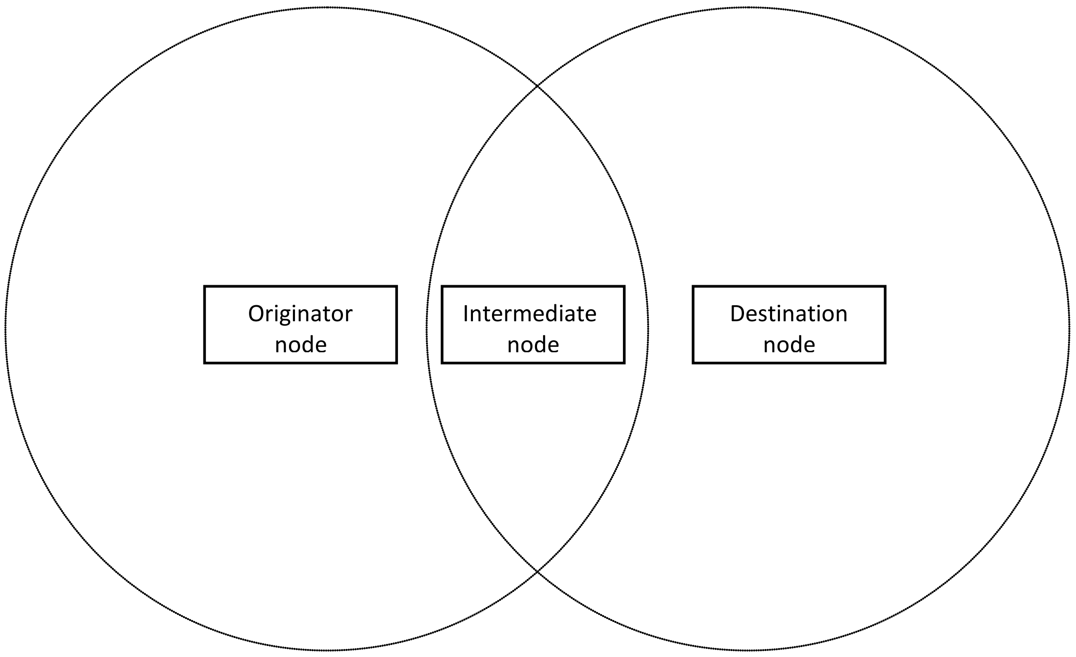
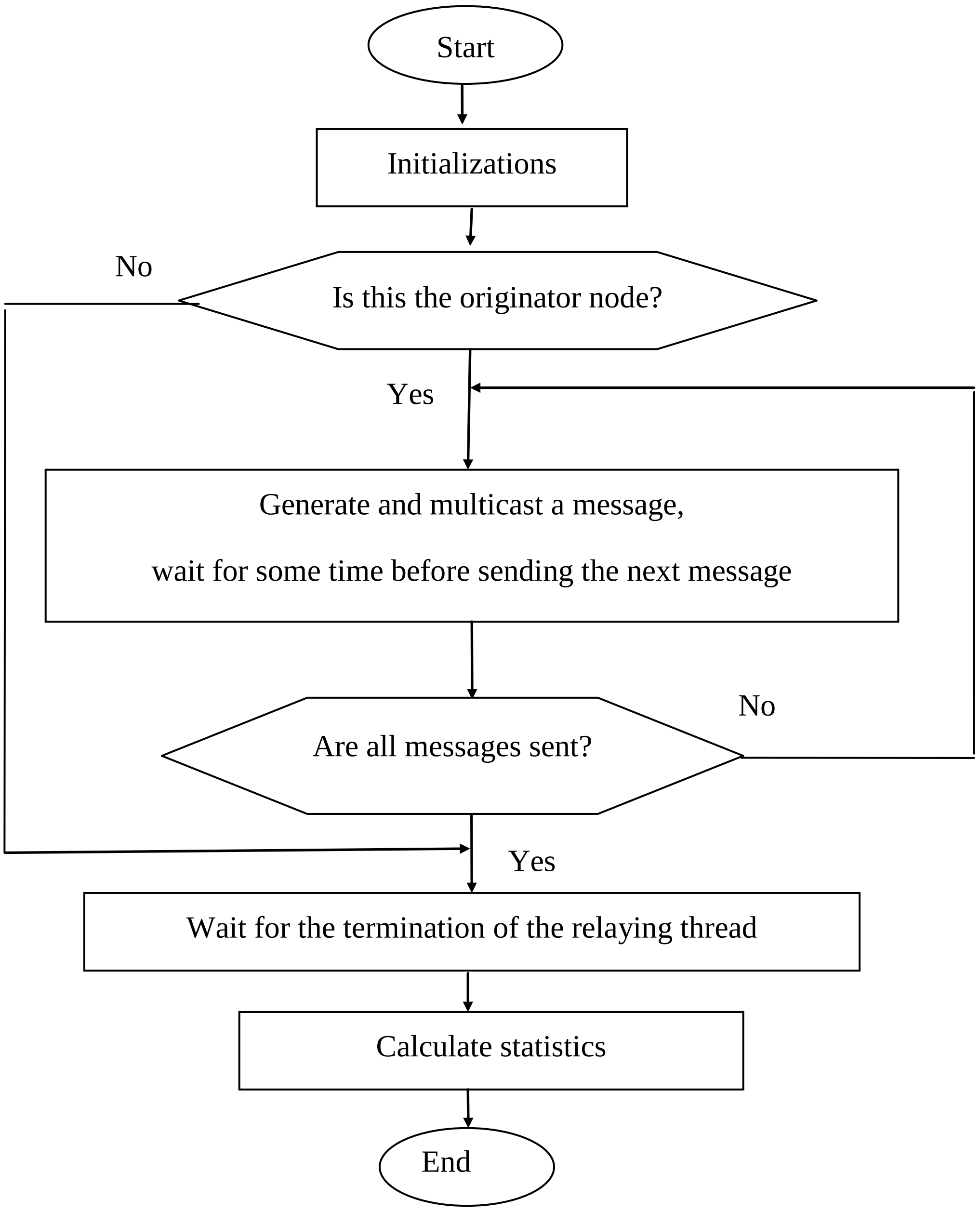


Figure 4.4: A simple wireless ad hoc network with three nodes.

The originator node generates and multicasts a request message to the destination node. This message is received by the intermediate node as a new message and by the originator node as a back message. The originator node discards back messages. On the other hand the intermediate node forwards the received message, in multicast mode to the destination node. This message is received by the destination node as a request message and by the originator node as a duplicate message.



Start
Initializations
No
Is this the originator node?
Yes
Generate and multicast a message,
wait for some time before sending the next message
No
Are all messages sent?
Yes
Wait for the termination of the relaying thread
Calculate statistics
End

Figure 4.5: The algorithm of the originating thread in the program.

The algorithm of the originating thread is shown in Figure 4.5. The originating thread is active on the originator node and is used to send request messages to a destination node through intermediate nodes in multicast mode. After sending all requests, the originating thread waits for the termination of the relaying thread, then collects statistics and terminates as well. On other nodes (destination and intermediate) the originating thread waits for the termination of the relaying thread and terminates.

(a)

Originator IP	Destination IP	Message Identifier(ID)	Number of messages	Remaining number of messages	Hop Count	Pad
long integer (8 bytes)	long integer (8 bytes)	long integer (8 bytes)	long integer (8 bytes)	long integer (8 bytes)	integer (4 bytes)	84 bytes

(b)

Originator IP	Originator IP	Message Identifier(ID)	Number of messages	Remaining number of message	Hop Count	Original destination IP	Pad
long integer (8 bytes)	long integer (8 bytes)	long integer (8 bytes)	long integer (8 bytes)	long integer (8 bytes)	integer (4 bytes)	long integer (8 bytes)	76 bytes

Figure 4.6: A 128 bytes request datagram with data types (a), A 128 bytes reply datagram with data types (b).

Figure 4.6 shows both request message and reply message attributes with their data types. In each request message, the originator IP, the destination IP and the number of messages are fixed. Message identifier (ID) and remaining number of messages and hop count are changing in each message. In each reply message, the destination IP field is set with the originator IP address. To distinguish between the request and the reply messages Original destination IP is used in the reply messages. Each

Requests		Replies	
Number of received messages	Number of sent Messages	Number of received messages	Number of sent messages
Long integer (8 bytes)	Long integer (8 bytes)	Long integer (8 bytes)	Long integer (8 bytes)

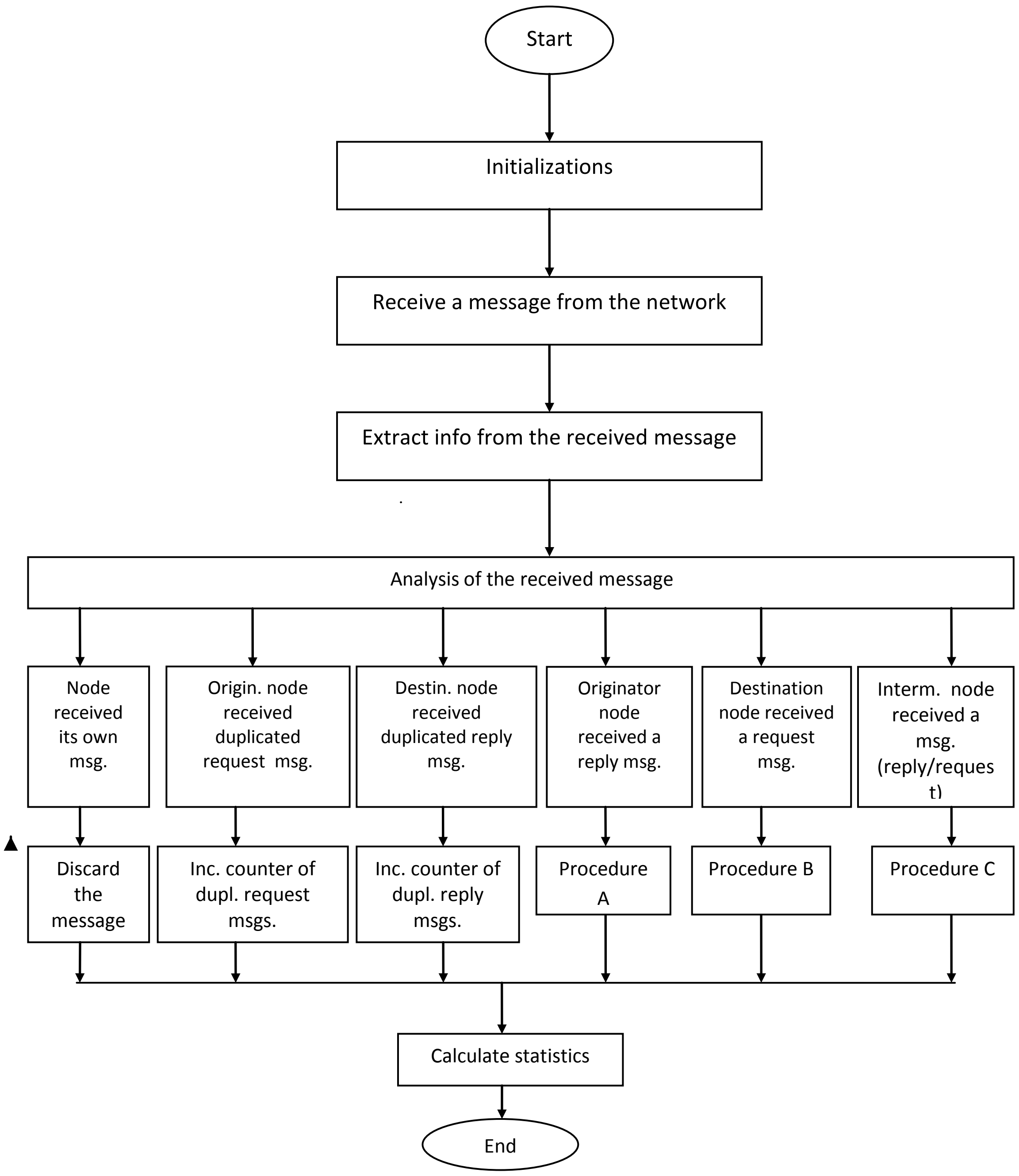
Figure 4.7: Data structure of a request and a reply message counter.

message has an identifier (IP addresses) of the source and by looking at this identifier the receiving side discards its own messages. Message ID is used to determine lost messages on any node. Hop count is used to determine number hops between the source and the destination nodes. Pad field is used to complete remaining data size.

Figure 4.7 presents the data structure of request and reply message counters. It counts number of sent and received, request and reply messages. The array length is also fixed to 2000 indexes. Data structure given in Figure 4.6 is used together with the data structure given in Figure 4.7, to find number of lost and duplicated messages on each node.

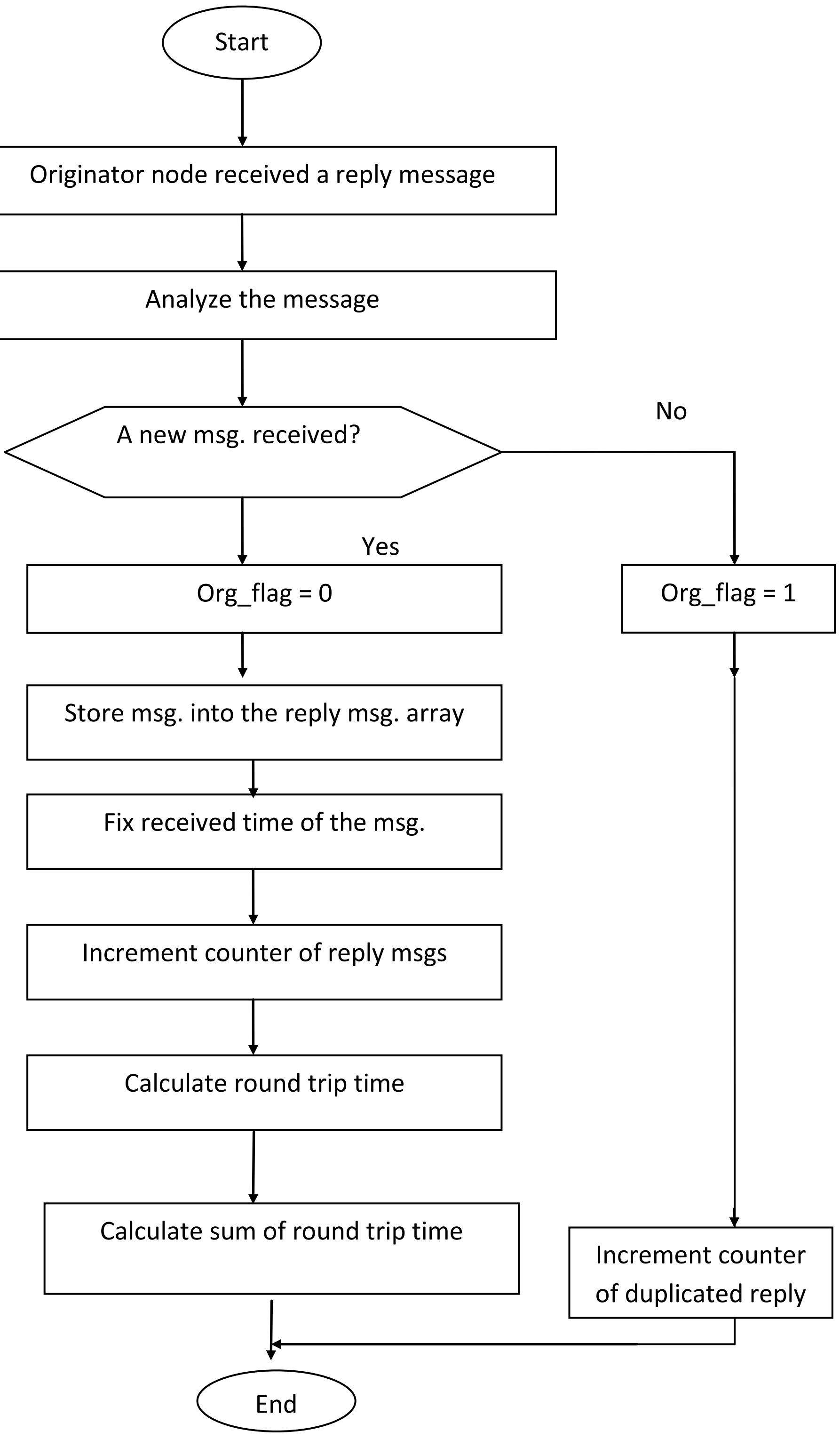
Figures 4.8 - 4.11 illustrate the algorithm of the relaying thread. The relaying thread is active on all nodes, and used to receive multicast messages from the network and analyze the received messages. The received message can be a request or a reply message for all nodes in the prototype system as shown in Figure 4.8. All nodes discard their own messages after receiving the message. In addition, any received, duplicated request and reply messages are counted at all nodes.

In the program, on the originator node, when the relaying thread receives a reply message, it checks if the message is received first time (new message) or it is a duplicated message. The originator node saves each new reply message into the reply messages array and compares each received new reply message with the contents of the reply messages array. For the duplicated messages, counter of the duplicated reply messages is increased. For the new messages, receive time of the message is figured out and round trip time of the message is calculated and added to the sum of the round trip times. Also hop count of the message is incremented and added to the sum of hop count for reply messages. The simplified algorithm of the relaying thread at the originator node is shown in Figure 4.9.



Start
Initializations
Receive a message from the network
Extract info from the received message
Analysis of the received message
Node received its own msg.
Origin. node received duplicated request msg.
Destin. node received duplicated reply msg.
Originator node received a reply msg.
Destination node received a request msg.
Interm. node received a msg. (reply/request)
Discard the message
Inc. counter of dupl. request msgs.
Inc. counter of dupl. reply msgs.
Procedure A
Procedure B
Procedure C
Calculate statistics
End

Figure 4.8: The algorithm of the relaying thread in the test-bed program.



Start
Originator node received a reply message
Analyze the message
A new msg. received?
No
Yes
Org_flag = 0
Org_flag = 1
Store msg. into the reply msg. array
Fix received time of the msg.
Increment counter of reply msgs
Calculate round trip time
Calculate sum of round trip time
Increment counter of duplicated reply
End

Figure 4.9: Algorithm for the relaying thread at the originator node after handling a reply message from the network.

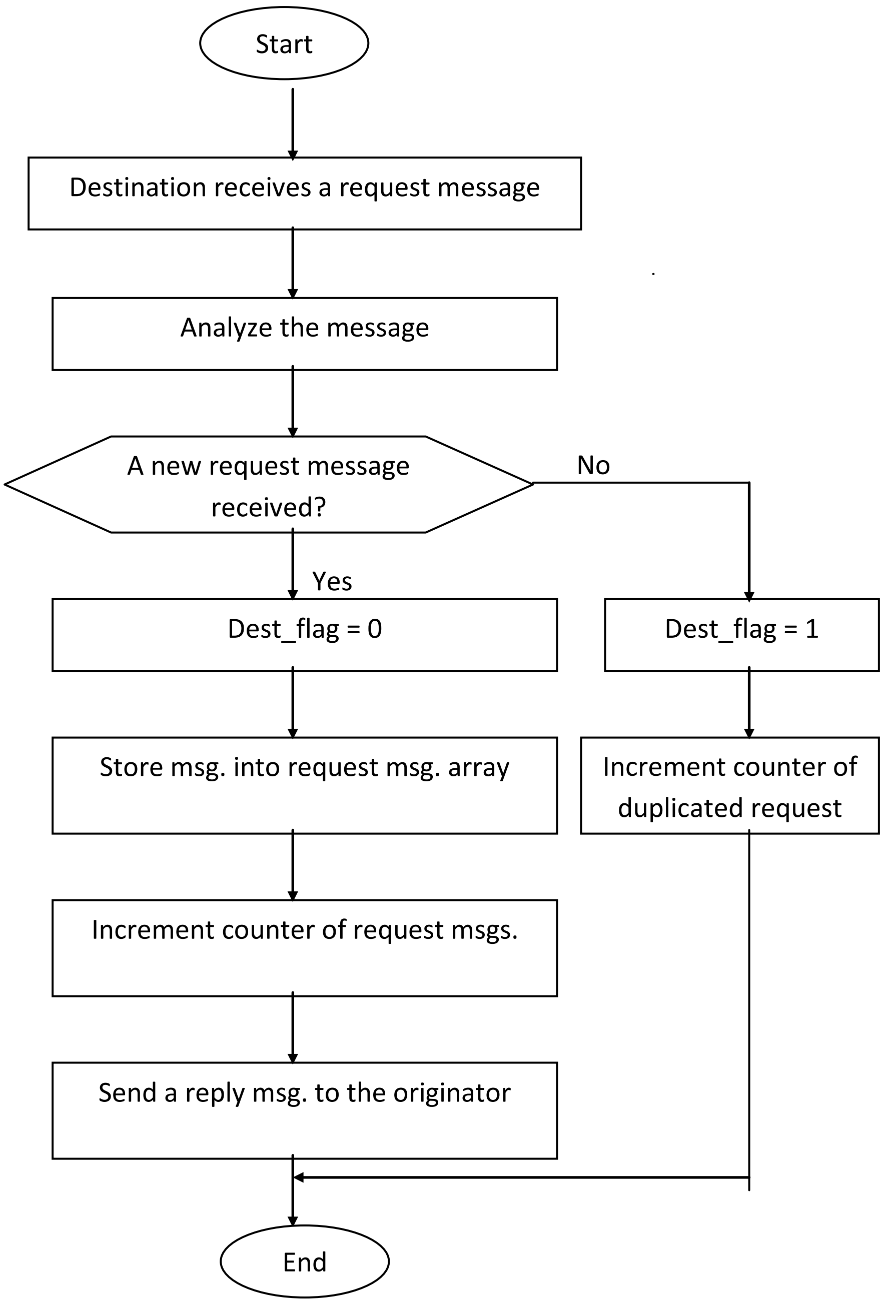

Start
Destination receives a request message
Analyze the message
A new request message received?
No
Yes
Dest_flag = 0
Dest_flag = 1
Store msg. into request msg. array
Increment counter of duplicated request
Increment counter of request msgs.
Send a reply msg. to the originator
End

Figure 4.10: Algorithm for the destination node after handling a request message from network.

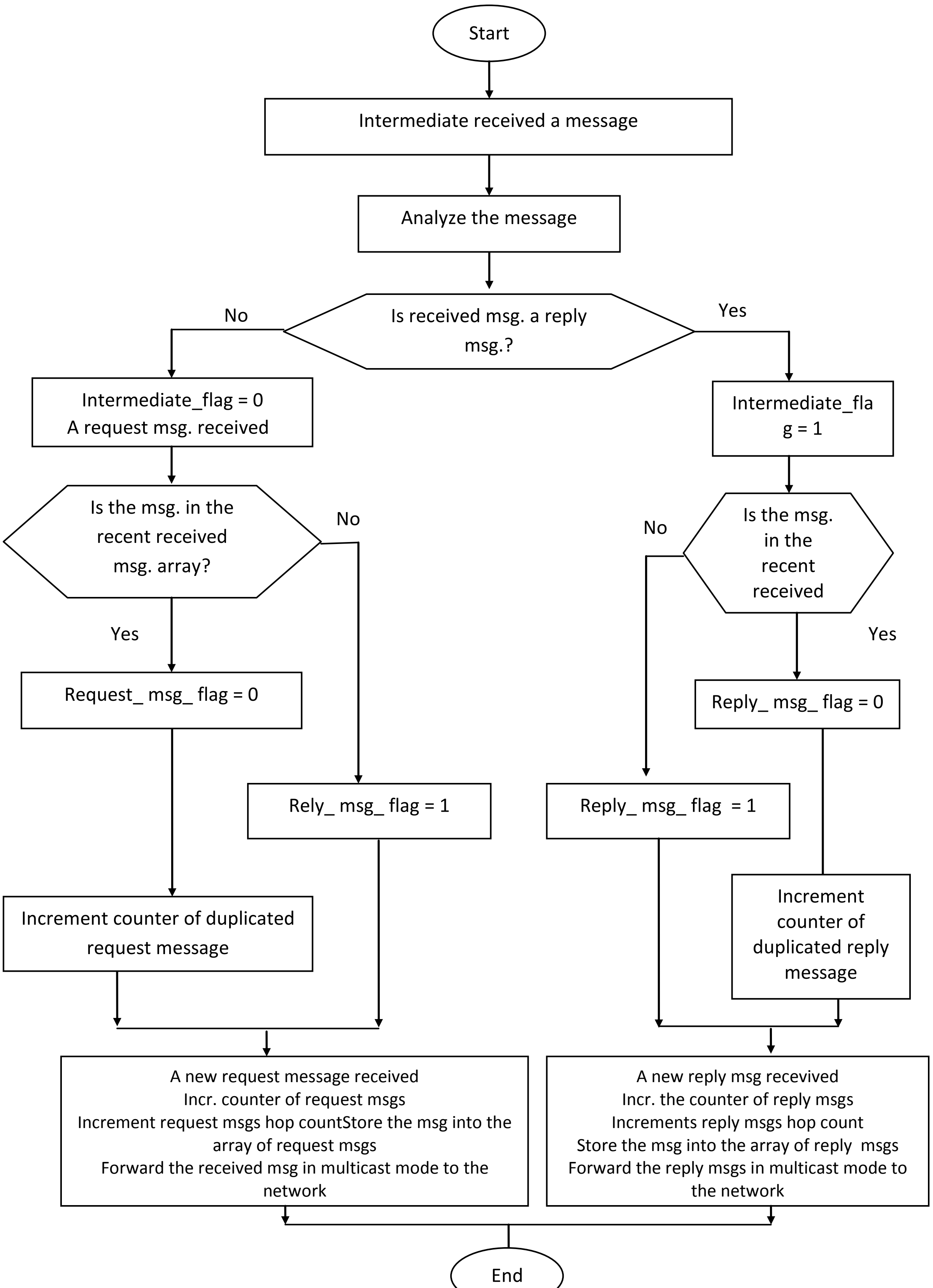

Start
Intermediate received a message
Analyze the message
Is received msg. a reply msg.?
No
Yes
Intermediate_flag = 0
A request msg. received
Intermediate_fla g = 1
Is the msg. in the recent received msg. array?
No
Yes
Is the msg. in the recent received
No
Yes
Request_ msg_ flag = 0
Rely_ msg_ flag = 1
Reply_ msg_ flag = 1
Reply_ msg_ flag = 0
Increment counter of duplicated request message
Increment counter of duplicated reply message
A new request message received
Incr. counter of request msgs
Increment request msgs hop countStore the msg into the array of request msgs
Forward the received msg in multicast mode to the network
A new reply msg recevived
Incr. the counter of reply msgs
Increments reply msgs hop count
Store the msg into the array of reply msgs
Forward the reply msgs in multicast mode to the network
End

Figure 4.11: Algorithm for the intermediate node after handling a message (request or reply) from the network.

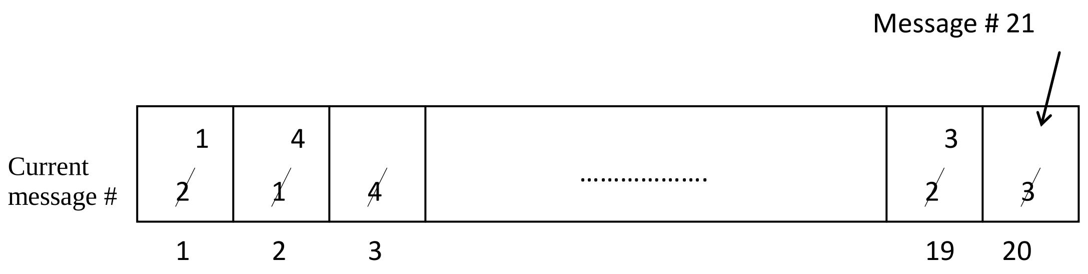


Figure 4.12: Received messages at the intermediate node within a sliding window consisting of 20 cells.

Almost same functions were performed at the destination node. When the destination node receives a request message as shown in Figure 4.10, it checks if the message is received first time (new message) or it is a duplicated message. The destination node saves each new request message into the request messages array and compares each received new request message with the contents of the request messages array. For the duplicated messages counter of duplicated request messages is increased. For the received new messages, a reply message is prepared and sent to the originator node in the multicast mode through the intermediate nodes. Also for each received request message, hop count of the message is incremented and added to the sum of hop count for request messages.

An intermediate node can receive a request or a reply message from the neighbor nodes (see Figure 4.11). For both cases, it checks if the message is received first time (new message) or it is a duplicated message. To store recent received messages, sliding window method is used on the intermediate nodes as outlined in Figure 4.12. The intermediate node stores each new message (request or reply) into the corresponding sliding window comprising 20 cells (each cell holds the received message number at a particular moment of time).

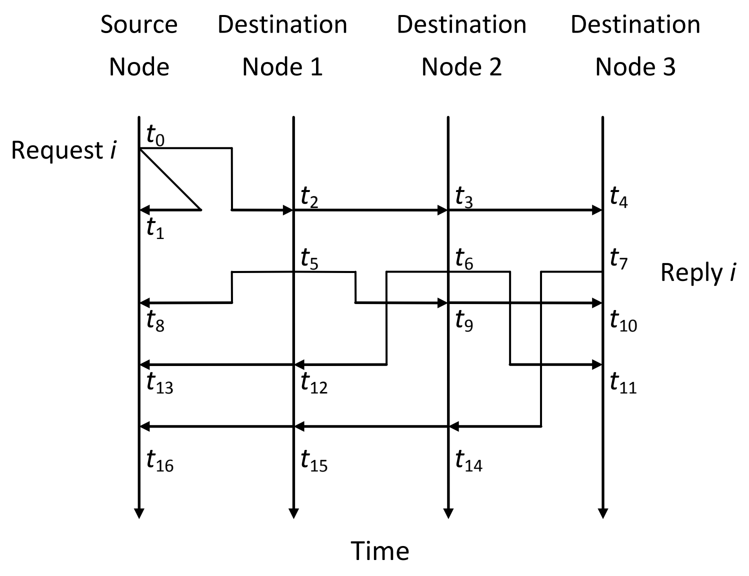


Figure 4.13: A scenario of messaging in the wireless network of four nodes.

Each received new message is compared with the contents of the sliding window. If the message is not a recently received one it is stored into the corresponding cell. For the duplicated messages counter of duplicated messages is increased. For the received new messages, after increasing the corresponding hop count a forwarding message is prepared and sent to the neighbor nodes in the multicast mode.

The version of the program presented above cannot be used by more than one destination nodes in an ad hoc network. The outlined program, under consideration is extended to send a request message that is generated by the source node, to more than one destination nodes and to receive replies from all destination nodes at the source node. Each destination node sends a reply message for each received request message. Source node calculates the average round trip time for the reply messages from individual destination nodes. The delivery ratio is calculated by each destination node. These performance metrics were discussed in the next section. In the extended program out of order received messages were also investigated at both source and destination nodes.

Figure 4.13 shows a timing diagram for one source node and three destination nodes in the wireless ad hoc network. The source node generates and multicasts a request message to the destination nodes at time t_0. This message is received by the source node and the destination nodes at times t_1, t_2, t_3, and t_4, respectively. The source node always discards its own messages. The reply messages from the destination nodes were sent at times t_5 , t_6 and t_7 respectively and were received at times t_8 , t_{13} and t_{16} by the source node. For simplicity back messages of the destination nodes were discarded in the figure. A reply message of any destination node is received by the other destination nodes as well.

4.3 Collected Information

In order to measure the performance of wireless ad hoc network, we need to collect some information during the experiments. The developed program has the responsibility of collecting information. The information that is collected is not exactly same in all the nodes. There are some differences between the collected information by the originator and destination or intermediate node. All the nodes fix start and stop time of each experiment with their local host ip addresses.

The originator node saves the parameter for each experiment. It computes the average round trip time of replies at the originator (source) node in terms of second. It also collects the average number of reply messages received, average hop count for the reply messages, duplicate ratio of the replies and the number of out of order reply messages at the source node. All collected information is saved to a text file by each node for future investigation. Each intermediate node collects average round trip time, total number of received request or reply messages, total number of duplicated request and reply messages at intermediate and total number of lost request and reply messages at intermediate node.

The destination node collects the total number of request messages received, total number of request messages sent, total number of duplicated request messages, total number of request lost messages, average hop count of all the received request messages, the number of out of order request messages and finally calculates the delivery rate of requests. At the destination node, all these collected information is used for the investigation of the performance of wireless ad hoc networks.

Chapter 5

ORGANIZATION OF EXPERIMENTS

Real-world experimental investigations can be categorized as indoor, fixed outdoor and mobile outdoor setups [32]. In fixed setup, the position of the nodes does not change in time. In mobile setup, the position of the nodes changes in time with different speed. In this study mobile and fixed outdoor setups are considered. The speed of the nodes is slow walking speed (~5 km/h). In our study, we conducted a group of experiments for the investigation of wireless ad hoc networks under different configurations and scenarios. It is important to see the behaviors of wireless ad hoc networks with more than one configuration and scenario to understand the overall performance in real-world. In the following sections of this chapter, conducted experiments will be described.

The laptop computers used in the experiments have Intel Core2 Duo Processor 2.2 GHz and are equipped with 802.11b/g Wi-Fi wireless interface. Windows Vista was used as an operating system and each laptop had 2 GB of ram and 250 GB Hard Disk. Each laptop was placed at 50 cm height from the ground in the experimental area. All the experiments were performed during daytime with temperature varying between 20^{o}C and 30^{o}C. In each experiment, the number of requests, which were sent from the source node to the destination node, was 2000 and the inter-packet time (delay between transmission of each packet) at the source node was set at 100 milliseconds. The maximum data size of IEEE 802.11 standard is 2312 bytes in a packet [29], with all headers of the upper layers. Therefore, for large application data sizes (4000 and 8000 bytes), more than one packet were sent from the source node to the destination node.

5.1 Experiments with Two Nodes

In this group of experiments, two nodes were used for the investigation of the performance of wireless ad hoc networks. One node was arranged as the originator node and the other one as the destination node. In the network configuration of this group of experiments, the distance was changed from 30 meters to 120 meters step by step and at each step the distance was increased by 30 meters. At each step, the data size of each packet was varied from 128 bytes to 4096 bytes. The total number of request messages was fixed at 2000 and the inter-packet transmission time between the packets was fixed at 100 milliseconds.

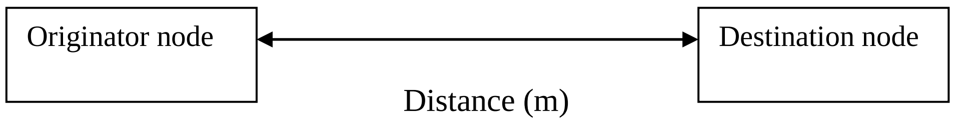


Figure 5.1 : Configuration of the experiments with using two nodes.

A wireless ad hoc network was conducted near the Computer Engineering Department of the Eastern Mediterranean University. There was no physical obstacle between the laptops in the first group of experiments as it is shown in figure 5.1. Each conducted experiment was repeated five times with the same distance and data size settings in order to achieve more efficient results that the average of the trials will give us better understanding of the performance of wireless ad hoc networks.

Experiments with two laptops, without any obstacles in between were used to investigate the maximum range of a wireless node in the network.

In the second group of experiments the effect of inter-packet time (the delay between each message) was investigated with the same configuration. In all conducted experiments, the number of request sent was fixed to 2000. Inter-packet transmission time is the time difference between two consecutive request packets that are sent. In order to have a better understanding of the effect of the inter-packet transmission time, a small group of experiments were conducted with two laptops. In Figure 5.1, we can see that the same configuration was used in the experiments in Section 5.1 except that the distance was constant in this one. The distance between the source node and the destination node was fixed to 150 meters while the data size was varied between 2000,4000 and 8000 bytes. No obstacles were used between the laptops to see the pure effect of the inter-packet transmission time on the network. The inter-packet transmission time was changed to 10, 30, 50, 70 and 100 ms at each step and three trials were made for each set of parameters.

Third group of experiments were done in the presence of an irregular obstacle (a building is used here) between the source node and the destination node in real-world environment. In this group, three different scenarios were used by changing the distance of the source node and the destination node to the building.

In the first scenario, the source node was placed 1m near the building and its position was kept fixed while the distance between the destination node and the source node was changed from 10m to 30m from the source node. The second scenario was the

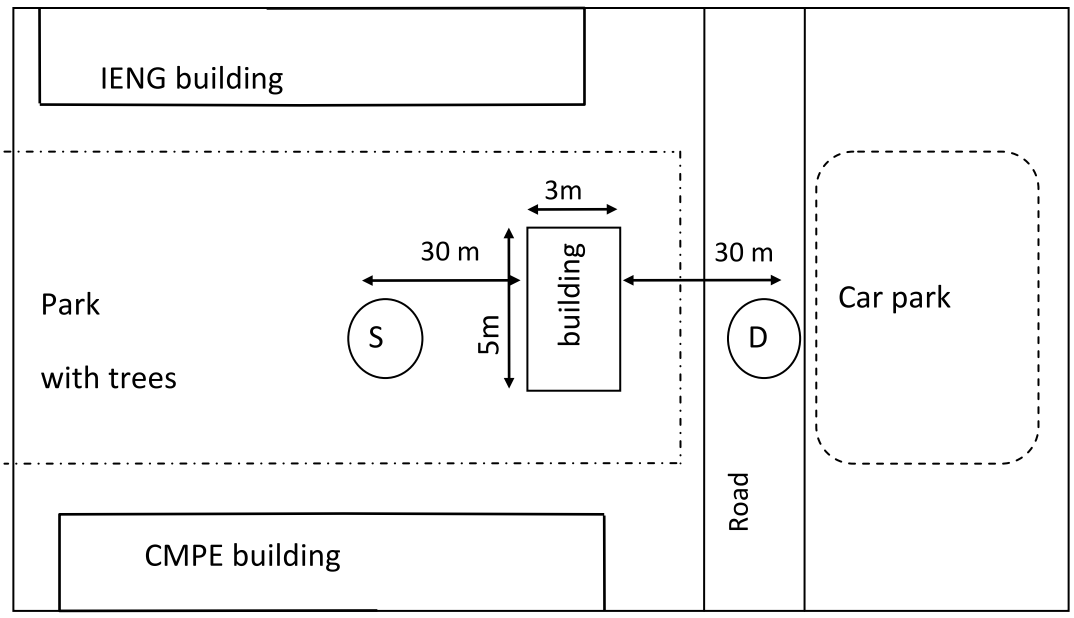


Figure 5.2: A configuration of a wireless ad hoc network consisting of source node and destination node with a building, scenario three.

reverse of the first scenario, where the destination node was placed 1m near the building and its position was kept fixed while the place of the source node was changed from 10m to 30m from the destination node. Figure 5.2 presents the third scenario, where both the source node and the destination node were placed at the same interval from the building. Then the position of the nodes was varied by an equal amount from the building in the range from 10m to 50m.

5.2 Experiments with more than Two Nodes

The experiments with more than two nodes, are categorized in two main groups which are, single path experiments and multi-path experiments. In single path experiments, there was only one path from source to destination node in the whole network. Figure 5.3 presents a complex scenario of the network configuration that was used in a real-world environment (deployed in EMU area) with five nodes. In all experiments there was only one originator or source node of data packets, while the

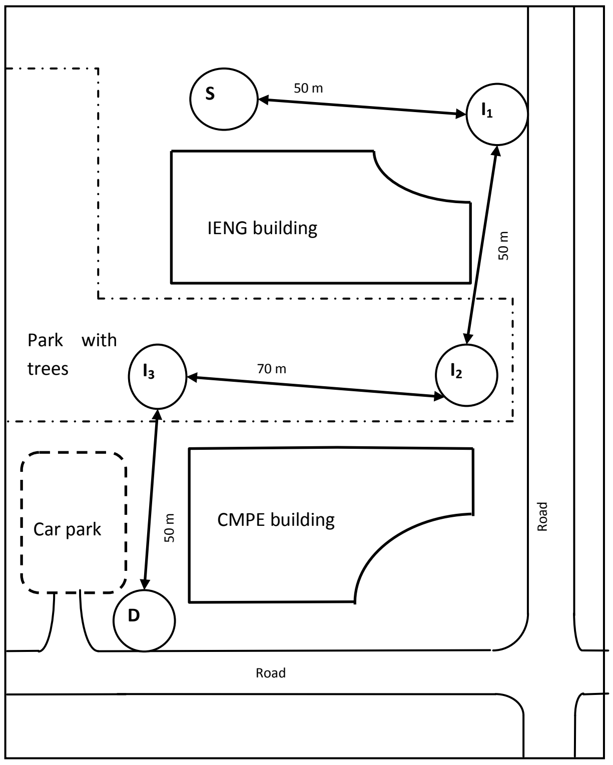


Figure 5.3: A configuration of a wireless ad hoc network consisting of five nodes.

positions of the intermediate nodes and destination node, depended on the specific scenario. In the experiments that were carried out with the use of the given network configuration, four different scenarios were considered, with the number of intermediate nodes varying between 0 and 3. To investigate routing in the network, the nodes were positioned in such a way that only adjacent nodes were within the coverage area of each other. As is shown in Figure 5.3, the source node S can only transmit and listen to intermediate node I_1. The intermediate node I_1 has the source node S and the intermediate node I_2 within its coverage area. Similarly, the intermediate node I_2 can only communicate with intermediate nodes I_1 and I_3. The neighbor of the destination node D is only the intermediate node I_3.

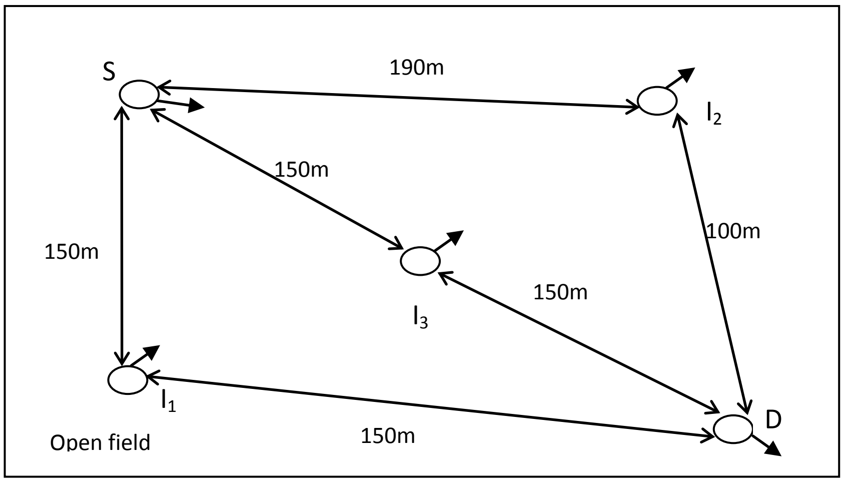


Figure 5.4: A configuration of a wireless ad hoc network consisting of five nodes in open area.

In the multi-path experiments, routing and data dissemination are considered in different ad hoc network configurations fixed nodes. Two set of experiments were contacted. Figure 5.4 shows settings for the first set of experiments where there exist a source node, destination node and three intermediate nodes.

At the beginning of the experiments, the nodes were distributed in the area randomly. For instance, the source node S could transmit and listen to intermediate nodes I_1, I_2 .and I_3. The neighbors of the destination node D were the intermediate nodes I_1, I_2 and I_3. The destination node D could not transmit or could not listen the source node S directly. The area of the experiment was 300m x 300m. The reason for choosing an open field area is that it was far enough from any wireless interference that could affect its performance. The second set of experiments was the extension of the first set of experiments.

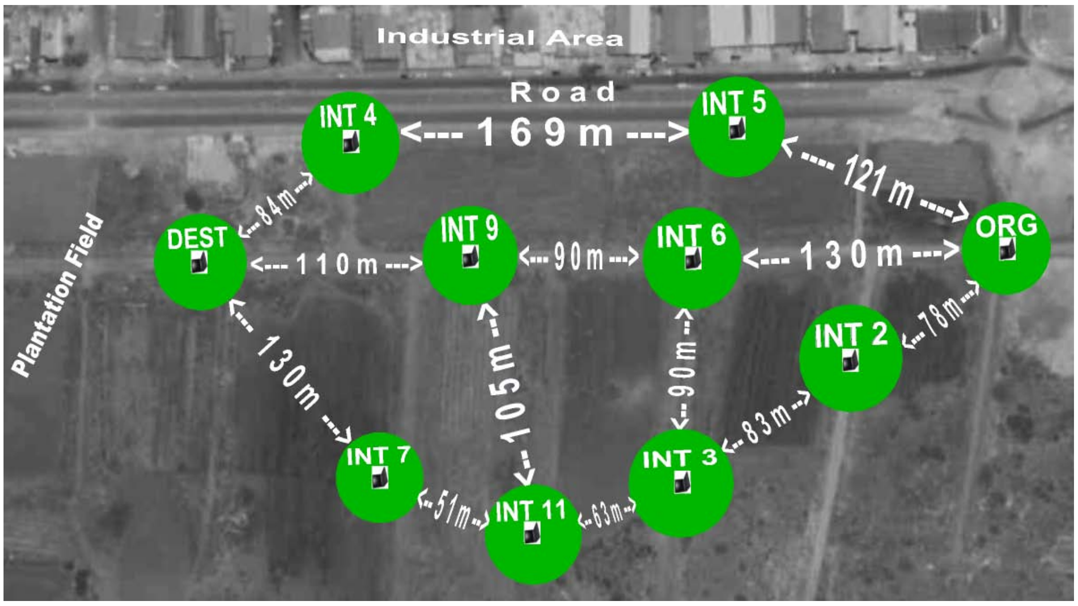


Figure 5.5: A configuration of a wireless ad hoc network consisting of ten nodes.

Figure 5.5 illustrates the area where all of the experiments took place. It is located inside the city, opposite of industrial area. As seen in Figure 5.5, there are only 1 source node, 1 destination node and 8 intermediate nodes. Source and destination nodes were positioned in such a way that they could not communicate directly while intermediate nodes were positioned by an arbitrary fashion. Due to the long distance between the source and the destination nodes packets were transmitted through intermediate nodes to the destination node. Flow of packets through intermediate nodes again followed an arbitrary fashion.

It is nearly impossible to achieve the same results from two trials even if network configuration and packet size remains stable due to real-world environmental factors such as fading, attenuation, and presence of other interfering factors are not stable [33]. Therefore, in order to get more statistical and realistic data, all of the experiments with 5 different packet sizes are iterated 3 times and only the average of these 3 trials was taken into the consideration.

5.3 Experiments with One Source Node and Three Destination Nodes

Figure 5.6 presents settings of the source node and three destination nodes at different directions for the network configuration deployed in an open field. The laptop computer, which was used as the source node, was placed at the center and three destination nodes were positioned on a circle with equal distances from the source node and from the neighbor destination nodes. In the experiments, the place of the source node was fixed and the place of the destination nodes was varied in the range

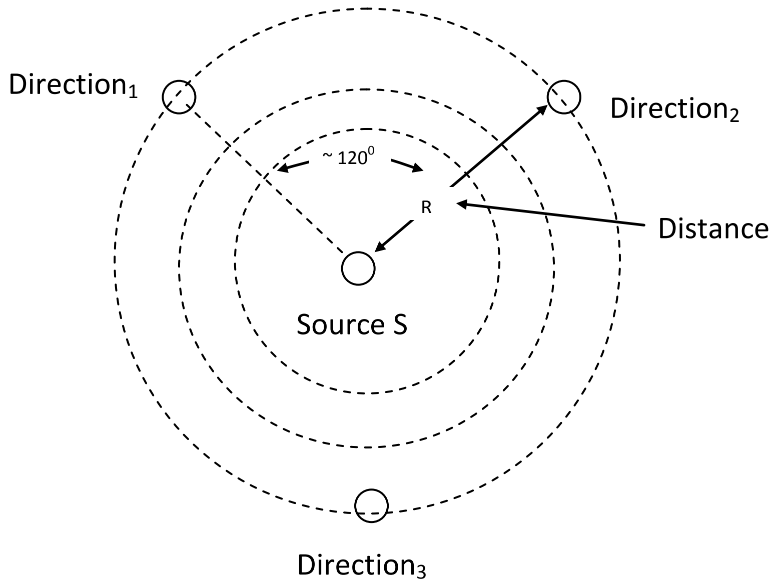


Figure 5.6: The position of source and destination nodes in the network.

from 30 m up to 120 m, to investigate the effect of the inter-node distance on the performance metrics that given be described later. During these settings, all destination nodes were within the coverage area of the source node. At each distance, the application data size was varied between 50, 800 and 4000 bytes. Again each set of experiment was repeated more than once in order to achieve better results.

Normally, during all experiments, each laptop was placed 50 cm high from the ground level. Under the same network configuration, series of experiments were conducted to understand the effect of the high of the laptops from the ground level. The laptops were placed 100cm height from the ground and the distance between the source and the destinations was 120 meters. The data size was varied with respect to 50, 800, 4000 bytes and the result of experiments where laptops stood 50cm high from the ground level was compared with the experiments where laptops were placed 100cm high from the ground.

Chapter 6

EXPERIMENTAL RESULTS AND THEIR ANALYSIS

6.1 Performance Metrics

In this study, the performance metrics that are used in experiments are delivery ratio, average end-to-end latency, round-trip-time (RTT) and number of hops. These performance metrics could be used in experimental studies with different parameters such as; distance, packet inter-arrival time, data size and number of hops between source and destination nodes. In this study, in some group of experiments, we considered the delivery ratio of the three destination nodes, that were calculated at destination nodes and the average round trip time at the source node for three destination nodes.

Formally, the delivery ratio measured at the destination on distance *D* is represented by the expression (6.1).

$$d_i\left(D\right) = \frac{N_i(D)}{N_s} \quad , \tag{6.1}$$

where N_s is the number of multicast data packets transmitted by the source node and N_i is the number of data packets delivered to the destination node *i*, *i* = 1,2,…,*m* placed at distance *D*. From this, the delivery ratio for one source node and *m* destination nodes placed at the same distance *D* from the source node is represented by the expression (6.2).

$$d(D) = \frac{\sum_{i=1}^{m} d_i N_i(D)}{\sum_{i=1}^{m} N_i(D)} \tag{6.2}$$

The average round trip time, measured at the source node for a destination, can be defined with the expression (6.3).

$$R = \frac{1}{N_r} \sum_{i=1}^{N_r} R_i , \tag{6.3}$$

where N_r is the number of replies at the source node and R_i is the round trip time for reply i, i = 1, 2, …, N_r.

The average round trip time at the source node for *m* destination nodes can be represented by the expression (6.4).

$$R = \frac{1}{m} \sum_{j=1}^{m} R_j \tag{6.4}$$

where R_j, is the average round trip time of destination *j* and *j* = 1,2, ..., *m*.

Another performance metric is the average number of hops, measured at the destination node, expressed with the expression;

$$h_d = \frac{1}{N_d} \sum_{i=1}^{N_d} N_i \tag{6.6}$$

where N_i is the number of hops for request i where i=1,2,3,….,N_d.

6.2 Results of Experiments

The result of experiments that was explained in section 5.1 and configured in Figure 5.1 is presented in Figures 6.1-6.2. Figures demonstrate the dependence of the average

Table 6.1: The average values of round trip time which varies with distance under different application data sizes with using two nodes.

Inter-node distance,m	Application data size(bytes)					
	100	500	1000	2000	4000	8000
30	0.176	0.944	15.779	15.984	32.600	77.801
60	0.187	1.063	15.745	15.938	32.200	78.210
90	0.227	1.005	15.743	15.903	31.700	78.210
120	0.837	1.192	15.894	16.133	32.270	74.580
150	0.160	0.903	15.708	15.801	31.600	74.750

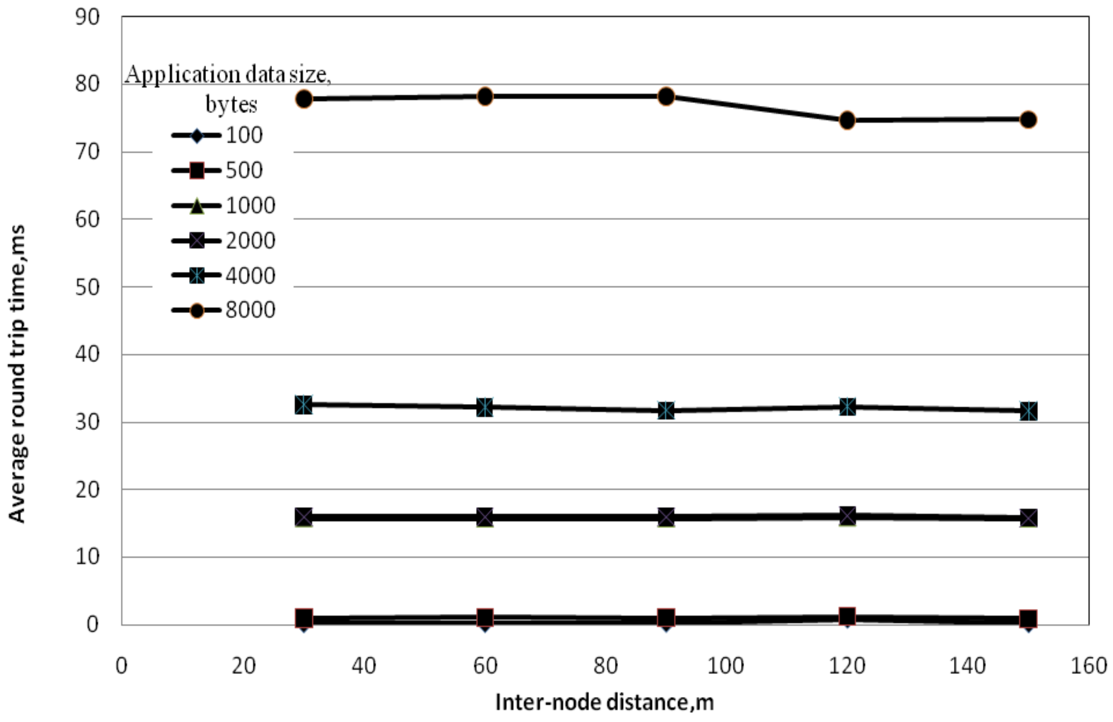


Figure 6.1: Average round trip time versus distance with different data sizes.

round trip time and delivery ratio on distance with different application data sizes. Also the exact average values of the results can be seen from the Table 6.1-6.2 which they were used to draw the figures.

Table 6.2: The average values of delivery ratio which varies with respect to distance under different application data sizes with using two laptops.

Inter-node distance,m	Application data size(bytes)					
	100	500	1000	2000	4000	8000
30	0.965	0.964	0.970	0.961	0.940	0.935
60	0.987	0.980	0.948	0.977	0.990	0.984
90	0.972	0.990	0.948	0.992	0.995	0.988
120	0.864	0.895	0.950	0.900	0.853	0.714
150	0.787	0.783	0.722	0.469	0.412	0.256

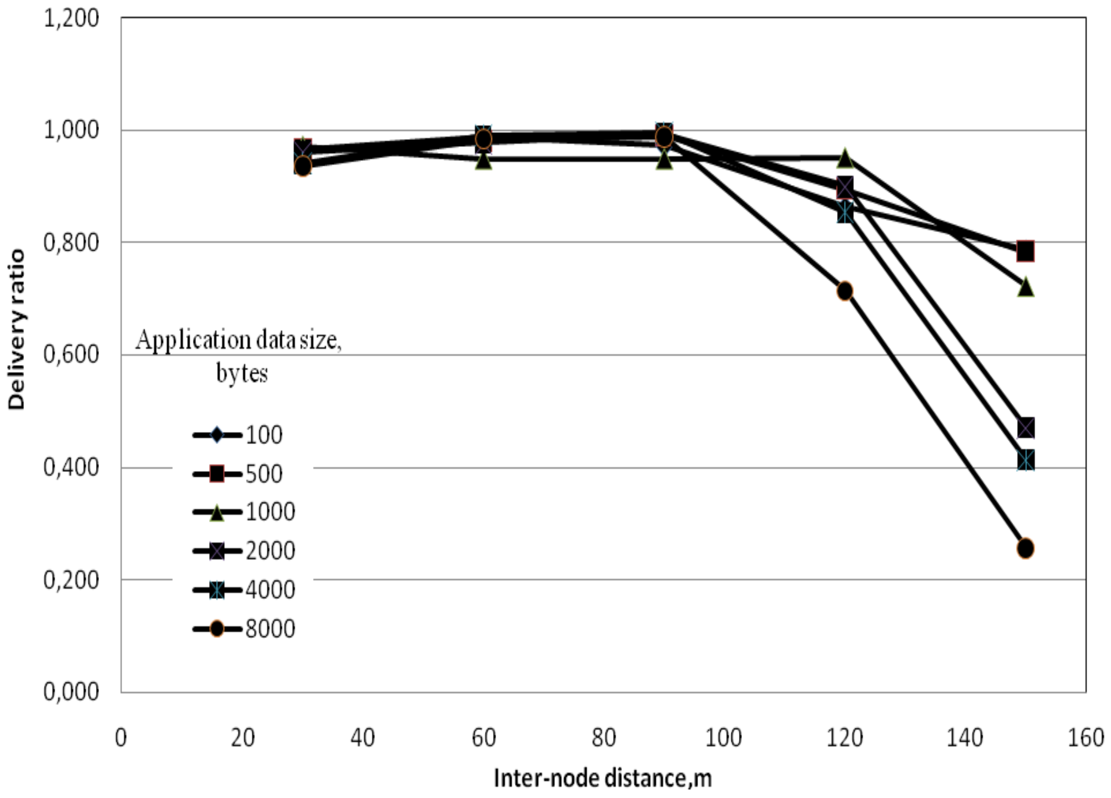


Figure 6.2: The delivery ratio versus inter-node distance with different data sizes.

The result of the experiments that was configured in Figure 5.1 is shown in Tables 6.3-6.4 and presented in Figures 6.3 and 6.4. In Figure 6.3, the effect of inter-packet transmission time on delivery ratio is demonstrated while in Figure 6.4, the effect of inter-packet transmission time on round trip time is demonstrated.

Table 6.3: The average delivery ratio with respect to the inter arrival packet time under different application data sizes with using two laptops.

Inter-packet transmission time, ms	Application data size(bytes)		
	2000	4000	8000
10	0.745	0.608	0.446
30	0.967	0.709	0.631
50	0.899	0.809	0.551
70	0.982	0.977	0.802
100	0.984	0.976	0.945

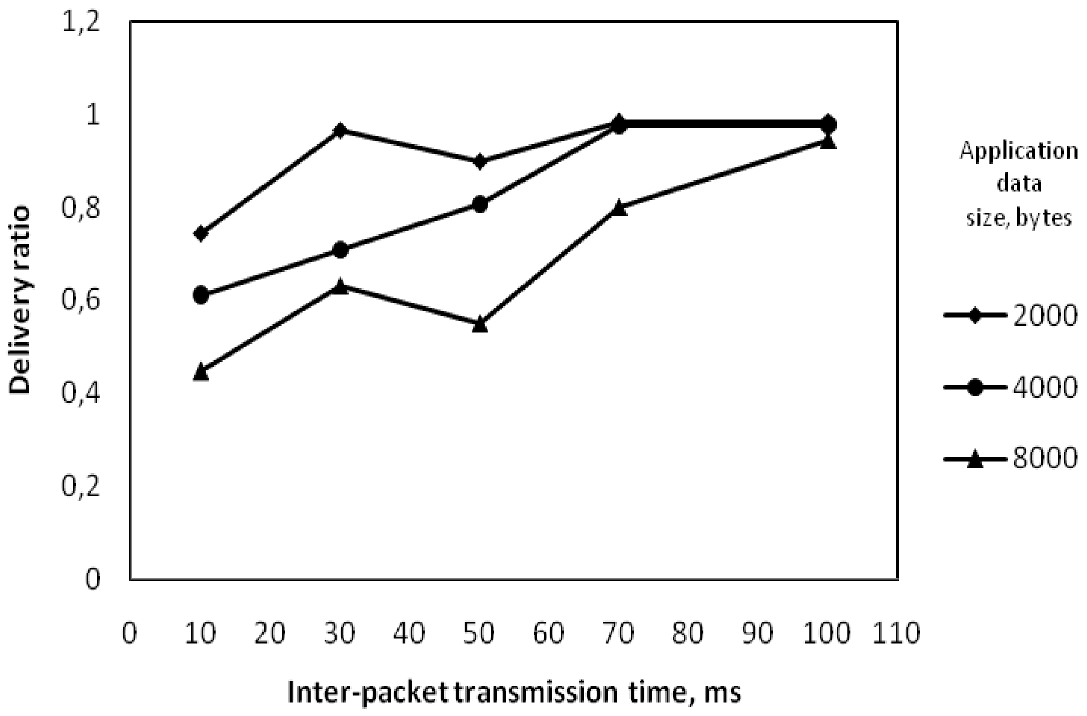


Figure 6.3: The delivery ratio versus inter-packet transmission time with different application data sizes.

Table 6.4: The average round trip time values with respect to distance under different application data sizes with using two nodes.

Inter-packet transmission time, ms	Application data size(bytes)		
	2000	4000	8000
10	22.867	52.07	97.939
30	16.23	37.926	89.373
50	21.64	38.998	76.130
70	15.89	31.887	77.38
100	15.7	31.68	77.897

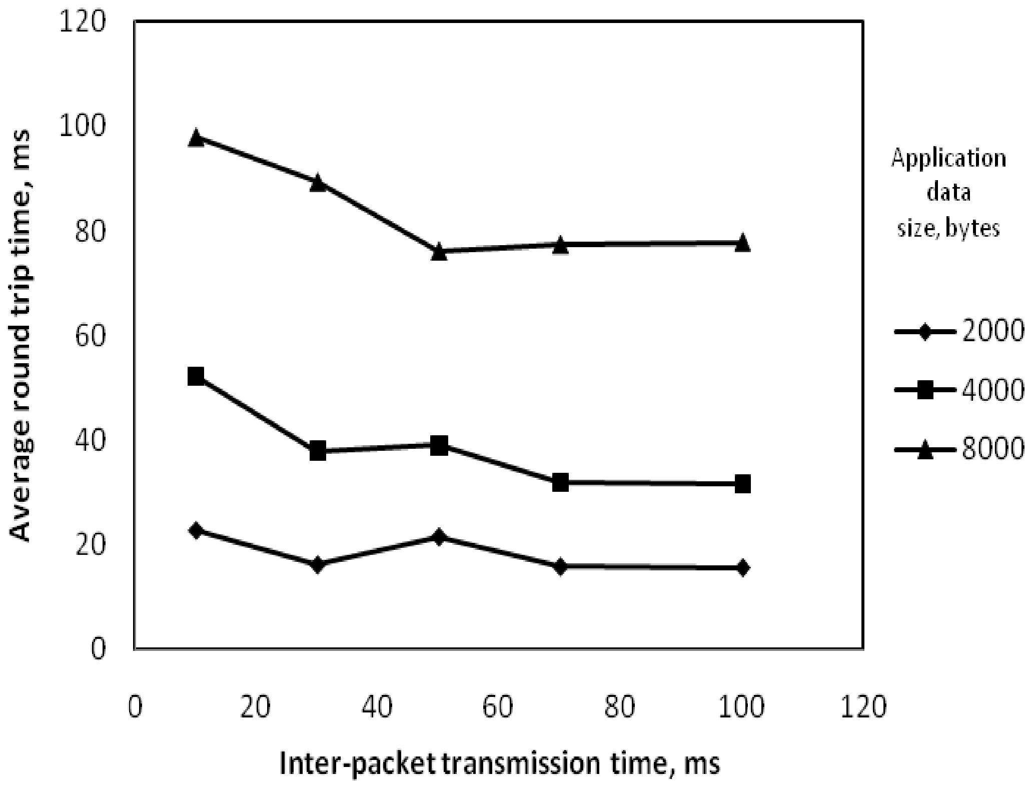


Figure 6.4: The average round trip time versus inter-packet transmission time with different application data sizes.

Table 6.5: The average delivery ratio values with respect to distance under different application data sizes using two nodes and an obstacle between them in scenario one.

Inter-node distance, m	Application data size (bytes)			
	100	1000	2000	4000
10	0.994	0.998	0.993	0.990
20	0.970	0.841	0.777	0.792
30	0.882	0.665	0.532	0.418

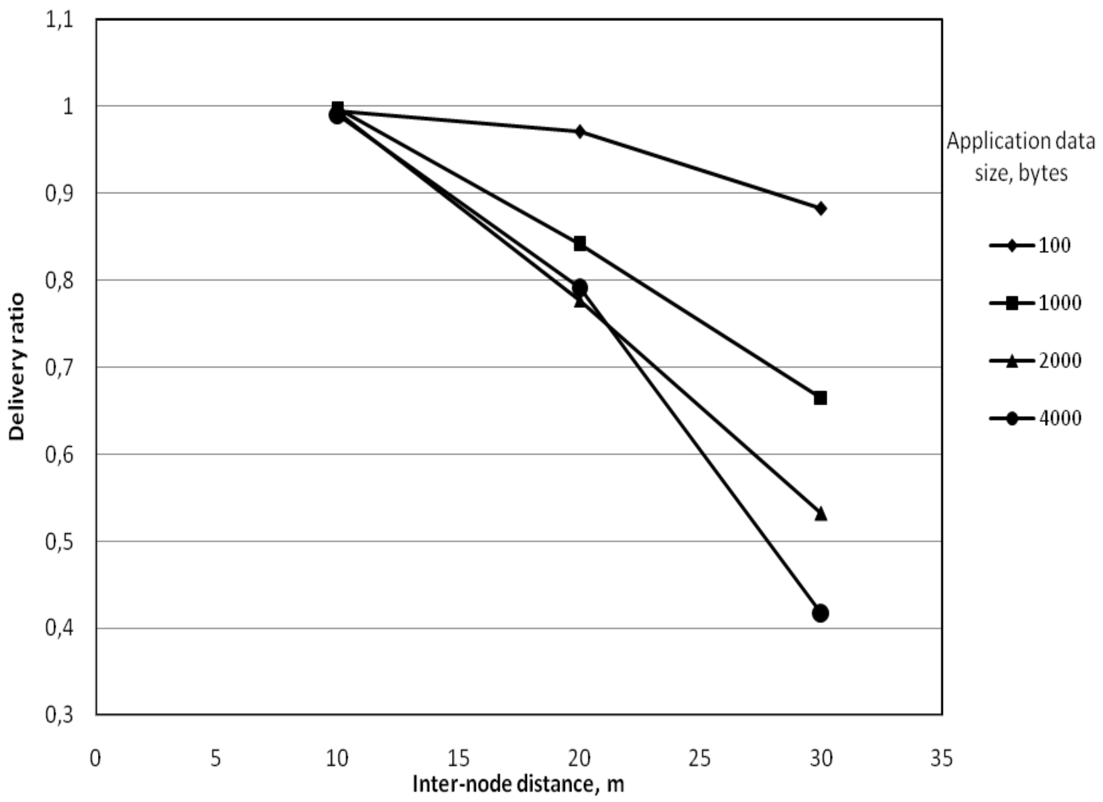


Figure 6.5: The delivery ratio versus inter-node distance, for different application data sizes in scenario one.

The result of the experiments that was configured in figure 5.2 with different scenarios are given in Tables 6.5-6.10 and presented in Figures 6.5-6.10. In Figures 6.5 and 6.6, the effect of inter-node distance on delivery ratio and round trip time with different application data sizes can be seen for the first scenario respectively. In Figures 6.7 and 6.8, again the effect of inter-node distance on delivery ratio and round trip time with different application data sizes can be seen but for scenario two. The effect of the inter-node distance on delivery ratio and round trip time is demonstrated in Figures 6.9 and 6.10 for third scenario.

Table 6.6: The average round trip time values with respect to distance under different application data sizes using two nodes and an obstacle between them in scenario one.

Inter-node distance, m	Application data size(bytes)			
	100	1000	2000	4000
10	0.191	15.832	16.009	32.510
20	0.193	15.824	15.938	31.826
30	0.229	16.278	15.815	31.712

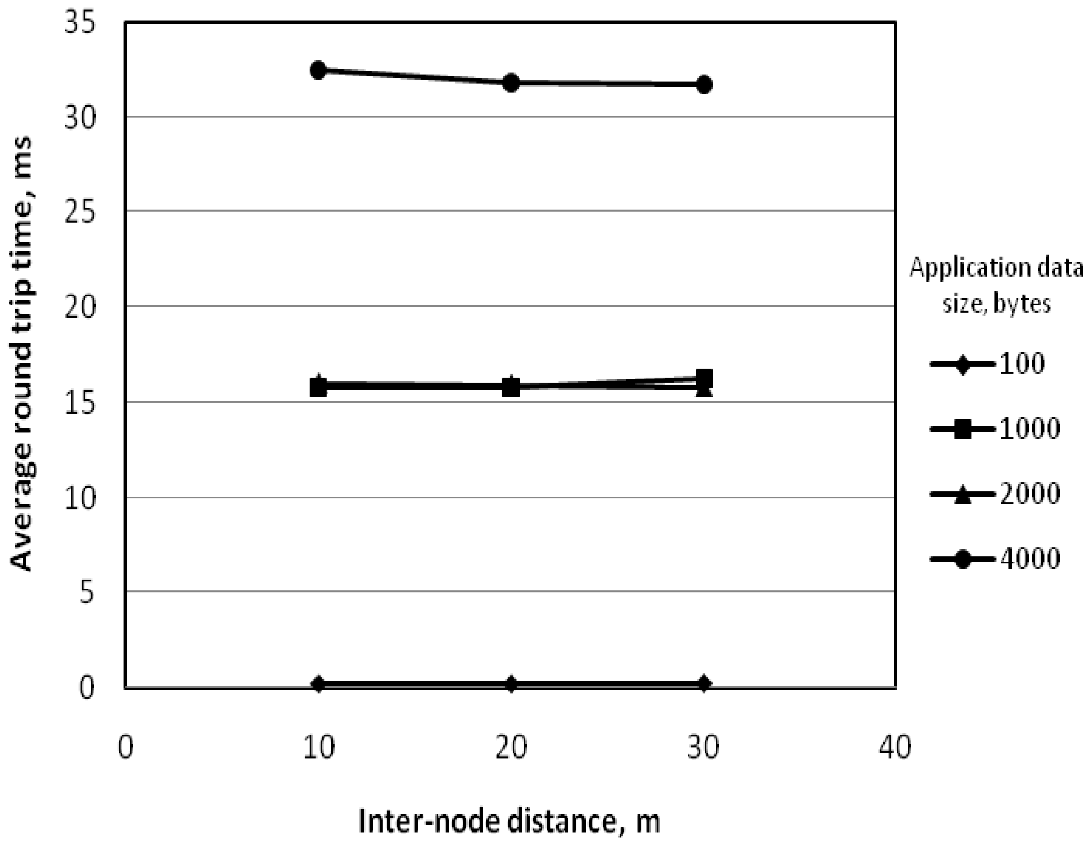


Figure 6.6: The average round trip time versus inter-node distance, for different application data sizes in scenario one.

Table 6.7: The average delivery ratio values with respect to distance under different application data sizes in scenario two.

Inter-node distance, m	Application data size (bytes)			
	100	1000	2000	4000
10	0.999	0.999	0.998	0.997
20	0.996	0.975	0.840	0.869
30	0.686	0.781	0.705	0.64

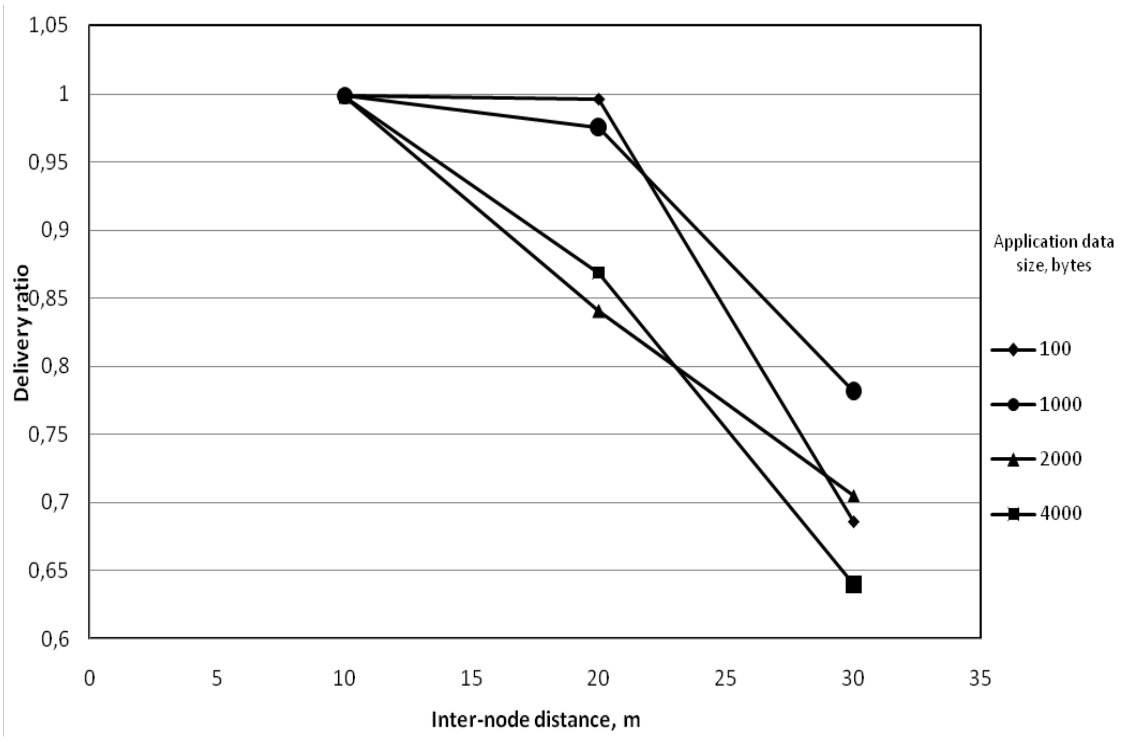


Figure 6.7: The delivery ratio versus inter-node distance, for different application data sizes in scenario two.

Table 6.8: The average round trip time values with respect to distance under different application data sizes in scenario two.

Inter-node distance, m	Application data size (bytes)			
	100	1000	2000	4000
10	0.198	15.69	15.836	31.780
20	0.062	15.657	15.853	31.78
30	0.548	17.507	15.913	32.13

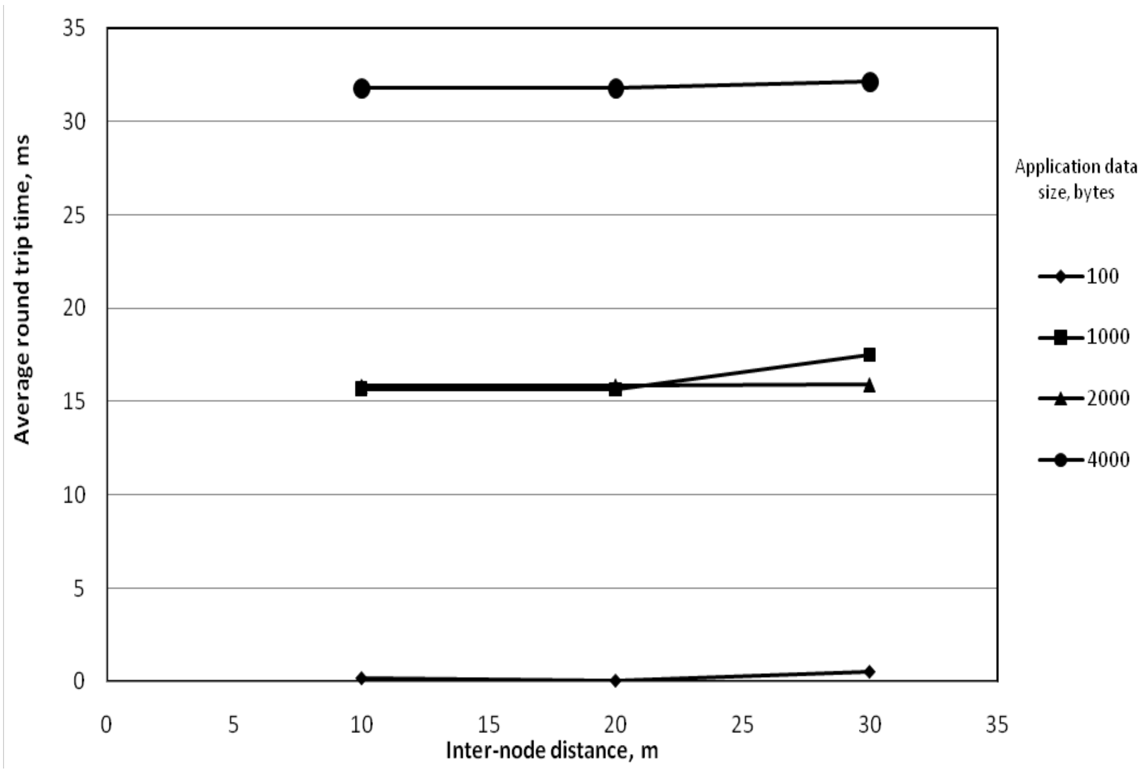


Figure 6.8: The average round trip time versus inter-node distance, for different application data sizes in scenario two.

Table 6.9: The average delivery ratio values with respect to distance under different application data sizes in scenario three.

Inter-node distance, m	Application data size (bytes)			
	100	1000	2000	4000
10	0.988	0.959	0.837	0.56
20	0.921	0.844	0.734	0.575
40	0.552	0.518	0.399	0.189

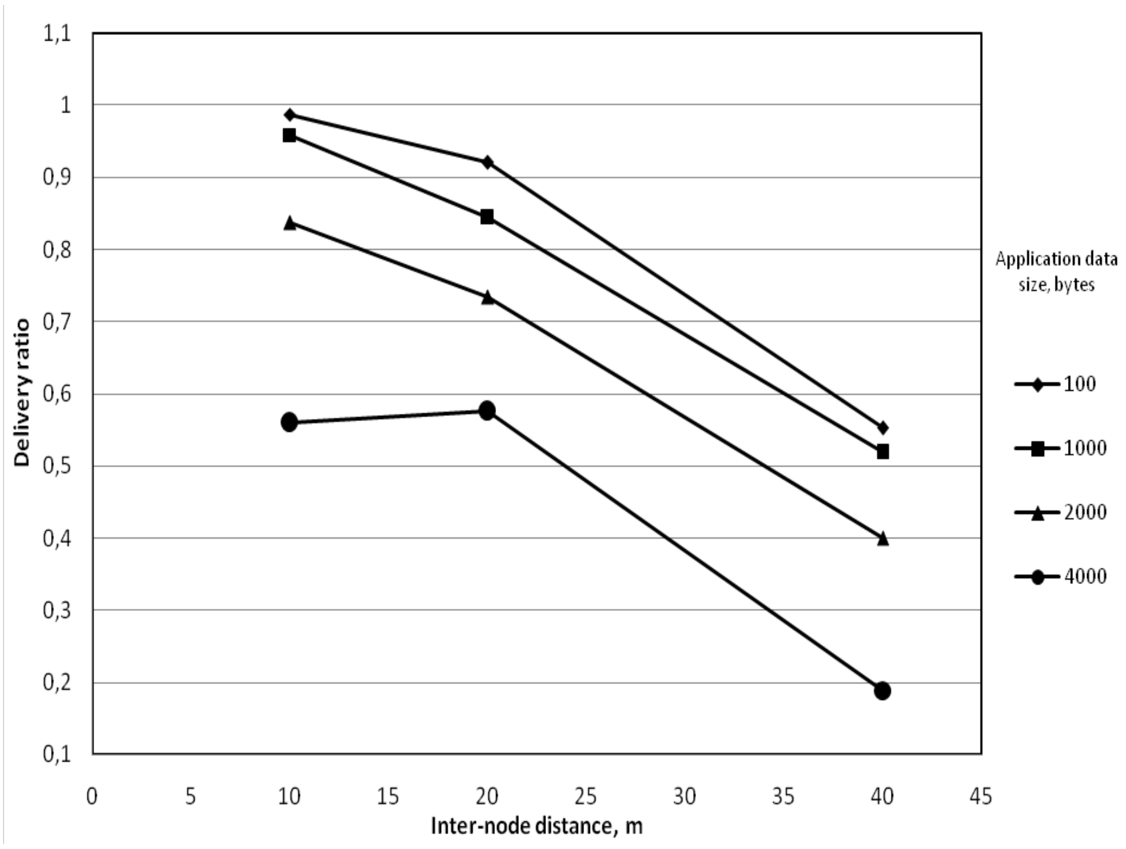


Figure 6.9: The delivery ratio versus inter-node distance, for different application data sizes in scenario three.

Table 6.10: The average round trip time values with respect to distance under different application data sizes in scenario three.

Inter-node distance, m	Application data size (bytes)			
	100	1000	2000	4000
10	0.227	15.748	15.902	31.862
20	0.156	15.718	15.971	31.999
40	0.295	15.775	15.816	31.603

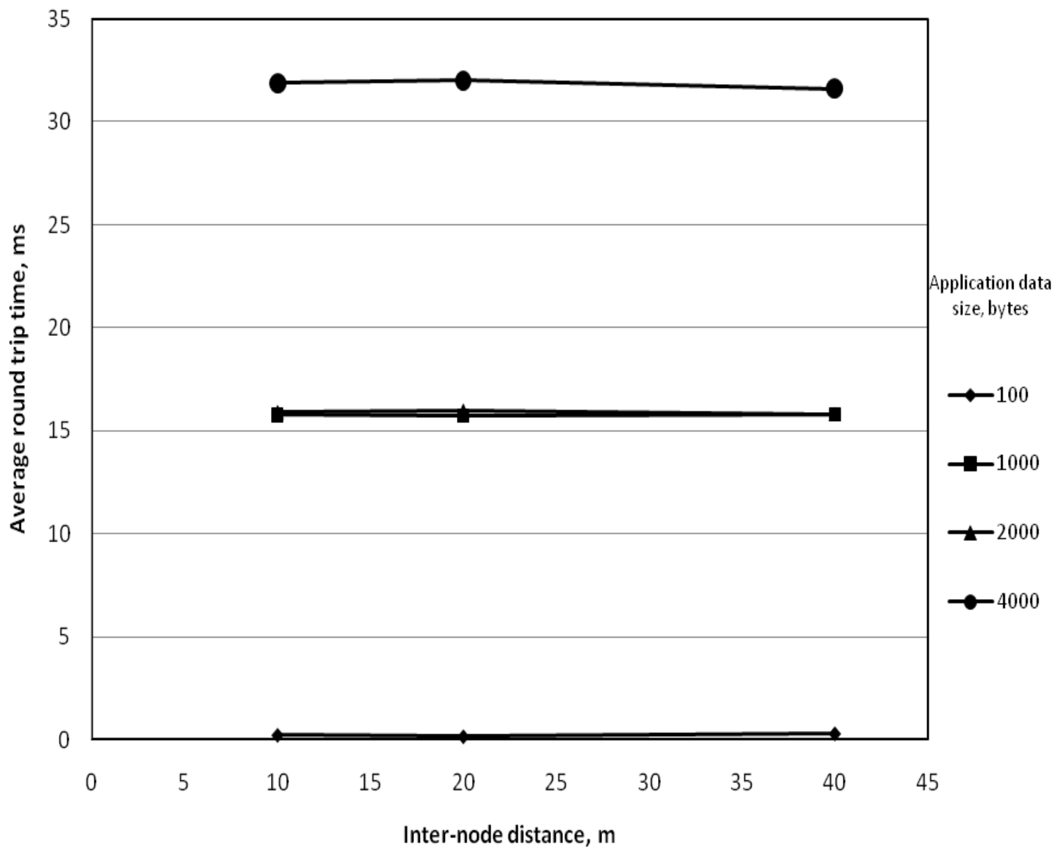


Figure 6.10: The average round trip time versus inter-node distance, for different application data sizes in scenario three.

Table 6.11: The average delivery ratio values with respect to number of intermediate nodes between the source and destination nodes, under different application data sizes.

Number of intermediate nodes	Application data size (bytes)				
	100	1000	2000	4000	8000
0	0.972	0.948	0.992	0.995	0.988
1	0.919	0.953	0.861	0.809	0.359
2	0.812	0.886	0.660	0.421	0.187
3	0.716	0.847	0.445	0.228	0.070

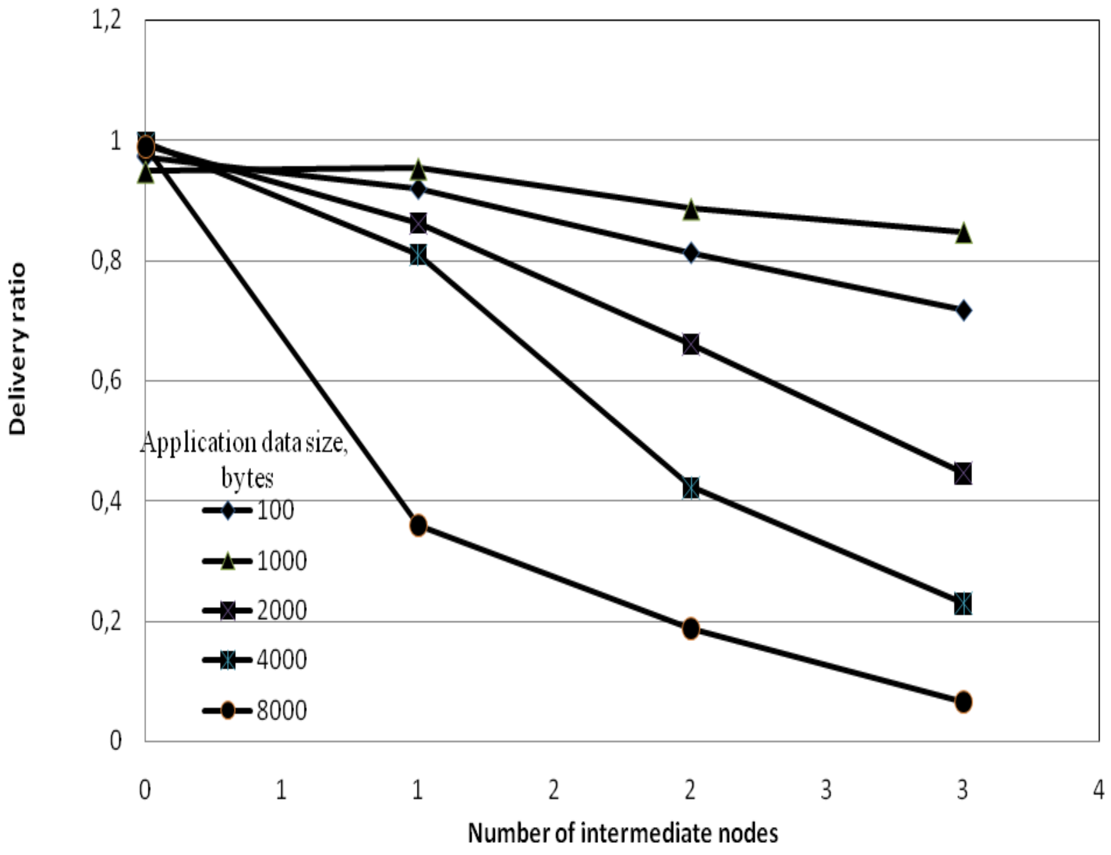


Figure 6.11: The delivery ratio versus the number of intermediate nodes between the source and destination nodes, for different application data sizes.

In single path group of experiments that is configured in figure 5.3 the number of intermediate nodes between the source and destination node was varied from 0 to 3 and a series of experiments were performed with different application data sizes. Table 6.11-6.12 and the corresponding figures 6.11 and 6.12 show the behavior of packet (message) delivery ratio and average round trip time on the number of intermediate nodes between the source and destination nodes.

Table 6.12: The average round trip time values with respect to number of intermediate nodes under different application data sizes.

Number of intermediate nodes	Application data size (bytes)				
	100	1000	2000	4000	8000
0	0.227	15.743	15.903	31.700	78.210
1	2.17	25.66	60.94	111.969	249.99
2	16.05	62.66	94.41	190.5	410.9
3	17.98	78.97	180.78	261.25	515

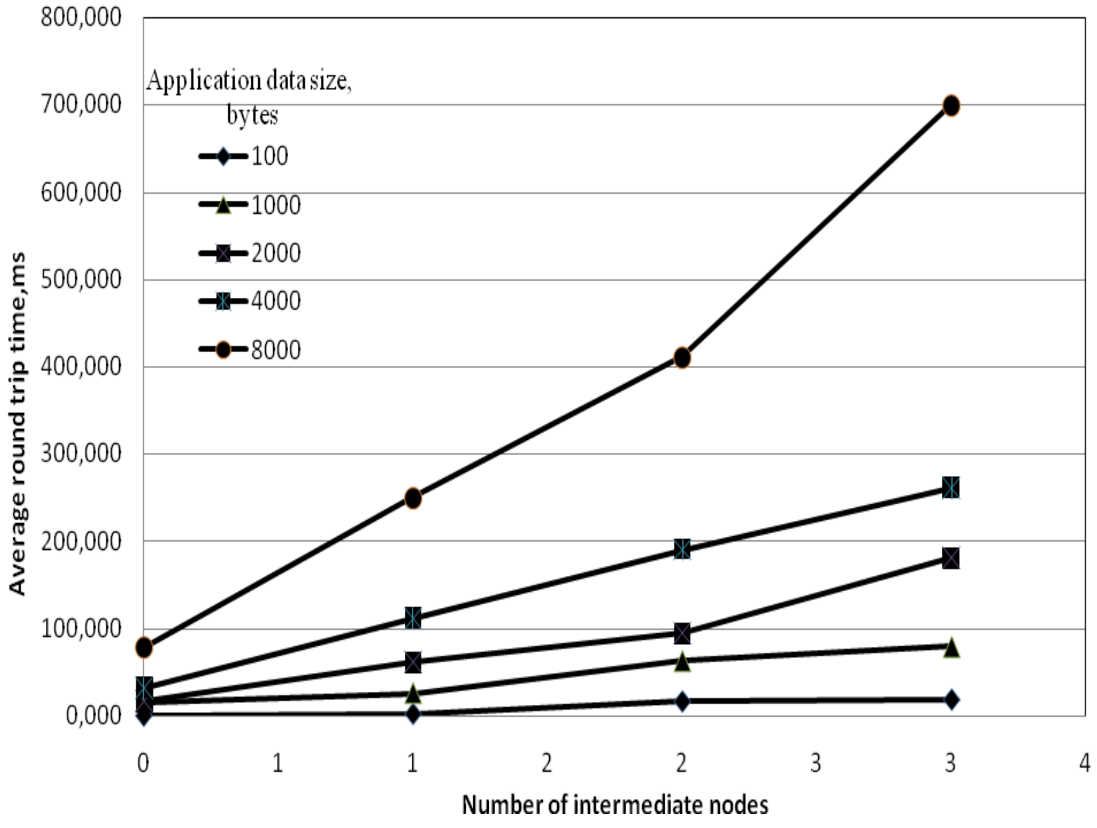


Figure 6.12: The average round trip time versus the number of intermediate nodes between the source and destination nodes, for different application data sizes.

Table 6.13: The average round trip time values with respect to data size under two different scenarios.

Application data size, bytes	Average Round Trip time (ms)	
	5 nodes	10 nodes
50	1.072	16.24
400	11.835	32.15
800	36.65	57.12
2000	61.461	157.7
4000	131.874	347.3

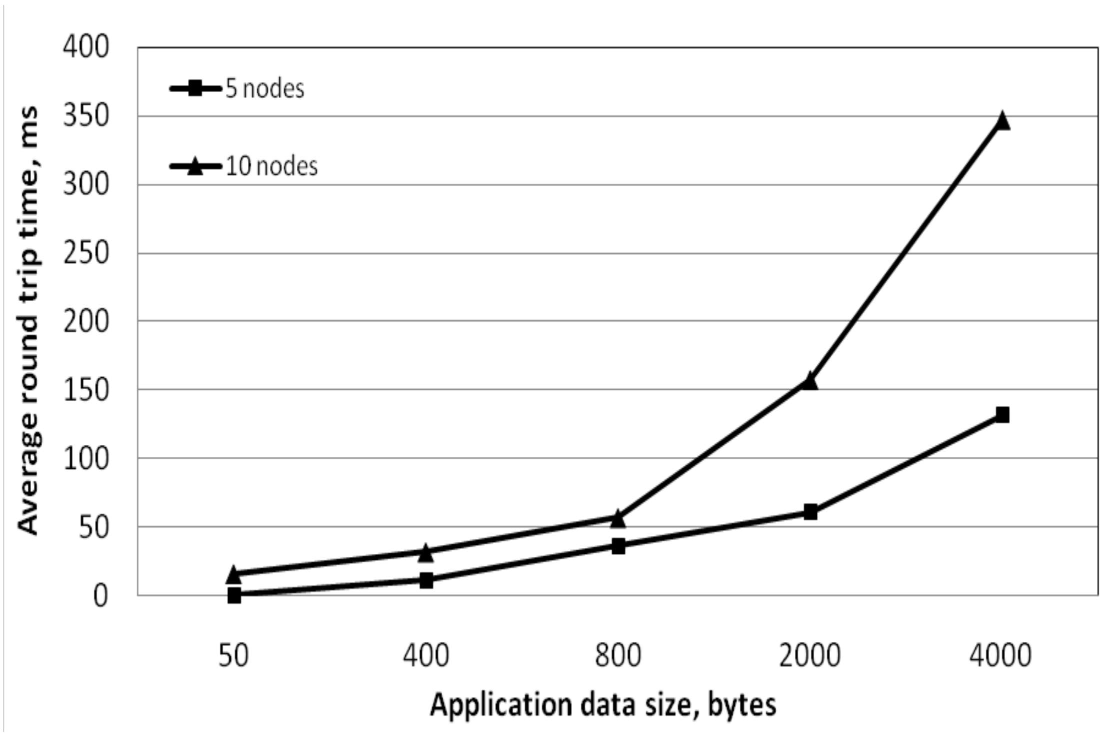


Figure 6.13: The average round trip time versus application data sizes between the source node and the destination node in an open area with different number of fixed nodes.

The result of multi-path experiments that was performed with the network configuration presented in Figure 5.4 and Figure 5.5 is shown in Tables 6.13-6.15 and presented in Figures 6.13-6.15. Routing and data dissemination from the source node to the destination node is investigated in these configurations with fixed and mobile nodes. The graphs display the comparative results of the experiments.

Table 6.14: The delivery ratio with respect to data size under two different scenarios.

Application data size, bytes	Delivery ratio	
	5 nodes	10 nodes
50	0.986	0.957
400	0.985	0.933
800	0,981	0,911
2000	0.951	0.652
4000	0.741	0.405

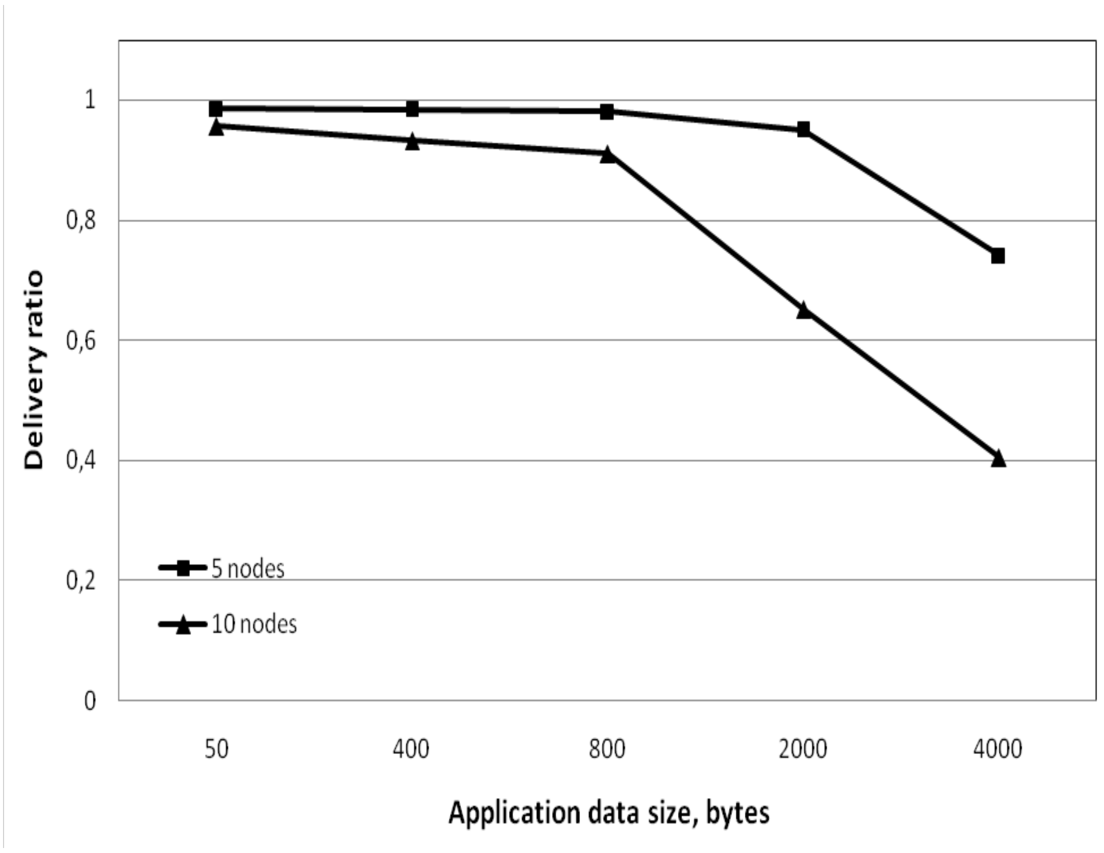


Figure 6.14: The delivery ratio versus application data sizes between the source node and the destination node in an open area with different number of fixed nodes.

Table 6.15: The average number of hop values with respect to application data size in two different scenarios.

Application data size, bytes	Average number of hops	
	5 nodes	10 nodes
50	1.091	2.255
400	1.166	2.285
800	1.372	2.39
2000	1.515	2.726
4000	1.575	3.135

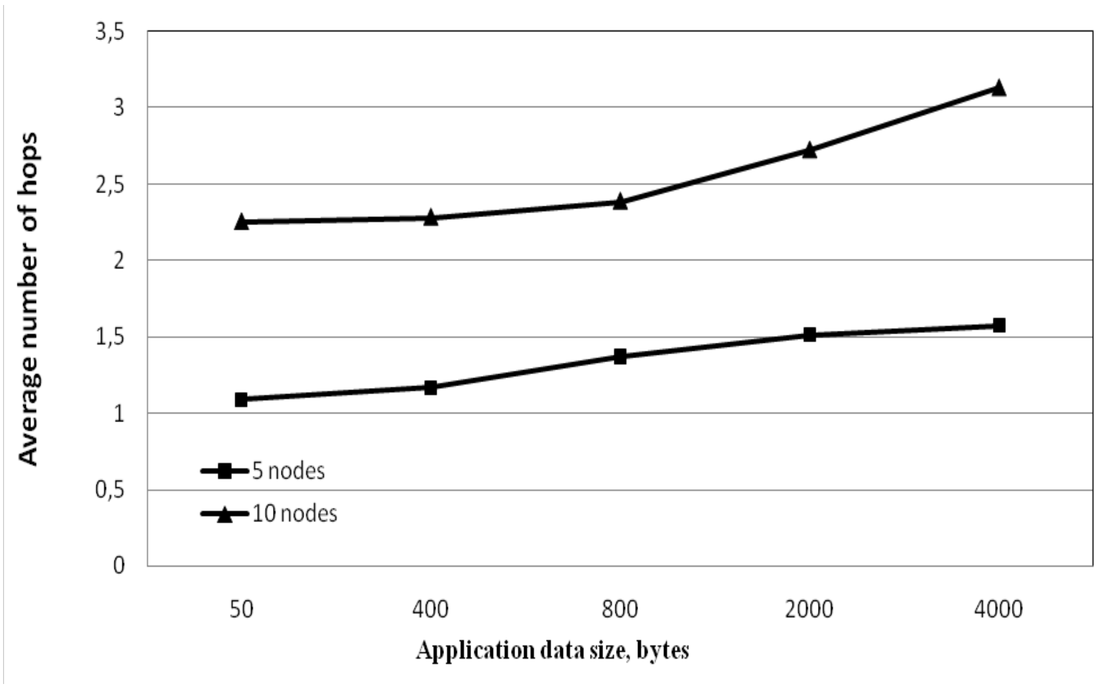


Figure 6.15: The average number of hop versus application data size with different number of fixed nodes.

The result of experiments that was performed with the network configuration shown in Figure 5.6 is given in Tables 6.16-6.24 and presented in figures 6.16-6.24. During the performance of the experiments, the source node was placed at the center and three destination nodes were positioned on a circle with equal distances from the source node and from the neighbor destination nodes as explained in the previous section of the thesis. The inter-node distance between the source node and the destination nodes was varied from 30m to 120m. In all these distances, the three destination nodes were in the coverage area of the source node. After 120m the source node could not reach to the destination node under the given conditions.

Graphs in Figures 6.16-6.21 demonstrate the dependence of the delivery ratio and average round trip time on inter-node distance with different application data sizes for the first group of experiments. In the graphs corresponding performance metric values are given for three different directions with the overall value of three destinations.

Graphs in Figures 6.22 and 6.23 present delivery ratio and overall average round trip time on inter-node distance with different application data sizes.

In Table 6.24 and the corresponding in Figure 6.24, the effect of the distance of the laptops to the ground level can be seen. A small group of experiments were conducted to see the delivery ratio difference between laptops 50cm high from the ground and laptops 100cm high from the ground with 120 meter distance between two nodes.

Table 6.16: The average delivery ratio values with respect to distance and the application data size is fixed to 50 bytes.

Inter-node distance, m	Delivery ratio			
	Direction 1	Direction 2	Direction 3	Overall
30	0.999	0.999	0.999	0.999
60	0.998	0.986	0.998	0.998
90	0.988	0.932	0.996	0.972
120	0.819	0.703	0.788	0.770

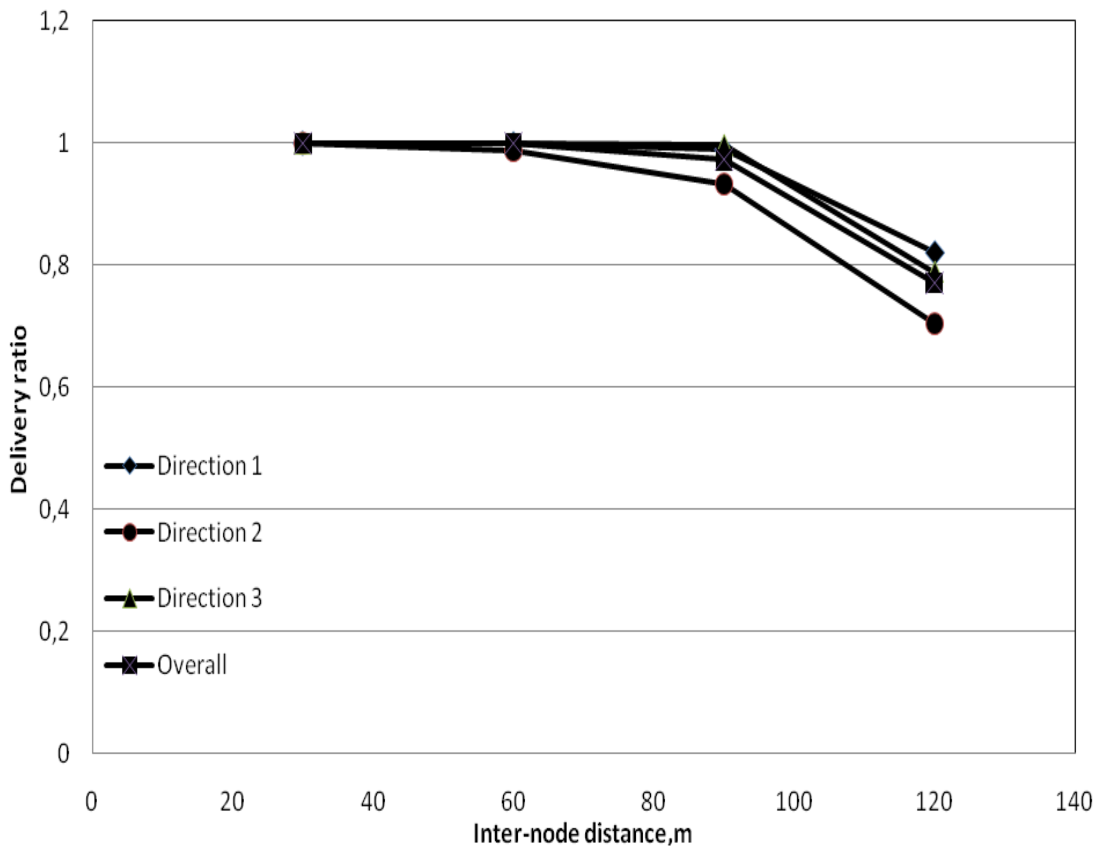


Figure 6.16: The delivery ratio versus inter-node distance, for different directions with application data size is 50 bytes.

Table 6.17: Average round trip time values with respect to distance when the application data size is 50 bytes.

Inter-node distance, m	Average round trip time (ms)			
	Direction 1	Direction 2	Direction 3	Overall
30	0.150	0.122	0.110	0.128
60	0.201	0.198	0.244	0.214
90	0.336	0.210	0.299	0.281
120	0.672	0.567	0.659	0.633

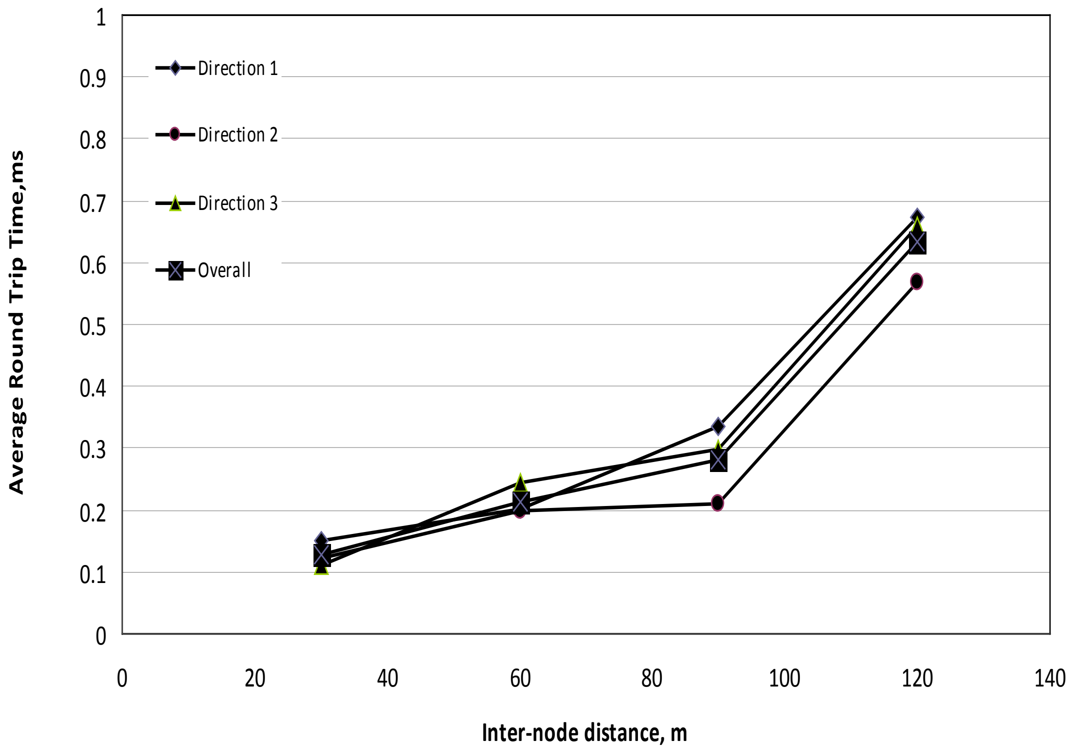


Figure 6.17: The average round trip time versus inter-node distance, for different directions with application data size = 50 bytes.

Table 6.18: Average delivery ratio values with respect to distance when the application data size is 800 bytes.

Inter-node distance, m	Delivery ratio			
	Direction1	Direction2	Direction3	Overall
30	0.999	0.999	0.999	0.999
60	0.994	0.988	0.993	0.992
90	0.863	0.944	0.727	0.844
120	0.830	0.684	0.752	0.755

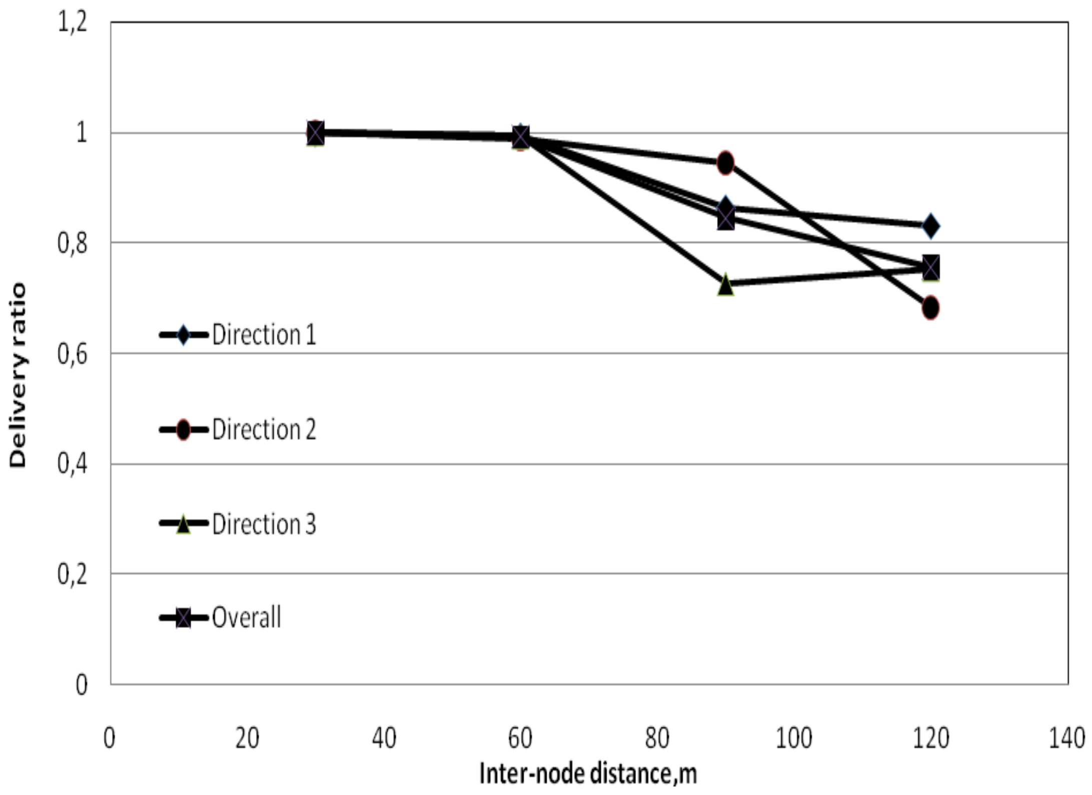


Figure 6.18: The delivery ratio versus inter-node distance, for different directions with application data size = 800 bytes.

Table 6.19: The average round trip time values with respect to distance when the application data size is 800 bytes.

Inter-node distance, m	Average round trip time (ms)			
	Direction1	Direction 2	Direction 3	Overall
30	23.947	19.787	18.7	20.811
60	18.72	17.463	18.446	18.212
90	16.492	17.018	20.991	18.167
120	21.845	24.977	17.244	21.355

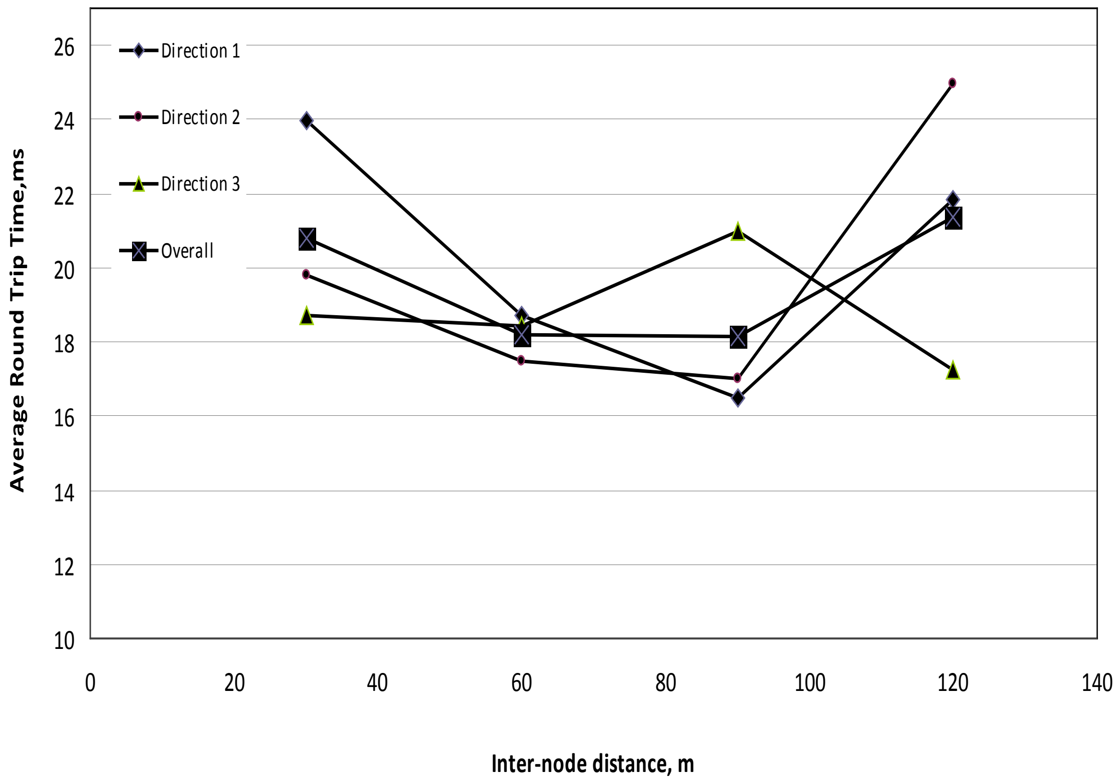


Figure 6.19: The average round trip time versus inter-node distance, for different directions with application data size = 800 bytes.

Table 6.20: The average delivery ratio values with respect to distance when the application data size is 4000 bytes.

Inter-node distance, m	Delivery ratio			
	Direction1	Direction 2	Direction 3	Overall
30	0.989	0.926	0.989	0.968
60	0.99	0.942	0.991	0.974
90	0.961	0.700	0.969	0.877
120	0.769	0.373	0.487	0.543

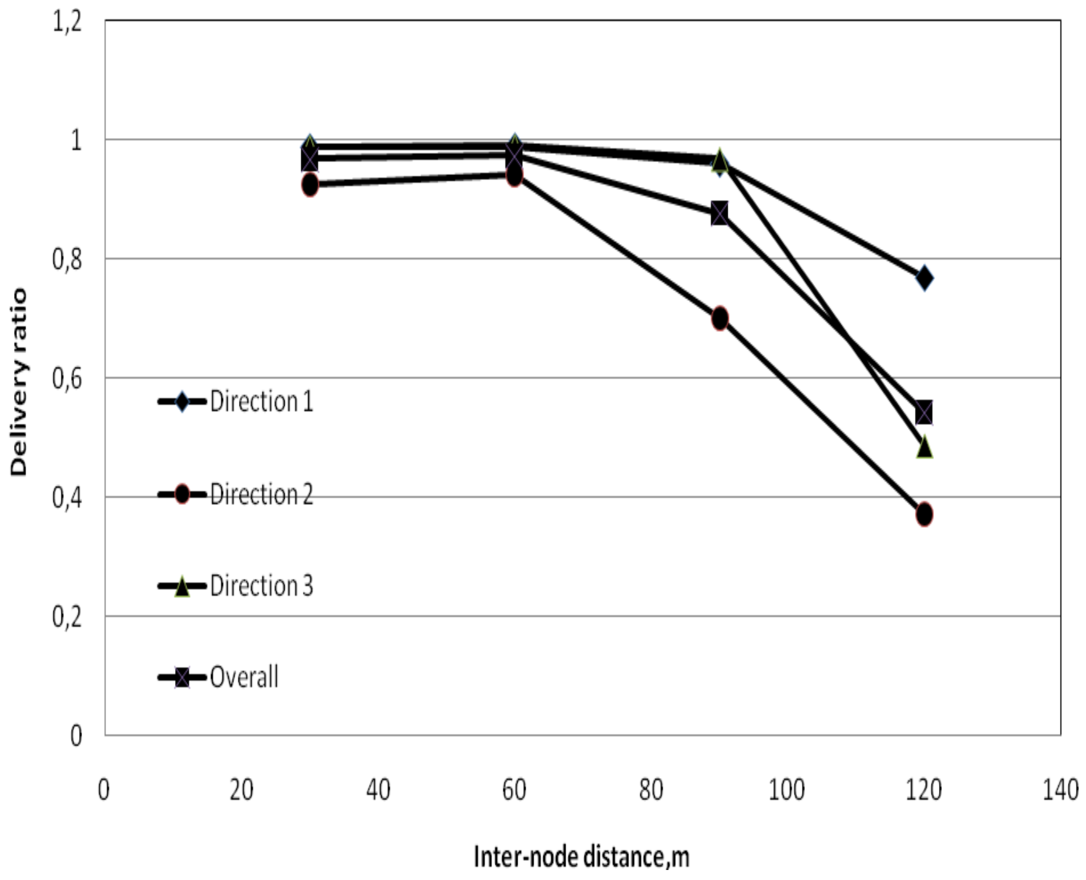


Figure 6.20: The delivery ratio versus inter-node distance, for different directions with application data size = 4000 bytes.

Table 6.21: The average round trip time values with respect to distance when the application data size is 4000 bytes.

Inter-node distance, m	Average round trip time (ms)			
	Direction1	Direction2	Direction3	Overall
30	32.846	33.223	28.781	31.617
60	36.2	32.905	30.275	33.127
90	46.954	35.786	46.5	43.08
120	41.366	39.751	31.25	37.456

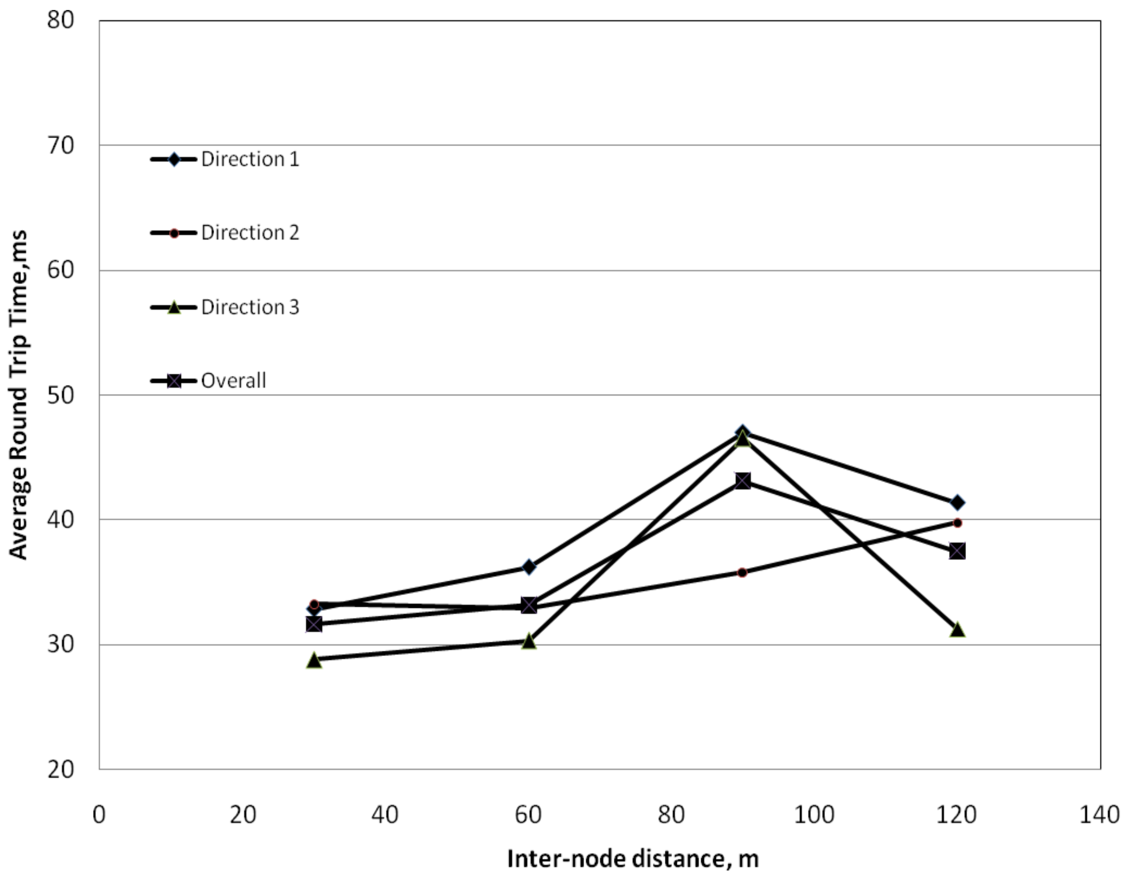


Figure 6.21: The average round trip time versus inter-node distance, for different directions with application data size = 4000 bytes.

Table 6.22: The overall round trip time values with respect to inter-node distance under different application data sizes.

Inter-node distance, m	Application data sizes (bytes)		
	50	800	4000
30	0.128	20.811	31.617
60	0.214	18.210	33.127
90	0.281	18.167	43.080
120	0.633	21.355	37.456

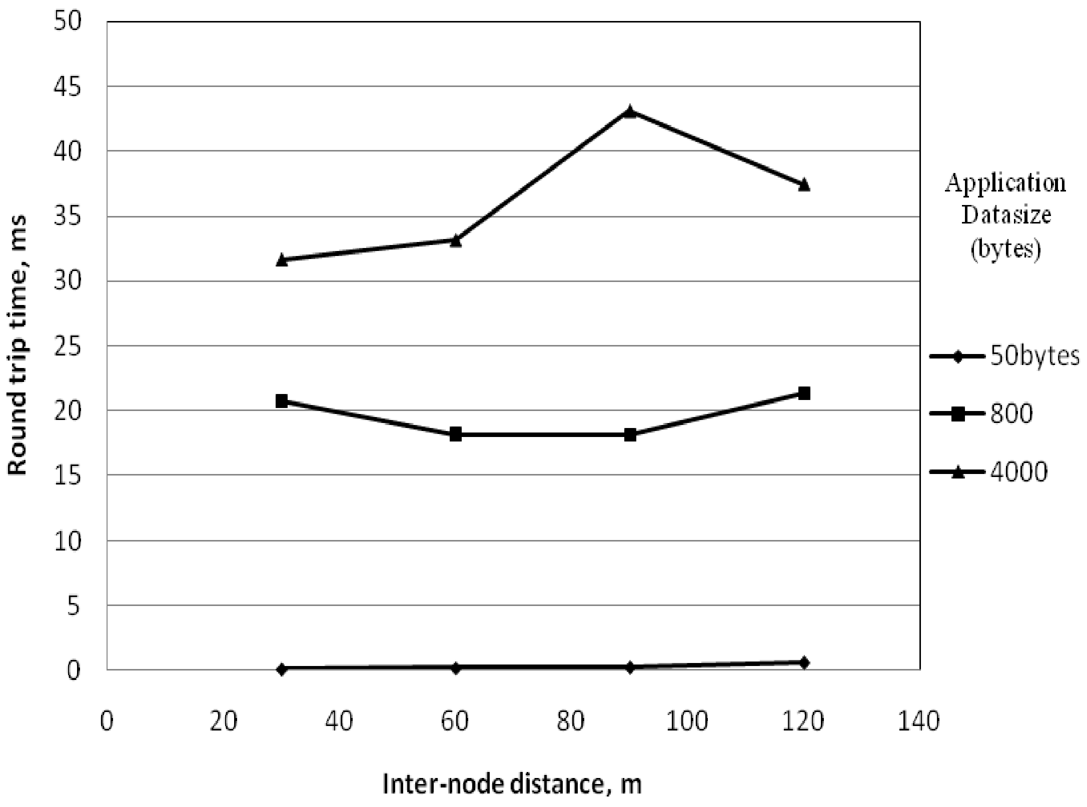


Figure 6.22: The average round trip time (overall) versus inter-node distance for different application data sizes.

Table 6.23: The overall delivery ratio values with respect to distance under different application data sizes.

Inter-node distance, m	Application data size (bytes)		
	50	800	4000
30	0.999	0.999	0.968
60	0.998	0.992	0.974
90	0.972	0.844	0.877
120	0.770	0.755	0.543

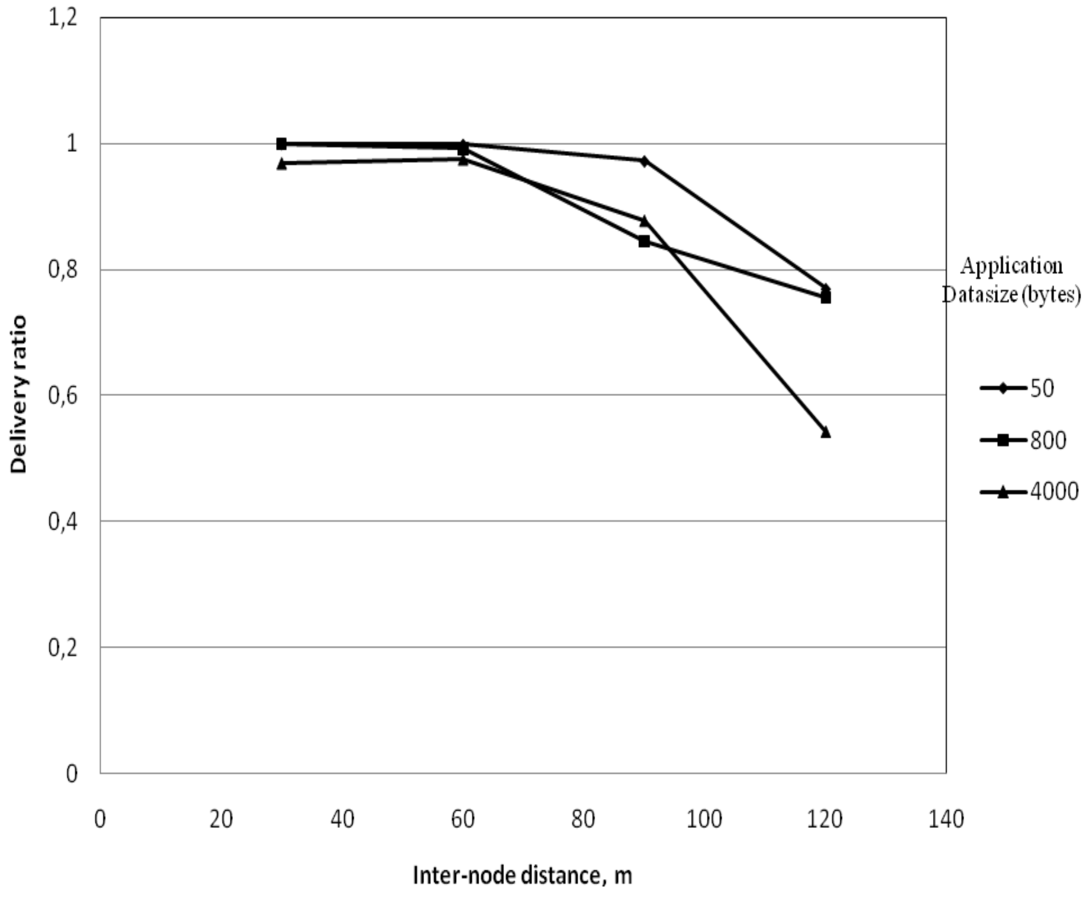


Figure 6.23: The delivery ratio (overall) versus inter-node distance, for different application data sizes.

Table 6.24: The average delivery ratio values with respect to data size under different height values from the ground.

Application data size, bytes	Delivery ratio	
	Height=50cm	Height=100cm
50	0.770	0.996
800	0.755	0.998
4000	0.543	0.995

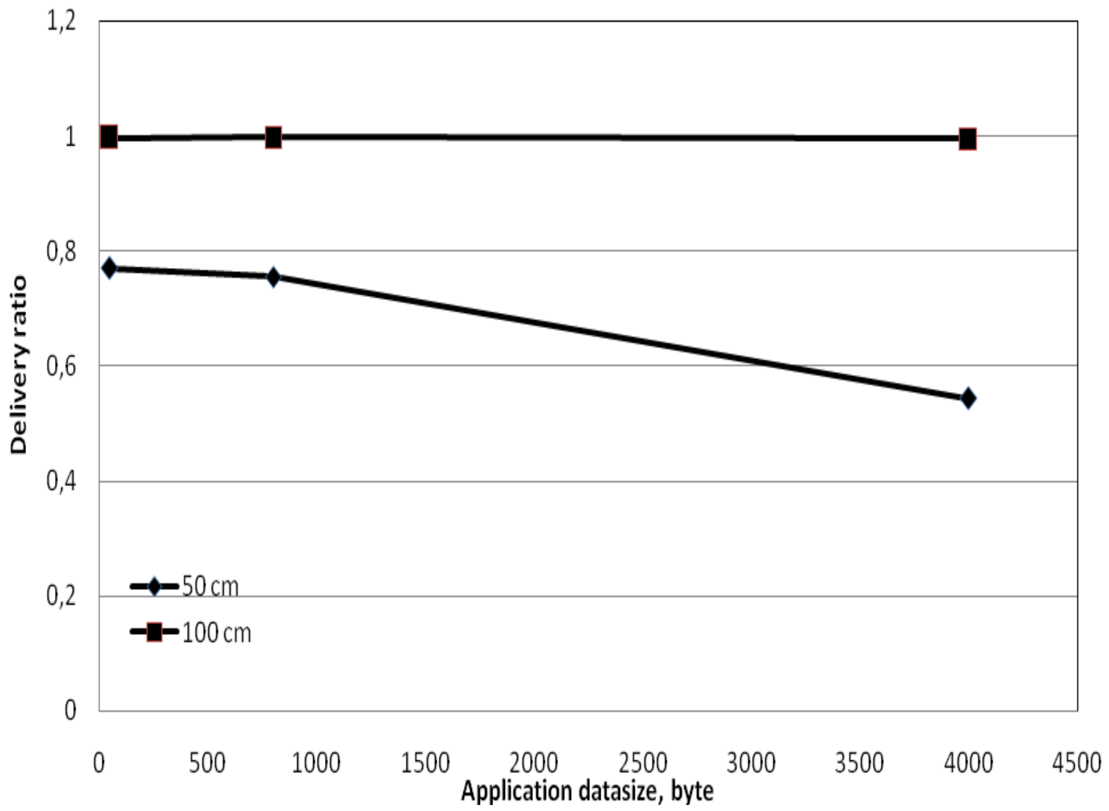


Figure 6.24: The average delivery ratio versus application data size under different height of the laptops to the ground level.

6.3 Discussion of the Experimental Results

In Section 5.1, some group of experiments were conducted between two nodes to understand the behavior of delivery ratio and the average round trip time with different application data size and inter-node distances. More information can be found in Section 5.1 in this study. Based on the obtained experimental results that were explained in Section 5.1, one can make the following inferences.

- The average round trip time and delivery ratio metrics depend on the number of intermediate nodes between the source node and the destination node and the size of the application data.
- As Figure 6.2 demonstrates, the delivery ratio which is almost constant at low distances (up to 90 m) starts to decrease at high inter-node distances for all application data sizes. The decrement in delivery ratio increases as application data size increases.
- The average round trip time increases with the increase of the application data size (Figure 6.1). Especially it is quite high for a large application data size, since in this case there is more than one packet transmission. For small application data size it remains quite low. From the same graph it is also clear that the average round trip time does not depend on the distance between two nodes, if the destination node is in the coverage area of the source node.

From the results of the second group of experiments that were conducted in Section 5.1, one can say that:

- As the inter-packet transmission time increases between the packets, the delivery ratio also increases (Figure 6.3). This will allow more packets to get to their destinations, as there is less possibility of collision since the load of the network is low. From the same graph, it can be seen that the delivery ratio is lower for high application data sizes since there is a fragmentation of the packets.

- Figures 6.4 show that for the same inter-packet transmission time, the average round trip time, is high for high application data size and on the other hand, it is decreasing slowly as the inter-packet transmission time is increased.

Based on the results of experiments that were configured in Figure 5.2, these inferences can be made.

- As it was expected, the average round trip time increases with the increase of the application data size (Figures 6.6, 6.8 and 6.10). Especially it is quite high for a large application data size, since in this case, there is more than one packet transmission. For small application data size it remains quite low. As it was explained before, if the destination node is in the coverage area of the source node, the distance between them do not affect the average round trip time.
- Figures 6.5, 6.7, and 6.9 show that the packet delivery ratio considerably decreases with the increase in the inter-node distance between the source node and the destination node. This performance metric is quite low for large number of inter-nodes distances, since large number of packets is lost on the way from the source node to the destination node.
- From the same graphs of Figures 6.5, 6.7, and 6.9 one can also see that, there is a large decrement in the packet delivery ratio when application packet size increases (especially for 4000 bytes and 8000 bytes). In wireless ad hoc

networks Bit Error Rate of a radio link is high, therefore, the probability of a packet to get corrupted or lost increases with the increasing packet size. For a large application data size, more than one packet is transmitted since there is a limitation on the frame size in IEEE 802.11 MAC layer [33].

In the experiments with the network configuration given in Figure 5.3, the number of intermediate node varied and the performance metrics are calculated for different application data sizes. The analyses of the results are as follows.

- The used performance metrics depend on the number of intermediate nodes between the source node and the destination node and the size of the application data.
- Figure 6.11 shows that the packet delivery ratio considerably decreases with the increase in the number of intermediate nodes between the source node and the destination node. This performance metric is quite low for large number of intermediate nodes, since large number of packets is lost on the way from the source node to the destination node.
- As the number of intermediate node increases between the source node and the destination node, the average round trip time also increases (Figure 6.12). Each intermediate node performs some processing of the received packets. With the increase of the number of the intermediate nodes, the total packet delay also increases.

Based on the obtained experimental results from experiments that were configured in Figures 5.4 and 5.5, one can make the following inferences.

- The average round trip time increases with the increase of the application data size (Figure 6.13). Especially it is quite high for a large application data size, since in this case, there is more than one packet transmission. For small application data size it remains quite low. From the same graph, it is also clear that, the average round trip time with 10 nodes experiments case is higher than the average round trip time with 5 nodes in the network. When there are more nodes, the source node and the destination node are too far from each other.

- From Figure 6.14, it is noticeable that the delivery ratio is quit lower for the 10 nodes experiments case especially for large application data sizes. This is because in the 10 nodes case, we have increased the inter-node distance between the source node and the destination node and then filled the in-between distance by the intermediate nodes. Therefore, we have a decrease in the delivery ratio.

- Graph in Figure 6.15, illustrates the average number of hops from the source node to the destination node. Number of hops taken by each packet is increasing due to increase in packet size. Additionally, it is clear that number of hops taken by packets whose sizes are greater than 2000 bytes is at higher level when compared to smaller sized packets.

Based on the obtained experimental results from experiments that are configured in Figure 5.6, one can make the following inferences.

- The average round trip time depends on the application data size and the delivery ratio depends on the inter-node distance between the source node and the destination node and the application data size.
- The average round trip time increases with the increase of the application data size as it can be seen from the investigation of Figures 6.17, 6.19, 6.21, 6.22. Especially it is quite high for a large application data size, since in this case, there is more than one packet transmission. For small application data size it remains quite low. From the same graph, it is also clear that, the average round trip time does not depend on the distance between two nodes, if the destination node is in the coverage area of the source node.
- Figures 6.16, 6.18, 6.20 and 6.23 show that the packet delivery ratio considerably decreases with the increase in the inter-node distance between the source node and the destination node. This performance metric is quite low for large number of inter-nodes distances, since large number of packets is lost on the way from the source node to the destination node.
- From the graph of Figure 6.23 one can also see that, there is a large decrement in the packet delivery ratio when application packet size increases (especially for 4000 bytes. The reason of this is the fragmentation of the packets).

- In Figure 6.24, it can be seen that the delivery ratio is high when the nodes are placed higher positions from the ground level. Putting the nodes 100cm height from the ground level gave us better results when it is compared with 50cm height.

6.4 Average Values and Confidence Intervals of the Investigated Performance Metrics

In this section, average values and confidence intervals of the investigated performance metrics of the experiments that are configured in Figure 5.3 are provided. The performance metrics that were used in the experiments are delivery ratio and average round trip time.

Table 6.25: Average values and 95% confidence intervals of the performance metrics for application data size = 100 bytes.

Metric	The number of intermediate nodes between source and destination when application data size is 100 bytes		
	0	1	2
Delivery ratio	0.972±0.003	0.919±0.029	0.812±0.145
Average round trip time	0.227±0.032	2.17±1.549	16.05±0.227

Table 6.26: Average values and 95% confidence intervals of the performance metrics for application data size = 1000 bytes.

Metric	The number of intermediate nodes between source and destination when application data size is 1000 bytes		
	0	1	2
Delivery ratio	0.948±0.083	0.953±0.014	0.886±0.108
Average round trip time	15.743±0.100	25.66±0.336	62.66±0.366

Table 6.27: Average values and 95% confidence intervals of the performance metrics for application data size = 2000 bytes.

Metric	The number of intermediate nodes between source and destination when application data size is 2000 bytes		
	0	1	2
Delivery ratio	0.992±0.013	0.861±0.079	0.660±0.367
Average round trip time	15.903±0.061	60.94±0.244	94.41±0.305

Table 6.28: Average values and 95% confidence intervals of the performance metrics for application data size = 4000 bytes.

Metric	The number of intermediate nodes between source and destination when application data size is 4000 bytes		
	0	1	2
Delivery ratio	0.995±0.002	0.809±0.153	0.421±0.027
Average round trip time	31.7±0.088	111.969±0.271	190.5±5.236

Table 6.29: Average values and 95% confidence intervals of the performance metrics for application data size = 8000 bytes.

Metric	The number of intermediate nodes between source and destination when application data size is 8000 bytes		
	0	1	2
Delivery ratio	0.988±0.003	0.359±0.048	0.187±0.097
Average round trip time	78.210±0.168	249.99±2.787	410.9±7.339

Chapter 7

CONCLUSION

An application layer multithreaded program which has been developed in [37] was used for experimental investigation of data transmission in wireless ad hoc networks. In this program, pure flooding method is used for packet transmission or routing between the nodes.

A large number of experiments were conducted in an attempt to investigate the characteristics of the wireless ad hoc network in outdoor real-world network environment using the program. As a result, more than 500 experiments were run and a vast amount of raw data, (more than 6000 values out of 1000 data files), was analyzed. A number of performance metrics were observed under different conditions.

In the thesis, first of all, an extensive survey of routing protocols is introduced. Based on this survey, a classification of existing routing protocols is done with respect to their transmission method and categorized as unicast, multicast and anycast routing protocols. Another classification is done based on the working mechanism for each given category as reactive, proactive and hybrid protocols. Survey results of the existing experimental studies for investigating the performance of wireless mobile ad hoc networks have been also presented in the thesis.

Through our survey, it is found that in the literature, some experimental work is done to evaluate wireless mobile ad hoc routing protocols [21][22][25][34] and investigate the effect of 802.11 on real ad hoc scenarios [20]. These evaluations are done based on the distance between mobile nodes, node mobility, number of hops, traffic load, data size, transmission speed and inter-packet transmission time. The common performance metrics are given as delivery ratio, hop count, round trip time or end-to-end delay, throughput and control overhead.

In this study, we have considered delivery ratio, round trip time, number of hops with respect to application data size, hop count and inter-node distance under different wireless ad hoc network scenarios in outdoor environment. In the literature generally the experiments were done using linux operating system where in this study windows operating system was used.

In [24], some experiments were done in rectangular area. In their network configuration, all the nodes generate traffic and send the generated traffic to randomly selected destinations. Message delivery ratio, communication efficiency, hop count and end-to-end latency were used as a performance metrics and they used fixed packet size which was randomly generated with a mean 1200 bytes. In [26], the performance of the ad hoc network was investigated through the condunted experiments. The parameter which was used in the experiments is the system traffic load. The inter-node distance and the application data size in the experiments were kept fixed.

In [36], some experiments were conducted inside the campus with four nodes with some parameters such as the application data size and the inter-arrival packet time.

End-to-end delay and throughput were used as a performance metrics. For measuring the end-to-end delay in the network, Ping utility was utilized as a data source. The application data size was varied from 64 bytes up to 48856 bytes and the inter-arrival packet time was varied between 10 ms and 15 seconds.

In this study, the application data size was varied from 100 bytes up to 8000 bytes during the experiments and the inter-arrival packet time varied from 10ms up to 100ms. We are not be able to compare our results with other researchers' results completely because of the difference in the used performance metrics, parameter types, routing method and network configurations. Although in [36], same performance metrics were used with this study and it can be seen that the end-to-end delay (round trip time) increases when the application data size is increased and end-to-end delay is higher when the inter-arrival packet time is very low such as 10ms or 30 ms. After 50 ms, end-to-end delay does not change with respect to inter-arrival packet time. However, in [36], they did not measure how the delivery ratio is affected with the inter-arrival packet time. In this thesis, it is also measured and it's effects are discussed.

The developed program and the results of the experiments can be used for investigation of different schemes of routing and information dissemination in real-world wireless ad hoc network and as data for sensible simulations. Also, as a future work, with the use of the program testing of existing routing protocols can be done in real-world environments as well as new routing protocols can be developed to improve performance of wireless mobile ad hoc networks.

REFERENCES

[1] Siva Ram Murthy C., & Manoj B. S.(2004). Ad hoc wireless networks: Architectures and protocols. *Prentice Hall*.

[2] Liu C., & Kaiser J. (2005). A Survey of mobile Ad Hoc network Routing Protocols. *University of Magdeburg.*

[3] Abolhasan M., Wysocki T.A., & Dutkiewicz E. (2004). A Review of Routing Protocols for Mobile Ad hoc Networks. *In Elsevier Journal of Ad hoc Networks,* 2 , 1-22.

[4] Perkins C. E., & Royer E. M. (1999). Ad Hoc On Demand Distance Vector routing. *Proceedings of 2nd IEEE Workshop, Mobile Computer System and Applications,* 90-100.

[5] Johnson D. B., & Maltz D. A. (1996). Dynamic Source routing in ad hoc wireless networks. *in Mobile Computing*, vol. 353, Chapter 5, pp. 153-181, Kluwer Academic Publishers.

[6] Jacquet P., Muhlethaler P., Clausen T., Laouiti A., Qayyum A., & Viennot L. (2001). Optimized link state routing protocol for ad hoc networks. *In Proceedings of the 5th IEEE Multi Topic Conference (INMIC 2001).*

[7] Garcia-Luna-Aceves J. J., & Spohn C. M. (1999). Source-tree routing in wireless networks. *In Proceedings of the Seventh Annual International Conference on Network Protocols*, Toronto, Canada, 273-282.

[8] Karp B., & Kung H. T. (1998). Dynamic neighbor discovery and loopfree, multi-hop routing for wireless, mobile networks. *Harvard University*.

[9] Haas Z. J. (1997). A New Routing Protocol For The Reconfigurable Wireless Networks. *In Proceedings of 6th IEEE International Conference on Universal Personal Communications, IEEE ICUPC'97,* San Diego, California, 562-566.

[10] Royer E. M., & Perkins C. E. (1999). Multicast Operation of the Ad hoc On-Demand Distance Vector Routing Protocol. In *Proc. Of the 5th annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, Seattle, WA, 207-218.

[11] Lee S., Gerla M., & Chiang C. (1999). On-Demand Multicast Routing Protocol. In *Proc. of the Wireless Communications and Networking Conference (WCNC),* New Orleans, LA,1298-1302.

[12] Jacquet P., Minet P., Laouiti A., Viennot L., Clausen T., & Adjih C. (2001). Multicast optimized link state routing. *IETF Internet Draft: draft-ietfmanet-olsr-molsr*.

[13] Devarapalli V., & Sidhu D. (2001). MZR: A Multicast Protocol for Mobile Ad Hoc Networks. *In IEEE International Conference on Communications (ICC),* Helsinki, Finland.

[14] Wang J., Zheng Y., Jia W. (2003). An AODV-Based Anycast Protocol in Mobile Ad hoc Network. *International Symp. of IEEE on Personal,Indoor, and Mobile Radio Communication Proceedings*.

[15] Peng G., Yang J., & Gao C. (2004). ARDSR: An Anycast Routing Protocol for Mobile Ad hoc Network. *Symp. On Emerging Technologies of IEEE on Mobile and Wireless Communication*.

[16] Saeed A., Khan L., Shah N., & Ali H. (2009). Performance Comparison of two Anycast based reactive routing protocols for mobile Ad Hoc networks. *International Conferance on 2nd Computer, Control and Communication (IC4 2009).* 1-6.

[17] Martin M., & Takuro S. (2009). Route-Count Based Anycast Routing in Wireless Ad Hoc Networks. *Vehicular Technology Conference Fall (VTC 2009-Fall), 2009 IEEE 70th*. 1-5.

[18] Kashif S., Lijuan C., Tu W., & Teresa D. (2008). A Hybrid Anycast Routing Protocol for Load Balancing in Heterogeneous Access Networks. *Computer Communications and Networks, 2008. ICCCN '08. Proceedings of 17th International Conference*.1-6.

[19] Kotz D., Newport C., Gray R.S, Liu J., Yuan Y., & Elliott C. (2004). Experimental Evaluation of Wireless Simulation Assumptions. *In Proceedings of MSWiM 2004*.

[20] Dhoutaut D., & Guerin-Lassous I. (2003). Experiments with 802.11b in ad hoc configurations. *The 14th IEEE 2003 International Symposium on Personal, Indoor and Mobile Radio Communication Proceedings*. 1618-1622.

[21] Srivastava V., Hilal A.B., Thompson M.S., Chattha J.N., MacKenzie A. B., & DaSilva L.A. (2008). Characterizing Mobile Ad Hoc Networks – The MANIAC Challenge Experiment. *In WiNTECH*.

[22] Kazemi H., Hadjichristofi G., & DaSilvia L. A. (2008). MMAN – A Monitor for Mobile Ad Hoc Networks: Design, Implementation, and Experimental Evaluation. *In WiNTEC*.

[23] Macker J., & Lee R. (2007). http://cs.itd.nrl.navy.mil/work/olsr/

[24] Gray R.S., Kotz D., Newport C., Dubrovsky N., Fiske A., Liu J., Masone C., McGrath S., & Yuan Y. (2004). Outdoor Experimental comparison of four ad hoc routing algorithms. *In Proceedings of MSWiM 2004*.

[25] Juwad M.F., & Al-Raweshidy H.S. (2008). Experimental Performance Comparison between SAODV&AODV. *Second Asia International Conference on Modelling & Simulation*. 247-252.

[26] Zhong X., Mei S., Wang Y., & Wang J. (2004). Experimental Evaluation of Stable Adaptive Routing Protocol. *In proceedings of WCNC 2004.* 1563-1567.

[27] Oz G., Kostin A., Oyeniyi A. M., Sharghi Z., & Seifzadeh S. (2008). Prototype Application-Layer Test Bed for Implementation and Investigation of Routing and Data Dissemination in Wireless Mobile Ad Hoc Networks. In *10th International Workshop on Computer Science and Information Technologies CSIT2008*, 27-31.

[28] Obraczka K., & Kumar V. (2001). Flooding for Reliable Multicast in Multi-hop Ad Hoc Networks. *Wireless networks, Springer,* vol. 7, no. 6.

[29] Crow B. P., Widjaja I., Kim J. G., & Sakaim P. T. (1997). IEEE 802.11 Wireless Local Area Networks. *IEEE communication Magazines*. 116-126.

[30] Brenner P. (1997). A Technical Tutorial on the IEEE 802.11 Protocol. *Breez-eCom*.

[31] Vincent L., & Margaret M. (2009). Repeatable and Realistic Experimentation in Mobile Wireless Networks. *IEEE Transactions on Mobile Computing, v*ol 8, Issue 12, 1718-1728.

[32] Yu S., Zhou W., & Wu Y. (2002). Research on Network Anycast. *Proc. of the Fifth Intl. Conf. on Algorithms and Architectures for Parallel Processing (ICA3PP02).* 154-161.

[33] Kotz D., Newport C., Gray R. S., & Liu J. (2007). Experimental Evaluation of Wireless Simulation Assumptions. *Transactions of the Society for Modeling and Simulation International*, vol. 83, 643-661.

[34] Cao J., & Wu W. (2008). A multi-metric QoS Routing Method for Ad Hoc Network. *The 4th International Conference on Mobile Ad hoc and Sensor Networks*, 99-102.

[35] Dow C.R., Hsuan P., & Hwang S.F. (2006). Design and Implementation of Anycast Protocols for Mobile Ad Hoc Networks. *ICACT2006*.419-424.

[36] Toh C. K., Chen R., Delwar M., & Allen D. (2000). Experimenting with an Ad Hoc wireless network on campus: insights and experiences. *ACM SIGMETRICS Performance Evaluation Review*. 21-29.

[37] Methods, Models and Algorithm of Dissemination of information in Wireless Mobile Ad Hoc Networks BAB-A-08-10, EMU, October 2009 - March 2011 (ongoing).

APPENDICES

Appendix A: The Source Text of the Application-Layer Program

```
/*       A protocol for an ad hoc wireless networks                                             */
/*                                                                                              */
/*       Usage: For the originator: prgname output_filename num_of_msg msg_size dest_IP delay   */
/*        For the others      :progname          output_filename                                */
/*
                                                                                                          */
/*                          num_of_msg is the number of sent message                            */
/*                          destination_IP is the destination host IP                           */
/*  Visual C++ Environment.                                                                     */
/*  In Project-->Settings-->Object/library:                                                     */
/*  the libraries LIBCMT.lib and WSOCK32.lib must be added                                      */
/*  and "Ignore all default libraries" be selected.                                             */
/*                                                                                              */
/*  Initially start destination, then intermediate and finaly start originator                  */
/*  on different wireless hosts.                                                                */
/*                                                                                              */
/*       On the originator host, originating thread will send request,                          */
/*                           relaying thread will discard its own message.                      */
/*       On the other hosts originating thread will not send anything,                          */
/*                          will wait for the termination of the relaying thread.               */
/*                                                                                              */
/*       Average number of hops( hop counts is added)                                           */
/*       originator and intermediate nodes                                                      */
/*       Filename  : adhoc.cpp                                                                  */
/*       Last Update         : March 16, 2010                                                   */
/*                                                                                              */
/*****************************************************************************************/


#define _MT                                                /*  to  use  a  declaration  of  _beginthreadex()  in
process.h */

#include <time.h>
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <memory.h> /* Not necessary */
#include <string.h>
#include <process.h>
#include "multcast.h"

/* Default multicast and destination port number to use */
#define DESTINATION_MCAST  "234.55.66.77"
#define MY_PORT  8888
#define DESTINATION_PORT  8888

// ************************** modified part 1 out of 4 ***********************************
#define MAX_NO_OF_NODES 10
// ************************** modified part 1 finished ***********************************
#define MAXWIN 20                                                 /* Maximum number in sliding window */
#define MAXMSGS 2100                                              /* Maximum number of messages */

/* Variables */
int WSAInitFailed;
char strDestMulti[MAXHOSTNAME] = {DESTINATION_MCAST};
char strSrcMulti[MAXHOSTNAME] = {DESTINATION_MCAST};

u_short nDestPort = DESTINATION_PORT;
u_short nMyPort  = MY_PORT;

SOCKET hSockSnd = INVALID_SOCKET;                                         /* To send in originating thread */
SOCKET hSockRcv = INVALID_SOCKET;                                         /* To receive in relaying thread */
SOCKET hSockFrwrd = INVALID_SOCKET;                                    /* To send in relaying thread */


struct sockaddr_in stDestAddr, stSrcAddr;
WSADATA stWSAData;

static int nOptName = IP_ADD_MEMBERSHIP;                                                /* Multicast option */
static int nLoopback = IP_MULTICAST_LOOP;                                       /* Multicast loopback option */
static int nRecvTimeout = SO_RCVTIMEO;                                                   /*    Time    of    for
recvfrom() */


/* Variables for roundtrip time calculations */
struct RoundTrip
{
          long msg_id;
                              /* message identifier */
          DWORD sndtime;                                             /*  Send  time  of  the  request  message
*/
          DWORD rcvtime;                                         /* Receive time of the reply message */
          DWORD rtttime;                                        /* Round trip time of the message */
};

/* Variable to find average round trip time */
DWORD sum_rtt_org = 0;
double average_rtt_org;
long total_ave_val = 0;

DWORD sum_rtt_inter = 0;
double average_rtt_inter;

/* Variables to calculate average hop count at the originator and the destination */
```

```
long sum_hop_cnt_org = 0;
double average_hop_cnt_org;
int hop_cnt_rply = 0;


long sum_hop_cnt_dest = 0;
double average_hop_cnt_dest;
int hop_cnt_rqst = 0;


struct RoundTrip OrgRtt[MAXMSGS];
struct RoundTrip InterRtt[MAXMSGS];


char rcvbuffer[9000];                                                    /* Storage for a received message */
char destbuffer[9000];                                            /* Storage for send message at destination */
char sndbuffer[9000];                                            /* Storage for send message at intermediate */

/* Message Attributes */
long msg_id;                  /* Msg id: 1, 2,....;incremented by the snding thread in generator  */
long dest_IP;                                                          /* Destination(receiver) IP address */
long originator_IP;                                                            /* Sender(source) IP adress */
long msg_num;
               /* Total number of messages */
long remain_msg;                                                       /* Remaining number of messages */
int hop_cnt;
                                          /* Number of hops */

long destination_IP;                                          /* Keeps the dest IP entered from command line */
long original_dest_IP;         /* dest IP, inserted into the send msg by the destination */
long msg_sndr_IP;                                                                 /* remote host(sender) IP */

/* Defines message type */
struct MsgCnt
{
          long Received;
          long Sent;
};

/* Defines received messages */
struct RcvdMsg
{
          struct MsgCnt Requests;
          struct MsgCnt Replies;
};

struct RcvdMsg OrgMsg;
                    /* Message at orginator  */
struct RcvdMsg DestMsg;
             /* Message at destination */
struct RcvdMsg InterMsg;                                                          /* Message at intermediate */

/* Defines lost messages */
struct LostMsg
{
          long Requests;
          long Replies;
};

struct LostMsg OrgLost;                                                          /* Lost messages at orginator  */
struct LostMsg DestLost;                                                        /* Lost messages at destination */
struct LostMsg InterLost;                                                         /* Lost messages at intermediate
*/

/* Defines duplicated messages */
struct DupMsg
{
          long Requests;
          long Replies;
};

struct DupMsg OrgDup;                                                           /*    Duplicated    messages    at
orginator  */
struct DupMsg DestDup;                                                          *    Duplicated    messages    at
destination */
struct DupMsg InterDup;                                        /* Duplicated messages at intermediate */

/* Array of requst msgs for intermediate nodes. */
long RelyLostMsgs[MAXMSGS]= {0};

/* Array of reply msgs for intermediate nodes. */
long RplyLostMsgs[MAXMSGS]= {0};

/* Array of structure for received messages at destination node and originating node */
struct ReceivedMessages
{
          long msg_id;
          int hop_cnt;
          int eflag;
          int indx;

};

struct ReceivedMessages RcvdMsgs[MAXMSGS]; /* Array of request msgs at destination node */
struct ReceivedMessages RplyMsgs[MAXMSGS];   /* Array of reply msgs at originating node */
struct ReceivedMessages AllRcvdSwindow[MAXMSGS];   /* Array of reply msgs at destination node */
struct ReceivedMessages AllRplySwindow[MAXMSGS];   /* Array of reply msgs at originating node */
long CombinedDest[MAXMSGS]={0};
long CombinedOrg[MAXMSGS]={0};


struct ReceivedMessages RcvdMsgsInter[MAXMSGS];/* Array of request msgs at intermediate */
struct ReceivedMessages RplyMsgsInter[MAXMSGS];  /* Array of reply msgs at intermediate */
```

```
/* Array of structure for sliding windows **/
struct SlidingWindow                                                    /* Structure of a  sliding window */
{                                                                               long msg_id;
                                                              /* Message Ids in the sliding window */
          int eflag;                                                    /* Shows the abcense or precense of array
*/
};

struct SlidingWindow RqstSwindow[MAXWIN];            /* Sliding IDs of recent requst messages */
struct SlidingWindow RplySwindow[MAXWIN];                   /* Sliding IDs of recent reply messages */

/* Counter for Request sliding window array elements */
int RqstSWcount = 0;

/* Counter for Reply sliding window array elements */
int RplySWcount = 0;

/* Shows if requst message is already in the sliding window array or not */
int msg_rely_flag = 0;

/* Shows if reply message is already in the sliding window array or not */
int msg_rply_flag = 0;

/* Counter for request/reply transmissions at intermediate */
int request_reply_inter = 0;

/* Message length */
int msg_length;

/* Variables needed to get local IP **/
          char                szErrorMessage[ 129 ];
          char                szLocalHostName[ 129 ];
          unsigned long       ulInetAddr;
          struct hostent      *pHostEnt;
          int                 nRC;
          long                                    Local_IP;

          FILE *fout;


// ****************************** modified part 2 out of 5 ********************************

struct received_message_ip_counter
{
          long received_message_ip;
          int received_message_counter;
};

struct received_message_ip_counter find_message_sender[MAX_NO_OF_NODES];
int flag_msg=0;

// ***************************** modified part 2 finished *********************************

/***************************************************************************************/
/*                                                        Relaying thread execution
                                                                                   */
/* It receives multicast datagrams from the network
                                         */
/***************************************************************************************/

unsigned _stdcall RelayingThread(LPVOID lpArg)
{

HANDLE hArg = (HANDLE) lpArg;                                          /* Convert parameter. Not used */

int nRet;
int WSAErr;
struct sockaddr_in rcvaddr, sndaddr, rmtaddr;
int addrlen, sndlen;
int cnt, i, j;

static struct ip_mreq stIp;                                           /* For setting multicast receiving */

//int RcvTimeOut = 50000;                                             /* Time out value for receiving, ms */
int RcvTimeOut = 100000;                                              /* Time out value for receiving, ms */

/* Initialize requst sliding window, create an empty array */
    for (j=0; j<MAXWIN; j++)
          {
                    RqstSwindow[j].eflag = 0;
                    RqstSwindow[j].msg_id = 0;
          }

/* Initialize reply sliding window, create an empty array */
    for (j=0; j<MAXWIN; j++)
          {
                    RplySwindow[j].eflag = 0;
                    RplySwindow[j].msg_id = 0;
          }

/* Initialize RcvdMsgs array elements */
for (j=0; j<MAXMSGS; j++)
          {
                    RcvdMsgs[j].eflag = 0;
                    RcvdMsgs[j].msg_id = 0;
                    RcvdMsgs[j].hop_cnt = 0;
          }

/* Initialize RplyMsgs array elements */
for (j=0; j<MAXMSGS; j++)
          {
```

```
                    RplyMsgs[j].eflag = 0;
                    RplyMsgs[j].msg_id = 0;
                    RplyMsgs[j].hop_cnt = 0;
          }

/* Initialize AllRcvdSwindow array elements */
for (j=0; j<MAXMSGS; j++)
          {
                    AllRcvdSwindow[j].eflag = 0;
                    AllRcvdSwindow[j].msg_id = 0;
          }

/* Initialize AllRplySwindow array elements */
for (j=0; j<MAXMSGS; j++)
          {
                    AllRplySwindow[j].eflag = 0;
                    AllRplySwindow[j].msg_id = 0;
          }

/* Initialize RcvdMsgsInter array elements */
for (j=0; j<MAXMSGS; j++)
          {
                    RcvdMsgsInter[j].eflag = 0;
                    RcvdMsgsInter[j].msg_id = 0;
          }

/* Initialize RplyMsgsInter array elements */
for (j=0; j<MAXMSGS; j++)
          {
                    RplyMsgsInter[j].eflag = 0;
                    RplyMsgsInter[j].msg_id = 0;
          }

/* Initialize InterMsg struct elements */
                    InterMsg.Requests.Received = 0;
                    InterMsg.Requests.Sent = 0;
                    InterMsg.Replies.Sent = 0;
                    InterMsg.Replies.Received = 0;

/* Initialize OrgMsg struct elements */
                    OrgMsg.Requests.Received = 0;
                    OrgMsg.Requests.Sent = 0;
                    OrgMsg.Replies.Sent = 0;
                    OrgMsg.Replies.Received = 0;

/* Initialize DestMsg struct elements */
                    DestMsg.Requests.Received = 0;
                    DestMsg.Requests.Sent = 0;
                    DestMsg.Replies.Sent = 0;
                    DestMsg.Replies.Received = 0;

/* Initialize InterLost struct elements */
                    InterLost.Requests = 0;
                    InterLost.Replies = 0;

/* Initialize OrgLost struct elements */
                    OrgLost.Requests = 0;
                    OrgLost.Replies = 0;

/* Initialize DestLost struct elements */
                    DestLost.Requests = 0;
                    DestLost.Replies = 0;

/* Initialize InterDup struct elements */
                    InterDup.Requests = 0;
                    InterDup.Replies = 0;

/* Initialize OrgDup struct elements */
                    OrgDup.Requests = 0;
                    OrgDup.Replies = 0;

/* Initialize DestDup struct elements */
                    DestDup.Requests = 0;
                    DestDup.Replies = 0;

// ****************************** modified part 3 out of 5 ********************************

for(i=0; i<MAX_NO_OF_NODES; i++)
{
          find_message_sender[i].received_message_ip=0;
          find_message_sender[i].received_message_counter=0;
}

// ***************************** modified part 3 finished *********************************

/************* Create a receive socket and check it *************************************/
hSockRcv = socket(PF_INET,SOCK_DGRAM,0);

if (hSockRcv == INVALID_SOCKET)
{
          WSAErr =WSAGetLastError();
          printf("SNDRCV1, rcv: WSAErr= %d\n",WSAErr); exit(1);
}
          printf("ADHOC, rcv: Socket %d for receiving was created\n",hSockRcv);


/* Now initialize my own address  */
rcvaddr.sin_family = PF_INET;
rcvaddr.sin_addr.s_addr = htonl(INADDR_ANY);                                               /* OS decides */
rcvaddr.sin_port = htons(nMyPort);
addrlen = sizeof(rcvaddr);

/************** Create a send socket and check it **************************************/
```

```
hSockFrwrd = socket(PF_INET, SOCK_DGRAM, 0);

if (hSockFrwrd == INVALID_SOCKET)
{
          WSAErr =WSAGetLastError();
          printf("SNDRCV1, rcv: WSAErr= %d\n",WSAErr); exit(1);
}
printf("ADHOC, rcv: Socket %d for forwarding was created\n",hSockFrwrd);

/* Now initialize sender socket address  */
          sndaddr.sin_family = PF_INET;
          sndaddr.sin_addr.s_addr = inet_addr(DESTINATION_MCAST);
          sndaddr.sin_port = htons(nDestPort);
          sndlen = sizeof(sndaddr);

/********************** Binding to my own IP address *********************************/
nRet = bind(hSockRcv, (struct sockaddr FAR *)&rcvaddr, sizeof(rcvaddr));
if (nRet == SOCKET_ERROR)
{
          perror ("ADHOC, rcv: bind(): err");
          WSACleanup(); exit(1);
}

/* Preparing to get datagrams multicast to IP = DESTINATION_MCAST */
stIp.imr_multiaddr.s_addr = inet_addr(DESTINATION_MCAST);
stIp.imr_interface.s_addr = htonl(INADDR_ANY);                                                      /*         Any
interface */

/* Set a multicast receiving option for itself */
nRet = setsockopt(hSockRcv, IPPROTO_IP, nOptName,
           (char * FAR)&stIp, sizeof(struct ip_mreq));

if (nRet == SOCKET_ERROR)
{
          perror("ADHOC, rcv: setsockopt():err");
          WSACleanup();exit(1);
}

printf ("ADHOC, rcv: Multicast socket option is OK\n");

/* Set a time out option for  receiving */
nRet = setsockopt(hSockRcv, SOL_SOCKET, nRecvTimeout,
           (char * FAR)&RcvTimeOut, sizeof(RcvTimeOut));

if (nRet == SOCKET_ERROR)
{
          perror("ADHOC, rcv: setsockopt():err");
          WSACleanup();exit(1);
}

printf ("ADHOC, rcv: Time out socket option is OK\n");


/********************************* Get Local IP **************************************/

          nRC = gethostname( szLocalHostName, sizeof( szLocalHostName ) );
          if ( nRC == -1 )
                    {
                    perror( szErrorMessage );    exit( EXIT_FAILURE );
                    }

          if (( ulInetAddr = inet_addr( szLocalHostName ) ) == ((unsigned long)-1L) )
          {
                    if (( pHostEnt = gethostbyname( szLocalHostName ) ) == NULL )
                              {
                              perror( szErrorMessage );
                              exit( EXIT_FAILURE );
                              }
          memcpy((char *)&rcvaddr.sin_addr, (char *)pHostEnt->h_addr,pHostEnt->h_length);
                    }
          else
                    {
                    memcpy( (char *)&rcvaddr.sin_addr, (char *)&ulInetAddr, sizeof(
                    ulInetAddr ));
                    }

          printf(" Local Host IP Address (Dot)  = [%s]\n", inet_ntoa( rcvaddr.sin_addr ));

/* Converts Internet Protocol dotted address into a proper address */
          Local_IP = inet_addr(inet_ntoa(rcvaddr.sin_addr));
//        printf("\tLocal Host IP address rcvaddr.sin_addr = %ld\n\n", rcvaddr.sin_addr);
          printf("\tLocal Host IP address (ulong) = %ld\n\n", Local_IP );
          fprintf(fout, "Local Host IP address (ulong) = %ld\n\n", Local_IP );


/*************************************************************************************/

/* Receiving multicast messages */
          while(1)
                                        /* Endless cycle */
          {
                    printf("\n****waiting in receiving loop.....\n");
                    cnt = recvfrom(hSockRcv, rcvbuffer, sizeof(rcvbuffer), 0,
                              (struct sockaddr *)&rmtaddr, &addrlen);

                    if (cnt < 0)                            /* A timeout has occured, nothing received! */
                    {
                              printf("Time out elapsed, Nothing to receive. Terminate the thread.\n\n");
                              break;
                    }

/* sender(remote) IP (can be originator or any other host before destination) */
                    msg_sndr_IP = inet_addr(inet_ntoa(rmtaddr.sin_addr));
```

```
//                  printf("A message received from: %ld\n", msg_ndr_IP);
//                  fprintf(fout, "A message received from: %ld\n", msg_sndr_IP);

/* Extract received info from the received messages and save */
//                  sscanf( rcvbuffer, "%ld %ld %ld %ld %ld %ld %ld\n", &originator_IP, &dest_IP,
//                            &msg_id, &msg_num, &remain_msg, &total_attempt, &original_dest_IP);

                    sscanf( rcvbuffer, "%ld %ld %ld %ld %ld %d %ld\n",
                              &originator_IP,    &dest_IP,    &msg_id,    &msg_num,    &remain_msg,    &hop_cnt,
&original_dest_IP);

//                  fprintf(fout, "Received buffer: %s\n", rcvbuffer);
                    printf("Received buffer: %s\n", rcvbuffer);


/****************************** Compare sender IP with local IP  ***********************/
/*                                                      Any node receives its own message
                                                    */
/*************************************************************************************/

                    if (msg_sndr_IP == Local_IP)
                    {
/* Node receives its own message, discard it */
//                            printf("\nNode received its own message from itself, discard it.\n\n");
//                            fprintf(fout, "\nNode received its own message from itself, discard it.\n\n");
                              for (i=0; i<sizeof(rcvbuffer); i++)
                                        rcvbuffer[i] = ' ';
                              continue;
                    }

/********************* Destination rcves a reply msge from neighbour nodes ************/
/************Increment duplicated reply mesg count at destination node ****************/
/*************************************************************************************/
                    if        ((original_dest_IP == Local_IP)&&(dest_IP == originator_IP))
                    {
//                            printf("\nDestination  received reply back message, count the message.\n");
//                            fprintf(fout, "\nDestination  received reply back message, count the message.\n");
/* Increment counter of duplicated request messages at the destination */
                              DestDup.Replies++;

/* Clear the receive buffer */
                              for (i=0; i<sizeof(rcvbuffer); i++)
                                        rcvbuffer[i] = ' ';
                              continue;
                    }

/************************ Compare originator IP with local IP  ************************/
/* Originator node receives a message, it can be a back msg from any node            */
/*                                  or  a reply message from the destination
                                */
/*************************************************************************************/

                    if (originator_IP == Local_IP)
                    {
                              if ((dest_IP == originator_IP) && (original_dest_IP == destination_IP))
                              {
/* Originator rcves a reply msge from dest */
//                                      fprintf(fout, "Originator receives a reply message \n");
//                                      printf("Originator receives a reply message \n");
                                        int org_flag = 0;                       /* message is received first time
*/

/* Check if message is already received, it is already in the reply msg array */
                                        for(i=0; i<MAXMSGS; i++)
                                                  if (RplyMsgs[i].msg_id == msg_id)
                                                  {
                                                            org_flag = 1;       /* duplicated message */
                                                            OrgDup.Replies++; /*  Incrmt  counter  of  duplctd  rply
msgs */
         //                                                 printf("Total  Number  of  duplicated  reply  msgs  at
origntr: %d\n",
         //                                                           OrgDup.Replies);
//                                                          fprintf(fout,  "Total  Number  of  dplctd  reply  msgs  at
orig:%d\n",
//                                                                    OrgDup.Replies);
                                                  }

                                        if (org_flag == 0)           /* Save received mssge in to the reply array
*/
                                        {
                                                  hop_cnt_rply = hop_cnt;/* Save received message hop cnt */
                                                  hop_cnt_rply++;     /* Increment hop count of reply messages */

                                                  RplyMsgs[msg_id-1].msg_id = msg_id;     /* save the message */
//                                                RplyMsgs[msg_id-1].hop_cnt = hop_cnt_rply;    /* Save hop count
*/
                                                  RplyMsgs[msg_id-1].eflag = 1;               /* Set the flag */
                                                  RplyMsgs[msg_id-1].indx = msg_id-1;        /* Save the index */

/* Fix receive time of the message and message id into the RttArray */
                                                  OrgRtt[msg_id-1].rcvtime = GetTickCount();
                                                  OrgRtt[msg_id-1].msg_id = msg_id;

/* Increment counter of reply messages */
                                                  OrgMsg.Replies.Received++;

/* Calculate sum of the hop count */
//                                                sum_hop_cnt_org = sum_hop_cnt_org + RplyMsgs[msg_id-1].hop_cnt;
                                                  sum_hop_cnt_org = sum_hop_cnt_org + hop_cnt_rply;


/* Calculate round trip time of the reply message */
```

```
                                                  OrgRtt[msg_id-1].rtttime = OrgRtt[msg_id-1].rcvtime -
                                                            OrgRtt[msg_id-1].sndtime;

/* Calculate sum of round trip time */
                                                  sum_rtt_org = sum_rtt_org + OrgRtt[msg_id-1].rtttime;

//                                                printf("Round trip time=%ld of message=%ld\n",
//                                                          OrgRtt[msg_id-1].rtttime, msg_id);

//                                                printf("Total Number of reply messages at originator: %d\n",
//                                                          OrgMsg.Replies.Received);

//                                                fprintf(fout,"Round trip time=%ld of msg =%ld and sum of rtt =
%ld\n",
//                                                          OrgRtt[msg_id-1].rtttime, msg_id, sum_rtt_org );

//                                                fprintf(fout,  "Total  Number  of  reply  messages  at  originator:
%d\n",
//                                                          OrgMsg.Replies.Received);

/* Save the message in all reply sliding window array */
                                                  i=0;
                                                  int swoflag = 0;
                                                  while((i<MAXMSGS) && (swoflag == 0))
                                                  {
                                                            if(AllRplySwindow[i].eflag == 0)
                                                            {
                                                                      AllRplySwindow[i].msg_id = msg_id;
                                                                      AllRplySwindow[i].eflag = 1;
                                                                      swoflag = 1;
                                                            }
                                                            i++;
                                                  }

                                        }
                                            /* end if (org_flag == 0) */

                                        org_flag = 0;
                              } /* end if ((dest_IP == originator_IP) && (original_dest_IP == destination_IP)) */


/* Originator received request back message */
                              if ((dest_IP == destination_IP)&&(msg_sndr_IP != originator_IP))
                              {
//                                        printf("\nOriginator   received   request   back   message,   count   the
message.\n");
//                                        fprintf(fout,"\nOriginator  received  request  back  message,  count  the
message.\n");

/* Increment counter of duplicated request messages at the originator */
                                        OrgDup.Requests++;
                              }

/* Clear the receive buffer */
                              for (i=0; i<sizeof(rcvbuffer); i++)
                                        rcvbuffer[i] = ' ';
                              continue;
                    } /* if((dest_IP == originator_IP) && (originator_IP == Local_IP))*/


/****************************** Compare received msg dest.IP with local IP  ************/
/*****************Destination node receives a request msg from the neighbors **********/
/*********************************************************************************/
                    if ((dest_IP == Local_IP)&&(Local_IP != originator_IP))/* I am the destintion */
                    {

// *****************************   modified part 4 out of 5   *************************
//                  fprintf(fout, "A message received from: %ld\n", msg_sndr_IP);
//                  fprintf(fout, "Received buffer: %s\n", rcvbuffer);


                              int flag_msg = 0; /* a message not receved from that ip before */
                              for(i=0; i<MAX_NO_OF_NODES; i++)
                              {
                                        if(find_message_sender[i].received_message_ip == msg_sndr_IP)
                                        {
                                                  find_message_sender[i].received_message_counter++;
                                                  flag_msg = 1;
                                        }
                              }

                              i=0;
                              if(flag_msg==0)
                              {
                                        while((flag_msg==0) && (i < MAX_NO_OF_NODES))
                                        {
                                                  if(find_message_sender[i].received_message_ip == 0)
                                                  {
                                                            find_message_sender[i].received_message_ip           =
msg_sndr_IP;
                                                            find_message_sender[i].received_message_counter++;
                                                            flag_msg = 1;
                                                  }
                                                  i++;
                                        }
                              }

                              flag_msg = 0;

// ****************************   modified part 4 finished   ***************************


//                            printf("Destination received a request message \n");
```

```
                              int dest_flag = 0;                      /* message is received first time */

/* Check if message is already received, it is already in the received msg array */
                              for(i=0; i<MAXMSGS; i++)
                                        if (RcvdMsgs[i].msg_id == msg_id)
                                        {
                                                  dest_flag = 1;
                  /* duplicated message */
                                                  DestDup.Requests++;/* Increment duplicated received messages  */
//                                                printf("Total  Number  of  duplicated  message  at  destination:
%d\n",
//                                                          DestDup.Requests);
//                                                fprintf(fout,"Total  Number  of  duplicated  msg  at  destination:
%d\n",
//                                                          DestDup.Requests);
                                        }

                              if (dest_flag == 0)              /* Save received message in to the array */
                              {
/* Fix receive time of the request message and message id into the RttArray at destn. */
//                                      RttArray[msg_id-1].rcvtimedest = GetTickCount();

                                        hop_cnt_rqst = hop_cnt;
                                        hop_cnt_rqst++;               /* Increment hop count of request */


                                        RcvdMsgs[msg_id-1].msg_id = msg_id;               /* Save the message */
//                                      RcvdMsgs[msg_id-1].hop_cnt = hop_cnt_rqst;         /* Save the hop cnt */
                                        RcvdMsgs[msg_id-1].eflag = 1;                      /* Set the message flag
*/
                                        RcvdMsgs[msg_id-1].indx = msg_id-1;                  /* Save the index */

/* Increment counter of received msgs  */
                                        DestMsg.Requests.Received++;
//                                      printf("Total number of request messages at destination %d\n",
//                                                DestMsg.Requests.Received);
//                                      fprintf(fout, "Total Number of request messages at destination %d\n",
//                                                DestMsg.Requests.Received);

/* Calculate sum of the hop count */
//                                                sum_hop_cnt_dest   =   sum_hop_cnt_dest   +   RcvdMsgs[msg_id-
1].hop_cnt;

/* Calculate sum of the hop count */
                                                  sum_hop_cnt_dest = sum_hop_cnt_dest + hop_cnt_rqst;


/* Save the message in all reply sliding window array */
                                                  i=0;
                                                  int swdflag = 0;
                                                  while((i<MAXMSGS) && (swdflag == 0))
                                                  {
                                                            if(AllRcvdSwindow[i].eflag == 0)
                                                            {
                                                                      AllRcvdSwindow[i].msg_id = msg_id;
                                                                      AllRcvdSwindow[i].eflag = 1;
                                                                      swdflag = 1;
                                                            }
                                                            i++;
                                                  }

/***************** Send a reply message to the originator ****************************/

//                                      hop_cnt = 0;/* reset reply hop counter for new message */
                                        int reply_hop_cnt = 0;/* reset reply hop counter for new rply message */

                                        sprintf(destbuffer, "%ld %ld %ld %ld %ld %d %ld\n", originator_IP,
                                                  originator_IP,   msg_id,   msg_num,   remain_msg,   reply_hop_cnt,
dest_IP);

                                        cnt = sendto(hSockFrwrd, destbuffer, cnt, 0,
                                                  (struct sockaddr *) &sndaddr, sndlen);

                                        if (cnt < 0)
                                        {
                                                  perror ("SNDRCV1,snd: sendto() err");
                                                  WSACleanup();
                                                  exit (1);
                                        }
//                                      printf("A msg %s  was sent from the dest. to orig. \n\n",
//                                                          destbuffer);
//                                      fprintf(fout, "A msg %s  was sent from the dest. to orig. \n\n",
//                                                          destbuffer);

                                        DestMsg.Replies.Sent++; /* Increment counter of sent reply mesgs  */
                              }
                        /* end if (dest_flag == 0) */

                              dest_flag = 0;


/* Clear the destination buffer */
                              for (i=0; i<sizeof(destbuffer); i++)
                                        destbuffer[i] = ' ';
                    }                                             /* end of if (dest_IP == Local_IP)  */


/************************************************************************************/
/*                                      Intermediate node received a message
                                */
/* Received msg can ve a reply msg from destination or request msg to the dest.    */
/************************************************************************************/
```

```
                    else
                    {
//                            printf("Intermediate node received a message \n");
//                            fprintf(fout, "Intermediate node receives a message \n");
                              int interm_flag = 0;     /* Used to separate reply and request message */

                              if (dest_IP == originator_IP)
                                        interm_flag = 1;                        /* This is a reply message */

                              if(interm_flag == 1)
                              {
//                                      printf("A reply msg received\n");
//                                      fprintf(fout, "A reply msg received\n");

/* Check if this message id is in the reply sliding array */
                                        msg_rply_flag = 1;                    /* Message is not in the array */
                                        for(i=0; i<MAXWIN; i++)
                                                  if(RplySwindow[i].msg_id == msg_id)
                                                  {
                                                            msg_rply_flag = 0;                /* Back msg, do not
save */
                                                            InterDup.Replies++;  /*  Increment  back  reply  message
count */
//                                                          printf("Total  number  of  received  back  reply  msgs
%d\n",
//                                                                    InterDup.Replies);
//                                                          fprintf(fout, "Total number of back reply msgs %d\n",
//                                                                    InterDup.Replies);
                                                  }

                              } /* end of reply message */

                              else /* this is a request message */
                              {
//                                      printf("A request msg received\n");
//                                      fprintf(fout,"A request msg received\n");
/* Check if this message id is in the request sliding array */
                                        msg_rely_flag = 1;                  /* Message is not in the array */
                                        for(i=0; i<MAXWIN; i++)
                                                  if(RqstSwindow[i].msg_id == msg_id)
                                                  {
                                                            msg_rely_flag = 0;                /* Back msg, do not
save */
                                                            InterDup.Requests++;/*   Increment   back   request   msg
conter */
//                                                          printf("Total  number  of  received  back  request  msgs
%d\n",
//                                                                    InterDup.Requests);
//                                                          fprintf(fout,  "Total  number  of  back  request  msgs
%d\n",
//                                                                    InterDup.Requests);
                                                  }
                              } /* End of request message */

                              interm_flag = 0;

                    }/* End of intermediate node receives a message */


/**************************** Request message computations starts ******************/
                    if (msg_rely_flag == 1)             /* Message is not in the array save it */
                    {
                              InterMsg.Requests.Received++;  /* Increment counter of requst messages */
//                            printf("Total number of received request msgs %d\n",
//                                      InterMsg.Requests.Received);
//                            fprintf(fout, "Total number of received request msgs %d\n",
//                                      InterMsg.Requests.Received);

/* Save the message in to the array of request messages for intermediate node */
                              RelyLostMsgs[msg_id-1]= msg_id;

/* Save this message id into the sliding Ids of the recent request messages */
                              i = 0;
                              int weflag = 0;
                              while ((i<MAXWIN) && (weflag == 0))
                              {
                                        if (RqstSwindow[i].eflag == 0)
                                        {
                                                  RqstSwindow[i].msg_id = msg_id;
                                                  RqstSWcount++;        /*  increment  #  of  elements  in  sliding
window */
                                                  RqstSwindow[i].eflag = 1;
                                                  weflag = 1;
                                        }
                              i++;
                              }

/* Print the array element */
/*                            for (k=0; k<MAXWIN; k++)
                              {
//                                      printf("Current msg nmbrs bfr sliding= %ld array entry status = %d \n",
//                                                RqstSwindow[k].msg_id, RqstSwindow[k].eflag);
                                        fprintf(fout,  "Current  request  msg  nmbrs  bfr  sliding=%ld  array  entry
status=%d\n",
                                                  RqstSwindow[k].msg_id, RqstSwindow[k].eflag);
                              }
*/
/* Reorganization of Sliding window(when there is no place in the window) */
                              if(RqstSWcount == MAXWIN)
                              {
                                        for (i=1; i<MAXWIN; i++)
                                                  RqstSwindow[i-1].msg_id = RqstSwindow[i].msg_id;
                                        RqstSwindow[MAXWIN - 1].eflag = 0;
                                        RqstSwindow[MAXWIN - 1].msg_id = 0;
                                        RqstSWcount--;     /* Decrement number of elements in sliding window */
                              }

/* Print the array element */
/*                            for (k=0; k<MAXWIN; k++)
                              {
                                        printf("Current msg numbers after sliding= %ld array entry status = %d
\n",
                                                  RqstSwindow[k].msg_id, RqstSwindow[k].eflag);
                              }
*/

/* Clear sndbuffer */
                              for(i=0; i<sizeof(sndbuffer); i++)
                                        sndbuffer[i] = ' ';

/* Forward the received message in multicast mode to the network */
                              for(i=0; i<sizeof(sndbuffer); i++)
                                        sndbuffer[i] = rcvbuffer[i];

//                            printf("Received buffer: %s\n", rcvbuffer);
//                            fprintf(fout, "\nSend request message from intermediate. node: %s\n", sndbuffer);

                              hop_cnt_rqst = hop_cnt;               /* Save received message hop cnt */
                              hop_cnt_rqst++;                         /* Increment hop count of requests */

/* Insert the new value of hop count into the message (sndbuffer) */
                              sprintf(sndbuffer, "%ld %ld %ld %ld %ld %d\n",
                              originator_IP, dest_IP, msg_id, msg_num, remain_msg, hop_cnt_rqst);

                              cnt = sendto(hSockFrwrd, sndbuffer, cnt, 0,
                                        (struct sockaddr *) &sndaddr, sndlen);

                         if (cnt < 0)
                               {
                                         perror ("SNDRCV1,snd: sendto() err");
                                         WSACleanup();
                                         exit (1);
                               }

/* Save the message into the request array of intermediate node */
                              RcvdMsgsInter[msg_id-1].msg_id = msg_id;

/* Fix send time of message */
                              InterRtt[msg_id-1].sndtime = GetTickCount();
//                  fprintf(fout, "Send time =  %ld of message= %ld\n", InterRtt[msg_id-1].sndtime, msg_id);


//                            printf("A request message %s  was sent from intermediate node\n", sndbuffer);
//                            fprintf(fout, "A request message %s  was sent from intermediate node\n", sndbuffer);
//                            fprintf(fout, "request hop cnt %d \n", hop_cnt_rqst);

                              InterMsg.Requests.Sent++;            /* Increment counter of request(sent) messages
*/

/* Reset msg_rqst_flag */
                              msg_rely_flag = 0;


                    }/* end of msg_rqst_flag == 1  */


/*********************** request message computations ends ********************************/

/******************** Reply message computations starts    ********************************/
                    if (msg_rply_flag == 1)                  /* Message is not in the array, save it */
                    {
                              InterMsg.Replies.Received++; /* Increment counter of reply(received) messages */
//                            printf("Total number of received reply msgs %d\n",
//                                      InterMsg.Replies.Received);
//                            fprintf(fout, "Total number of received reply msgs %d\n",
//                                      InterMsg.Replies.Received);

/* Save the message in to the array of reply messages for intermediate node */
                              RplyLostMsgs[msg_id-1] = msg_id;

/* Save the message into the reply array of intermediate node */
                              RplyMsgsInter[msg_id-1].msg_id = msg_id;

                              if((RplyMsgsInter[msg_id-1].msg_id == RcvdMsgsInter[msg_id-1].msg_id) &&
                                        (RplyMsgsInter[msg_id-1].msg_id!= 0))
                              {
/* A reply is received already sent request message calculate round trip time of the message */
                                        request_reply_inter++;  /*  Incr.  request/reply  transmission  counter  at
inter  */

/* Fix receive time of the message and message id into the InterRtt */
                                        InterRtt[msg_id-1].rcvtime = GetTickCount();
                                        InterRtt[msg_id-1].msg_id = msg_id;

//                                      fprintf(fout,  "Receive  time =   %ld  of  message= %ld\n",  InterRtt[msg_id-
1].rcvtime, msg_id);

/* Calculate round trip time */
                                        InterRtt[msg_id-1].rtttime = InterRtt[msg_id-1].rcvtime -
                                                  InterRtt[msg_id-1].sndtime;

/* Calculate sum of round trip time */
                                        sum_rtt_inter = sum_rtt_inter + InterRtt[msg_id-1].rtttime;
```

```
//                                      printf("Round trip time=%ld of message=%ld at intermediate\n",
//                                              InterRtt[msg_id-1].rtttime, msg_id);

                              }
/* Save this message id into the sliding Ids of the recent request messages */
                              i = 0;
                              int weflag = 0;
                              while ((i<MAXWIN) && (weflag == 0))
                              {
                                      if (RplySwindow[i].eflag == 0)
                                      {
                                              RplySwindow[i].msg_id = msg_id;
                                              RplySWcount++;         /*   increment   number   of   elements   in
sliding window */
                                              RplySwindow[i].eflag = 1;
                                              weflag = 1;
                                      }
                              i++;
                              }

/* Print the array element */
/*                            for (k=0; k<MAXWIN; k++)
                              {
//                    printf("Current msg numbers before sliding=%ld array entry status = %d \n",
//                            RplySwindow[k].msg_id, RplySwindow[k].eflag);
                      fprintf(fout, "Current reply msg numbers before sliding=%ldarray entry status=%d \n",
                              RplySwindow[k].msg_id, RplySwindow[k].eflag);
                              }
*/
/* Reorganization of Sliding window(when there is no place in the window) */
                              if(RplySWcount == MAXWIN)
                              {
                                      for (i=1; i<MAXWIN; i++)
                                              RplySwindow[i-1].msg_id = RplySwindow[i].msg_id;

                                      RplySwindow[MAXWIN - 1].eflag = 0;
                                      RplySwindow[MAXWIN - 1].msg_id = 0;
                                      RplySWcount--;          /* Decrement number of elements in sliding window
*/
                              }

/* Print the array element */
/*                            for (k=0; k<MAXWIN; k++)
                              {
                                      printf("Current msg numbers after sliding=%ld array entry status = %d \n",
                                              RplySwindow[k].msg_id, RplySwindow[k].eflag);
                              }
*/

/* Clear sndbuffer */
                              for(i=0; i<sizeof(sndbuffer); i++)
                                      sndbuffer[i] = ' ';

/* Forward the received message in multicast mode to the network */
                              for(i=0; i<sizeof(sndbuffer); i++)
                                      sndbuffer[i]=rcvbuffer[i];

                    //        printf("Received buffer: %s\n", rcvbuffer);
//                            fprintf(fout, "\nSend reply message from intermediate node: %s\n", sndbuffer);

                              hop_cnt_rply  = hop_cnt;
                              hop_cnt_rply++;
                                /* Increment number of hops */

/* Insert the new value of hop count in the message (sndbuffer) */
//                            sprintf(sndbuffer, "%ld %ld %ld %ld %ld %d %ld\n", originator_IP,
//                                            dest_IP,    msg_id,    msg_num,    remain_msg,    hop_cnt_rply,
original_dest_IP);

/* Insert the new value of hop count in the message (sndbuffer) */
                              sprintf(sndbuffer, "%ld %ld %ld %ld %ld %d %ld\n", originator_IP,
                                              originator_IP,   msg_id,   msg_num,   remain_msg,   hop_cnt_rply,
original_dest_IP);


                              cnt = sendto(hSockFrwrd, sndbuffer, cnt, 0,
                                      (struct sockaddr *) &sndaddr, sndlen);

                         if (cnt < 0)
                                   {perror ("SNDRCV1,snd: sendto() err"); WSACleanup(); exit (1);}

//                            fprintf(fout, "A reply message %s  was sent from intermediate node.\n", sndbuffer);
//                            printf("A reply message %s  was sent from intermediate node.\n", sndbuffer);
//                            fprintf(fout, "reply hop cnt %d \n", hop_cnt_rply);

                              InterMsg.Replies.Sent++;           /* Increment counter of reply (sent) messages */

/* Reset msg_rply_flag */
                              msg_rply_flag = 0;


                    }/* end of msg_rply_flag == 1 case  */

/******************************** Reply message computations ends ***********************/

          for (i=0; i<sizeof(rcvbuffer); i++)
                                      rcvbuffer[i] = ' ';

          for (i=0; i<sizeof(sndbuffer); i++)
                                      sndbuffer[i] = ' ';

          for (i=0; i<sizeof(destbuffer); i++)
                                      destbuffer[i] = ' ';
          } /* End of while */

/****************************** Find the number of lost messages at the nodes ***********/
/* Find the number of lost messages(reply) at the originator */

          int initiate_cnt1=0;/* Used to discard the lost at the beginning of the experiment */

          for(i=0; i<msg_num; i++)
          {
                    if(RplyMsgs[i].eflag != 0)
                              initiate_cnt1=1;/* Lost countings start after receiving first reply message*/

                    if(RplyMsgs[i].eflag == 0 && initiate_cnt1 == 1)
                              OrgLost.Replies++;

          }
          initiate_cnt1=0;
          printf("Total Number of reply lost msgs at originator node: %d\n", OrgLost.Replies);

/* Find number of lost messages(request) at the destination */
          for(i=0; i<msg_num; i++)
          {
                    if(RcvdMsgs[i].eflag != 0)
                    initiate_cnt1=1;

                    if(RcvdMsgs[i].eflag == 0 && initiate_cnt1==1)
                              DestLost.Requests++;
          }
          initiate_cnt1=0;
          printf("Total Number of request lost messages at dest: %d\n", DestLost.Requests);

/* Find number of lost messages(reply) at the intermediate */
          for(i=0; i<msg_num; i++)
          {
                    if(RplyLostMsgs[i] != 0)
                    initiate_cnt1=1;

                    if(RplyLostMsgs[i] == 0 && initiate_cnt1==1)
                              InterLost.Replies++;
          }
          initiate_cnt1=0;
          printf("Total Number of reply lost messages at interm: %d\n", InterLost.Replies);

/* Find number of lost messages(request) at the intermediate */
          for(i=0; i<msg_num; i++)
          {
                    if(RelyLostMsgs[i] != 0)
                    initiate_cnt1=1;

                    if(RelyLostMsgs[i] == 0 && initiate_cnt1 == 1)
                              InterLost.Requests++;
          }
          initiate_cnt1=0;
          printf("Total Number of request lost messages at interm: %d\n\n", InterLost.Requests);

          printf("********Relaying thread terminating....\n");


return 0;
}


/****************************************************************************************/
/*        The originating thread.                                                       */
/*                  It initializes all threads.                                         */
/****************************************************************************************/

int main(int argc,  char *argv[])

{
HANDLE hRcvThread;
HANDLE hArg;
                                                  /* Not used */
int i,j;
int WSAErr;
struct sockaddr_in addr;
int addrlen, cnt;
unsigned  uWorkThreadId;
DWORD dwResult;

static struct ip_mreq stIpReq;

int flag=0;
                                 /* For generator */

char msgbuffer[9000];
           /* Send message buffer  */

/* Message Attributes */
long msgid = 0;        /* Msg id: 1, 2,....;incremented by the snding thread in gnrator  */
long destIP;
                                /* Destination IP */
long originatorIP;                                                /* Sender(source) IP adress */
long msgnum;                                                       /* Total number of messages */
long remainmsg;                                                      /* Remaining messages */

int hopcnt;                                                            /* Number of hops */

int Dest_out_of_order_cnt=0;  /* Counter for our of order request messages */
int Org_out_of_order_cnt=0;   /* Counter for our of order replies messages */

char tmpbuf[128];
char tmpbuf1[128];


/* For the originator */
if (argc == 6)
                       /* For the originator */
{
          printf("Usage:%s [output_fname][num_of_msg] [msg_size][dest_IP][delay] \n", argv[0]);
          flag = 1;
}

/* For intermediate and destination nodes */
if (argc == 2)                                                            /*    For    intermediate    nodes    and
destination */
          printf("Usage:%s [output file name] \n", argv[0]);

/* Open a file for the output messages */

fout = fopen(argv[1], "w");
if (!fout)
          {
          printf("The file could not be open\n");
          exit(1);
          }

/* Initialize WinSock DLL */

WSAInitFailed= WSAStartup(WSA_VERSION, &stWSAData);
if(WSAInitFailed != 0)
{
          printf("SNDRCV1: InitFailed = %d\n",WSAInitFailed);
          exit(1);
}

/* Create manual reset event */

//hMsgOfTypeACK = CreateEvent(NULL, TRUE, FALSE, NULL);
/* Collect all statistics */

fprintf(fout, "    Statistics for mobile nodes \n");
fprintf(fout, " ---------------------------------- \n\n");

/* Display operating system-style date and time. */
          _strdate( tmpbuf );
    fprintf(fout, "Start date   = %s\n", tmpbuf );
          printf("Start date   = %s\n", tmpbuf );
    _strtime( tmpbuf1 );
    fprintf(fout, "Start time   = %s\n\n", tmpbuf1 );
          printf("Start time   = %s\n\n", tmpbuf1 );

/*********************************************************************************/
/*        Create child thread (RelayingThread) for receiving multicast datagrams */
/*********************************************************************************/

hRcvThread = (HANDLE) _beginthreadex(NULL, 0, RelayingThread,
             (void *) hArg, 0, &uWorkThreadId);
if(!hRcvThread)
     {
     printf("SNDRCV1: thread error creating\n");
     WSACleanup();
           exit (0xFFFFFFFF);

          }

if (flag == 1)                                                        /* originator is performing sending */
{  /* beginning of if (flag == 1) */

/* Now create a socket for sending and check it */

          hSockSnd = socket(PF_INET,SOCK_DGRAM,0);
          /* printf ("hSock = %d\n", hSock); */

          if (hSockSnd == INVALID_SOCKET)
          {
                    WSAErr =WSAGetLastError();
                    printf("SNDRCV1, snd: WSAErr = %d\n",WSAErr); exit(1);
          }
          printf("ADHOC, snd: Socket %d was created for sending\n",hSockSnd);

/* Initialize the address for sending */

          addr.sin_family = PF_INET;
          addr.sin_addr.s_addr = inet_addr(DESTINATION_MCAST);
          addr.sin_port = htons(nDestPort);
          addrlen = sizeof(addr);

/* Sleep a bit for another process to start */
          Sleep (5000);

/* Sending  messages, with one  second delay  between them */

     for (i=0; i<(atoi(argv[2])); i++)
     {
                    originatorIP = Local_IP;
                               /* Sender IP */
                    destIP = inet_addr(argv[4]);                                /* Converts IP to unsigned long
*/
                    destination_IP = destIP;                                  /* Global, used in relying thread
*/
                    msgnum = atoi(argv[2]);
                    msgid = msgid+1;                                /*   Increment   and   send   the   message
identifier */
                    remainmsg = msgnum-msgid;
                    hopcnt = 0;
                                              /* Number of hops */


//                  printf("\nSend msg from the orgntr:\norgntr IP=%ld\n dest.IP=%ld\n msgID=%ld\n",
//                            originatorIP, destIP, msgid);

/* inet_addr(inet_ntoa(addr.sin_addr)): unsigned  */
/* long inet_ntoa(addr.sin_addr):dotted from */


                    msg_length = atoi(argv[3]);                                  /* Fix size of send message */

                    sprintf(msgbuffer, "%ld %ld %ld %ld %ld %d\n",
                              originatorIP, destIP, msgid, msgnum, remainmsg, hopcnt);


                    cnt = sendto(hSockSnd, msgbuffer, msg_length, 0,
                              (struct sockaddr *) &addr, addrlen);
                    if (cnt < 0)
                    {
                              perror ("SNDRCV1,snd: sendto() err");
                              WSACleanup();
                              exit (1);
                    }

/* Fix send time of message */
                    OrgRtt[msgid-1].sndtime = GetTickCount();
//                  fprintf(fout, "Send time = %ld of message= %ld\n", OrgRtt[msgid-1].sndtime, msgid);

//        printf("ADHOC, snd: Message number %ld  was sent\n\n", msgid);
//                  printf("Send message from the originator: %s\n", msgbuffer);

            /*end while(attempt < 5) */

          Sleep (atoi(argv[5])); /* Sleep time before the next message */
//                  Sleep (1000); /* Sleep 1000  ms, 1 second */
     }                                                         /* end of for (i=0; i<(atoi(argv[2])); i++)
*/
}
                      /* end of if (flag == 1)  */

/* Wait until the relaying thread has exited */
          dwResult = WaitForSingleObject(hRcvThread,INFINITE);

/* Display operating system-style date and time. */
          _strdate( tmpbuf );
    fprintf(fout, "Stop date   = %s\n", tmpbuf );
          printf("Stop date   = %s\n", tmpbuf );
    _strtime( tmpbuf1 );
    fprintf(fout, "Stop time   = %s\n\n", tmpbuf1 );
          printf("Stop time   = %s\n\n", tmpbuf1 );


/* Calculate average round trip time */
          if (flag == 1)
          {
                    fprintf(fout, " Originator Node Results: \n");
                    fprintf(fout, " Parameters: \n");
                    fprintf(fout, "-------------\n\n");
                    //printf("Message size = %d\n", atoi(argv[3]));
                    fprintf(fout, "Number of messages = %d\n", atoi(argv[2]));
                    fprintf(fout, "Message size = %d\n", atoi(argv[3]));
                    fprintf(fout, "Intermediate time at the originator = %d ms\n\n", atoi(argv[5]));

/*********************************************************************/

                    fprintf(fout, "Total number of reply messages at originator: %d\n",
                              OrgMsg.Replies.Received);

                    fprintf(fout, "Total number of duplicated request msgs at originator: %d\n",
                              OrgDup.Requests);

                    fprintf(fout, "Total number of duplicated reply messages at originator:%d\n",
                              OrgDup.Replies);

                    fprintf(fout, "Total number of lost reply msgs at originator: %d\n",
                              OrgLost.Replies - DestLost.Requests);

                    fprintf(fout,   "Total   number   of   lost   msgs   at   originator(request+replies):   %d\n\n",
OrgLost.Replies);


                    /* Calculate average round trip time */
                    average_rtt_org = (double) sum_rtt_org/OrgMsg.Replies.Received;

//                  printf("Average round trip time=%lf for %ld messages\n",
//                            average_rtt_org, OrgMsg.Replies.Received);

//                  fprintf(fout, "Sum of Round trip time=%ld for %ld messages\n",
//                            sum_rtt_org, OrgMsg.Replies.Received);

                    fprintf(fout, "Average Round trip time = %.3lf milliseconds\n\n",
                              average_rtt_org);


                    fprintf(fout, "Delivery ratio of replies : %.3lf\n\n",
                              (double)      OrgMsg.Replies.Received/(OrgMsg.Replies.Received+OrgLost.Replies      -
DestLost.Requests));

                    fprintf(fout, "Delivery ratio (RTT): %.3lf\n\n",
                              (double) OrgMsg.Replies.Received/(OrgMsg.Replies.Received+OrgLost.Replies));
```

```
/* Calculate average number of hops for reply messages */

                    average_hop_cnt_org = (double)sum_hop_cnt_org/OrgMsg.Replies.Received;

                    fprintf(fout, "Sum of hop cnt=%ld for %ld messages\n\n",
                              sum_hop_cnt_org, OrgMsg.Replies.Received);

//                  printf("Average hop count =%.3lf for %ld reply messages\n",
//                            average_hop_cnt_org, OrgMsg.Replies.Received);

                    fprintf(fout, "Average hop count =%.3lf for %ld reply messages\n\n",
                              average_hop_cnt_org, OrgMsg.Replies.Received);

//                  printf("Duplicate Ratio (of received replies) =%.3lf \n",
//                            (double)OrgDup.Replies/OrgMsg.Replies.Received);

                    fprintf(fout, "Duplicate ratio (of received replies) =%.3lf \n\n",
                              (double)OrgDup.Replies/OrgMsg.Replies.Received);

/* Create combined array */
                    for(i=0,j=0; i<msgnum; i++)
                    {
                              if(RplyMsgs[i].msg_id!=0)
                              {
                                        CombinedOrg[i]=AllRplySwindow[j].msg_id;
                                        j++;
                              }

                    }

/*                  fprintf(fout, "\nCombined array content at originator:\n\n");
                    for(i=0; i<msgnum; i++)
                    {
                              fprintf(fout, "%5ld  ", CombinedOrg[i]);
                              if ((i+1) % 10 == 0)
                                        fprintf(fout,"\n");
                    }

*/

/* Find number of out of order receiving messages */

                    for(i=0; i<msgnum; i++)
                              if((CombinedOrg[i]!= 0) && (CombinedOrg[i] != i+1)) /* Do not count lost msgs */
                                        Org_out_of_order_cnt++;

                    fprintf(fout, "\nNumber of out of order reply msgs = %d\n", Org_out_of_order_cnt);


                    fprintf(fout, "Reply array content at originator:\n");
                    for(i=0; i<msgnum; i++)
                    {
                              fprintf(fout, "%5ld  ", RplyMsgs[i].msg_id );
                              if ((i+1) % 10 == 0)
                                        fprintf(fout,"\n");
                    }

                    fprintf(fout, "\nRound trip times for each messages at originator:\n");
                    for(i=0; i<msgnum; i++)
                    {
                              fprintf(fout, "%5ld ", OrgRtt[i].rtttime);
                              if ((i+1) % 10 == 0)
                                        fprintf(fout,"\n");
                    }
/*
                    fprintf(fout, "\nAll Reply Sliding window array content at originator:\n");
                    for(i=0; i<msgnum; i++)
                    {
                              fprintf(fout, "%5ld  ", AllRplySwindow[i].msg_id );
                              if ((i+1) % 10 == 0)
                                        fprintf(fout,"\n");
                    }
*/


          }

          else
          {
                    fprintf(fout, " \n\n\nDestination Node Results: \n\n");

                    fprintf(fout, "Total number of request messages at destination: %d\n",
                              DestMsg.Requests.Received);

                    fprintf(fout, "Total number of sent reply messages from destination: %d\n",
                              DestMsg.Replies.Sent);

                    fprintf(fout, "Total number of duplicated request messages at destination: %d\n",
                              DestDup.Requests);

                    fprintf(fout, "Total number of duplicated reply msgs at destination: %d\n",
                              DestDup.Replies);

                    fprintf(fout, "Total number of lost request msgs at dest: %d\n\n", DestLost.Requests);

                    fprintf(fout, "Delivery ratio (requests): %.3lf\n\n",
                              (double) DestMsg.Requests.Received/(DestMsg.Requests.Received+DestLost.Requests));

/* Calculate average number of hops for request messages */

                    average_hop_cnt_dest = (double)sum_hop_cnt_dest/DestMsg.Requests.Received;
```

```
//                  printf("Average hop count =%.3lf for %ld request messages\n",
//                          average_hop_cnt_dest, DestMsg.Requests.Received);

                    fprintf(fout, "Sum of hop cnt=%ld for %ld messages\n\n",
                            sum_hop_cnt_dest, DestMsg.Requests.Received);

                    fprintf(fout, "Average hop count =%.3lf \n\n", average_hop_cnt_dest);
//****************************** modified part 5 out of 5 **********************************************

                    for(i=0; i<MAX_NO_OF_NODES; i++)
                    {
                            if(find_message_sender[i].received_message_ip!=0)
                            {
                                    if(find_message_sender[i].received_message_ip==originator_IP)
                                    {
                                            fprintf(fout, "Node with IP: %d (originator) sent %d messages to
the destination\n",
                                            find_message_sender[i].received_message_ip,
find_message_sender[i].received_message_counter);
                                    }
                                    else
                                    {
                                            fprintf(fout,  "Node  with  IP:  %d  sent  %d  messages  to  the
destination\n",
                                            find_message_sender[i].received_message_ip,
find_message_sender[i].received_message_counter);
                                    }
                            }
                    }

//****************************** modified part 5 finished **********************************************

/* Create combined array */
                    for(i=0,j=0; i<msg_num; i++)
                    {
                            if(RcvdMsgs[i].msg_id!=0)
                            {
                                    CombinedDest[i]=AllRcvdSwindow[j].msg_id;
                                    j++;
                            }

                    }
/*
                    fprintf(fout, "\nCombined array content at destination:\n\n");
                    for(i=0; i<msg_num; i++)
                    {
                            fprintf(fout, "%5ld  ", CombinedDest[i]);
                            if ((i+1) % 10 == 0)
                                    fprintf(fout,"\n");
                    }

*/

/* Find number of out of order receiving messages */

                    for(i=0; i<msg_num; i++)
                            if((CombinedDest[i]!= 0) && (CombinedDest[i] != i+1)) /* Do not count lost msgs */
                                    Dest_out_of_order_cnt++;

                    fprintf(fout, "\nNumber of out of order request msgs = %d\n", Dest_out_of_order_cnt);


                    fprintf(fout, "\nRequest array content at destination:\n\n");
                    for(i=0; i<msg_num; i++)
                    {
                            fprintf(fout, "%5ld  ", RcvdMsgs[i].msg_id );
                            if ((i+1) % 10 == 0)
                                    fprintf(fout,"\n");
                    }

/*                  fprintf(fout, "\nAll Request Sliding window array content at destination:\n");
                    for(i=0; i<msg_num; i++)
                    {
                            fprintf(fout, "%5ld  ", AllRcvdSwindow[i].msg_id );
                            if ((i+1) % 10 == 0)
                                    fprintf(fout,"\n");
                    }
*/


                    fprintf(fout, " \n\nIntermediate Node Results: \n\n");

                    /* Calculate average round trip time at intermediate */
                    average_rtt_inter = (double) sum_rtt_inter/request_reply_inter;

//                  printf("Average    round    trip    time=%lf    for    %ld    messages\n",    average_rtt_inter,
request_reply_inter);

//                  fprintf(fout,   "Sum   of   Round   trip   time=%ld   for   %ld   messages\n",   sum_rtt_inter,
request_reply_inter);

                    fprintf(fout, "\nAverage Round Trip time at intermediate= %.3lf milliseconds\n\n",
                            average_rtt_inter);

                    fprintf(fout, "Total Number of request/reply transmissions at intermediate: %d\n",
                            request_reply_inter);

                    fprintf(fout, "Total number of request msgs at intermediate: %d\n",
                            InterMsg.Requests.Received);

                    fprintf(fout, "Total number of sent request msgs at intermediate: %d\n",
                            InterMsg.Requests.Sent);
                    fprintf(fout, "Total number of reply msgs at intermediate: %d\n",
                              InterMsg.Replies.Received);

                    fprintf(fout, "Total number of sent reply msgs at intermediate: %d\n",
                              InterMsg.Replies.Sent);

                    fprintf(fout, "Total number of duplicated reply msgs at intermediate: %d\n",
                              InterDup.Replies);

                    fprintf(fout, "Total number of duplicated request msgs at intermediate: %d\n",
                              InterDup.Requests);

                    fprintf(fout, "Total Number of lost request msgs at interm: %d\n", InterLost.Requests);

                    fprintf(fout, "Total Number of lost reply msgs at interm: %d\n", InterLost.Replies);

/*                  fprintf(fout, "\n\nRequest lost message array content at intermediate:\n");
                    for(i=0; i<msg_num; i++)
                    {
                              fprintf(fout, "%5ld  ", RelyLostMsgs[i]);
                              if ((i+1) % 10 == 0)
                                        fprintf(fout,"\n");
                    }

                    fprintf(fout, "\nReply lost message array content at intermediate:\n");
                    for(i=0; i<msg_num; i++)
                    {
                              fprintf(fout, "%5ld  ", RplyLostMsgs[i]);
                              if ((i+1) % 10 == 0)
                                        fprintf(fout,"\n");
                    }

*/
                    fprintf(fout, "\nRequest array content at intermediate:\n");
                    for(i=0; i<msg_num; i++)
                    {
                              fprintf(fout, "%5ld  ", RcvdMsgsInter[i].msg_id );
                              if ((i+1) % 10 == 0)
                                        fprintf(fout,"\n");
                    }

                    fprintf(fout, "\nReply array content at intermediate:\n");
                    for(i=0; i<msg_num; i++)
                    {
                              fprintf(fout, "%5ld  ", RplyMsgsInter[i].msg_id );
                              if ((i+1) % 10 == 0)
                                        fprintf(fout,"\n");
                    }

                    fprintf(fout, "\nRound trip times for each messages at intermediate\n");
                    for(i=0; i<msg_num; i++)
                    {
                              fprintf(fout, "%5ld ", InterRtt[i].rtttime);
                              if ((i+1) % 10 == 0)
                                        fprintf(fout,"\n");
                    }

          }

CloseHandle(hRcvThread);
fclose(fout);
WSACleanup();   /* Matching cleanup for startup */
return 0;
}
```

Appendix B: Raw Results of the Experiments

Results of experiments with 10 nodes (10 laptops) varying message size without mobility

Time: 10:00 **Date: 21/03/2010**

Number of sent requests = 2000 **Inter packet time (delay) = 100 ms**

Size of data = 100 bytes **Distance = mobility**

Performance Metric	**Data size (bytes)**	**Trials**		
		1st	**2nd**	**3rd**
Average Round Trip Time (ms)		15.55	16.98	16.2
Total number of reply messages at originator		1640	1424	1468
Total number of duplicated request messages at originator		7773	7716	7963
Total number of duplicated reply messages at originator		4718	4059	3997
Total number of lost reply messages at originator		360	576	532
Total number of lost messages at originator (request + replies)		360	576	532
Delivery ratio (replies)		0.82	0.71	0.73
Delivery ratio (RTT)		0.82	0.71	0.73
Average hop count		2.359	2.171	2.270
Duplicate ratio (replies)		2.877	2.850	2.723
Number of out of order reply msgs		15	16	20

Performance Metric	Data size (bytes	Trials		
		1st	2nd	3rd
Destination Results				
Total number of request messages at destination		1908	1931	1905
Total number of sent reply messages from destination		1908	1931	1905
Total number of duplicated request messages at destination		4406	3650	4207
Total number of lost request messages at destination		92	69	95
Delivery ratio		0.954	0.9655	0.9525
Average hop count		2.137	2.264	2.366
Number of received msgs from originator				
Number of out of order request messages		4	15	19
Intermediate 3 Results				
Average Round Trip (ms)		**3.812539**	**15.908096**	**15.491128**
Total number of request/reply transmissions at intermediate		**1595**	**1371**	**1409**
Total number of received request messages at intermediate		**1934**	**1930**	**1918**
Total number of received reply messages at intermediate		**1598**	**1375**	**1414**
Total number of duplicated reply messages at intermediate		**4491**	**3248**	**3619**
Total number of duplicated request messages at intermediate		**6515**	**6073**	**6637**
Total number of lost request messages at intermediate		**402**	**625**	**586**
Total number of lost reply messages at intermediate		**66**	**70**	**82**
Intermediate 4 Results				
Average Round Trip (ms)		14.111817	14.376006	14.024665
Total number of request/reply transmissions at intermediate		1574	1367	1419
Total number of received request messages at intermediate		1921	1921	1922
Total number of received reply messages at intermediate		1577	1372	1426
Total number of duplicated reply messages at intermediate		3639	3279	3757
Total number of duplicated request messages at intermediate		5338	4445	4907
Total number of lost request messages at intermediate		79	79	78
Total number of lost reply messages at intermediate		473	628	574

Intermediate 5 Results				
Average Round Trip (ms)		16,876894	11,185294	14,025460
Total number of request/reply transmissions at intermediate		1584	1360	1414
Total number of received request messages at intermediate		1933	1923	1916
Total number of received reply messages at intermediate		1587	1363	1417
Total number of duplicated reply messages at intermediate		3431	2580	3017
Total number of duplicated request messages at intermediate		6106	5530	6293
Total number of lost request messages at intermediate		67	77	84
Total number of lost reply messages at intermediate		413	637	583
Intermediate 6 Results				
Average Round Trip (ms)		16.155473	15.752511	14.161469
Total number of request/reply transmissions at intermediate		1608	1394	1443
Total number of received request messages at intermediate		1976	1977	1960
Total number of received reply messages at intermediate		1618	1406	1454
Total number of duplicated reply messages at intermediate		1618	2318	2926
Total number of duplicated request messages at intermediate		5592	4618	5723
Total number of lost request messages at intermediate		24	23	40
Total number of lost reply messages at intermediate		382	594	546
Intermediate 7 Results				
Average Round Trip (ms)		3.318874	8.386555	4.811573
Total number of request/reply transmissions at intermediate		1314	1309	1348
Total number of received request messages at intermediate		1763	1890	1927
Total number of received reply messages at intermediate		1522	1424	1464
Total number of duplicated reply messages at intermediate		3693	3081	3500
Total number of duplicated request messages at intermediate		2236	2788	2827
Total number of lost request messages at intermediate		74	110	73
Total number of lost reply messages at intermediate		315	576	536
Intermediate 9 Results				
Average Round Trip (ms)		7.537927	4.984074	5.048276

Total number of request/reply transmissions at intermediate		1582	1193	1160
Total number of received request messages at intermediate		1953	1738	1629
Total number of received reply messages at intermediate		1608	1288	1238
Total number of duplicated reply messages at intermediate		2691	1772	1883
Total number of duplicated request messages at intermediate		3145	1776	1827
Total number of lost request messages at intermediate		47	114	110
Total number of lost reply messages at intermediate		392	564	501
Intermediate 11 Results				
Average Round Trip (ms)		7.860299	9.932318	11.1138 39
Total number of request/reply transmissions at intermediate		1539	1182	1344
Total number of received request messages at intermediate		1950	1822	1935
Total number of received reply messages at intermediate		1643	1360	1462
Total number of duplicated reply messages at intermediate		3259	1733	2279
Total number of duplicated request messages at intermediate		2354	1441	2328
Total number of lost request messages at intermediate		50	178	65
Total number of lost reply messages at intermediate		357	640	538
Intermediate 12 Results				
Average Round Trip (ms)		14.055625	16.63815 8	16.9731 64
Total number of request/reply transmissions at intermediate		1600	1368	1416
Total number of received request messages at intermediate		1935	1933	1922
Total number of received reply messages at intermediate		1600	1370	1418
Total number of duplicated reply messages at intermediate		3673	1925	2328
Total number of duplicated request messages at intermediate		6441	4396	5295
Total number of lost request messages at intermediate		65	67	78
Total number of lost reply messages at intermediate		400	630	582

Results of experiments with 10 nodes (10 laptops) varying message size without mobility

Time: 10:00 **Date: 21/03/2010**

Number of sent requests = 2000 **Inter packet time (delay) = 100 ms**

Size of data = 400 bytes **Distance = mobility**

Originator Results

Performance Metric	**Data size (bytes)**	**Trials**		
		1st	**2nd**	**3rd**
Average Round Trip Time (ms)		28.98	33.74	33.74
Total number of reply messages at originator		1272	1265	1405
Total number of duplicated request messages at originator		6616	6695	7018
Total number of duplicated reply messages at originator		3531	2866	3510
Total number of lost reply messages at originator		725	734	595
Total number of lost messages at originator (request + replies)		725	734	595
Delivery ratio (replies)		0.636	0.632	0.702
Delivery ratio (RTT)		0.637	0.633	0.703
Average hop count		2.298	2.274	2.243
Duplicate ratio (replies)		2.776	2.266	2.498
Number of out of order reply msgs		6	8	6

Performance Metric	**Data size (bytes**	**Trials**		
		1st	**2nd**	**3rd**
Destination Results				
Total number of request messages at destination		1861	1880	1861
Total number of sent reply messages from destination		1861	1880	1861
Total number of duplicated request messages at destination		2412	2945	2998
Total number of lost request messages at destination		139	119	139
Delivery ratio		0.9305	0.94047	0.9305
Average hop count		2.171	2.346	2.339
Number of received msgs from originator				
Number of out of order request messages		6	14	6
Intermediate 3 Results				

Average Round Trip (ms)		**20.549035**	**28.657348**	**32.142857**
Total number of request/reply transmissions at intermediate		**1244**	**1252**	**1379**
Total number of received request messages at intermediate		**1941**	**1919**	**1914**
Total number of received reply messages at intermediate		**1250**	**1254**	**1389**
Total number of duplicated reply messages at intermediate		**2861**	**2508**	**2823**
Total number of duplicated request messages at intermediate		**4257**	**4903**	**4532**
Total number of lost request messages at intermediate		**59**	**81**	**86**
Total number of lost reply messages at intermediate		**747**	**732**	**611**
Intermediate 4 Results				
Average Round Trip (ms)		27.003265	22.595097	
Total number of request/reply transmissions at intermediate		1225	1183	
Total number of received request messages at intermediate		1923	1851	
Total number of received reply messages at intermediate		1236	1244	
Total number of duplicated reply messages at intermediate		2336	2151	
Total number of duplicated request messages at intermediate		3439	2554	
Total number of lost request messages at intermediate		77	149	
Total number of lost reply messages at intermediate		761	755	
Intermediate 5 Results				
Average Round Trip (ms)		32.526837	26.004085	29.911194
Total number of request/reply transmissions at intermediate		1211	1224	1340
Total number of received request messages at intermediate		1933	1931	1907
Total number of received reply messages at intermediate		1218	1231	1346
Total number of duplicated reply messages at intermediate		2230	1821	2115
Total number of duplicated request messages at intermediate		4147	4111	4387
Total number of lost request messages at intermediate		67	69	93
Total number of lost reply messages at intermediate		779	768	654
Intermediate 6 Results				
Average Round Trip (ms)		28.359098	28.016800	24.441531
Total number of request/reply transmissions at intermediate		1242	1250	1411

Total number of received request messages at intermediate		1979	1951	1950
Total number of received reply messages at intermediate		1250	1265	1416
Total number of duplicated reply messages at intermediate		2243	1921	3120
Total number of duplicated request messages at intermediate		4309	3813	3987
Total number of lost request messages at intermediate		21	49	50
Total number of lost reply messages at intermediate		747	734	584
Intermediate 7 Results				
Average Round Trip (ms)		10.635870	14.693817	14.156897
Total number of request/reply transmissions at intermediate		736	1019	1160
Total number of received request messages at intermediate		1455	1582	1659
Total number of received reply messages at intermediate		1287	1159	1331
Total number of duplicated reply messages at intermediate		1990	2395	2456
Total number of duplicated request messages at intermediate		795	1652	1870
Total number of lost request messages at intermediate		421	144	200
Total number of lost reply messages at intermediate		588	565	528
Intermediate 9 Results				
Average Round Trip (ms)		25.264151	17.234275	18.839354
Total number of request/reply transmissions at intermediate		1166	1097	1301
Total number of received request messages at intermediate		1920	1782	1826
Total number of received reply messages at intermediate		1207	1250	1399
Total number of duplicated reply messages at intermediate		1111	1563	1615
Total number of duplicated request messages at intermediate		1685	1242	1162
Total number of lost request messages at intermediate		80	165	174
Total number of lost reply messages at intermediate		790	698	601
Intermediate 11 Results				
Average Round Trip (ms)		14.300334	21.000000	16.786030
Total number of request/reply transmissions at intermediate		899	1165	1131
Total number of received request messages at intermediate		1600	1850	1719
Total number of received reply messages at intermediate		1309	1249	1337

Total number of duplicated reply messages at intermediate		1389	1346	1257
Total number of duplicated request messages at intermediate		1118	1546	1105
Total number of lost request messages at intermediate		400	147	281
Total number of lost reply messages at intermediate		688	750	663
Intermediate 12 Results				
Average Round Trip (ms)		23.601942	30.571543	36.238653
Total number of request/reply transmissions at intermediate		1236	1251	1366
Total number of received request messages at intermediate		1947	1930	1926
Total number of received reply messages at intermediate		1241	1257	1371
Total number of duplicated reply messages at intermediate		2105	1717	1755
Total number of duplicated request messages at intermediate		4209	3785	4077
Total number of lost request messages at intermediate		53	66	74
Total number of lost reply messages at intermediate		756	735	629

Results of experiments with 10 nodes (10 laptops) varying message size without mobility

Time: 10:00 **Date: 21/03/2010**

Number of sent requests = 2000 **Inter packet time (delay) = 100 ms**

Size of data = 800 bytes **Distance = mobility**

Originator Results

Performance Metric	**Data size (bytes)**	**Trials**		
		1st	**2nd**	**3rd**
Average Round Trip Time (ms)		60.82	52.42	58.12
Total number of reply messages at originator		1269	1424	1517
Total number of duplicated request messages at originator		6161	5695	6362
Total number of duplicated reply messages at originator		2835	3161	3532
Total number of lost reply		731	575	483

messages at originator				
Total number of lost messages at originator (request + replies)		731	575	483
Delivery ratio (replies)		0.634	0.712	0.758
Delivery ratio (RTT)		0.634	0.712	0.758
Average hop count		2.331	2.336	2.417
Duplicate ratio (replies)		2.234	2.22	2.328
Number of out of order reply msgs		30	17	9

Performance Metric	**Data size (bytes**	**Trials**		
		1st	**2nd**	**3rd**
Destination Results				
Total number of request messages at destination		1794	1834	1843
Total number of sent reply messages from destination		1794	1834	1843
Total number of duplicated request messages at destination		2205	2214	2553
Total number of lost request messages at destination		206	165	157
Delivery ratio		0.897	0.917459	0.9215
Average hop count		2.488	2.258	2.426
Number of received msgs from originator				
Number of out of order request messages		37	9	7
Intermediate 3 Results				
Average Round Trip (ms)		**43.898632**	**46.315018**	**41.313179**
Total number of request/reply transmissions at intermediate		**1243**	**1365**	**1472**
Total number of received request messages at intermediate		**1888**	**1885**	**1894**
Total number of received reply messages at intermediate		**1260**	**1385**	**1489**
Total number of duplicated reply messages at intermediate		**2536**	**2395**	**2827**
Total number of duplicated request messages at intermediate		**4112**	**3715**	**3680**
Total number of lost request messages at intermediate		**112**	**115**	**106**
Total number of lost reply messages at intermediate		**740**	**614**	**511**
Intermediate 4 Results				
Average Round Trip (ms)		27.003265	22.595097	27.003265

Total number of request/reply transmissions at intermediate		1225	1183	1225
Total number of received request messages at intermediate		1923	1851	1923
Total number of received reply messages at intermediate		1236	1244	1236
Total number of duplicated reply messages at intermediate		2336	2151	2336
Total number of duplicated request messages at intermediate		3439	2554	3439
Total number of lost request messages at intermediate		77	149	77
Total number of lost reply messages at intermediate		761	755	761
Intermediate 5 Results				
Average Round Trip (ms)		62.442942	57.965035	55,631542
Total number of request/reply transmissions at intermediate		1183	1144	1433
Total number of received request messages at intermediate		1893	1897	1893
Total number of received reply messages at intermediate		1195	1165	1446
Total number of duplicated reply messages at intermediate		1564	994	1879
Total number of duplicated request messages at intermediate		3746	3406	3732
Total number of lost request messages at intermediate		107	103	107
Total number of lost reply messages at intermediate		805	833	554
Intermediate 6 Results				
Average Round Trip (ms)		57.157109	50.261637	41.109375
Total number of request/reply transmissions at intermediate		1273	1246	1536
Total number of received request messages at intermediate		1942	1943	1948
Total number of received reply messages at intermediate		1284	1265	1543
Total number of duplicated reply messages at intermediate		1870	1664	3328
Total number of duplicated request messages at intermediate		4381	3558	4175
Total number of lost request messages at intermediate		58	56	52
Total number of lost reply messages at intermediate		716	733	457
Intermediate 7 Results				
Average Round Trip (ms)		26.372851	20.676069	21.088406
Total number of request/reply transmissions at intermediate		1105	1099	1380
Total number of received request messages at intermediate		1749	1595	1834

Total number of received reply messages at intermediate		1295	1546	1574
Total number of duplicated reply messages at intermediate		2148	2350	2774
Total number of duplicated request messages at intermediate		1162	1057	1206
Total number of lost request messages at intermediate		188	404	166
Total number of lost reply messages at intermediate		642	453	426
Intermediate 9 Results				
Average Round Trip (ms)			30.761	28.166667
Total number of request/reply transmissions at intermediate			1000	1302
Total number of received request messages at intermediate			1457	1726
Total number of received reply messages at intermediate			1169	1507
Total number of duplicated reply messages at intermediate			1172	1879
Total number of duplicated request messages at intermediate			831	1083
Total number of lost request messages at intermediate			256	274
Total number of lost reply messages at intermediate			544	493
Intermediate 11 Results				
Average Round Trip (ms)		28.021079	33.558319	30.550308
Total number of request/reply transmissions at intermediate		1186	1166	1461
Total number of received request messages at intermediate		1814	1687	1890
Total number of received reply messages at intermediate		1322	1411	1550
Total number of duplicated reply messages at intermediate		1892	1127	2744
Total number of duplicated request messages at intermediate		1647	1196	2135
Total number of lost request messages at intermediate		186	312	110
Total number of lost reply messages at intermediate		678	588	450
Intermediate 12 Results				
Average Round Trip (ms)		54.608273	51.148494	58.486413
Total number of request/reply transmissions at intermediate		1233	1394	1472
Total number of received request messages at intermediate		1896	1917	1922
Total number of received reply messages at intermediate		1245	1406	1480
Total number of duplicated reply messages at intermediate		1774	1476	2113

Total number of duplicated request messages at intermediate		3224	3019	4208
Total number of lost request messages at intermediate		104	83	78
Total number of lost reply messages at intermediate		755	593	520

Results of experiments with 10 nodes (10 laptops) varying message size without mobility

Time: 10:00 **Date: 21/03/2010**

Number of sent requests = 2000 **Inter packet time (delay) = 100 ms**

Size of data = 2000 bytes **Distance = mobility**

Originator Results

Performance Metric	**Data size (bytes)**	**Trials**		
		1st	**2nd**	**3rd**
Average Round Trip Time (ms)		149.66	150.74	173.08
Total number of reply messages at originator		495	441	411
Total number of duplicated request messages at originator		3132	3321	3026
Total number of duplicated reply messages at originator		513	439	403
Total number of lost reply messages at originator		1505	1549	1589
Total number of lost messages at originator (request + replies)		1505	1549	1589
Delivery ratio (replies)		0.247	0.221	0.205
Delivery ratio (RTT)		0.248	0.222	0.205
Average hop count		2.78	2.737	3.088
Duplicate ratio (replies)		1.036	0.995	0.981
Number of out of order reply msgs		2	2	7

Performance Metric	**Data size (bytes**	**Trials**		
		1st	**2nd**	**3rd**

Destination Results				
Total number of request messages at destination		1361	1398	1158
Total number of sent reply messages from destination		1361	1398	1158
Total number of duplicated request messages at destination		865	981	680
Total number of lost request messages at destination		639	602	842
Delivery ratio		0.6805	0.699	0.579
Average hop count		2.647	2.569	2.964
Number of received msgs from originator				
Number of out of order request messages		19	10	13
Intermediate 3 Results				
Average Round Trip (ms)		**75.079167**	**87.703883**	**91.213793**
Total number of request/reply transmissions at intermediate		**480**	**412**	**435**
Total number of received request messages at intermediate		**1479**	**1543**	**1397**
Total number of received reply messages at intermediate		**519**	**430**	**444**
Total number of duplicated reply messages at intermediate		**480**	**333**	**415**
Total number of duplicated request messages at intermediate		**1533**	**1680**	**1478**
Total number of lost request messages at intermediate		**521**	**457**	**603**
Total number of lost reply messages at intermediate		**1481**	**1560**	**1556**
Intermediate 4 Results				
Average Round Trip (ms)		58.919450	68.703407	55.466790
Total number of request/reply transmissions at intermediate		509	449	542
Total number of received request messages at intermediate		1462	1502	1239
Total number of received reply messages at intermediate		558	525	612
Total number of duplicated reply messages at intermediate		505	464	502
Total number of duplicated request messages at intermediate		1066	1301	718
Total number of lost request messages at intermediate		538	498	761
Total number of lost reply messages at intermediate		1442	1465	1388
Intermediate 5 Results				
Average Round Trip (ms)		102.687919	105.834270	123.243323

Total number of request/reply transmissions at intermediate		298	356	337
Total number of received request messages at intermediate		1432	1559	1353
Total number of received reply messages at intermediate		345	370	354
Total number of duplicated reply messages at intermediate		141	181	186
Total number of duplicated request messages at intermediate		984	1405	1197
Total number of lost request messages at intermediate		568	441	647
Total number of lost reply messages at intermediate		1655	1620	1646
Intermediate 6 Results				
Average Round Trip (ms)		89.350711	87.945946	96.912935
Total number of request/reply transmissions at intermediate		422	370	402
Total number of received request messages at intermediate		1475	1467	1361
Total number of received reply messages at intermediate		484	431	431
Total number of duplicated reply messages at intermediate		339	367	346
Total number of duplicated request messages at intermediate		1507	1340	1233
Total number of lost request messages at intermediate		524	533	639
Total number of lost reply messages at intermediate		1516	1559	1569
Intermediate 7 Results				
Average Round Trip (ms)		33.701205	35.932331	30.007449
Total number of request/reply transmissions at intermediate		415	399	537
Total number of received request messages at intermediate		831	1061	1083
Total number of received reply messages at intermediate		716	578	709
Total number of duplicated reply messages at intermediate		665	552	705
Total number of duplicated request messages at intermediate		402	585	541
Total number of lost request messages at intermediate		917	782	917
Total number of lost reply messages at intermediate		1028	1262	1291
Intermediate 9 Results				
Average Round Trip (ms)		63.397297	35.204188	57.530435
Total number of request/reply transmissions at intermediate		370	382	230
Total number of received request messages at intermediate		1210	1150	714

Total number of received reply messages at intermediate		511	597	405
Total number of duplicated reply messages at intermediate		210	346	150
Total number of duplicated request messages at intermediate		450	280	120
Total number of lost request messages at intermediate		727	850	1180
Total number of lost reply messages at intermediate		1423	1393	1489
Intermediate 11 Results				
Average Round Trip (ms)		60.492925	59.238938	58.467706
Total number of request/reply transmissions at intermediate		424	339	449
Total number of received request messages at intermediate		1013	1115	1075
Total number of received reply messages at intermediate		656	492	583
Total number of duplicated reply messages at intermediate		216	274	249
Total number of duplicated request messages at intermediate		357	475	411
Total number of lost request messages at intermediate		987	885	925
Total number of lost reply messages at intermediate		1344	1498	1417
Intermediate 12 Results				
Average Round Trip (ms)		124.084536	129.590698	134.687192
Total number of request/reply transmissions at intermediate		485	430	406
Total number of received request messages at intermediate		1660	1717	1575
Total number of received reply messages at intermediate		497	434	410
Total number of duplicated reply messages at intermediate		261	204	240
Total number of duplicated request messages at intermediate		2006	1861	1801
Total number of lost request messages at intermediate		340	283	425
Total number of lost reply messages at intermediate		1503	1542	1590

Results of experiments with 10 nodes (10 laptops) varying message size without mobility

Time: 10:00 **Date: 21/03/2010**

Number of sent requests = 2000 **Inter packet time (delay) = 100 ms**

Size of data = 4000 bytes **Distance = mobility**

Originator Results

Performance Metric	**Data size (bytes)**	**Trials**		
		1st	**2nd**	**3rd**
Average Round Trip Time (ms)		314.79	330.75	396.33
Total number of reply messages at originator		155	76	150
Total number of duplicated request messages at originator		1884	1966	1766
Total number of duplicated reply messages at originator		110	53	129
Total number of lost reply messages at originator		1839	1843	1850
Total number of lost messages at originator (request + replies)		1839	1843	1850
Delivery ratio (replies)		0.077	0.039	0.075
Delivery ratio (RTT)		0.078	0.04	0.075
Average hop count		2.987	3.039	3.307
Duplicate ratio (replies)		0.71	0.697	0.86
Number of out of order reply msgs		0	0	0

Performance Metric	**Data size (bytes**	**Trials**		
		1st	**2nd**	**3rd**
Destination Results				
Total number of request messages at destination		811	895	725
Total number of sent reply messages from destination		811	895	725
Total number of duplicated request messages at destination		353	285	409
Total number of lost request messages at destination		1189	1105	1275
Delivery ratio		0.4055	0.4475	0.3625
Average hop count		3.051	2.985	3.37
Number of received msgs from originator				
Number of out of order request messages		8	10	6

Intermediate 3 Results				
Average Round Trip (ms)		**167.769737**	**200.405405**	**187.517045**
Total number of request/reply transmissions at intermediate		**152**	**74**	**176**
Total number of received request messages at intermediate		**1062**	**1203**	**886**
Total number of received reply messages at intermediate		**166**	**78**	**190**
Total number of duplicated reply messages at intermediate		**122**	**49**	**124**
Total number of duplicated request messages at intermediate		**610**	**556**	**452**
Total number of lost request messages at intermediate		**938**	**797**	**1114**
Total number of lost reply messages at intermediate		**1828**	**1841**	**1810**
Intermediate 4 Results				
Average Round Trip (ms)		121.450292	111.465517	112.013825
Total number of request/reply transmissions at intermediate		171	116	217
Total number of received request messages at intermediate		862	1034	797
Total number of received reply messages at intermediate		207	134	256
Total number of duplicated reply messages at intermediate		111	65	233
Total number of duplicated request messages at intermediate		346	357	476
Total number of lost request messages at intermediate		1138	966	1203
Total number of lost reply messages at intermediate		1787	1854	1744
Intermediate 5 Results				
Average Round Trip (ms)		234.439394	209.529412	265.890323
Total number of request/reply transmissions at intermediate		66	51	155
Total number of received request messages at intermediate		878	1228	1034
Total number of received reply messages at intermediate		96	66	165
Total number of duplicated reply messages at intermediate		24	14	56
Total number of duplicated request messages at intermediate		323	426	658
Total number of lost request messages at intermediate		1122	772	966
Total number of lost reply messages at intermediate		1898	1853	1835
Intermediate 6 Results				
Average Round Trip (ms)		168.655172	189.187500	224.719298
Total number of request/reply transmissions at intermediate		116	80	171

Total number of received request messages at intermediate		871	998	999
Total number of received reply messages at intermediate		176	105	185
Total number of duplicated reply messages at intermediate		92	49	115
Total number of duplicated request messages at intermediate		444	541	620
Total number of lost request messages at intermediate		1129	1002	1001
Total number of lost reply messages at intermediate		1818	1883	1815
Intermediate 7 Results				
Average Round Trip (ms)		45.725490	77.227723	54.315972
Total number of request/reply transmissions at intermediate		204	101	288
Total number of received request messages at intermediate		581	576	749
Total number of received reply messages at intermediate		371	183	367
Total number of duplicated reply messages at intermediate		221	101	344
Total number of duplicated request messages at intermediate		184	225	441
Total number of lost request messages at intermediate		1242	1287	1251
Total number of lost reply messages at intermediate		1464	1676	1633
Intermediate 9 Results				
Average Round Trip (ms)		111.587719	95.953488	156.125828
Total number of request/reply transmissions at intermediate		114	86	151
Total number of received request messages at intermediate		651	796	715
Total number of received reply messages at intermediate		189	133	201
Total number of duplicated reply messages at intermediate		48	41	88
Total number of duplicated request messages at intermediate		100	101	417
Total number of lost request messages at intermediate		1343	1204	1174
Total number of lost reply messages at intermediate		1805	1864	1680
Intermediate 11 Results				
Average Round Trip (ms)		111.226131	126.816092	93.498371
Total number of request/reply transmissions at intermediate		199	87	307
Total number of received request messages at intermediate		713	637	796
Total number of received reply messages at intermediate		288	156	368

Total number of duplicated reply messages at intermediate		82	32	303
Total number of duplicated request messages at intermediate		243	158	438
Total number of lost request messages at intermediate		1285	1360	1204
Total number of lost reply messages at intermediate		1706	1703	1632
Intermediate 12 Results				
Average Round Trip (ms)		268.967320	289.625000	321.515337
Total number of request/reply transmissions at intermediate		153	64	163
Total number of received request messages at intermediate		1510	1603	1276
Total number of received reply messages at intermediate		154	66	166
Total number of duplicated reply messages at intermediate		58	18	68
Total number of duplicated request messages at intermediate		1070	1073	1014
Total number of lost request messages at intermediate		490	397	724
Total number of lost reply messages at intermediate		1840	1853	1834

Average of 3 experiments with 10 nodes (10 laptops) varying message size without mobility

Time: 10:00 **Date: 21/3/2010**

Number of sent requests = 2000 **Inter packet time (delay) = 100 ms**

Size of data = variable bytes

Performance Metric	**Message size (bytes)**				
	100	**400**	**800**	**2000**	**4000**

Originator Results					
Average Round Trip Time (ms)	16.24	32.15	57.12	157.7	347.3
Total number of reply messages at originator	1510	1314	1403	449	127
Total number of duplicated request messages at originator	7817	6776	6072	3159	1872
Total number of duplicated reply messages at originator	4258	3302	3176	451	97.33
Total number of lost reply messages at originator	489	684	596	1547	1844
Total number of lost messages at originator (request + replies)	489	684	596	1547	1844
Delivery ratio (replies)	0.75	0.65	0.701	0.224	0.063
Delivery ratio (RTT)	0.76	0.65	0.701	0.225	0.064
Average hop count	2.266	2.271	2.361	2.868	3.111
Number of out of order reply msgs	17	6.66	18	3.66	0
Destination Results					
Total number of request messages at destination	1914.666667	1867.333333	1823.666667	1305.666667	810.3333333
Total number of sent reply messages from destination	1914.666667	1867.333333	1823.666667	1305.666667	810.3333333
Total number of duplicated request messages at destination	4087.666667	2785	2324	842	349
Total number of lost request messages at destination	85.33333333	132.3333333	176	694.3333333	1189.666667
Delivery ratio	0.957333333	0.933823333	0.911986333	0.652833333	0.405166667
Average hop count	2.255666667	2.285333333	2.390666667	2.726666667	3.135333333
Number of received msgs from originator					
Number of out of order request messages	12.6666	8.6666	17.6666	14	8

Performance Metric	**Message size (bytes)**				
	100	**400**	**800**	**2000**	**4000**
Intermediate 3 Results					
Average Round Trip (ms)	**11.737254333**	**27,1164133**	**43,84227633**	**84,66561433**	**185,230729**
Total number of request/reply transmissions at intermediate	**1458.33**	**1291,66**	**1360**	**442,33**	**134**
Total number of received request messages at intermediate	**1927.33**	**1924,66**	**1889**	**1473**	**1050,33**
Total number of received reply messages at intermediate	**1462.3**	**1297,66**	**1378**	**464,33**	**144,666**
Total number of duplicated reply messages at intermediate	**3786**	**2730,66**	**2586**	**409,33**	**98,33**
Total number of duplicated request messages at intermediate	**6408,33**	**4564**	**3835,66**	**1563,66**	**539,33**
Total number of lost request messages at intermediate	**537,666**	**75,33**	**111**	**524**	**997,88**
Total number of lost reply messages at intermediate	**72.666**	**696,666**	**621,666**	**1532,33**	**1826,33**
Intermediate 4 Results					
Average Round Trip (ms)	14.1708293	24.7991775	45.4023397	61.0298823	114.976545
Total number of request/reply transmissions at intermediate	1453.3	1204	958.333	500	168
Total number of received request messages at intermediate	1921.3	1887	1377.6	1401	897.66
Total number of received reply messages at intermediate	1458.3	1240	969.666	565	199
Total number of duplicated reply messages at intermediate	3558.3	2243.5	1682.666	490.3	136.33
Total number of duplicated request messages at intermediate	4896.6	2996.5	2489	1028.3	393
Total number of lost request messages at intermediate	78.666	113	71	599	1102.33
Total number of lost reply messages at intermediate	558.333	758	479	1431.66	1461.66

Performance Metric	Message size (bytes)				
	100	**400**	**800**	**2000**	**4000**
Intermediate 5 Results					
Average Round Trip (ms)	14,02922	29,480705	40,163011	110,588504	236,619710
Total number of request/reply transmissions at intermediate	1452,667	1258,333	1253,333	330,3333	90,66667
Total number of received request messages at intermediate	1924	1923,667	1894,333	1448	1046,667
Total number of received reply messages at intermediate	1455,667	1265	1268,667	356,3333	109
Total number of duplicated reply messages at intermediate	3009,333	2055,333	1479	169,3333	31,33333
Total number of duplicated request messages at intermediate	5976,333	4215	3628	1195,333	469
Total number of lost request messages at intermediate	76	76,33333	105,6667	552	953,3333
Total number of lost reply messages at intermediate	544,3333	733,6667	730,6667	1640,333	1862
Performance Metric	**Message size (bytes)**				
	100	**400**	**800**	**2000**	**4000**
Intermediate 6 Results					
Average Round Trip (ms)	15.35648433	26.939143	49.50937367	191.40319733	194.1873233
Total number of request/reply transmissions at intermediate	1481.666667	1301	1351.6666667	398	122.33333
Total number of received request messages at intermediate	1971	1960	1944.333333	1434.3333	956
Total number of received reply messages at intermediate	1492.666667	1310.333333	1364	448.6666667	155.3333333
Total number of duplicated reply messages at intermediate	2287.333333	2428	2287.3333333	350.666667	85.33333333
Total number of duplicated request messages at intermediate	5311	4036.333333	4038	1360	535
Total number of lost request messages at intermediate	29	40	55.33333	565.33333	1044
Total number of lost reply messages at intermediate	507.33333	688.3333333	635.3333	1548	1838.6666

	3				667
Performance Metric	**Message size (bytes)**				
	100	**400**	**800**	**2000**	**4000**
Intermediate 7 Results					
Average Round Trip (ms)	5.5056673	13.162195	22.712442	33.213662	59.089728
Total number of request/reply transmissions at intermediate	1323.6667	971.66667	1194.6667	450.33333	197.66667
Total number of received request messages at intermediate	1860	1565.3333	1726	991.66667	635.33333
Total number of received reply messages at intermediate	1470	1259	1471.6667	667.66667	307
Total number of duplicated reply messages at intermediate	3424.6667	2280.3333	2424	640.66667	222
Total number of duplicated request messages at intermediate	2617	1439	1141.6667	509.33333	283.33333
Total number of lost request messages at intermediate	85.666667	255	252.66667	872	1260
Total number of lost reply messages at intermediate	475.66667	560.33333	507	1193.6667	1591
Performance Metric	**Message size (bytes)**				
	100	**400**	**800**	**2000**	**4000**
Intermediate 9 Results					
Average Round Trip (ms)	5.856759	20.445926	29.463833	52.043973	121.2223
Total number of request/reply transmissions at intermediate	3935	1188	1151	327.3333	117
Total number of received request messages at intermediate	1773.3333	1842.6666	1591.5	1024.6667	720.6667
Total number of received reply messages at intermediate	1378	1285.3333	1338	504.3333	174.3333
Total number of duplicated reply messages at intermediate	2115.3333	1429.6667	1525.5	235.3333	59
Total number of duplicated request messages at intermediate	2249.3333	1363	957	283.3333	206
Total number of lost request messages at intermediate	90.33333	139.6667	265	919	1240.3333
Total number of lost reply messages at intermediate	485.66667	696.3333	518.5	1435	1783
Performance Metric	**Message size (bytes)**				
	100	**400**	**800**	**2000**	**4000**

Intermediate 11 Results					
Average Round Trip (ms)	9.635485333	17.36212133	30.709902	59.39985633	110.513513
Total number of request/reply transmissions at intermediate	1355	1068	1271	404	197.66
Total number of received request messages at intermediate	1902.33	1723	1797	1067.66	715.33
Total number of received reply messages at intermediate	1488.33	1298.33	1427.66	577	270.66
Total number of duplicated reply messages at intermediate	2423.66	1330.66	1921	246.33	139
Total number of duplicated request messages at intermediate	2041	1256.33	1659.33	414.33	279.66
Total number of lost request messages at intermediate	97.66	276	202.66	932.33	1283
Total number of lost reply messages at intermediate	511.66	700.33	575	1419.66	1680.33
Performance Metric	**Message size (bytes)**				
	100	**400**	**800**	**2000**	**4000**
Intermediate 12 Results					
Average Round Trip (ms)	15.8889823	30.1373793333	54.7477266	129454142	293.369219
Total number of request/reply transmissions at intermediate	1416.33333	1284.3333333	1366.33333333333	470.33333333	126.666666667
Total number of received request messages at intermediate	1298.3	1934.3333333	1911.66666667	1650.666666667	1463
Total number of received reply messages at intermediate	1462.6666667	1289.666666667	1377	447	128.666666667
Total number of duplicated reply messages at intermediate	2642	1859	1787.66666667	235	48
Total number of duplicated request messages at intermediate	5377.3	3768.3333333	3483.66666667	1889.33333333	1152.33333333
Total number of lost request messages at intermediate	70	64.3	88.3333333333	349.333333333	537
Total number of lost reply messages at intermediate	537.3	706.6	622.666666667	1545	1842.33333333