A New Routing Protocol for Ad Hoc Wireless Networks
Design, Implementation and Performance Evaluation

Idris Skloul Ibrahim

Submitted for the degree of Doctor of Philosophy

Heriot Watt University

School of Mathematical and Computer Sciences

26- July - 2011

To My Parents, my wife Salwa, my sisters and brothers

Atya, Ibrahim, Saleh and Azdeen.

To my Son Nour and my daughters

Amira, Aya, Aiman, Ibtihal, Adean and Ayat.

*In memory of my beloved oldest brother "**Rajab**" who gave me tremendous support and encouragement in achieving my Ph.D. research.*

I will always miss him.

Table of Contents

List of Figures

List of Tables

Acknowledgements

In the name of Allah, the Compassionate, the Merciful.

All praise is due to **Allah**, the **Lord** of worlds, and peace and blessings of Allah be upon his noblest messenger and servant, the prophet *Mohamed*, and upon his family and his companions.

First, and foremost, I must thank almighty Allah who has always been helping me. I would like to express my sincere thanks and gratitude to Dr. Peter King, the first supervisor, who showed me the real objectives and indescribable technique of ad hoc networks architecture and protocols. Thus, without the means of his useful scholarly notes, this work would not have been possible. He has guided my work from the very beginning, and maintained his valuable comments and guidance throughout the long and painful stages of the thesis writing. Pre-eminent in this regard is also, Professor Robert J. Pooley, my second supervisor, for his useful feedback. Without his expert advice, this work would not be in this final form.

I would be indeed remiss if I failed to express my sincere thanks and appreciation to the all administrative staff and staff members of the school. I also would like to express my gratefulness to my colleagues, the students of the department for their feedback on my presentations. Last, but not least I would like to thank all the members of my family for encouraging and supporting me to present this work in the best form possible. A special word of thank is also due to the Libyan students' community based in Edinburgh for its constant and invaluable support. Finally, I shall be forever indebted to the Libyan People out of whose budget my scholarship was financed.

Abstract

A collection of mobile nodes can form a multi-hop radio network with a dynamic topology and without the need for any infrastructure such as base stations or wired network. Such a Mobile Ad Hoc Networks (MANETs) maintain their structure and connectivity in a decentralised and distributed fashion. Each mobile node acts as both a router for other nodes traffic, as well as a source of traffic of its own

In this thesis we develop and present a new hybrid routing protocol called *Multipath Distance Vector Zone Routing Protocol*, which is referred to as MDVZRP. In MDVZRP we assume that all the routes in the routing table are active and usable at any time, unless the node received or discovered a broken link. There is no need to periodically update the routing tables, therefore reducing the periodic update messages and hence reducing the control traffic in the entire network.

The protocol guarantees loop freedom and alternative disjoint paths. Routes are immediately available within each routing zone. For destinations outside the zone, MDVZRP employs a route discovery technique known as routing information on demand. Once the node is informed by either the MAC layer or itself that it should discover the non- reachable nodes, MDVZRP adopts a new technique.

First, we discuss the Ad Hoc networks and routing in general, then the motivation of MDVZRP regarding the nodes' flat view, and the selection and acquisition of multipath getting and selection. Furthermore, we describe the stages of MDVZRP and the protocol routing process with examples. The performance of MDVZRP is then evaluated to determine its operating parameters, and also to investigate its performance in a range of different scenarios.

Finally, MDVZRP is compared with DSDV and AODV ordinary routing protocols (standard) delivering CBR traffic. Simulation results show that MDVZRP gives a better performance than DSDV in all circumstances, it is also better than AODV in most of the scenarios, especially at low mobility.

CHAPTER 1: INTRODUCTION

The mobile computer users have had a dream that is to access the internet where they are and while walking into their offices. Consequently, various solutions have been provided to achieve this goal. The most practical approach presented is to install short-range radio receivers and transmitters in offices and the portable mobile computers too to allow them to communicate easily. Such techniques have rapidly led to Wireless Local Area Networks (Jianfeng, 2009; Martin, 2011). Increasing demand for cheap, portable and mobile devices for general business and applications have made mobile computing enjoy a tremendous rise in popularity.

Projections have been made that there are billions of wireless devices in use; also independent market sources show that every 1½ year the number of users of wireless devices have been doubling. In addition, a report from DUBLIN - Research and Markets finds that, the wireless adapter’s (NIC) volume show double-digit sequential growth in the fourth quarter of 2009. Also, It was projected that notebook and netbook PC volume will still experience double-digit increase in the future. The content of this report is based on primary data obtained through interviews with WLAN makers (DUBLIN, 2010).

Wired devices and networks also provide effective options for information, data and resources exchange via networks, but sending via a wired medium is not free of the communication problems and issues. Non flexibility is the main issue of the wired system, the wire was extended and, if installed, it is relatively difficult to re-install in another location without the effort and inconvenience to users, as wired devices do not provide a reliable communication for phone users with high mobility.

In recent past, the wireless networks and communications are considered to be an effective choice; the demand for these networks has increased because of:

- ❑ Successive developments in wireless technologies and products.
- ❑ Continued decline in prices.
- ❑ The great freedom available to users of movement without affecting their business.

This can be likened to the wireless networks of mobile phone networks, where the user can move anywhere he wants to and stay connected with the other terminal as long as it is within that network reception. Wireless networks may be a misleading term, as the majority of networks are not completely wireless, in most cases, these networks are a mixture of devices connected to wires and other devices connected to wireless networks, this type of hybrid systems are called *Hybrid Networks*. Of course, the infrastructure based networks provide a reliable, easy and secure connectivity for mobile devices; however, it takes time to establish such type of network, additional to the time taken, the costs associated with the installation of infrastructure can be quite high.

1.1 Classification of Wireless Networks

Once the dream of the mobile computers users became true by the existence of many wireless networks models and techniques that brings the trouble of compatibility to wireless communications, where networks equipments were incompatible. In reality, that means a computer equipped with a brand X radio would not work in a room equipped with a brand Y base station (Jianfeng, 2009; Martin, 2011).

Finally, to overcome the wireless LAN issues that related to compatibility, the IEEE standardisation committee has been tasked by industry to draw up a standard for wireless LAN as tasked and as had previously been established for wired LAN. IEEE named the wireless standard in name of 802.11, at the beginning it was known as WiFi, however, we will call it by its proper name, 802.11.

The IEEE proposed to classify the WLAN into two models:

- ❑ Infrastructure based or access point based.
- ❑ Non-infrastructure based or in the absence of an access point.

In the former model, the communication takes place through fixed base stations or Access Points (AP), which coordinate the communication between the mobile devices as shown in Figure 1.1.

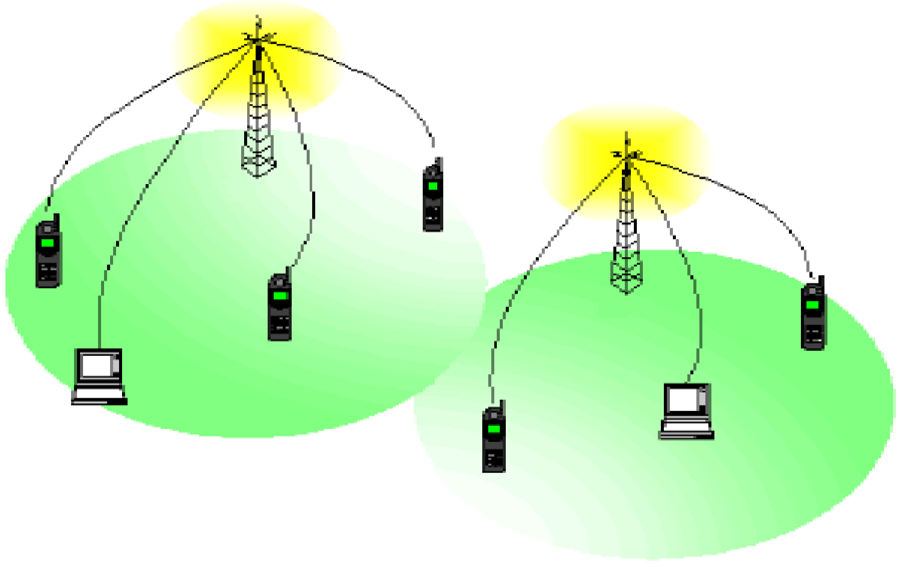


Figure 1.1: Infrastructure based WLAN

While in the second model, the mobile devices (Sensors / Computers) help each other to establish such communication, where each device works as a workstation and a router at the same time, as shown in Figure 1.2. Such networks work in absence of any static support of infrastructure (Ad Hoc Networks).

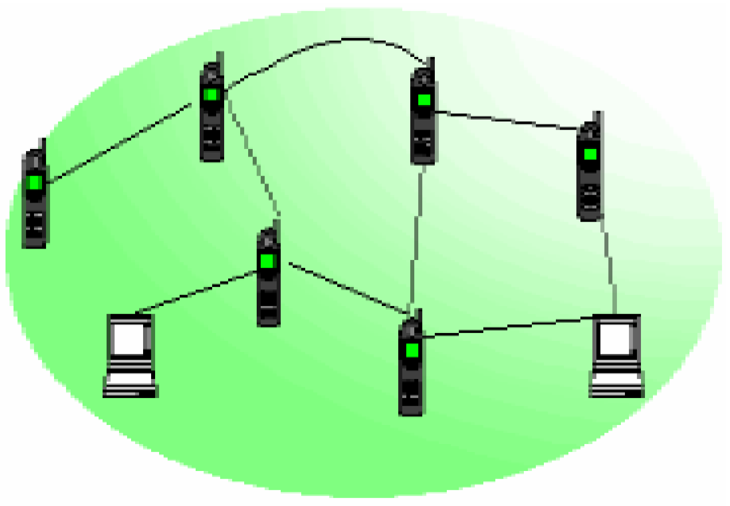


Figure 1.2: Non-Infrastructure based WLAN

Furthermore, in some situations the infrastructure is not available, or cannot be installed for economical, natural or geographical reasons such as cost, the available time, or natural disasters, etc. the non-infrastructure network can provide the necessary communications and network services in such situations (Roy, 2011). The nodes in such networks have to announce their presence periodically and listen for their neighbours announcements broadcast to discover and learn how to reach each other. In mobile non-infrastructure model, mobility and scalability are the main challenges, with mobility implying a non predictable topology and stale routes, while scalability means more traffic and overhead control packets. Non-infrastructure networks can be realized by different types such as wireless sensor networks (WSN) (Rodig, 2009), body area network (BAN) (Yang, 2006), and vehicular Ad Hoc networks (VANET) (Yousefi, 2006; Nasui, 2010).

Hence, efficient routing algorithms (protocols) are needed to make the communication between the mobile nodes possible, easy and reliable over multi-hop paths, consisting of several links, dynamic and non-predictable topology in a way that saves the network resources as much as necessary.

Many routing protocols have been proposed to provide routes in such dynamic environments. In mobile non-infrastructure networks, the main purpose of a conventional or standard protocol is to control the way in which the mobile nodes decide how to exchange the routing information between each other.

Multipath routing is a technique that provides multiple alternative paths between each source node and destination; the benefit of this technique is fault tolerance, increasing bandwidth, and security improvements. However, overlapping, looping (infinity loop) and optimum disjointed paths or node-disjointed are the main problems in such algorithms. Therefore, finding node disjoint multiple paths in mobile networks is not an easy task.

Designing and testing of such routing protocols in reality consume a lot of time and cost a lot of money. Even though there is no efficient simulator in 100 %, using tested known network simulators is a good way to test and evaluate a new protocol design where that technique saves the researchers time and money. To design or study a routing protocol it is recommended to simulate it and evaluate its performance. Mobility models and communication traffic patterns are the key parameters of the protocol simulation technique in the mobile non-infrastructure networks (Ad Hoc networks).

The most widely used simulator for wired and wireless networks is the Network Simulator (Kevin, 2010). We have used Ns version 2 (Ns2) in designing, testing, evaluating and improving our new hybrid routing protocol, Multipath Distance Vector Zone Radius Protocol MDVZRP.

We also used the Random Waypoint (RWP) (Ganapathi, 2008) model which is the most widely used mobile model in such research. While, Constant Bit Rate (CBR) is the communication traffic pattern used in the evaluation of our new routing protocol. This is explained in detail in chapter 2, which includes history, challenges, classifications, applications and protocols of mobile Ad Hoc networks.

1.2 Summary

This introductory chapter has briefly overviewed both infrastructure and non-infrastructure wireless networks in general. A simple idea on routing protocols for multi-hop mobile Ad Hoc networks is also given. However, they are discussed in more detail in chapter 2. In addition, we briefly mentioned the simulator used in this research (Ns2), the mobility model (Random Waypoint), and traffic pattern (Constant Bit Rate).

1.3 About This Thesis

In this section we will present the overall scope of the thesis, our contribution and finally, an overview of its structure.

1.3.1 Statement of the Problem and Scope of the Study

Multi-hop mobile Ad Hoc networks presents an open area for research and opportunities for making significant contributions to it, and many complex issues are related to their features of random dynamic topology including the lack of centralisation (monitoring, management, and security), the fact its an open medium, and the cooperative algorithms involved.

This thesis takes the routing protocol algorithms for multi-hop mobile Ad Hoc networks as an area of research. This topic has received a lot of concern and attention during the last few years, where this has been extensively reviewed, and discussed in many conferences and academic researches. Furthermore, because of the increasing concern for the multi-hop mobile Ad Hoc network applications and use, this area of research still receives a lot of industry, and government interesting and funding.

1.3.2 Contributions of this thesis

Most of the multi-hop mobile systems are dependent on batteries to perform their functionality. Hence, power consumption becomes one of the mobile Ad Hoc networks issues, especially in an environment where the power charge from time to another is a difficult job, such as under water and vehicles in hazardous area applications. We have designed a new routing protocol presents a solution for power consumption for the mobile Adhoc networks by reducing the number of routing packets sent per data packet delivered at the destination. Hence, the average control overhead is reduced. This research focuses on the performance evaluation of the new routing protocol compared with standard routing protocols for mobile Ad Hoc network. The contribution includes the protocol design, implementation and performance evaluation based on dynamic network scenarios and topologies.

1.3.3 Thesis Roadmap

Chapter One, the *Introduction*, has briefly outlined the development of wireless networking and the growing desire for such networks. Initially, it presents the wireless networks aims and classifications; it also has briefly discussed and contrasted multi-hop mobile Ad Hoc networks, their various types of routing algorithms protocols, simulator that used in this thesis, the mobility model, and the traffic pattern used. Finally, the remainder of the thesis is summarised.

Chapter Two presents this thesis's *multi-hop mobile Ad Hoc networks* (MANETs) background. It is a continuation of the introductory chapter, with more discussion and details. Initially, it presents the *multi-hop mobile Ad Hoc networks* in detail, which is the objective of this thesis, where it gives an introduction to its history, characteristics, and applications. Problems specified to mobile Ad Hoc networks (tradeoffs) are pointed out and some examples of the routing protocols that are used for mobile Ad Hoc networks are mentioned. This chapter goes on to discuss the models that are used in such networks as route determination models, data packets forwarding techniques, broadcasting or communication models, mobility models, and propagation models. Finally, it presents the simulation overview and computer networks simulators in more detail, specially the simulator that has been used in this thesis (Ns2).

Chapter Three concerns the *problem and motivation*. It presents the aims of the research and related work regarding the new routing protocol stages and algorithms. This chapter includes detailed explanation of the new routing protocol motivation, selection of optimum available path, zone radius and nodes density, routing initialization, node movement and route on demand.

Chapter Four introduces our new routing protocol (MDVZRP) *implementation* using the network simulator (Ns2), after a brief look at the early stages of the design process that has ultimately led to the current version of MDVZRP. It discusses and describes in detail the structure and components of the second version of MDVZRP v2.00, such as, routing table structure, packet format, packet types and their mechanisms. Finally, it presents in more detail the packet implementation, routing agent implementation, and all its functions using pseudocode for clearance.

Chapter Five provides a number of trace-based simulation scenarios for *testing the functionality* of MDVZRP v2.00 in Ns2. Several test-runs are conducted using trace-based simulation support, and their results are compared to expected real world results and discussed.

Chapter Six presents the figures and discussion of the new routing protocol (MDVZRP) *performance evaluation* with DSDV (Perkins, 1994) and AODV (Perkins, 2001) standard protocols. Several test-runs are conducted using trace-based simulation support, where the results based on known metrics are evaluated, discussed and graphically presented. We have also shown and explained the performance improvements of MDVZRP v2.0 compared to MDVZRP v 1.0. The primary metrics we considered to evaluate the performance of MDVZRP were Packet Delivery Fraction (PDF), End to End Delay (EED), Throughput, Normalised Routing Load (NRL) and Overhead (OH).

Chapter Seven concludes this thesis, summarises the outcome of the entire research, contributions and achievements, and then provides some notes on future work.

Appendices list and outline the relevant papers (Publication authored) and other related papers during the period that the work for this thesis was carried out.

CHAPTER 2: MANETS

2.1 Introduction

Currently, wireless networks have started to be the choice for effective networking because of successive developments in wireless product technology, a continued decline in prices, and the great freedom available for users to move location without affecting their business. This can be likened to the wireless communication of mobile phone networks, where the users can move anywhere they like and stay connected as long as they stay within the coverage area and have a good reception. Much wireless technology is based upon the principle of direct point-to-point communication as shown in Figure 2.1. In most popular communication models such as Wireless Local Area Network and Group Standard for Mobile communications (GSM) (GSM Association, 2010), mobile nodes use an approach, where communication takes place by nodes connecting to each other via some centralised access points. Therefore, centralisation and infrastructure are a part of the characteristics of such networks, where they are necessary for their configuration and operation (Martin, 2011).

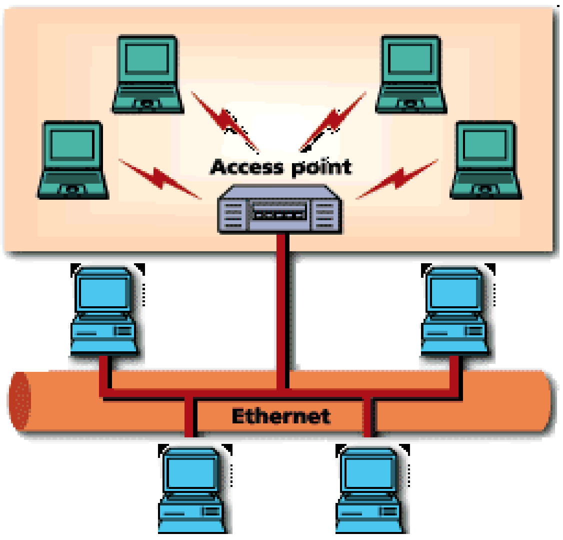


Figure 2.1: Infrastructure based Network

There is also another approach, where mobile nodes utilise each other as access point or relays for traffic when they cannot establish direct communication with endpoints (Out of direct communication range). That model of communication is called multi-hop or Mobile Ad Hoc networks or MANETs (Roy, 2011) as shown in Figure 2.2. This type of network uses the multi-hop model. It can be set up randomly and when needed (on-demand), and should be self configuring. All nodes can be mobile resulting in a possibly dynamic network topology (Xiang, 2008).

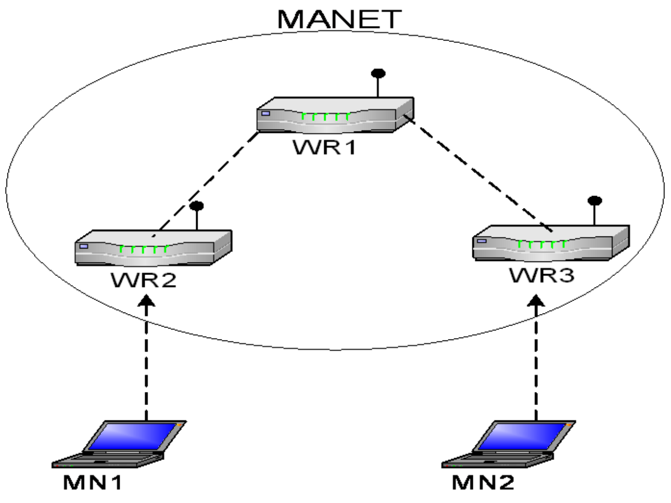


Figure 2.2: Mobile ad-hoc networks (MANETs)

Since we are interested in Computer networks, and based on the previous sections and media of propagation, we can say the Computer local area networks can be classified as shown in Figure 2.3, where the red marks show our area of interest.

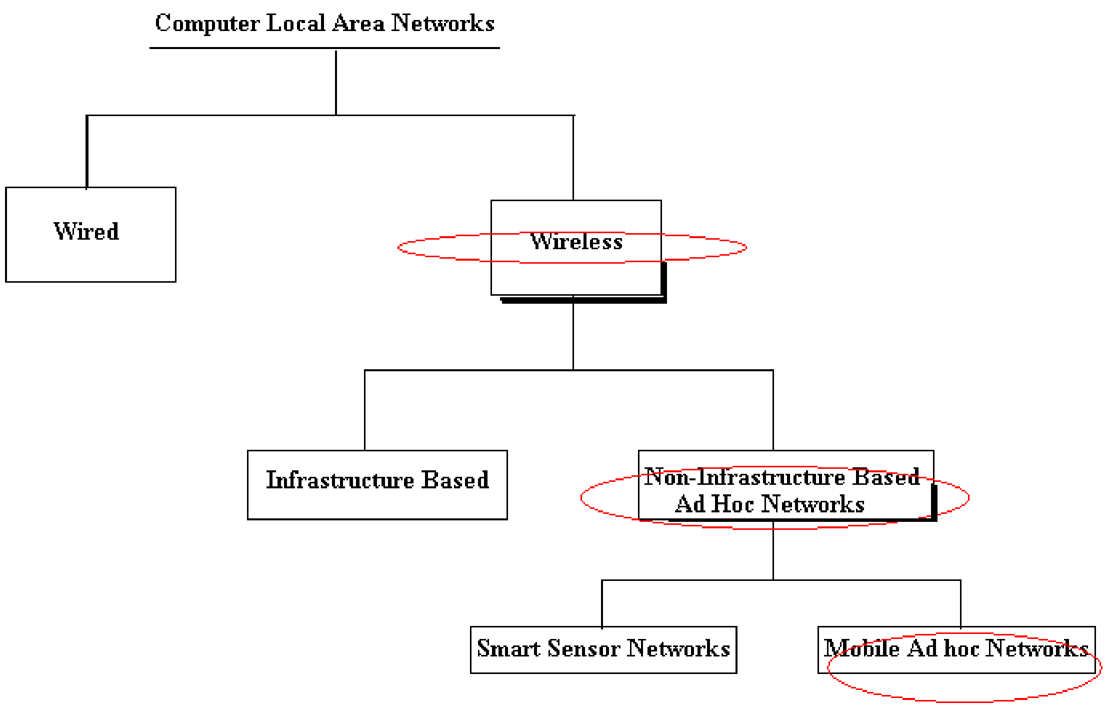


Figure 2.3: Computer local area networks

2.2 MANETs

First, the term “Ad Hoc” has been borrowed from the Latin term which means “for this purpose” or: "to manufacture or to use a special form”. For this reason, this type of wireless computer networks is called Mobile Ad Hoc Networks or (MANET) (Michel, 2007; Roy, 2011).

MANET is a collection of mobile nodes, where data packets are transferred from one node to another without passing through any access point by forming a temporary multihop radio network to maintain connectivity in a decentralised manner. Since the communication between the nodes in MANET is based on the wireless approach, it also suffers from the effects of radio communication, such as interference, noise, and fading. Furthermore, MANET has less bandwidth in the links between nodes than in a wired network. In general, topology in MANET is dynamic due to the departure of nodes and arrival of new nodes, so it varies with time. Therefore, links between nodes are not stable and not fixed, *changeable*. Some of MANET’s features are based on the packet radio networks that were studied extensively in the 1970s and 1980s (Martin, 2011).

2.2.1 History of MANETs

- ❑ The first generation dates back to the early **70s**, as research sponsored by Defence Advanced Research Projects Agency DARPA into using packet switched radio communication to provide reliable computer networks. At the time they were called Packet Radio Networks (PRNET) (Redi, 2002).

- ❑ The second generation of MANET emerged in the **1980s** from a project by DARPA too, this was called Survivable Adaptive Radio Networks (SURAN) and its aim was to enhance and implement a set of MANET systems. That project provided packet switched networks in a non-Infrastructure environment to the mobile battlefield (Redi, 2002).

- ❑ In the **1990s,** adoption of "Ad Hoc networks" term by the IEEE 802.11 subcommittee, and notebook revolution brought a commercial approach to public Ad Hoc networks as a part of mobile wireless computers and some other communication equipment. In addition, mobile networks became a focus of discussion at several research conferences (Friisø, 2003; Redi, 2002).

- ❑ In the late 90s and early **2000s**, the availability of simple intelligent equipment such plug-in and play systems allowed the establishment and management of personal wireless local area networks, hence becoming affordable; even in areas with no available infrastructure for such communication (Chaudet , 2005).

- ❑ In **early / mid 2000s** MANET’s commercial applications were launched. Bluetooth, a commercial application of MANETs, provides a quick communication between the personal area networks users to eliminate use of wired networks (Bluetooth, 2007; Martin, 2011). Research into the concept of m-commerce trading systems using MANETs is being developed (Osman, 2008; 2011).

- ❑ The **next-generation** of ad hoc networks need to be able to handle high mobility in order to support a wide range of emerging applications such as vehicular networks and mobile sensor networks (Ai Hua Ho, 2009).

2.2.2 MANETs Characteristics

From the introduction, we can summarise that MANET (Taruna, 2011) is an autonomous system of mobile nodes moving at any time in random dynamic topology, these mobile nodes are self organised and deployed with routing capabilities, and communicate over wireless links in the form of peer-to-peer and multi-hop forwarding connectivity independent of centralised authority. The system may operate in isolation, because of a lack of any fixed infrastructure, or may have gateways to and interface with a fixed network (Roy, 2011)

2.2.3 MANETs Issues

In addition to the effects of radio communication issues that MANETs are vulnerable to and which were inherited from the wireless communication system (interference, noise, and fading), there are also other constraints in the network security and energy, for example, when the MANETs nodes depend on batteries or other exhaustible energy systems for their operations. Regarding security, MANETs are generally more prone to physical security threats than centralised authority (wired / wireless) networks (Ramanarayana, 2007; Hoang, 2008; Nishu, 2009).

2.2.4 Some Applications of MANETs

Unlike a fixed wireless network, MANETs are suited for use where infrastructure is unavailable. One of many possible applications of MANETs is in some environments, where the need for collaborative mobile devices might be more important, for instance in an outside environment rather than inside offices. There are many applications where MANETs can be more beneficial than other networks such as in emergency response networks, search and rescue, policing and disaster recovery, where rapid communication is crucial (Luis, 2008; Osman, 2008).

2.2.5 Difficulties for Routing in MANETs

- ❑ Transmission range limitation.
- ❑ Low bandwidth.
- ❑ Higher error rates.
- ❑ Vulnerable to interference.
- ❑ Power consumption.
- ❑ No specific devices to do routing.
- ❑ Dynamic nature - frequent topological changes.

2.2.6 Routing Algorithm in MANETs

Moving a data packet from a specific source to a destination in an internetworking is called routing (Goldsmith, 2005). This process is the Internet key feature, where it enables the user’s messages to be transferred from one computer to another until they reach the target user or machine. There is a dedicated device that usually performs such a process in some types of wired and wireless networks, called a router.

While in MANET, the node moves at different speeds in independent random form, connected by any number of wireless links, where each intermediate node is ready to pass or forward both data and control traffic unrelated to its own use ahead. Determining and selecting of the best route to the target machine or node is also part of routing protocol process in MANET (Sharvani, 2009).

2.2.7 MANETs Routing Protocols

Because of some MANET characteristics and tradeoffs such as a lack of fixed infrastructure, no centralised authority, the mobile nature of the nodes, limitation in bandwidth and security, the traditional routing protocols are not suitable for communication over such networks. Hence, communication between nodes in such networks is needed for efficient routing protocols to allow the nodes to communicate in such an environment. Since these networks pose many complex issues, extensive studies and research have been carried out to provide solutions to such issues, for example, efficient routing protocols (Boukerche, 2009; Taruna, 2011).

Routing has become a major area of MANETs research. In recent years, many routing algorithms for mobile MANETs have been proposed, but it is not clear how different algorithms behave in different environments. An algorithm may be better in a particular network but worse in another. Therefore, many routing protocols are provided in such unpredictable dynamic environments.

These protocols can be broadly classified into three categories, namely, ***proactive***, ***reactive***, and ***hybrid***. Figure 2.4 shows the classification of Ad Hoc routing protocols with some examples of standard (known) routing protocols (Abolhasan, 2004; Boukerche, 2009).

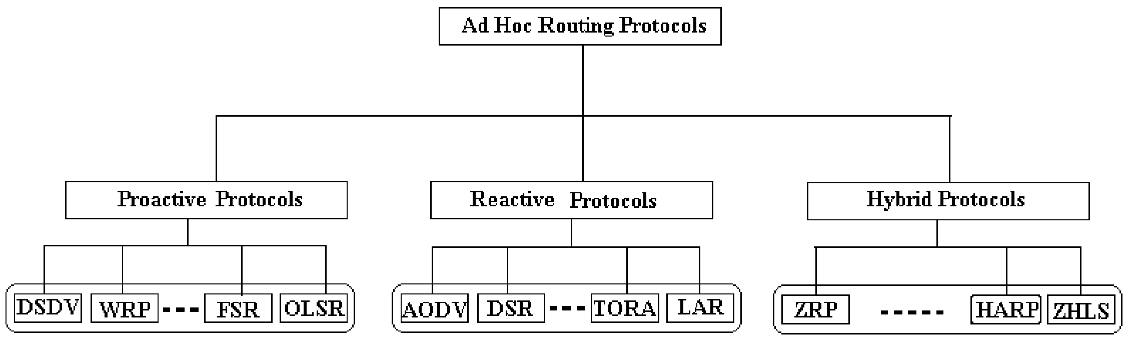


Figure 2.4: Classification of Ad Hoc routing protocols

The next few sections include a brief overview of some previous Ad Hoc routing protocols, and we are going to focus in detail on a number of routing protocols that have a close relationship to our new protocol in some phases.

2.2.7.1 Proactive Routing Protocols

Proactive protocols are also called table-driven, and traditionally are classified as either *distance-vector* or *link-state* protocols. They perform their tasks by periodically maintaining fresh lists of routes for each destination in the entire network. Therefore, routes are calculated in advance to all included nodes (needed or not needed) even those in which no data packets are sent (Xiang, 2008).

The advantage of such protocols is that they initiate low latencies, because routing information is already available at the transmission of the first data packet. However, they continuously react to topology changes, and use resources to provide up-to-date routing information, even when those changes have not affected any traffic, when that increases the amount of routing overhead which counts as the weakness of such routing protocols. Furthermore, any change notified by any node, propagates through the entire network to provide routing information for mobile nodes. Otherwise, some nodes routing table information would remain stale, where that may lead to the risk of link failure in some cases.

In general, due to the amount of overhead routing, this family of routing protocols tends to have difficulty in managing large scale mobile networks. Furthermore, if the size of the network is large, the routing tables will occupy a large space of physical storage or Memory, and updates may lead to inefficient resources of the network if they occur too frequently. The following protocols are some known types from the proactive family (Xiang, 2008; Boukerche, 2009; Taruna, 2011).

Destination Sequenced Distance Vector Routing protocol (DSDV): is one of the earliest protocols for MANETs. It was developed by Perkins and Bhagwat in 1994, but this description is based on the Bellman-Ford algorithm (Thomas, 2001). The concept of Perkins's algorithm is based on each node constructing its own routing table containing routes to each destination, where each entry in that table is marked with an updated, even sequence of numbers generated by the destination for each active workable route and an odd or an infinite number for broken routes (Taruna, 2011).

Nodes update their routing tables in two ways, an infrequent full dump (whole routing table of transmitting node) when there are significant changes, and smaller more frequent incremental updates, since the last full dump. Figure 2.5 shows a simple Ad Hoc network for 7 nodes, while Table 2.1 illustrates node (A) routing table and all possible reachable paths by node A. Each path in fact is an entry in the node (A) routing table, containing a destination number, next hop, number of hops, and sequence number

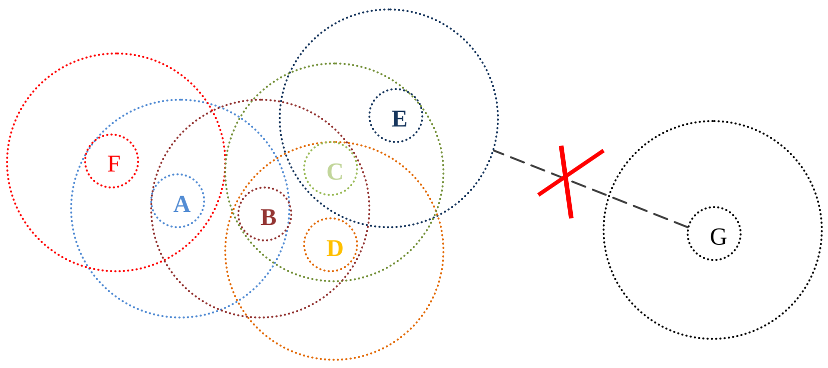


Figure 2.5: A simple Ad Hoc network for 7 nodes

Table 2.1: Routing Table of Node A

Destination	Next Hop	Number of Hops	Sequence Number	Install Time
A	A	0	A-22	T1
B	B	1	B-12	T2
C	B	2	C- 48	T3
D	B	2	D- 26	T4
E	B	3	E- 48	T5
F	A	1	F-50	T6
G	B	4	G-∞ (G-250)	T7

Route selection depends on two fields, the sequence number and number of hops, where the sequence number has the highest precedence (Priority). If new route information is received, the route with latest sequence number is used, if the new route has same sequence number as the one already saved in the routing table, the one that has a better metric (less number of hops) is chosen (Perkins, 1994; Xiang, 2008).

Wireless Routing Protocol (WRP) is a distance vector proactive unicast routing protocol for MANETs. WRP was introduced by SHREE MURTHY in 1995 (Murthy, 1995). It is also based on Bellman-Ford's algorithm and it is in fact an enhanced version of DSDV, where each node maintains an up to date routing table and informs its neighbours by a single update message. Hence, it expects to receive an acknowledgment message (AKC) from each neighbour. Therefore, WAR's mechanisms guarantee a reliable message exchange and a readily available route to every destination in the entire network. It differs from DSDV in that, the DSDV maintains only one routing table to provide routing information, while WRP uses four tables as follows: Routing Table (RT), Distance Table (DT), Link-Cost Table (LCT), and Message Retransmission List (MRL) to provide more accurate routing information (Taruna, 2011).

There are several proactive routing protocols presented as uniform routing protocols for MANETs such as Fisheye State Routing (FSR). It is based on link state routing, the routing information is immediately provided when needed (Pei, 2000). Distance Routing Effect Algorithm for Mobility (DREAM) is another proactive routing protocol. It is a location based routing, and uses two techniques. The first, called the distance effect, the location information in routing tables can be updated as a function of the distance separating nodes without compromising the routing accuracy. The greater the distance separating two nodes, the slower they appear to be moving with respect to each other. Accordingly, the second technique is that of triggering the sending of location updates by the moving nodes autonomously, based only on a node's mobility rate (Stefano, 1998).

In addition, some other proactive routing protocols classified as non-uniform are presented as well, they actually Core-node based routing, such as Landmark Ad Hoc Routing (LANMAR). In large-scale ad-hoc networks an enhanced version of LANMAR protocol by the same authors (Hong, 2000) is presented to dramatically reduce routing table size and routing update overhead. It combines the features of FSR and LANMAR routing algorithm (Hong, 2000). Also, Core-Extraction Distributed Ad Hoc Routing (CEDAR), and Optimised Link State Routing protocol (OLSR) (Jacquet, 1998), are presented as non-uniform proactive routing protocols as well. CEDAR (Sinha, 1999) is a hierarchical routing approach. A set of nodes called the core tries to maintain stable high-bandwidth links. The selection of routes is done with the consideration of the quality of service a link could provide.

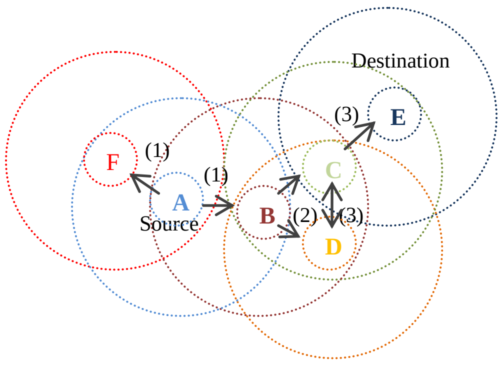
2.2.7.2 Reactive Routing Protocols

In this type of protocols, the data transmission is on demand based. Therefore, they are called on demand or source initiated (Xiang, 2008; Taruna, 2011). These protocols are not continuously affected by the topology changes as the proactive are, so the network is silent until data transmission is needed. Therefore, they dramatically reduce routing overhead, leaving more network resources available for other network traffic.

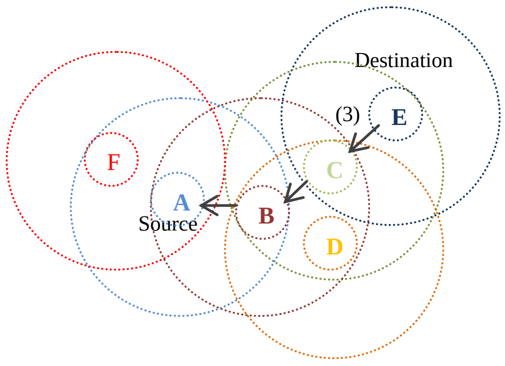
The node creates routes when it is explicitly desired to forward packets, by initiating and flooding the network with route request packets using the route discovery mechanism, where any node that receives that request replies it if it has routing information regarding it. Otherwise, it rebroadcasts the same route request again. This process continues until the route is found or all possible routes have been examined. For this reason reactive protocols suffer from high latencies in route discovery, and route look-ups could take some time. The next sub-sections are a few examples under this category:

Ad Hoc On Demand Distance Vector (AODV) routing protocol, is the most well known reactive protocol for MANETs by Perkins and Das (Perkins, 2001). It is a multicast and unicast routing protocol based on DSDV, introduced in 1997. AODV is not continuously affected by the topology changes, for this reason it has less overhead than the proactive protocols. The network is mainly silent in case of AODV, unless a source node needs to establish a connection with another destination node in the network, at which time it creates and broadcasts a route request or RREQ marked with the requested destination address. This process is called *Route discovery*, as shown in Figure 2.6. (Xiang, 2008; Taruna, 2011).

The RREQ message is forwarded by other intermediate AODV nodes in the network. As this message is spread through the network, each node that receives it sets up a reverse route or a route towards the requester node. As soon as the RREQ reaches an intermediate node that already has a fresh enough route to the specified destination, or the destination itself. The node sends a route reply RREP unicast message backwards to the requester node once such a message has been received. Intermediate nodes use the reverse routes created earlier for forwarding RREP message. AODV such as all the reactive protocols has high latencies in route discovery. AODV utilises three types of routing messages: Route Requests (RREQs), Route Replies (RREPs) and Route Errors (RERRs).



A. Flooding in AODV



B. RREP Establishment

Figure 2.6: Route discovery process in AODV, based on Perkins and Royer 2001

Since AODV is based on DSDV, it ensures loop freedom by using sequence numbers. The sequence number is generated by a destination node and included in the route request or route reply sent to desired nodes. It is also used by other nodes to determine stale routing information. In AODV, each node has its own routing table to save routing information for only those nodes it has already communicated with; while in DSDV the routing table saves routing information for all destinations in the entire network.

During packet transmission, the AODV source nodes always select the route with greatest sequence number and the least number of hops as the DSDV nodes do. In case of link failure, a list of unreachable destinations is put into a routing error message RERR and passed to the neighbouring nodes (*precursors*) that are likely to use the current node as their next hop towards those destinations. For this reason, nodes maintain a precursor list for each routing table entry. Figure 2.7 illustrates the route discovery process and route error. Routes are only kept as long as they are needed. If a route is not used for a certain period of time, its corresponding entry in the routing table is invalidated and subsequently deleted (Xiang, 2008).

AODV is currently one of the most popular Ad Hoc routing protocols and has enjoyed numerous reviews including Broch (1998), Johansson (1999) and Larsson (1998).

Furthermore, several independent AODV implementations exist, such as AODV-UU (Wiberg, 2002).

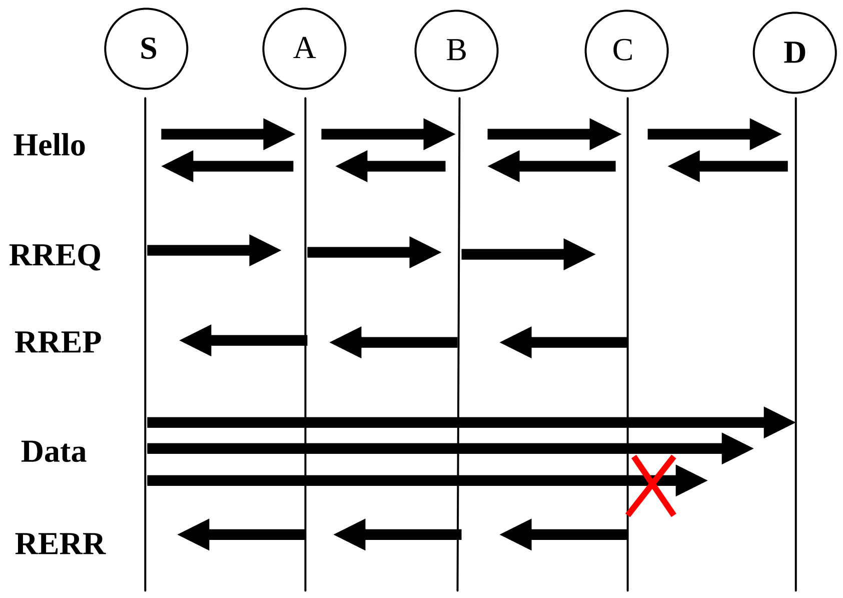


Figure 2.7: AODV route discovery and protocol messaging

Dynamic Source Routing (**DSR**) which is also a reactive routing protocol, was introduced in 1996 by Johnson and Maltz (Maltz, 1996). To reduce overhead, DSR only acquires routes when needed. It is also a beacon-less protocol that means it does not use periodic table-update messages to manage or keep an up-to-date view over the entire network like proactive protocols do. It also differs from AODV and reacts very quickly to any change in the network topology.

DSR is Link State based Algorithm, where each DSR source node is capable of keeping the best route to a destination node. The protocol composed of two mechanisms, route discovery and route maintenance, when data packet transmission needs to take place from a source node to a destination, the source node checks its route cache. If no route is available to that destination, the source node broadcasts RREQ that initiates a route discovery process similar to AODV. The only difference is that DSR intermediate node appends its own addresses before forwarding to RREQ if does not have a valid route to that destination in its route cache as shown in Figure 2.7 (Xiang, 2008).

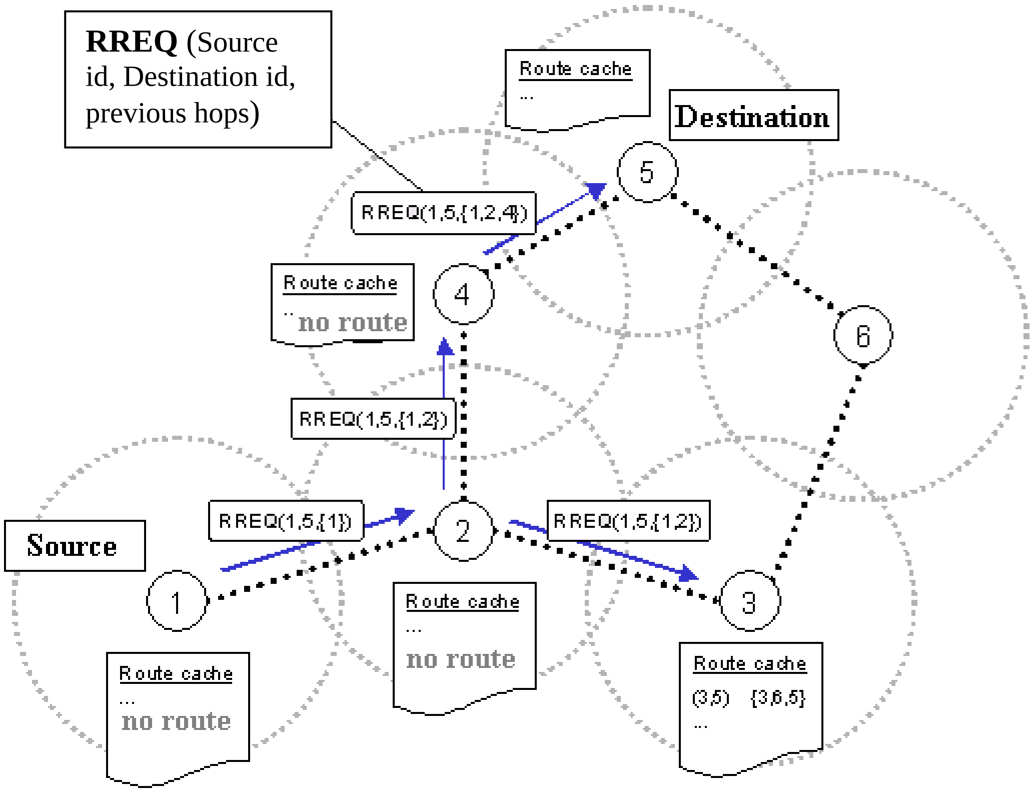


Figure 2.8: Route discovery process in DSR- (A) RREQ

When the RREQ reaches a node has an available route (node 3 for example) or the requested destination itself (node 5) as shown in the route discovery process in Figure 2.8. The node sends an RREP back to the source node using the available routing information (the reverse route included in RREQ) as shown in Figure 2.9. The RREP contains the requested route to the destination node.

In case of link failure occurring during the data packet transmission, the RERR is sent back to the source node such as AODV, where each node is responsible for confirming that the next hop has received the transmitted packet. This stage is called route maintenance. In general, due to route discovery and maintenance, DSR exhibits high overhead especially in high traffic networks (Xiang, 2008).

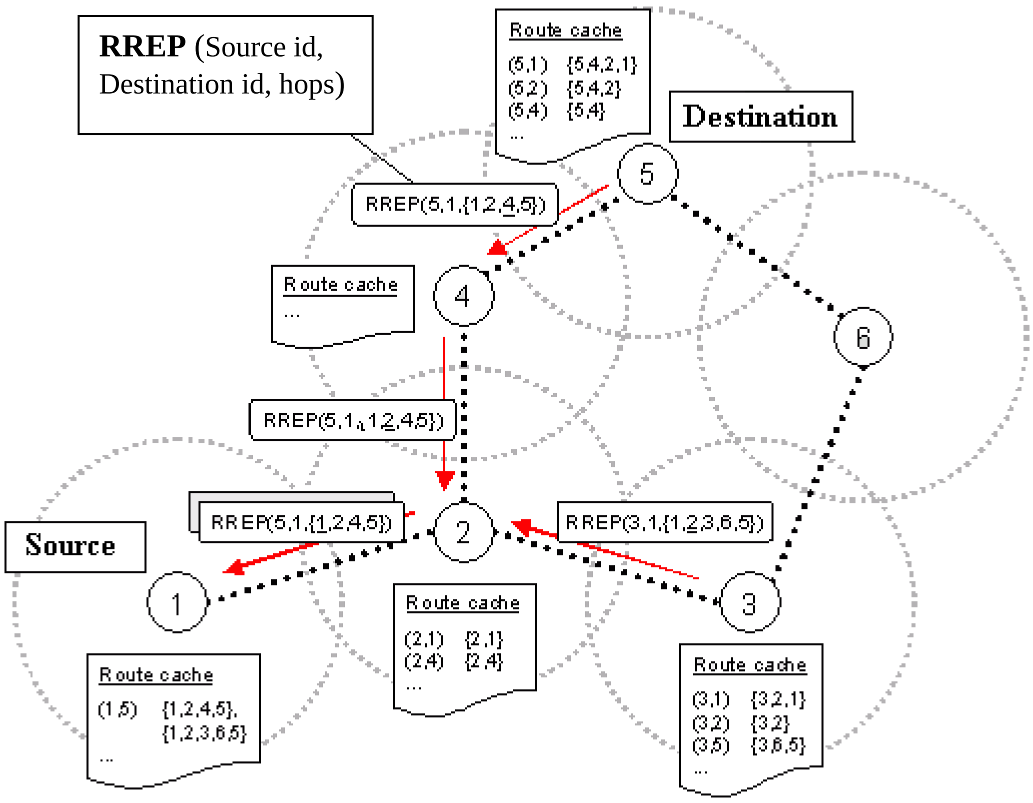


Figure 2.9: Route discovery process in DSR- (B) RREP

In (Zafar, 2007) a Shortest Multipath Source Routing (SMS) protocol is presented as a multipath extension to DSR for real time data and multimedia applications. SMS is based on multiple partial-disjoint paths from a source to a specific destination node. We presented this protocol in more details in related work section.

Many other reactive routing protocols are presented for MANET’s such as LAR, ABR, and SSR/SSA. Location-Aided Routing Protocol (LAR) is a location based on-demand routing protocol. It is designed to reduce the overhead routing, using the location information it derives from GPS (Global Positioning System). Instead of using flooding technique to obtain a route to a destination as the case in AODV and DSR, LAR sends the route request only into the destination’s area (Ko, 1998).

Associatively Based Routing (ABR) is a source initiated routing protocol that eliminates the need to update the routes periodically. It uses flooding of route request messages as the case in AODV and DSR to obtain a route to a destination. Two advantages of this protocol are: stable routes have a higher preference compared to shorter routes; Also, it repairs the broken link locally, so the source node doesn’t need to generate route request (finding-process) when discovers a broken link. Unfortunately, it’s preference for stable paths sometimes leads longer than the shortest path, may result in high delays during the packets delivery (Toh, 2001; Murthy, 2004).

In addition, Signal Stability-based adaptive Routing Protocol (SSR/SSA) is a proactive on-demand routing protocol as well, it is based on link-stability, which means that the routes between the nodes selection is based on the signal strength. This technique has effect of choosing routes that have stronger connectivity (Dube, 1997).

2.2.7.3 Hybrid Routing Protocols

These types of protocols combine both proactive and reactive approach (Boukerche, 2009). They use the proactive approach to determine the best routes to the destination node, and only report the routing information if there is a change in the network topology as the case of the link state approach. Since these types of protocols combine the reactive and proactive approaches, they also carry the problems associated with them. There is a number of hybrid routing protocols such as: Zone Routing Protocol (ZRP), Hybrid Routing Protocol for Large Scale mobile Ad Hoc networks with mobile backbones (HRPLS) and The Temporally-Ordered Routing Algorithm (TORA) (Yang, 2002; Taruna, 2011).

Zone Routing Protocol (**ZRP**) was the first hybrid routing protocol introduced in 1997 by Haas and Pearlman (Haas, 1997), combining the advantage of proactive and reactive protocols. It is based on zones concept (Clustering) as its name implies. It divides the network into a couple of routing zones to reduce the control overhead of a proactive approach, and decreases latency caused by the flooding technique that is used in route discovery in a reactive approach. Zone is a group of neighbours around a node. Figure 2.10 illustrates the routing zone of node S surrounded by its neighbours, where distance (zone radius) is 2. The nodes that lie inside the routing zone, where the distance from S is less than zone radius (1 hop) called interior nods (A, B, C, E, and F). The nodes that lie at distance exactly equal to zone radius (2 hops) such as G, H, I, and J are called peripherals. Each node may be within more than one routing zone (multiple overlapping zones). Node K is out of the node S routing zone, because the shortest distance from S to K is greater than zone radius (3 hops).

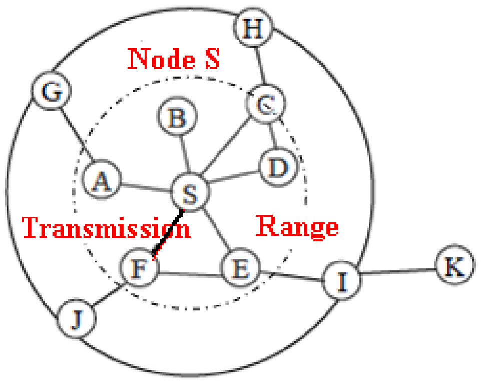


Figure 2.10: Routing zone where zone radius = 2

The number of routing zones in each network depends on the assumed zone radius or distances expressed in number of hops. The routing zone size is not a physical or geographical measurement. It is a radius length in number of hops. The routing zone size is also affected by transmission range also. Increasing the transmission power increases the number of nodes in the routing zone, and decreasing the transmission power decreases the number of nodes in the routing zone.

As we mentioned, the ZRP combines two sub-protocols, a proactive protocol for local routing between nodes within routing zone called IntrA-zone routing protocol (IARP) (Haas, 2001), and reactive protocol for global routing between zones called IntEr-zone (Haas, 2001) routing protocol (IERP).

Local neighbours are detected using Neighbour Detecting Protocol (NDP). That means, if both the source node and destination are in the same routing zone then the route is already established and must already be in the source node routing table. Hence the packet is delivered immediately. For destinations beyond the local routing, the nodes outside the source node’s routing zone, the reactive approach takes place and establishes route discovery technique by sending RREQ only to its border neighbours (Peripherals) using *Bordercasting* by BRP (Broadcasting routing protocol) instead of broadcasting which sends it to all neighbours.

The RREQ packet is forwarded in same context of bordercasting, till it reaches a node the requested destination is a member of its routing zone, and then sends an RREP back to the source node. The source node uses the routing information (path) saved in the RREP to send the data packets to the destination node.

The main advantage of ZRP is less control overhead than both proactive and reactive protocols. Furthermore, some routing protocols have no hierarchy in the treatment of the network nodes, where all the nodes deal with routing information in same manner, these routing protocols called uniform protocols (Kuosmanen, 2002; Jayakumar, 2007). Other routing protocols called non-uniform routing protocols treat the network nodes in a different manner and in hierarchical form by clustering or partitioning the network nodes in dealing with control messages. Table 2.2 shows some MANET’s protocols classified according to node uniformity.

Table 2.2: MANET Routing Protocols Classifications (Source MiNEMA)

Uniform routing	Proactive routing	Wireless Routing Protocol (WRP)	
		Destination Sequence Distance Vector (DSDV) routing protocol	
		Fisheye State Routing (FSR)	
		Distance Routing Effect Algo. for Mobility (DREAM)	Location-based routing
	Reactive routing	Dynamic Source Routing (DSR) protocol	
		Temporally-Ordered Routing Algorithm (TORA)	
		Ad Hoc On-demand Distance Vector Routing (AODV)	
		Location Aided Routing (LAR)	Location-based routing
		Associativity Based Routing (ABR) protocol	Link-stability based routing protocol
		Signal Stability-base adaptive Routing (SSR)	Link-stability based routing protocol
Non-Uniform	Zone-based routing	Zone Routing Protocol (ZRP)	Hybrid routing protocol
		Hybrid Ad hoc Routing Protocol (HARP)	Hybrid routing protocol
		Zone-based Hierarchical Link State routing (ZHLS)	Hybrid routing protocol
		Grid Location Service (GLS)	Location service
	Cluster-based routing	Clusterhead Gateway Switch Routing (CGSR)	
		Hierarchical State Routing (HSR)	
		Cluster Based Routing Protocol (CBRP)	
	Core-node based routing	Landmark Ad Hoc Routing (LANMAR)	Proactive routing
		Core-Extraction Distributed Ad Hoc Routing (CEDAR)	Proactive routing
		Optimised Link State Routing protocol (OLSR)	Proactive routing

2.2.8 MANET's Models

In this section, and its sub-sections, we will discuss the primary routing and communication modules used in MANETS. These modules include path selection methods, types of data packet forwarding, communication channels, and techniques of control packets broadcasting.

2.2.8.1 Route Selection Models

Providing the right algorithms to find and select the most effective routes, after the information caching stage for the data packet journey from the destination node to the source, is another stage of MANET's routing protocols. Sometimes, they are called route determination models. Those models need to read and understand the structure and standard of routes databases and to perform successful calculations for getting effective routes. Most single path routing protocols select and store the best route into the source node routing table to the requested destination node, where most multi-path routing protocols select and store more routes for backup purposes. The routing protocols can be classified based on the route determination into the following types:

- ❑ **Signal Strength**: It refers to the radio signal propagation magnitude, or signal's electric field at a reference point. The performance of wireless network and network's bandwidth total amount depends on signal strength between nodes of the network (Route life-time), as the signal is weak as the route life-time is short. The accurate packets deliver over the best signal strength route (Agarwal, 2000). As we mentioned previously, this model has been used in Signal Stability-base Adaptive Routing protocol (SSR/SSA) (Dube, 1997).

- ❑ **Link Stability:** Associatively Based Routing ABR (Toh, 1997) protocol is based on link stability as shown in Table 2.2. The strength of the weakest link in a route, informs how stable that route is. Quality and accuracy of data packets along a route depend on the stability of that route (Nandi, 2007).

- ❑ **Shortest Path / Link State:** Most of the proactive routing protocols are based on this metric, where the shortest path algorithm is used to provide the optimum paths, such as Wireless Routing Protocol (WRP) and Optimised Link State Routing protocol OLSR (Clausen, 2003) as classified in Table 2.2.

- **Distance Vector:** Such protocols select the optimum routes based on the number of hops metric (distance) form the source node to the destination. For example, Destination Sequence Distance Vector DSDV routing protocol (Perkins, 1994), Distance Routing Effect Algorithm for Mobility DREAM (Basagni, 1998), and On-demand Distance Vector Routing AODV (Perkins, and Royer 2001) are all classified under this criteria.

- **Directional Routing:** Routing this type of protocol is based on the number of hops metric (distance) and location. Where routing information is less updated for nodes with a slow movement than nodes with high mobility. Location Aided Routing LAR (Young-Bae, 2000), and Distance Routing Effect Algorithm for Mobility DREAM (Basagni, 1998), are examples of directional routing.

2.2.8.2 Data forwarding Models

Routing of data packets in MANET depends on the routing information available at the source and intermediate nodes. Therefore, we can also classify the routing algorithms based on the data packets forwarding techniques as follow:

- **Single Path:** Most of the MANET protocols are *single path*, where each source node uses a single path to each destination to forward the data packets. DSDV, AODV and DSR are examples of single path routing protocols (Mueller, 2004).

- **Multi-Path:** Multi-path routing is a way of improving the reliability of the transmitted information. Where the routing algorithms provide more than one route (Multipath) between a single source and a single destination node. The advantage of multi-path routing mainly depends on the disjoint paths from a source to destination availability. There are two types of disjoint paths; node *disjoint* and *link disjoint*. A node disjoint path is at path (route) which does not have any nodes in common with another path, except for the source and the destination. While the link disjoint paths do not have any common links, but may have common nodes (Tachtatzis, 2008; Natarajan, 2010).

2.2.8.3 Broadcasting Models

Packets exchange between the nodes is the main purpose of any network. The same message may be sent by a single source to a single destination, multiple recipients or to a specific group, such the case in the Internet applications (Email). Therefore, we can classify the broadcasting models in MANET into the following types (Roy, 2011):

- **Broadcast:** Communication is established between a single source and all the nodes in its transmission range (neighbours).
- **Unicast:** Communication is established between a single source and single destination.
- **Border-cast:** Communication is established between a single source and its peripheral nodes (nodes lie in the edge of the routing zone). This is used by ZRP routing protocol.
- **Multi-cast:** Communication is established between a single source and a list of selected recipients.

2.2.8.4 Mobility models

Mobility models represent the way in which the mobile users may move from place to another. In MANET the mobile user movement can take more than one form in different speeds over time, where they can move freely within the field in random directions. Such models are frequently used for simulation purposes. To test and evaluate a protocol performance for MANET, it is important to test it under realistic conditions in different scenarios including nodes mobility model and traffic load. A Survey of Mobility Models for Ad Hoc Network Research by Tracy Camp and others (Camp, 2002; Roy, 2011) includes all the mobility models used in MANET. There are several mobility models; in the next sub-sections we are going to focus and discuss only the most commonly used models.

- ❑ **Random Waypoint Mobility Model (RWP):** This is the mobility model used in this research. It is a normally used as a simulation tool for mobility in wireless networks. It was proposed by Johnson and Maltz (Maltz, 1996). It has become a benchmark model to evaluate the MANET routing protocols. According to some surveys it is used in 60% of simulation experiments.

  It is based on random movement and several speeds over the time, where a mobile node remains fixed for an interval of time called thinking time or Pause Time [Minimum pause time, Maximum time], then chooses a random destination from its location and starts its journey towards it with a randomly selected speed, uniformly distributed between [0- Maximum Speed] m/sec.

  Each time the node reaches the destination location, the pause process will be repeated (Ganapathi, 2008; Hyytia, 2005).

- ❑ **Random Walk Mobility Model (RWM):** A mobile node in this model walks in random directions between 0 and 2π [0-180], and random speeds [0, Vmax] from its current location to a new location. Specifying a short time or distance that means the mobile nodes are only permitted to walk in a very restricted simulation area within the simulation time.

Each node movement occurs in either a constant interval of time or a constant distance. Once the mobile node reaches its desirable location (destination) a new direction and speed are recalculated after a given time or distance walked. A mobile node's new directions and speeds in a random waypoint model are independent of the previous direction and speed information.

Unusual movements such as sharp turns or sudden stops can be generated because its mobility pattern is a memory-less, where the information about the previous movement is not used for the next (future) movement as we mentioned previously. We can say a mobile node moves with a zero pause time in RWP model is same as in a Random Walk model (Camp, 2002).

- **Random Direction Mobility Model (RDM):** This model was presented to reduce the high probability of density waves by the RWP. Density waves are the nodes' gathering (caching) and deployment in part of the simulation area or its centre. A mobile node movement in this model is similar to the RWM, which chooses a random direction, and then travels to the border of the simulation area in that direction. Once it reaches that boundary, it pauses for a specified time, and then chooses a new direction between 0 and 2π [0-180] and the process continues (Gloss, 2005). Random Direction Mobility Model (RDM) has a much higher hop count than most other models because of pause time at the borders of simulation area.

Furthermore, there are other mobility models in use, but not mentioned or discussed previously in this thesis, such as Random drunken (RD), and Trace based (Vetriselvi, 2007). In the Random drunken model a mobile node, randomly chooses a changeable direction after every unit of distance, and moves independently towards it, with the same average speed for a certain time, continuously within the topology without pausing (Wu, 2006). The RD model is not used to design or configure realistic scenarios, but used as a template for the development of new mobility models. While, a mobile node in a Trace based model moves according to the mobility specification provided by the user.

2.2.8.5 Radio Propagation Models

Characteristics of propagation are different from one model to another, and depend on many factors. For an example, signal power, this differs from packet to another. Each mobile node's physical layer contains a receiving threshold. If a received packet's signal power is below the receiving threshold, that packet is marked as an Error and neglected (dropped) by the MAC layer (Kevin, 2010; Roy, 2011). Therefore, propagation models are important in the planning process, they are used to predict the path loos along a distance and an antenna effective coverage area (Itoua, 2008). There are three radio propagation models implemented in ns-2; the *free space* model, the *two-ray ground reflection* model and the *shadowing* model. Originally, all these models come from the domains of radio engineering and physics (Wiberg, 2002).

- ❑ **Free Space:** This propagation model is ideal for clear line-of-sight path between transmitter and receiver (Itoua, 2008; Kevin, 2010).

- ❑ **Two-Ray Ground Model:** This is the model that has been used in this research. It is more accurate than the free space model, especially over long distances, where it considers the path between transmitter and receiver for both cases in ground reflected, and direct propagation, while it does not provide guaranteed results in short distances (Itoua, 2008; Kevin, 2010).

- ❑ **The Shadowing Model:** In reality, the received power as a deterministic function of distance is not ideal circle as the above two models are predicted, but due to multipath propagation effects, it is at certain distance a random variable (Kevin, 2010). The shadowing model, attempts to more realistically model multi-path propagation effects, in other words fading. This model has two parts; a path loss model, which predicts the mean received signal power at the distance from the transmitter, and a log-normal random variable, which models probabilistic communication between nodes at the edge of the radio range (Wiberg, 2002; Itoua, 2008).

Many other models are presented but not implemented in Ns2 simulator, these models such as Terrain, City and Band-specific models. The Terrain models are mainly used for getting a fast overview over a landscape. John Egli has introduced one of the terrain models in (Egli, 1957), Egli's model is typically suitable when one of the receivers is fixed (antenna) and another is mobile (cellular communication) (Elsallabi, 2007). Longley–Rice (LR) and ITU are two other examples of terrain models. The LR was created for the needs of frequency planning in television broadcasting, it is an example of free-space transmission for frequencies between 20 MHz and 40 GHz, it covers area where its path lengths between 1 - 2000 km(Rice, 1967). ITU is another example of terrain models (an example of line-of-sight propagation models); it predicts the path loss as a function of the height of path blockage and the First Fresnel zone (electrodynamics, acoustics, and gravitational radiation) for the transmission link.

The Hata and Okumura are two examples of the city models for Urban Areas. The Okumura is typically ideal for using in cities with many urban structures but not many tall blocking structures, this model is not typically ideal for modern US cities because of the high towers buildings this model has been presented based on measurements made in the city of Tokyo, Japan to determine the median field strength and numerous correction factors. This model supports frequencies between 150MHz to 1920 MHz, it covers area between 1 - 100 km. The Hata model is an extension (developed) version of the Okumura Model, it coverage frequencies between 1500 to 2000 MHz (Seybold, 2005).

2.3 Simulation Overview

A Networks simulation concept is based on the modelling of real world networks on a computer screen, using computer software designed and implemented for this purpose, such as Network Simulator 2, which has been used in this research. It is intended by the real networks, the networks that are implemented on present, or the networks that wish to implement in future. Computer simulations have many benefits, for example: as long as these networks can be modelled using the computer simulators, it is possible to change and easily control all their characteristics, and all related components. Furthermore, a simulation and modelling process, using computer software, is a low-cost alternative when compared to the real implementation without a prior study. Particularly taking into account all the tests that may have to be carried out and then repeated again, parts and equipment which may be changed each time to reach the objective.

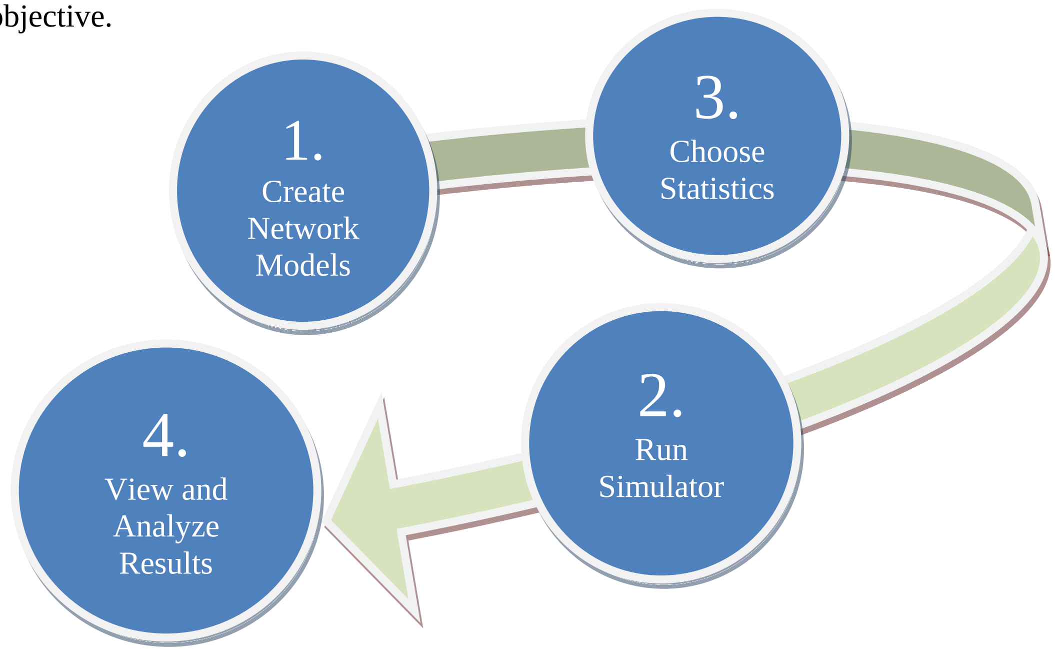


Figure 2.11: Simulation and modelling cycle

In general, to simulate any model, it has to pass through four phases, called the modelling cycle, as shown in Figure 2.11. Creating the network, choosing its components and standards such as, network size, topology, protocol type, and its standard IEEE 802.11, IEEE 802.11a, or IEEE 802.11b for an example, also the type of modulation method, whether we will use the option of RTC / CTS or not, or the option of Fragmentation, and medium access mechanism CASMA/CD (Roy, 2011), CASMA/CA, PCF, or DCF, are carried in the first phase.

In the second phase, we choose the metrics, based on how we wish to evaluate the network performance, such as Throughput, Delay … etc. Where in the third phase, we set up the simulation time duration, which we simulated the network, for example, for a period of an hour a day, for a week or a month ... etc., all that depends on the network type, characteristics, and purpose.

For example, if the simulation program was run on the basis of one day, all the results we get are the modelling of the network for one day in real life, the operation of the network in reality for a one-day, may be offset by the simulation program for five minutes in the fourth and final stage, we receive the simulation results in a file for viewing, and we use our program part to analyse those results (Black, 2009).

There is now the question of what the accuracy and validity of the results that obtained from simulation are. In fact, there is no simulator that gives 100% guaranteed and perfect results, although there are differences between the simulators, they do try to provide as accurate as possible results. The accuracy of these results depends on several factors, the most important two are:

First, the human factor, i.e., the extent of the skills of the person who is using the simulator, this includes simulator knowledge and the model subject concept in general, computer networks for example. Second the efficiency of the simulator itself.

2.3.1 Computer Network Simulators

A Simulation Modelling is a popular method for network research, and performance analysis. There are several simulators for Computer networks in use, some of them are free, and known as Open Source, while some others are commercial. Here are some examples of network simulator in general:

Network Simulator 2 (Kevin, 2010), OPNET IT (OPNET, 2007), Scalable Simulation Framework (SSFNet, 1998-1999), OMNeT++ (OMNeT++, 2009; Varga, 2001), J-Sim (Barnett Iii, 1993), and last but not the least OPNET Modeler (Chang, 1999).

2.3.2 Network Simulation 2 (Ns2)

Ns2 (Kevin, 2010), is an open source simulator, more general and widely-used in researches and performance evaluations of wired and wireless (local and satellite) networks (e.g. MANET). It is a discrete event simulator, written in C++ and OTcl (OTcl, 1996) based on object oriented concept and design, which was developed as a part of VINT (VINT, 1996). It is a DARPA-funded research project, whose aim is to provide a reliable network simulator for scale and protocol interaction study based on the present and future network and protocols. It is supports IP, TCP, UDP, routing, multicast protocols simulation, QoS mechanisms, and more.

MANETs area of interest is received an extensive study and development over Ns2, where additional set of software models are provided for wireless networks in Ns2. The Monarch project at Carnegie Mellon University (CMU) is an extension of Ns2 (Johnson, 1999). CMU has provided new components and elements to the wireless networks (Wireless extensions) at the physical layer (Radio propagation models, Antennas and Network Interfaces), link layer (Media Access Control Protocols), network (routing) layers of the simulation environment, and Scenario Creation. Figure 2.12 shows the flow diagram of developing and testing MANETs routing protocols in Ns2. CMU model allows simulation of pure wireless LANs or MANET networks. Mobile scenarios and traffic pattern generation for MANETs are very simple in Ns2 using Mobility scenario, and Traffic pattern generators by CMU wireless extension.

2.3.2.1 Packets

A simulation fundamental unit that is used for information exchange between its objects is called a *Packet*. It is built up of packet headers and packet data. The packet headers contain many fields. Their size and number are different from a protocol to another see section 4.5 for further details on packet structure.

Accessing packet data and headers is made available through access methods (Nikos, 2010; Kevin, 2010). Figure 4.6 shows our new routing protocol’s packet structure, where its own packet header types are added to the available ones. For further information regarding to the new routing protocol packet types and their structure see section 4.6.

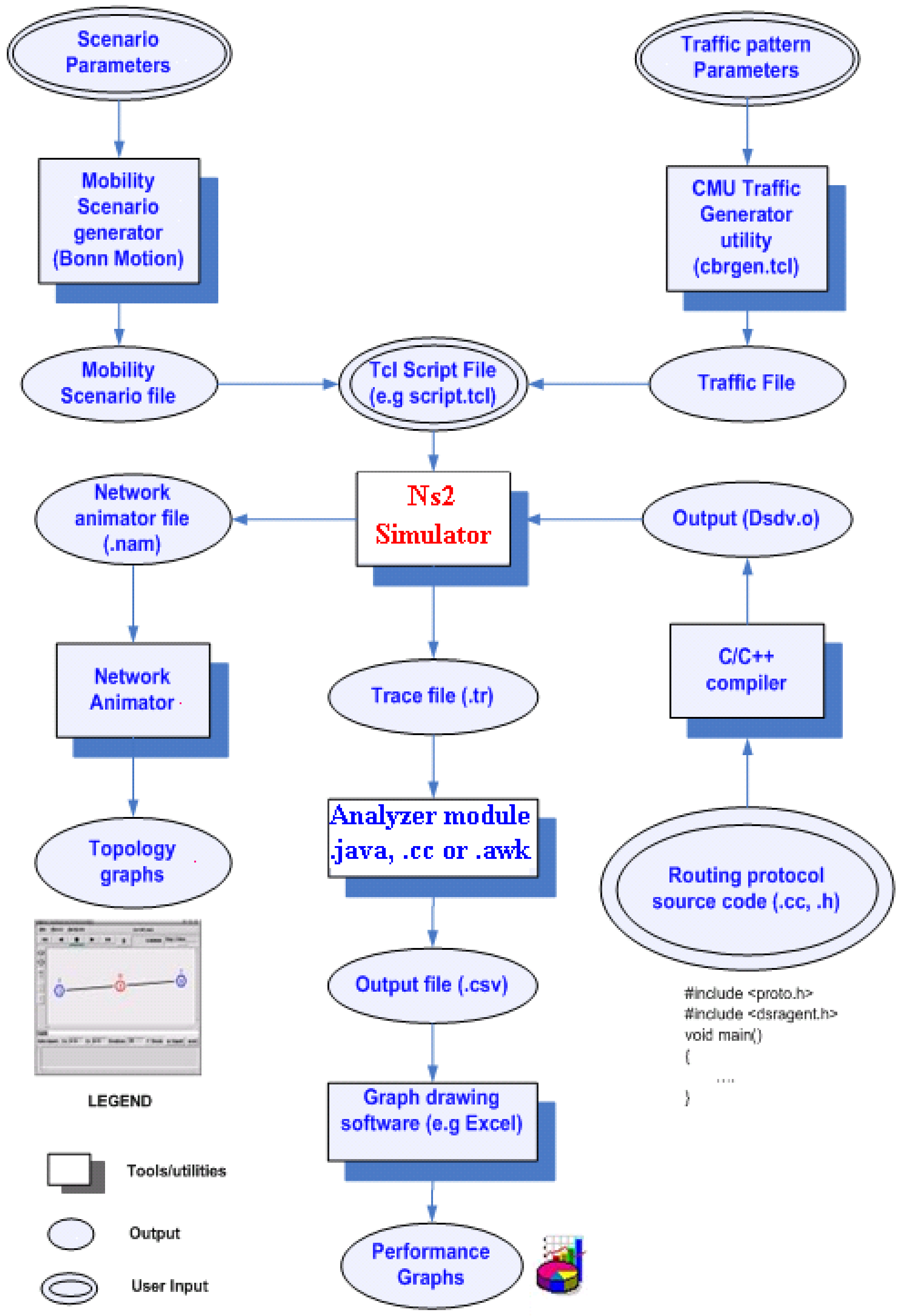

Scenario Parameters
Traffic pattern Parameters
Mobility Scenario generator (Bonn Motion)
CMU Traffic Generator utility (cbrgen.tcl)
Mobility Scenario file
Tcl Script File (e.g script.tcl)
Traffic File
Network animator file (.nam)
Ns2 Simulator
Output (Dsdv.o)
Network Animator
Trace file (.tr)
C/C++ compiler
Topology graphs
Analyzer module .java, .cc or .awk
Routing protocol source code (.cc, .h)
Output file (.csv)
#include <proto.h>
#include <dsragent.h>
void main()
{
....
}
LEGEND
Graph drawing software (e.g Excel)
Tools/utilities
Output
User Input
Performance Graphs

Figure 2.12: MANET's routing protocols flow diagram in Ns2 Simulator

2.3.2.2 Mobile Networking in Ns2

It is essential to include a mobile node in any wireless model, additional to a number of simulation features that support MANETs, and wireless local area networks. Mobile nodes connected to wireless channels concept is introduced in CMU wireless extensions, which allow for MANET and wireless networks simulation (Kevin, 2010).

The node object is the most important entity among all the Ns2's network components, where it is responsible for most of the packet processes such as creating, sending, and forwarding, also, receiving and reading the packet's headers. Therefore, it is fundamental in all network simulators. There are two types of nodes in Ns2, a *mobile* node, which is an extension of *wired unicast* node.

A *wired node* (Wiberg, 2002; Kevin, 2010) is a compound object, consisting of a node entry object, where packets first arrive, and two classifiers as shown in Figure 2.13. The first one, called the address classifier, is responsible for packet's address identification, where it examines its address and then decides whether the packet belongs to the current node or not. The second classifier is the port classifier, it determines which one of that node's protocols should receive and deal with that packet.

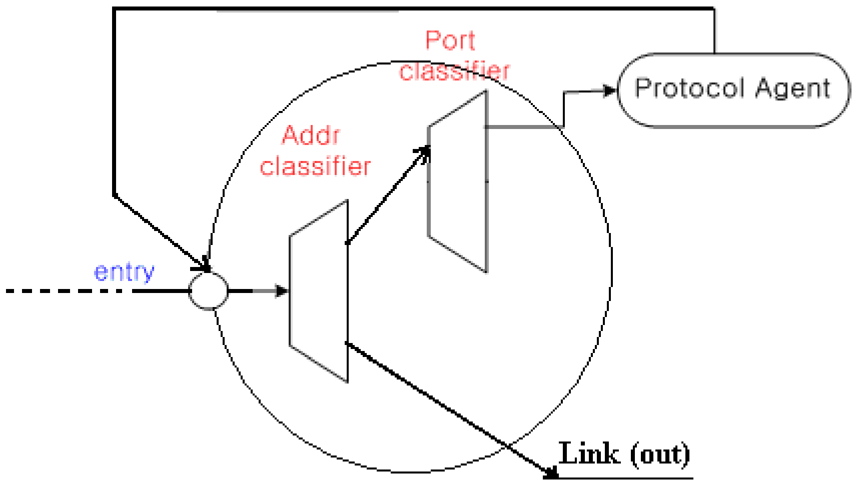


Figure 2.13: Composite construction of wired unicast node, based on Kevin, 2010

A *mobile* node (Kevin, 2010) is a wired node with extra functions to perform mobile network tasks (mobility). The main differences between the two nodes object are: regarding the communication media, the mobile node is connected to wireless channels, whereas the wired node is connected by links. Also, regarding the mobility concept, the mobile nodes may move within a specific topological area, while the wired nodes are mainly fixed (remain stationary). Figure 2.14 shows the schematics of a mobile node in Ns2.

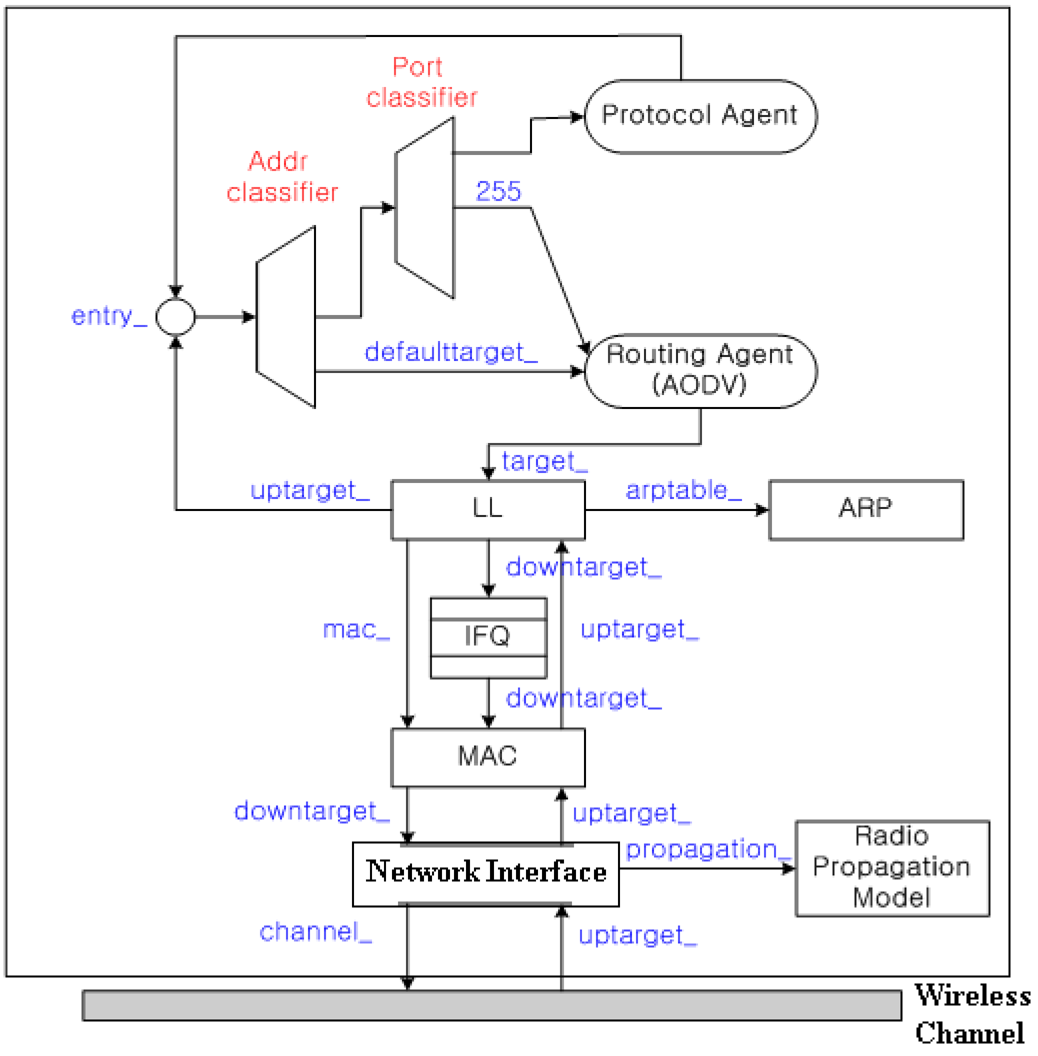


Figure 2.14: Mobile node schematics, based on (Wiberg, 2002; Kevin, 2010)

The following models, agents and functions are the main parts of the mobile node compound object.

- An *address classifier*, the received packet handler, which decides whether to forward the packet to the port classifier, or to the default target (routing agent).
- A *port classifier*, the packet handler for the mobile node's attached agents.
- A *routing agent*, for packet forwarding, which sets the next hop field of a packet indicates its next hop towards the requested destination using routing table.
- A *link layer*, used for converting node's network address to a hardware address, with help of *address resolution protocol* (ARP), which responsible for resolving

network address to physical address (MAC).

- An *interface queue*, used for sorting the outgoing packets.
- A *MAC* layer used for providing control mechanisms for addressing and media (wireless channel) access.
- A *network interface* connects a node to the wireless channel over which it can send and receive packets.
- A *radio propagation model* used for determining whether the network interface can receive that packet or not using the packet's signal strength.
- A *wireless channel* over which the node can transmit and receive packets.

2.3.2.3 Wireless Links

MANETs also realised by wireless links techniques such as IEEE 802.11, Bluetooth, and Ultra-Wide Band (UWB) (Green, 2004; Bluetooth, 2007). However, each one of these communication technologies poses various challenges in its algorithm design. Ns2 provides a compound object to perform a connection between two nodes, called link. It supports the two types of links, a simplex *Unidirectional* and duplex *Bidirectional*. A duplex link is simply two simplex links in two directions as shown in Figure 2.15. The queue object is to save the sent packet in queued event and before pass to the delay object in de-queued event. Simulating a packet drop is presented by sending it to a null agent from the queue object, while simulating a link delay is performed by a delay object.

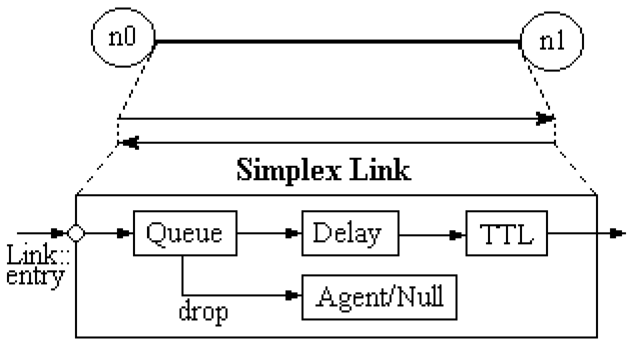


Figure 2.15: Simplex Link

A node that wishes to establish a connection simply puts a packet in the queue object of the link. Hence, if not dropped by passing it to null agent, it will pass to the delay of the link object. Each received packet time to live be calculated and its TTL field is updated at TTL object of the link.

2.3.2.4 SYMMTRIC AND ASYMMTRIC NETWORKS

In a symmetric computer network, all nodes can transmit and receive data at equal rates. Asymmetric networks, on the other hand, support disproportionately more bandwidth in one direction than the other. This can be a problem in wireless networks which adopt a TCP technique where TCP relies on ACKs for reliable delivery and for congestion control. If ACKs are not reliably returned the smooth of packets will be disrupted by retransmissions. Most of MANET protocols have been designed assuming that the underlying technology was bidirectional (Symmetrical Network). As an example, a set of nodes which are connected through a single physical network assume they can exchange routing information with each other as shown in Figure 2.16. Exchanging routing information enables the discovery of the underlying network topology, and the routing traffic via discovered networks (Skloul, 2008).

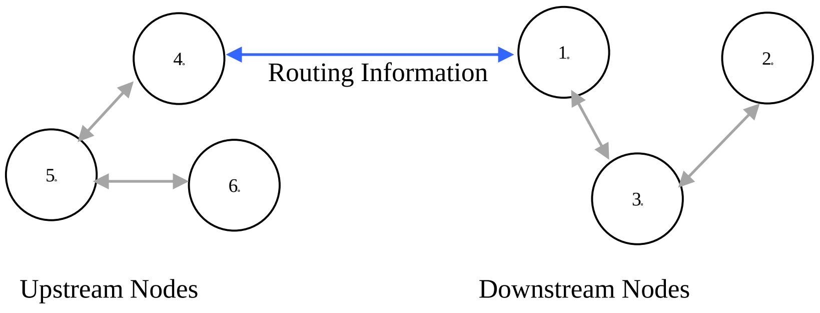


Figure 2.16: Symmetric Network

However, if the link connecting these nodes is unidirectional (Asymmetrical Network), we can say that all downstream nodes have received only capabilities and therefore cannot send routing information to upstream nodes as shown in Figure 2.17. As a result, upstream nodes cannot discover downstream network topologies dynamically and will therefore never forward information towards them.

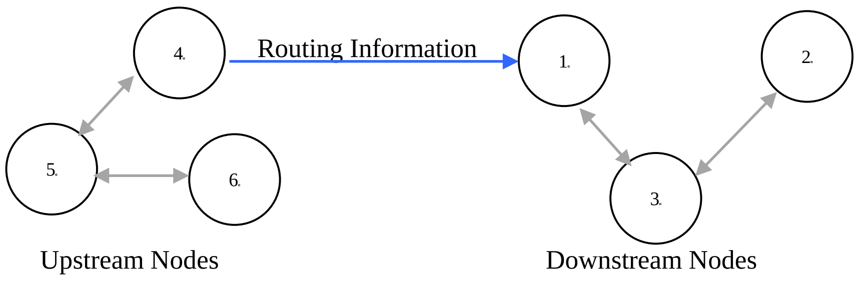


Figure 2.17: Asymmetric Network

Generally, in the presence of a unidirectional link, many routing protocols, will fail to operate and lose to send data therefore, to provide full network connectivity we need to make the node to discover if the link is a bidirectional or unidirectional link before sending over any data.

2.3.2.5 Packet Transmission

The packet by mobile node is first generated by an agent or a traffic source. The first entity that receives the generated packet is the address classifier of the mobile node entry as shown in Figure 2.14. Based on the packet's address, the address classifier determines whether to destine the received packet for the current node, or forward it to another mobile node by handing it to the routing agent (a default target) for processing through a default target of the address classifier, where and before passing it down to the link layer, the routing agent, fills in the *next_hop* field of the packet.

At the link layer, the packet's destination address translates into hardware (MAC) address, by initiating the address resolution protocol (ARP) to map IP network addresses to the hardware addresses, and fill in the MAC header of the packet (Fairhurst, 2005; Wiberg, 2002). Hence, the packet should be queued in the interface queue for its transmission turn. The interface queue length can be specified, depending on the queue type used. In this research we specified the interface queue for 50 packets length. Packets are retrieved from the interface queue by the MAC layer when appropriate, i.e., when the wireless channel is free to use. Finally, the packets are propagated onto the wireless channel once handed to the network interface.

2.3.2.6 Packet Reception

At the packet reception time, not all the packets are necessarily received correctly. It is the radio propagation model's role to decide (i.e., based on the transmission speed and the distance between nodes), whether the packet has been correctly received and should be handed to the MAC layer by the network interface or not. At each wireless node's physical layer, the radio propagation model is used to predict each packet's received signal power (Wiberg, 2002; Nikos, 2010).

In turn, the packet is handed to the link layer by the MAC layer, and hence, to the address classifier, which determines whether the received packet matches the current node address to be handed it to the port classifier, or handed it to a default target for further processing and possible forwarding.

2.3.2.7 Trace Logs

In order to get results from the simulator, we need to figure out what happens exactly during a simulation run time. There are a number of ways to collect data output during the simulation time. The network simulator Ns2 generates event logs called *trace* or *log* files (Kevin, 2010), which can be analysed after a simulation of each scenario (offline). The log files would gather information that could be used in a performance study, which record and provide information for events of packets being sent, received or dropped. This is called the trace of packet. In this thesis we have used these log files to read and gather the information of packets events, and hence, we could provide files contain all the necessary information needed for the analysis and evaluation of our new routing protocol's performance. Ns2 provides three different formats of log file. These formats are:

- ❑ **Old Trace Format**: It is the most commonly used format because of its simplicity, its fields are grouped to provide different information from the packet's fields, and easy to read (Kevin, 2010).

- ❑ **New Trace Format**: This is the trace format that we have used in this thesis. It offers more information than the previous one on each event during the simulation time. Therefore, the length of its lines means it is not as easy to read as the old trace format, which contain a lot of fields and tag – value pairs (Kevin, 2010). Because this format is the one we have used in this thesis, we give an overview of the new wireless trace format in more detail in chapter 5.

- ❑ **Tagged Trace Format**: This trace format is the last version (recently) added to Ns2. It is also contains tags and values as the new trace format do. Tagged trace format analysis is more difficult than in the previous two formats, because the tag names must be determined by each object to be traced. Therefore, tag clashes are expected and likely to occur.

To visualise the simulation results and real world packet traces, the **NAM trace** file is used by the visualisation tool (Nam). NAM is a Tcl/TK based animator used for replying events during simulation time. If the events happen intensively or the simulation time is long the NAM trace file can be huge. Full details are found in (Kevin, 2010).

2.4 Summary

Ad Hoc networks’ simplicity, low cost compared to conventional networks, their special properties, and their applications all make them attractive, but that come at a certain price, especially regarding network security, due to the features (e.g. lack of centralised monitoring and management) of MANETs, they are often vulnerable to security attacks. On the other hand, because of their use in public applications, Ad Hoc networks have the potential to become very useful and popular.

In this chapter various types of routing protocols for MANET are presented, where they gave us evidence that the selection of the optimum routing protocol for a suitable application in MANETs needs more study. Choosing the most suitable protocol for the right application in MANET is like other networks and needs tests and evaluation. Practically, in reality such work is expensive. Therefore, we should carry out such work in simulation time first, using one of the network simulators, such as Ns2 which was briefly described in this chapter.

The network simulator (Ns2) is a product of a hard work of large number of researchers, programmers and users over many years. Its open environment and flexibility has made it one of the best simulators that have been used for researches in networks and their applications. It is also widely used for wireless Ad Hoc networking simulation, because of its support for mobile networking. It is continuously updated, and a new version is released each year with minor or major changes.

CHAPTER 3: PROBLEM & MOTIVATION

3.1 Introduction

Providing a convenient routing protocol for MANETs is a challenge because of its open (no centralization) and dynamic environment. Therefore, the suitability of each routing protocol depends on many parameters such as, network size, node mobility speed, and traffic load. All that, together with the dynamic nature of MANETs, made an optimum routing protocol selection a complicated task. Extensive nodal mobility of MANETs makes multi-hop routing a genuine challenge. The frequent topology changes and variable propagation conditions make a routing table obsolete very quickly, which results in enormous control overhead for route discovery and maintenance. However, MANETs suffer from severe constraints on communication resources (bandwidth, radio propagation, energy supply, interference… etc.) (Blum, 2004; Michel, 2007; Boukerche, 2009).

In some scenarios, route maintenance itself may consume so much in the way of resources that no bandwidth might remain for the transmission of data packets. Even worse, the short lifetime of routing information means that a portion of the information may no longer be useful and thus the bandwidth used to distribute the routing update information could be wasted. Also, that increases delay and overall control packets (overhead). Single path, on-demand routing protocols rely on a uni-path route for each data session. In the case of a failure of an active link between source and destination, the routing protocol must invoke a route discovery process and in so doing, increases delay and overhead. This increase in overhead to deliver a data packet leads to many issues. For an example, power consumption where in some environments the battery charges is a difficult task (battle field, underwater and hazardous areas …etc.). (Michel, 2007; Haseeb, 2007).

This research introduces a new multipath routing protocol based on hybrid mechanisms to reduce the overhead and hence increase battery life. In the investigation of this issue, we pay specific attention to the scalability of algorithms in respect to the network size and nodal mobility (Michel, 2007; Boukerche, 2009).

3.2 Related Work

The first multipath routing was introduced for conventional wired networks for the purpose of load balance and error tolerance (Li, 2005). Multipath routing applications can be classified as either transmitting data packets in multiple paths simultaneously or as using the multiple paths for backup only. Some protocols in MANETs, such as the Dynamic Source Routing (DSR) and the Temporally Ordered Routing Algorithm (TORA), use multiple paths. However, the multiple paths are utilized as a backup or auxiliary method in these protocols (Wu, 2001).Concurrent transmission on multiple paths performs better than the single path transmission in end-to-end delay, and network throughput. Due to these features, multipath routing protocols have been extensively studied in MANETs (Haseeb, 2007).

When multiple paths are used simultaneously, the criteria under which the data packets are assigned to a particular route vary. Commonly, round-robin is used. More complicated route selection technique and rules may be implemented, to achieve a better system performance. Research on multipath routing is mainly focused on the case of disjoint paths. The multipath routing protocols for MANETs have been introduced in 90s. For an example, An On-Demand Multipath Routing protocol (Nasipuri, 1999) is presented as an extension of DSR (Maltz, 1996). It utilises one of the alternative paths that is stored in the source node's routing table (as a backup) for data packets transmission if primary route fails.

The Alternate Path Routing (APR) by Pearlman and others, is mainly used in the conventional wired networks. The performance degradation caused in MANETs when transmission by different paths or routes sharing a node (Coupling) has been studied using ARP (Pearlman, 2000; Li, 2005). During data packets communication, when two node disjoint routes are located physically close enough to interfere with each other the routes are said to be coupled. Those nodes which are participating in simultaneous active multipath communications, as a result are competing with each other to access the medium. This finally leads to worse performance than in a single path protocol. Also, Split Multipath Routing (SMR) presented in (Lee, 2001) focuses on transmitting data packets simultaneously using different disjoint paths techniques.

A new application of multipath is found by Tsirigos and Hazz. In this application and technique of multipath they divide each data packet into sub-packets and then, transmit those pieces simultaneously in different routes to the destination. They include some redundant pieces to help the receiver node (destination) to re-build the original packet successfully even when some pieces are lost (Tsirigos, 2001).

Zafar and Harle have presented in (Zafar, 2009) a summary of multipath routing schemes, and they proposed an on-demand multipath routing mechanism referred to as SMS (Shortest Multipath scheme). SMS is an extension and modification to DSR (Maltz, 1996; Xiang, 2008). The performance of on-demand multipath routing schemes often degrades in terms of end-to-end delay and routing overheads when mobility rates and traffic loads increase, a consequence of both long and stale routes (Zafar, 2007).

3.3 MDVZRP Algorithms

MDVZRP is a multipath hybrid routing protocol for MANET, in simple functions for getting the most optimal routes to each required destination in the entire network, as it combines the common characteristics of proactive (Table driven), and reactive (on-demand) protocols. This technique combines the characteristics of two protocols with different mechanisms. Its main purpose is to capitalise on the strengths that almost exclusively belong to one protocol and at the same time minimise their weaknesses (Khengar, 2003).

Generally, each node in the proactive protocols needs to build its own routing table by actively seeking out routes to all nodes in the network, whether these routes are needed at the time or not. This can lead to very short initial latencies when the node transmits to an arbitrary destination in the network and can create a large table by saving unused routes. When a node needs to transmit information or data to a specific destination, one is simply taken from the node’s routing table. In addition, nodes need to maintain (Keep up to date) their routing tables by periodically sending control messages, this can lead to the consumption of the bandwidth in order to propagate these messages.

In contrast to proactive protocols, reactive protocols only attempt to find routes when they are needed. This type of protocol provides a significant reduction in the amount of bandwidth used for control messages. However, they also experience significant initial delays due to the high probability that a route is not immediately available when needed. The next sections present and clarify the algorithms of the current and pre-MDVZRP versions.

3.3.1 MDVZRP: Motivation

In a MANET, the largest part of the traffic is assumed to be directed to nearby nodes. In MDVZRP, we assume that all the paths (routes) in any routing table are active (workable), usable and only need updating when a new node joins the network and new routing information regarding it is received (update message), or an error is received regarding a non reachable node or a broken active link, then partial updates are needed for some entries such as the ones which have the non reachable node as a destination or an intermediate node.

Therefore, the current version of MDVZRP restricts the scope of proactive to a zone centred on each node. When a new node joins a network and broadcasts its first beacon message (packet), the early version of MDVZRP algorithm gives that node a flat view over the entire network by receiving a full dump (full routing information) from all its nearest (one hop) neighbours to build its own routing table, and obtains multipath to each known destination in the network. On demand requests can be more efficiently performed without querying nodes of the entire network, where all nodes proactively store some local routing information.

When an error message regarding a non reachable node or a broken link is received, MDVZRP utilises an alternative path getting process to find a suitable alternative route among the multipath were stored into the routing table instead of wasting time in route repair, or activating a route request process every time.

3.3.2 MDVZRP: Optimum Multiple Routes Selection

Finding node disjoint multiple paths in mobile networks is not an easy task. Optimum routes are selected by MDVZRP by filtering potential updates to the routing table based on destination, first hop and cost metric. In the following example Figure 3.1, the Source node (1) normally has 15 routes to the Destination (9) as shown in Table 3.1, where the 1st hop is one of the nearest neighbours (node 2, 3 or 4).

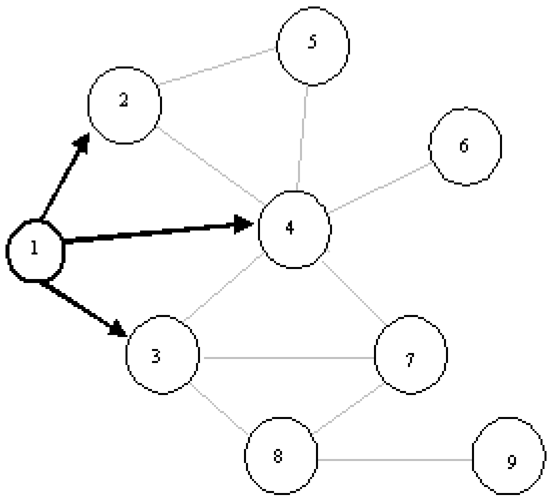
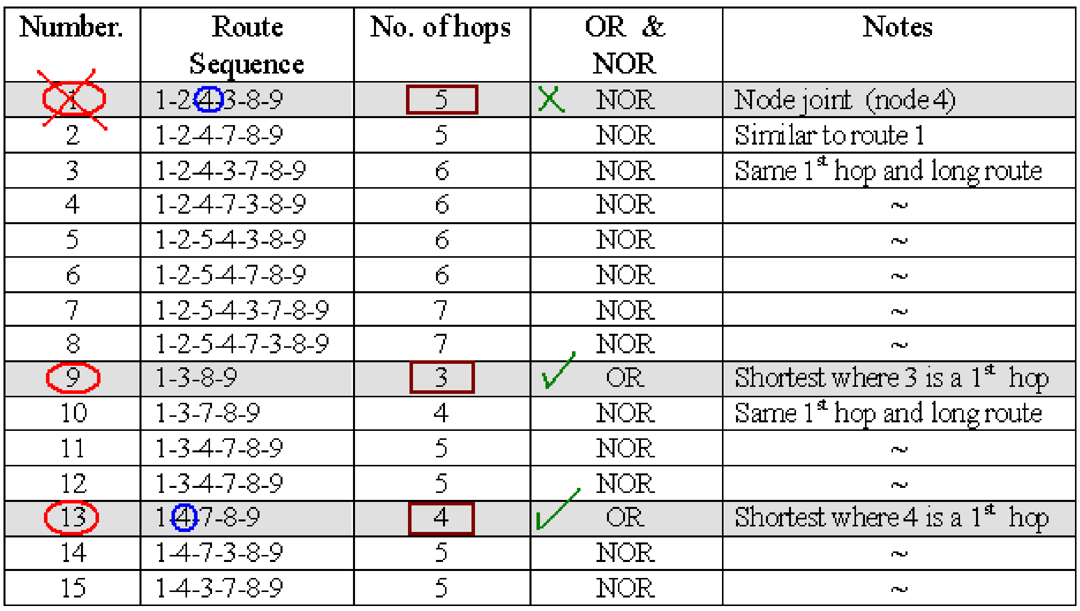


Figure 3.1: An example of a 9 node network

The maximum number of optimum routes (OR) to a destination node depends on the number of neighbours of the source node. The source node (Node 1) has 3 neighbours. Therefore, we need to select only one optimum route from each one hop neighbour, this means there are 3 routes 1, 9 and 13 among the 15 as shown in Table 3.1. We have selected only 2 optimum routes (9, 13), where route 1 is excluded (NOR) because it has node 4 as a 2nd hop, which is a common node (joint) with less number of hops route (shorter route 13). Route 9 and 13 are kept in spite of the common node (8), because they are the only last 2 optimum routes remaining, and we assume that the source node should have at least two routes to each known destination if available.

Table 3.1: Optimum (OR) and Non Optimum (NOR) Routes



Number.	Route Sequence	No. of hops	OR & NOR	Notes
1	1-2-4-3-8-9	5	✗ NOR	Node joint (node 4)
2	1-2-4-7-8-9	5	NOR	Similar to route 1
3	1-2-4-3-7-8-9	6	NOR	Same 1st hop and long route
4	1-2-4-7-3-8-9	6	NOR	~
5	1-2-5-4-3-8-9	6	NOR	~
6	1-2-5-4-7-8-9	6	NOR	~
7	1-2-5-4-3-7-8-9	7	NOR	~
8	1-2-5-4-7-3-8-9	7	NOR	~
9	1-3-8-9	3	✓ OR	Shortest where 3 is a 1st hop
10	1-3-7-8-9	4	NOR	Same 1st hop and long route
11	1-3-4-7-8-9	5	NOR	~
12	1-3-4-7-8-9	5	NOR	~
13	1-4-7-8-9	4	✓ OR	Shortest where 4 is a 1st hop
14	1-4-7-3-8-9	5	NOR	~
15	1-4-3-7-8-9	5	NOR	~

3.3.3 MDVZRP: Routing Initialisation

During the initialisation stage (process) as each node joins the MANET, it adds an entry (route) to itself in its routing table, and then starts to broadcast a periodic beacon (Heart beat message). In Figure 3.2 we assume that, a new node (6) joined the network. This assumption is based on a symmetrical links network (nodes have the same transmission range).

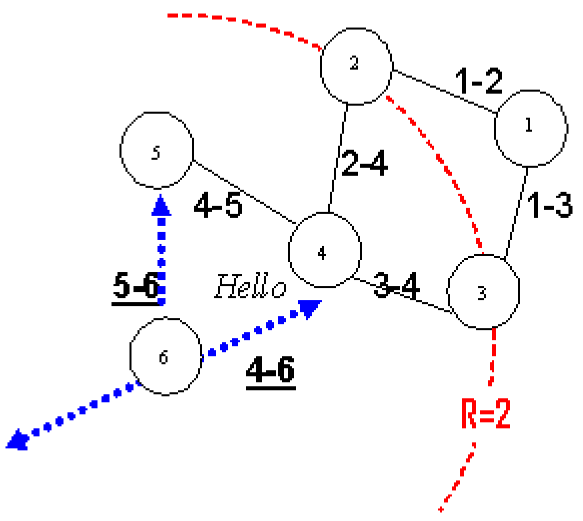


Figure 3.2: Hello message along a symmetrical network

A node that receives the beacon message (i.e. 4, 5) checks if it has a direct route to the beacon message sender (a known neighbour), then it updates that entry regarding to the next heart beat beacon expecting time, and discards that beacon message. Otherwise, it adds a new entry (route) where its destination address and 1st hop are the address of the node that sent the beacon, while the link ID (link_num field) is the (Beacon receiver - Beacon message sender) addresses (e.g. 4-6, 5-6) and sets the number of hops to 1. Then, unicasts its routing table (Full dump) to the new node, and broadcasts a route update packet to its one hop neighbours. Table 3.2 shows the new routes to node (6) that are added by both nodes 4 and 5 in their routing table respectively.

Table 3.2: Routes are obtained after receiving a Hello Message

Node 4:

Destination	1st hop	2nd hop	Metric	Link_num
6	6	-	1	4-6

Node 5:

Destination	1st hop	2nd hop	Metric	Link_num
6	6	-	1	5-6

When node 4 replies, it sends its update packet to the 1st hop neighbours (2, 3 and 5), as shown in Figure 3.2, where these nodes are the 2nd hop neighbours of the new node. Each one of the new node 2nd hop neighbours adds an entry in its routing table and discards the beacon message, where the address of the route destination is the node that sent the beacon address (node 6), the 1st hop address is the node that sent the update message address (node 4), while the link_num is the same one as in the update message

4-6, and the distance metric is incremented by 1 as shown in Table 3.3. The beacon message is discarded once the metric field = radius (R).

Table 3.3: Routes are obtained from the update message sent by node 4

Node 2:

Destination	1st hop	2nd hop	Metric	Link_num
6	4	6	2	4-6

Node 3:

Destination	1st hop	2nd hop	Metric	Link_num
6	4	6	2	4-6

Node 5:

Destination	1st hop	2nd hop	Metric	Link_num
6	6	-	1	5-6
6	4	6	2	4-6

Similarly, node 5 sends an update message to its 1st hop neighbour (node 4, in this case). Table 3.4 shows the new entry that node 4 has obtained from node's 5 update message.

Table 3.4: Routes are obtained from the Update message sent by node 5

Node 4:

Destination	1st hop	2nd hop	Metric	Link_num
6	6	-	1	4-6
6	5	6	2	5-6

Once the new node receives complete routing information (full dump) from all nearest nodes (node 4, 5), it starts to build its own routing table entry by entry excluding any comparable (similar), long and disjoint routes. The following tables are the entire network nodes routing tables. *The notes column in the routing table is not part of the protocol's routing table; it is just to show how the routes are obtained*

Table 3.5: Routing table of node 1

Destination	1st hop	2nd hop	Metric	Link_num		Notes
1	1	-	0	1-1		Initialization
2	2	-	1	1-2		Hello
3	3	-	1	1-3		Hello
4	2	4	2	2-4		Update
4	3	4	2	3-4		Update
5	2	4	2	..		….

Table 3.6: Routing table of node 2 after receiving the Update messages

Destination	1st hop	2nd hop	Metric	Link_num	Notes
1	1	-	1	1-2	Full dump
2	2	-	0	2-2	Initialization
3	1	3	2	1-3	Update
3	4	3	2	3-4	Update
4	4	-	1	2-4	Hello
5	4	5	2	4-5	Update
6	4	6	2	4-6	Update

Table 3.7: Routing table of node 3 after receiving the Update messages

Destination	1st hop	2nd hop	Metric	Link_num	Notes
1	1	-	1	1-3	Full dump
2	1	2	2	1-2	Full dump
2	4	2	2	2-4	Update
3	3	-	0	3-3	Initialization
4	4	-	1	3-4	Hello
5	4	5	2	4-5	Update
6	4	6	2	4-6	Update

Table 3.8: Routing table of node 4 after receiving the Update messages

Destination	1st hop	2nd hop	Metric	Link_num	Notes
1	2	1	2	1-2	Full dump
1	3	1	2	1-3	Full dump
2	2	-	1	2-4	Full dump
3	3	-	1	3-4	Full dump
4	4	-	0	4-4	Initialization
5	5	-	1	4-5	Hello
5	6	5	2	5-6	Update
6	5	6	2	5-6	Update
6	6	6	1	4-6	Hello

Table 3.9: Routing table of node 5 after receiving the Update messages

Destination	1st hop	2nd hop	Metric	Link_num	Notes
1	4	2	3	1-2	Full dump
2	4	2	2	2-4	Full dump
3	4	3	2	3-4	Full dump
4	4	-	1	4-5	Full dump
4	6	4	2	4-6	Update
5	5	-	0	5-5	Initialization
6	6	-	1	5-6	Hello
6	4	6	2	4-6	Update

Table 3.10: Routing table of the new node (6) after receiving the full dump

Destination	1st hop	2nd hop	Metric	Link_num	Notes
1	4	2	3	1-2	Full dump
2	4	2	2	2-4	Full dump
3	4	3	2	3-4	Full dump
4	4	-	1	4-6	Full dump
4	5	4	2	4-5	Full dump
5	5	-	1	5-6	Full dump
5	4	5	2	4-5	Full dump
6	6	-	0	6-6	Initialization

3.3.4 MDVZRP: Route on Demand

If a node needs to communicate with another one in the same network and it has no an available route in its routing table to that destination node because it is outside its routing zone or for any reason, in this case the source node initiates a route request (route on demand). In the following example we assume that, the Source node (1) needs to communicate with the Destination node (6), the S node broadcasts a route request (RREQ) message with the D address asking for a route to the required destination (node D), as shown in Figure 3.3.

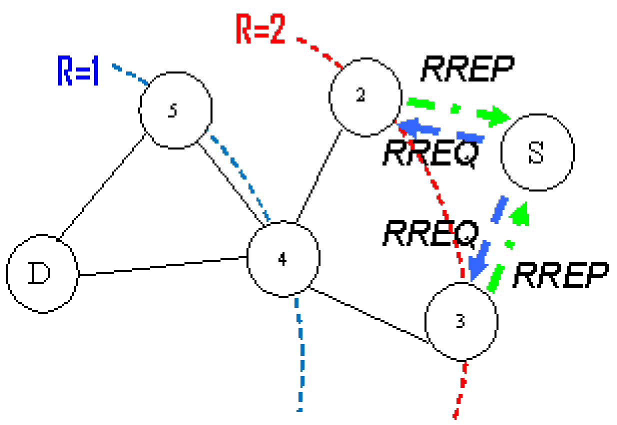


Figure 3.3: Route on demand along a symmetrical links network

A route can be determined, once the RREQ reaches a node that can offer accessibility, to the requested destination (e.g. destination's peripherals node). As shown in Figure 3.3 both nodes (3,2) have a route in 2 hops distance to that destination. The route requested is made available by unicasting a RREP back and reached to the Source node and stored in its routing table. The source node (1) has got 2 routes to the required destination (6) each in 3 hops distance. The source node has selected only one route, because both routes have the same metric (3 hops), as shown in Table 3.11.

Table 3.11: Routing table of node S (1) after RREP message

Destination	1st hop	2nd hop	Metric	Link_num	Notes
1	1	-	0	1-1	Initialization
2	2	-	1	1-2	Hello
3	3	-	1	1-3	Hello
4	2	4	2	2-4	Update
4	3	4	2	3-4	Update
5	2	4	3	4-5	Update/RREQ
6	2	4	3	4-6	RREQ

3.3.5 MDVZRP: Node Movement

In MANETs, the node movement is routing's main challenge, where mobile nodes cause broken links as they move from place to place. Any node that discovers a broken link should initiate and broadcast a route error RERR (a forwarded message), where each node receiving that message updates its routing table if it has information regarding the broken link and rebroadcasts it. Otherwise, drops it. In Figure 3.4, we assume that node (6) has moved away, and node (5) discovered node (6) is non-reachable node.

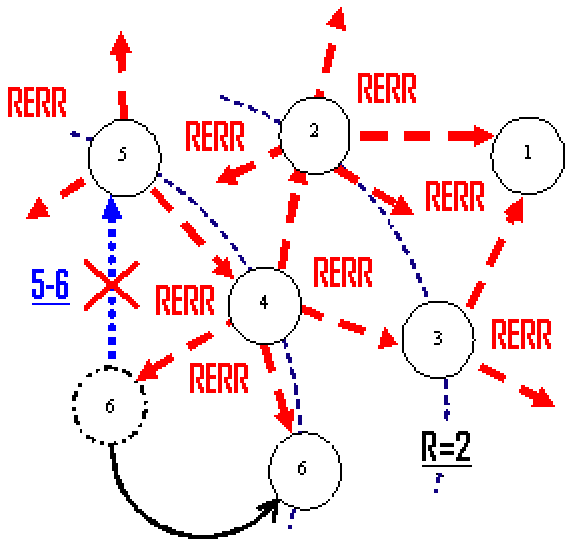


Figure 3.4: Route error message along a symmetrical network

Therefore, node (5) has to search in its routing table for the direct route to node (6) to get its link_num to delete it, and deletes any route (information) it has, where the 1st hop is node (6) as shown in Table 3.12. Then, Node (5) broadcasts a RERR message carrying the non reachable node address (6) and the link_num (5-6) to be deleted by any neighbour that has a route carrying the same link_num in its routing table. Each node receiving an RERR message, deletes any entry that has the same link_num, rebroadcasting the same RERR (error message) and so on, unless the node has no route carrying the same link_num, in this case it discards that message (RERR).

Table 3.12: Routing table of node 5 after discovering the broken link

Destination	1st hop	2nd hop	Metric	Link_num	Notes
1	4	2	3	1-2	
2	4	2	2	2-4	….
3	4	3	2	3-4	….
4	4	-	1	4-5	….
4	**6**	4	∞	4-6	Deleted
5	5	-	0	5-5	….
6	6	-	∞	**5-6**	Deleted
6	4	6	2	4-6	….

Table 3.13: Routing table of node 4 after receiving the RERR message

Destination	1st hop	2nd hop	Metric	Link_num	Notes
1	2	1	2	1-2	….
1	3	1	2	2-2	….
2	2	-	1	2-4	….
3	3	-	1	3-4	….
4	4	-	0	4-4	….
5	5	-	1	4-5	….
5	6	5	∞	**5-6**	Deleted
6	5	6	∞	**5-6**	Deleted
6	6	6	1	4-6	….

Table 3.14: Routing table of node 6 after receiving the RERR message

Destination	1st hop	2nd hop	Metric	Link_num	Notes
1	4	2	3	1-2	….
2	4	2	2	2-4	….
3	4	3	2	3-4	….
4	4	-	1	4-6	….
4	**5**	4	2	4-5	Deleted
5	5	-	1	**5-6**	Deleted
5	4	5	2	4-5	….
6	6	-	0	6-6	….

3.3.6 MDVZRP: Zone Radius and Node Density

The zone radius metric is the distance in the number of hops from a specific source node to the last node in its routing zone as shown in Figure 3.5. Each node has its own routing zone. An important consequence is that, in most of the cases there are overlaps between the zones of neighbouring nodes.

The routing zone has a radius (R) expressed in the number of hops. Thus, the routing zone includes all nodes whose distance from the source node in the equation is at most R number of hops. Figure 3.5 shows a new node (6) for an example, and its routing zone when it joined the network for the first time. We call each node that has only one hop distance from the new node a 1st hop neighbour if zone radius ***R =1*** (i.e. node 4, 5). While, we call the node that has 2 hops distance from the new node, a 2nd hop neighbour and so on.

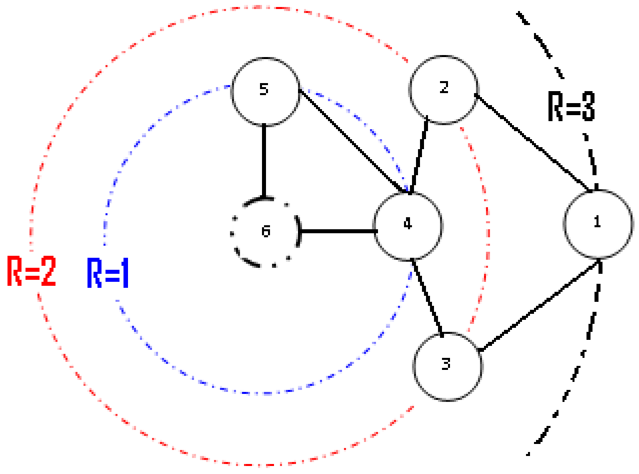


Figure 3.5: Routing zone radius, where R=1, 2 or 3

Nodes with their shortest route to the source equal to the zone radius are called "Peripheral". Nodes with their shortest route to the source larger than the zone radius are called "out of routing zone nodes".

3.4 Summary

Because of characteristics and dynamic environment of the MANET, routing was and still is one of the most significant issues that receive a lot of study and investigation. A new Multi-path Distance Vector Zone Routing Protocol (MDVZRP) for MANET was presented in detail in this chapter.

MDVZRP is a multipath routing protocol based on 2nd hop, link-num and a new assumption (no need to update the routes, unless a node receives routing information regarding to a specific route). MDVZRP presents a solution to the power consumption by reducing the overhead, and hence increases the battery life. The protocol adapts both reactive and proactive mechanisms to perform a reliable communication between mobile nodes via multi backup paths. The new routing protocol algorithms and routing tables are provided and pointed out. In addition, the packet reception algorithm flowchart is provided in Appendix A. The next two chapters present the MDVZRP implementation and its functionality testing respectively.

CHAPTER 4: MDVZRP IMPLEMENTATION

4.1 Introduction

MDVZRP is a multipath routing protocol that maintains routes for as long as the route is active. It is a combination of the two extremes of proactive and reactive protocols that aims to provide the best of both techniques. There are number of other hybrid protocols already existing and each of them has its own technique. However, as the following sections will clarify, the hybrid approach used in MDVZRP is significantly different to the route request and zone approaches employed in other hybrid protocols. We assume in our routing protocol, that all the entries are workable (fresh not stale), and there is no need to update them unless a node receives an error message regarding a specific route, in this case the node assigns that route as a broken link.

This chapter presents implementation details of the MDVZRP protocol, after a brief look at the early stages of the design process that ultimately led to the current version of MDVZRP.

4.2 Pre-MDVZRP Design

MDVZRP resulted from many stages of design and improvement over a period of time into what it is today. Some of the improvements were in the code itself and others were in the protocol's algorithm technique. This section will briefly describe these stages.

4.3 MDVZRP V1.0

First of all, the MDVZRP protocol was originally a purely proactive protocol, where nodes actively seek to get as many entries as they can to each node in the entire network by sending periodic beacons to inform adjacent nodes of their presence and by receiving routing information packets. Nodes that receive that beacon in turn propagate immediately routing information messages, such as Full Routing Information Packet (FRIP) and Routes Update Packet (RSUP). As shown in Figure 4.1 where these routing messages would consist of routing information regarding the node that sent the beacon (Node 6 for an example), and the node that was now broadcasting the routing update message.

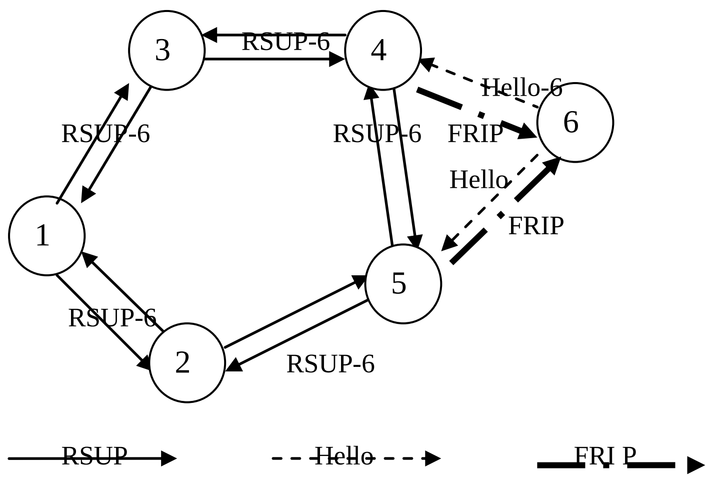


Figure 4.1: Hello and RSUP messages in an early version of MDVZRP

This effectively informed the nodes in the network of the presence of a new node, as well the next hop towards it. As mentioned, this protocol technique is based on the assumption that all the routes in a node's routing table are workable and fresh unless an error message is received. Therefore, not all the nodes are concerned with link changes that may occur, one of the design goals was to avoid the propagation of route update messages across the entire network, which was carried out in the later version of MDVZRP v2.0 and hence decreased the overhead in general.

Once a node discovers a non reachable adjacent node, it deletes any entry using that node as a next hop, and immediate broadcasts an error message that carries the link number of that broken route. Nodes that receive such error message and have entries in their routing tables with the same link number *broken link number*, in turn rebroadcast the same error message after deleting those entries. This routing information only propagates so far as it stops at the point where a node hasn't any route in its routing table with the same link number. Take for example the case when node 6 moves away from node 5, but is still an adjacent to node 4 *within its transmission range* as shown in Figure 4.2.

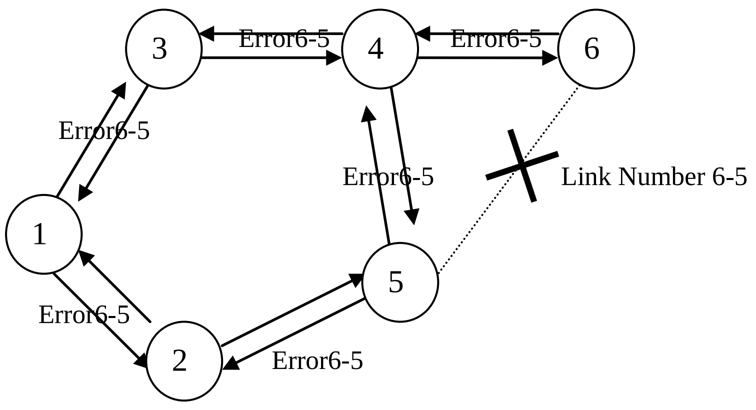


Figure 4.2: Error message in an early version of MDVZRP

This type of route error and the previous routes update messages may cause the flooding of an entire network once broadcast and could lead to the network clogging, especially in case of the protocol operates a purely proactive, and the network is highly dynamic.

A number of problems existed in this early version of MDVZRP v1.0. Firstly, although the propagation of route update messages allowed nodes in a network to become aware of a new arrival, the new node would not learn of the existence of all the nodes in the entire network, and routes to all previously present nodes. Secondly, there was a serious problem of routing table size because of the gradual increase that could occur due to the form of the update messages, and by actively seeking each node out routes to all nodes in the network, whether these routes are needed at the time or not.

This issue would eventually result in MDVZRP evolving to become zones hybrid protocol. As well as this nodes do not delete the broken route from their routing tables; instead they assign the metric number field of that broken entry to infinity. This led to the decision to include the error messages in the *RSUP* packet. Also a node has the right to broadcast a *RSUP* message at the periodic beacon time instead of broadcasting a beacon whenever it needs to advertise any changes in its routing table. Nodes that receive these update messages treat them for the two purposes at the same time (As a Beacon and an Update message).

Finally, most if not all of the wireless networks are asymmetrical (Skloul, 2008), while this early version of MDVZRP v1.0 treats all the links as a symmetrical (Skloul, 2008). For example, as shown in Figure 4.3, the nodes delete any route carrying the link number 6-5 or 5-6 when they receive an error message from node 6 or 5 regarding that link in between.

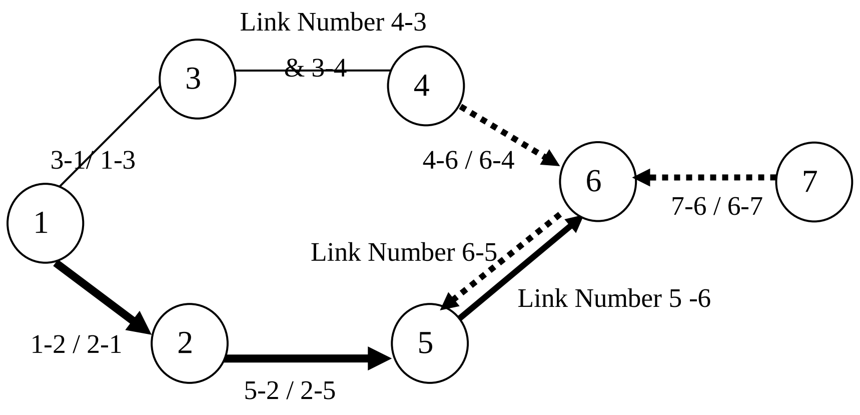


Figure 4.3: Asymmetrical links in an early version of MDVZRP (Skloul, 2008)

In matter of fact, the link number 5-6 is different from 6-5, where the link number 5-6 is generated by node 5 when receiving a beacon from 6 (depends on node 5 transmission range) and used by the nodes that need to get to node 6 or any other node behind it via node 5 (i.e. node 1,2), while the link number 6-5 is generated by node 6 when receiving a beacon from node 5 (depends on node 6 transmission range) and using by the nodes that need to get to node 5 or any other node behind it via node 6 (i.e. node 4,7).

Node 1 for an example, uses link number 5-6 to reach node 6 in 3 hops, while node 7 and 4 both use link number 6-5 to reach node 5 in 2 hops as shown in Figure 4.3 by the dashed line. Logically, link 5-6 and 6-5 are the same, but physically different because each node has a different transmission range in most cases. They might also be different because of different radio interference, or radio propagation conditions. For these reasons, in the early version of MDVZRP we have treated the error message that comes from node 6 regarding this link (link between node 6 and 5) as different from that error message that comes from node 5 for the same link. The next section describes the present state of MDVZRP, which addresses these issues.

4.4 Protocol Description of MDVZRP v2.0

Using MDVZRP, nodes in an Ad Hoc network can discover other nodes, as well as routes to those nodes, using a combination of proactive and reactive techniques. The proactive part of the protocol is based upon the use of periodic beacon packets known as BEACON, and the messages that inform nodes of changes in the topology, known as RSUP (*Routes Update Packets*). In the event that a route to a known node is unavailable, and a transmission needs to take place immediately, the protocol exhibits its reactive nature. This is through broadcasting a RREQ (Route Request) and unicasting a RREP (Route Reply) messages respectively.

Before the description of MDVZRP v2.0 and its implementation, we list the most important changes that have been made from v1.0 and the main differences between MDVZRP v1.0 and MDVVZRP v2.0. Table 4.1 lists these differences, while the performance comparison of the two versions is explained briefly and described graphically in the next chapter.

Table 4.1: MDVZRP v1.0 v MDVZRP v2.0

Differences Between MDVZRP v1.0 and v2.0		
Term	v1.0	v2.0
Beacon	Periodic	Periodic
RSUP	Yes	At the Beacon time
FRIP	Yes	No
Error Message	Yes, and Immediately	Included with Update packet
RREQ technique	No	Yes
Zones	No	Yes
Links	symmetrical	Asymmetrical

In the next few sections we are going to briefly explain the structure and design of MDVZRP v2.0, and also the pseudocode of the most important functions and classes. In the next chapters we will show some graphs of the MDVZRP's v2.0 performance compared to v1.0 and other reactive and proactive protocols.

4.5 Routing Table Structure

The routing table (RT) lists a number of multi-paths (backup) to each destination of the network. The routing table is used to transmit packets through the Ad Hoc network. The nodes have to update their routing tables when there is a significant change in the network. Table 4.2 shows MDVZRP's routing table fields. Each node participating in a MDVZRP will maintain a routing table with the following fields:

Table 4.2: MDVZRP Routing table fields

Field	Description	Type	Size	Notes
dst	Destination	Nsaddr_t	32	address
f_nxt_hop	First next hop	Nsaddr_t	32	First next hop towards destination.
s_nxt_hop	Second hop	Nsaddr_t	32	For discovering joint paths and selecting disjoint only to minimise routing table.
metric	Number of hops	Uint	32	Distance to a destination in number of hops
link_num	Link id or number	Uint	32	A unique number
Packet_id	Packet number	Nsaddr_t	32	Helps the node to determine whether received (RREQ and RREP) before or not.
advertise	Advertise	Boolean	1	When it is true, that means advertise this route.
Changed_at	Changed at	Double	32	Route last change time.
timeout_event	Event	Pointer	32	Event used to schedule timeout action.
q	Queue pointer	Pointer	32	Packet queued for destination

The *dst* (destination) field is the address of the destination node, while the *f_nxt_hop* indicates which node should be used as the next hop towards that destination. The 2nd next hop *s_nxt_hop* field is used to minimise the routing table size by getting and saving only the best disjoint routes (shortest routes that have no any common nodes to the same destination). This indicates the second node towards that destination with *Link_num* field (link number) these fields are used to construct the best routes to each destination, as shown in Figure 4.4 where node 1 has only one route to node 5 despite having two adjacent nodes (nod 2 and 3). That is because node 4 is a node joint (common) in both routes. The *metric* field represents the distance to that destination in number of hops, which are used for obtaining the best or shortest route to that destination from each adjacent. For knowing when each route was created or updated last the *changed_at* field is used.

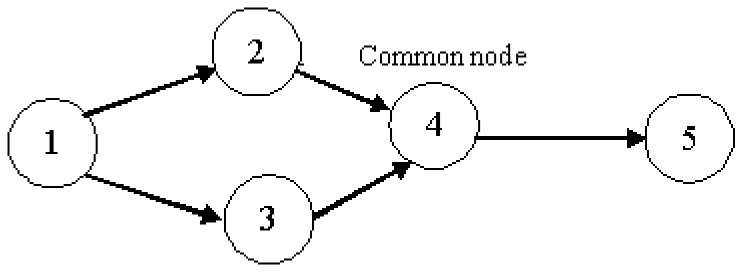


Figure 4.4: Routes with a common node (node joint)

Finally, the *timeout-event* is a field that is only present for adjacent nodes; the event is used to schedule timeout action, in other words nodes that are within wireless transmission range, while for all other nodes it is simply set to Null. For example, if an adjacent node beacon time arrives at T=t1 (i.e. T=15 sec) while the timeout interval is x (i.e. 3 sec), then the scheduler timeout event occurs at T=t1+x sec (T=15+3=18 sec), this means that the node will be considered as non reachable if nothing is received after 18 sec.

When a node was unable to send a packet to a specific destination because no route was available, it uses the *q* field to queue that packet (where the *Packet_id* field is set to the destination id) till it gets a route to that destination or drops later when the queue gets full.

If the route reply is not received yet when no route is available to a specific destination and a route request is broadcast, the node that made the route request cannot make another request till it receives the route reply or waits for certain amount of time. In another sense, it helps the node receiving the same RREQ or RREP to determine whether or not it has received it before for period of time. Otherwise, the node can make the same RREQ again if the RREP for that required destination is not received within that period of time. RREQs that have already been received and processed are promptly discarded.

4.6 General MDVZRP Packet Format

Figure 4.5 illustrates the general structure of an 802-11 PACKET which is the standard packet structure in most of the Ad Hoc wireless networks, while Figure 4.6 and 4.7 show the general format of MDVZRP packet which is base on the 802-11 packet format.

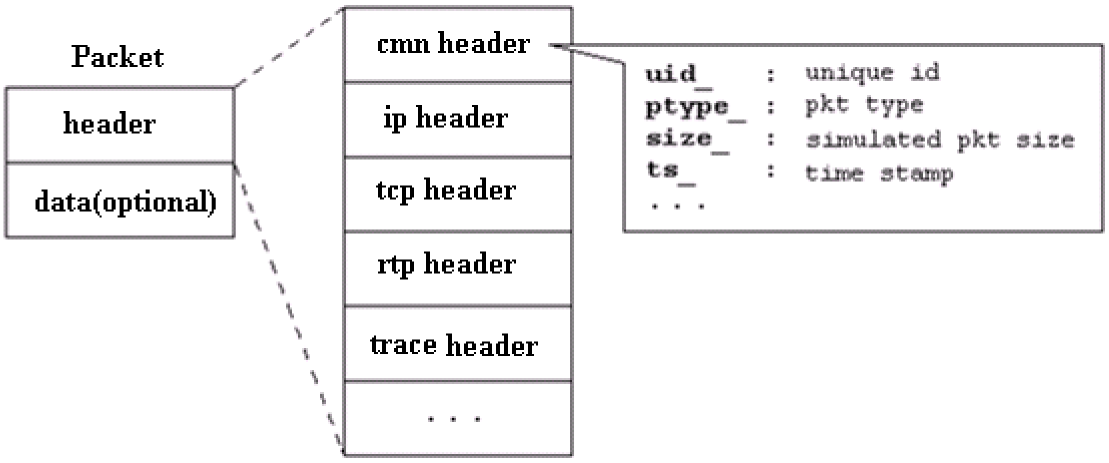


Figure 4.5: 802-11 Packet Format

The fields of the CMN header are set to indicate that the received packet should be MDVZRP's packet, as shown in Figure 4.7. The first byte of the MDVZRP cmn header indicates the packet's type. This can be one of these packets, a Beacon, RSUP, FRIP, RREQ or RREP, while the next eight bytes indicate the packet size and the packet id. The IP header (in dark colour) and CMN header fields are listed in the figures below, (not all the fields are listed). We have listed only the MDVZRP fields that are used frequently.

1st Byte	2nd Byte	3rd Byte	4th Byte
0 7	8 15	16 24	24 31
Source IP address			
Destination IP address			
Time to live			
flow id			
priority			
...			

Figure 4.6: IP header fields of MDVZRP

1st Byte	2nd Byte	3rd Byte	4th Byte
0 7	8 15	16 24	25 31
packet type	packet size	…	packet id
…	…	…	…
…			
IP address of forwarding hop			
IP next hop address for this packet			
IP address of first hop forwarded this packet			
IP address of pre-forwarding hop			
IP address of last hop			
how many times this packet was forwarded			
time for this packet in sec			
time for this packet in sec			
………			

Figure 4.7: CMN header fields of MDVZRP

4.7 Specific Packet Formats

There are five types of MDVZRP packets. These packets' format vary according to the packet type, where some fields in the packet headers have been omitted, leaving only ones that are relevant to this description.

4.7.1 BEACON Packet (HELLO)

The BEACON packet that is common in most of the Ad Hoc protocols is called the Hello packet. The main purpose of the beacon is to inform the adjacent nodes of the existence of the node that sent that packet, where it contains all the information required such as the node's own unique IP address. Nodes in any Ad Hoc network broadcast these packets periodically in turn when a node has nothing to broadcast, where any other broadcasting such as RSUP can serve as a beacon. The beacon messages are essentially empty packets, broadcast only to one hop adjacent by setting the destination field to -1 *Broadcasting*, and not rebroadcasting messages, where in the IP header the TTL field *Time to Live* is set to 1 as shown in Figure 4.8.

1st Byte	2nd Byte	3rd Byte	4th Byte
0 7	8 15	16 24	25 31
Source IP address = my address			
Destination IP = -1 (BROADCAST)			
TTL=1			
………			
Type=20(BEACON)			
…..			

Figure 4.8: A BEACON Packet

4.7.2 RSUP Packet

The RSUP (Routes Update Packet) Figure 4.9 is a non periodical rebroadcasting message. These types of messages are essentially, to keep the routing tables of all the nodes are up to date. The RSUP packet can serve as beacon when broadcast as we mentioned previously, and also used instead of the ERROR message as a part of the protocol early version development. Therefore, a node that broadcasts such packets wouldn't broadcast beacons at that period of time (this period is known as the beacon Interval) to reduce number of broadcast packets and hence reducing the overhead in general. Transmission and broadcasting an update packet represents the proactive part mechanism in MDVZRP protocol.

This packet is used when a node has a partially or completely change in its routing table, such as getting new entries, or has an old broken ones, in these cases the node needs to advertise them by presenting itself as a viable next hop to any of these new addresses included within the broadcast packet, and as a broken link explorer to any of these addresses included where metric is infinity. These addresses are ordered so that they form the new entire source route from the first entry to the last.

1st Byte 0 7	2nd Byte 8 15	3rd Byte 16 24	4th Byte 25 31
Source IP address = my address			
Destination IP = -1 (BROADCAST)			
TTL=1			
………			
Type=23(RSUP)			
next_hop_ // BROADCAST			
………			
1st entry need to advertise and included in this packet			
2nd entry need to advertise and included in this packet			
………			
last entry need to advertise and included in this packet			

Figure 4.9: A RSUP packet

Figure 4.10 is an example to show the RSUP packet use, where node 4 broadcasts a routes update message which is received by its adjacent nodes (One hop neighbours). The first entry informs all recipients that node 5 is not reachable through node 4; while the second and third entries inform that node 6 and 7 are accessible through node 4 in 2 and 3 hops respectively.

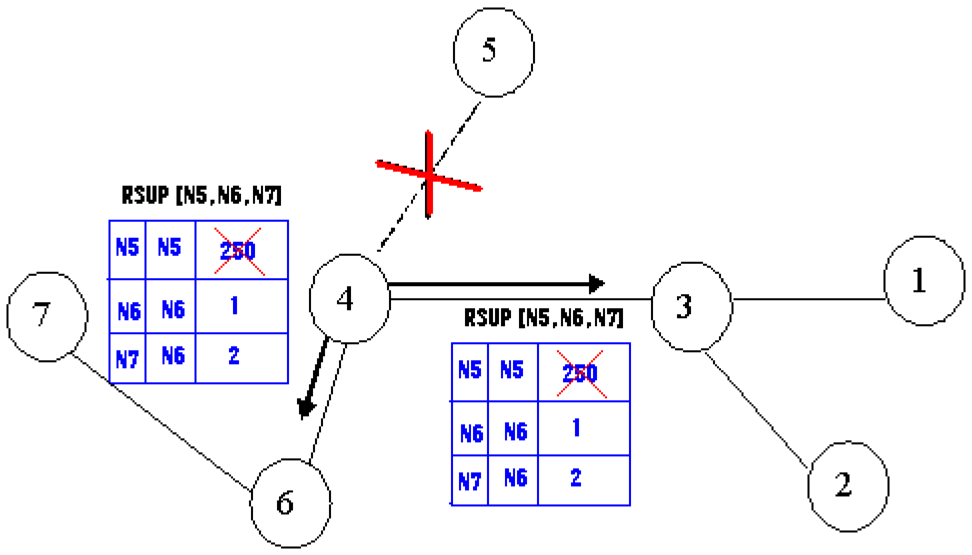


Figure 4.10: A simple example of Routes Update propagation

4.7.3 FRIP Packet

The third packet type that is another part of MDVZRP's proactive mechanism is the FRIP (Full Routing Information Packet).A FRIP is addressed to a single recipient, not a broadcast. It is not repeated automatically. The FRIP is sent as a reply to the beacon that a newly arrival node broadcasts. The new node receives FRIPs from all its neighbours and so is able to acquire an initial view of the whole network. In Figure 4.11 the format of FRIP is shown, while Figure 4.12 shows the FRIP mechanism.

1st Byte 0 7	2nd Byte 8 15	3rd Byte 16 24	4th Byte 25 31
Source IP address = my address			
Destination IP = New node's IP Address (Unicast)			
TTL=1			
………			
Type=22(FRIP)			
next_hop_ // New node's IP Address(dst)			
………			
Number of entries that are included in this packet			
1st entry			
2nd entry			
…………………………			
last entry			

Figure 4.11: A FRIP Packet

4.7.3.1 FRIP Mechanism

Figure 4.12 shows the FRIP mechanism where a new node (N) broadcasts a beacon to inform of its presence to any node that receives that beacon, and receives back two FRIP packets from nodes 3 and 5. The entries included in node 3 FRIP inform the new node that node 1, 2 and 4 are accessible through node 3 in 3 and 2 hops respectively, while node 3 itself is accessible directly in one hop (adjacent).

Entries received from node 5 inform the new node that node 5 is a one hop adjacent and node 6 is accessible through node 5 in 2 hops distance. Because of the transmission range, node 4 did not receive the new node's beacon. Therefore, node N did not receive a FRIP from node 4.

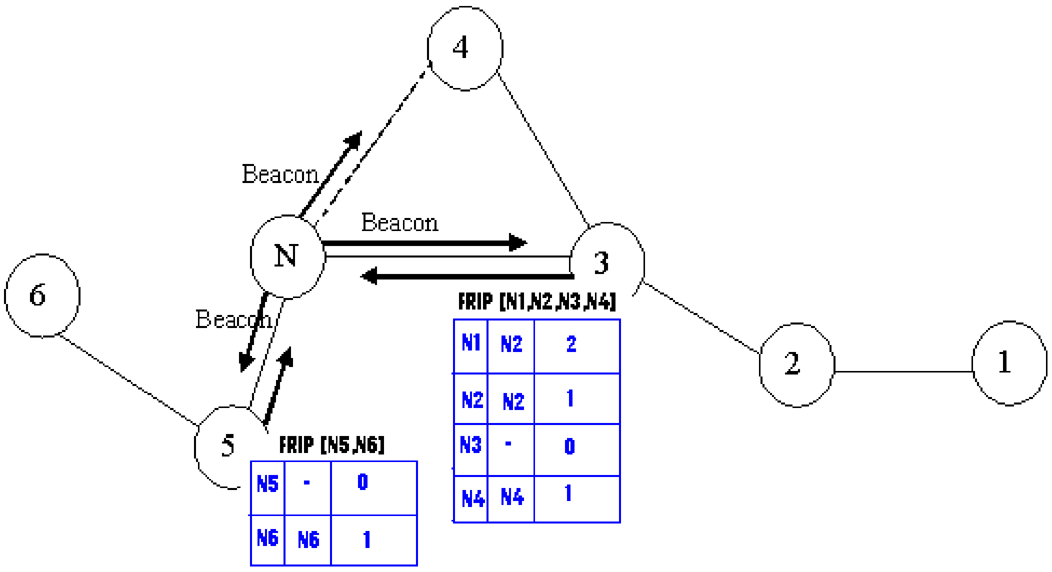


Figure 4.12: A simple example of FRIP propagation

4.7.4 RREQ Packet

Route Request and Route reply Packets are the fourth type that is part of MDVZRP's reactive mechanism. Figure 4.13 shows the format of the RREQ packet, divided into three groups of fields. The 1st group is the IP header fields; the 2nd group is the CMN header fields; while the 3rd one is the payload fields belonging to the requested destination. RREQ is a broadcast packet, which is rebroadcast by very receiver node if it has no route available to replay.

TTL was 32 (Ns2 default) in early version.TTL in current version is set according to the values in Table (6.5). RREQ use on demand once a node needs to send a data to a specific destination and has no route available in its routing table to that destination. The first entry in the RREQ payload is for the node that is being requested (Requested destination Address1), while the second entry is the source of the request (Requester Address2).

The requester node (Address 2) sets *prev_hop_* field to its IP address, *next_hop_* field to -1 (Broadcast), and resets the next two fields *fst_fwd_hop_* which indicates the IP address of the first adjacent node that forwards that packet if it does not know the requested destination, and *p_prev_hop_* field which is the IP address of pre-forwarding node both to the initial value (Null).

1st Byte 0 7	2nd Byte 8 15	3rd Byte 16 24	4th Byte 25 31
Source IP address = my address			
Destination IP = -1 (BROADCAST)			
TTL=32			
………			
Type=25(RREQ)			
next_hop_ =-1 (broadcast) // next hop for this packet			
prev_hop_ = myaddress // IP addr of forwarding hop			
fst_fwd_hop_ = Initial value (Reset=Null)			
p_prev_hop_= Initial value (Reset=Null)			
num_forwards_// number of forwarded hops(distance)			
………………………………			
Destination being requested(Address 1)			
Requester(Address 2)			

Figure 4.13: Format of RREQ Packet

Each neighbour (first hop node) either satisfies the RREQ by unicasting (sending) a RREP back to the source if it has a route to the requested destination (Address 1) and discards RREQ packet (drops it), or checks the two previous fields, if they are set to initial value (Null) as in this case where TTL= 32 (*first forward*), sets the *fst_fwd_hop_* field to its IP address and the *p_prev_hop_* field to the IP address of the forwarding hop which came before the previous hop. This is the RREQ packet sender in this case, as shown in the following three algorithm lines, then it re-broadcasts the RREQ to its own neighbours after increasing the hop count:

```
If (fst_fwd_hop_ =intial value)        // Initial value =Null
    fst_fwd_hop_ =my address   // set the first forwarding hop field to my address
p_prev_hop_ =prev_hop_         // set the pre- previous hop field to the previous hop address
```

The purpose of these two fields is that any intermediate node can get all the necessary information to maintain a route to the previous node and the source (Requester) node as well, that gives any intermediate node the ability to unicast RREP to the source node once the RREQ reaches a node that has a route to the required destination. The following example clarifies the RREQ / RREP mechanisms.

4.7.4.1 RREQ Mechanism

In Figure 4.14 we assume that node 6 is looking for a route to node 1, and it broadcasts a RREQ packet which has a unique id (i.e. xyz). Node 5 and 7 are the only neighbours and both receive the RREQ, we assume that both nodes have no available routes yet to node 1. Node 5 and 7 can easily create new routes to node 6 *Source* from the information included within the RREQ packet in case of any one (Node5 or 7) has no route to node 6 as shown in table 4.3.

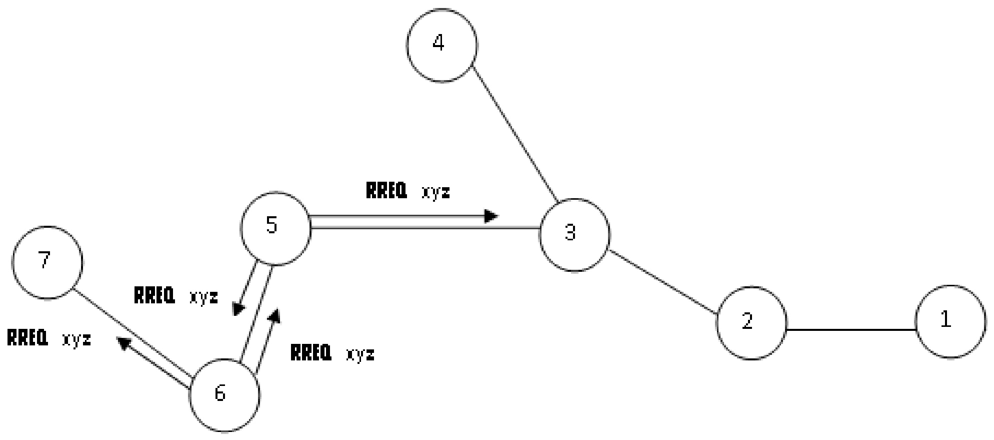


Figure 4.14: A simple example of RREQ propagation

The following simple lines in Figure 4.15 are a part of RREQ packet receive algorithm, for a new route creation to RREQ packet sender:

```
dst  (Destination) =  Requester;                  // Address 2
First next hop (f_nxt_hop) = prev_node;           // previous node
Second next hop (s_nxt_hop) = p_prev_node;        // pre previous node
Metric     (metric) = num_forwards_;              // Number forwarded

If (fst_fwd_hop_==initial)
      link_num = myaddress * 10000+ Requester;    // creating link id
Else
     link_num = fst_fwd_hop_ * 10000+Requester;

changed_at = now;     // Time at which this route is created or updated
```

Figure 4.15: Creating a new route to RREQ packet sender algorithm

Table 4.3: New routes to RREQ sender that created by RREQ receivers

Dst	f_nxt_hop	s_nxt_hop	Metric	Link_id	Change_at	Packet_id
Address2	prev_hop	p_prev_hop	num_of_hop	myaddress – address2	Now	RREQ packet id

Node 5:

6	6	null	1	50006	Now	xyz

Node 7:

6	6	null	1	60005	Now	xyz

After getting the routes, node 7 discards (drops) the RREQ packet because it has no other neighbours. While node 5 sets *fst_fwd_hop_* and *prev_hop_* fields to its IP address, *p_prev_hop_* filed to *prev_hop_* (the source node IP address), increases *num_forwards_* by 1 (Number of hops) and then broadcasts the RREQ packet again.

Each node maintains a field called *packet_id* in its routing table to save the RREQ / RREP packet id as a simple way of avoiding routing loops. An intermediate node may receive multiple copies of the same RREQ from various neighbours. Therefore, if it has already received a RREQ with the same packet_id, it drops the redundant RREQ (same Packet_id) and does not rebroadcast it.

4.7.5 RREP Packet

Figure 4.16 shows the format of RREP packet, which is divided into three groups of fields as well. The 1st group is the IP header fields, the 2nd group is the CMN header fields, while the 3rd one are some fields that represent the payload entry belonging to the requested destination. RREP is a unicast packet, where TTL is 32 at the first forward it uses on demand once a node receives a RREQ and has an available route to the destination being requested in its routing table. The first field in the RREP payload is for the node that is being requested (Requested destination) as shown in the last five lines in Figure 4.16, while the next four fields are the routing information needed by each intermediate node which RREP will pass through to create a route to the requested destination till RREP reaches the source of the request *Requester*.

1st Byte	2nd Byte	3rd Byte	4th Byte
0 7	8 15	16 24	25 31
Source IP address = my address			
Destination IP = Requester(Address 2)			
TTL=32			
………			
Type=26(RREP)			
next_hop_ = prev_hop_ or hop in the best route to the Requester			
prev_hop_ = myaddress // IP addr of forwarding hop			
fst_fwd_hop_ = Initial value (Reset=Null)			
p_prev_hop_= Initial value (Reset=Null)			
num_forwards_// number of forwarded hops(distance)			
……………………………………………………			
Destination being requested(Address 1)			
f_nxt_hop // 1st hop towards the requested node(Address 1)			
s_nxt_hop // 2nd hop towards the destination node(Address 1)			
metric // Num. of hops from me to the requested node(Address 1)			
link_num // link number between destination (address1) and f_nxt_hop			

Figure 4.16: Format of RREP Packet

In the previous example, we assume that node 3 is the only adjacent that received the rebroadcast RREQ from node 5, and it is not the source of RREQ as shown in Figure 4.14. Therefore, it has to carry out the same procedure as node 5 did to create a route to the RREQ packet sender *Requester* by using the algorithm in Figure 4.15. Table 4.4 shows the new route that node 3 has created as an intermediate node to node 6 *RREQ sender*.

Table 4.4: A new route to RREQ sender created by an intermediate node 3

Dst	f_nxt_hop	s_nxt_hop	Metric	Link_id	Change_at	Packet_id
Address2	prev_hop	p_prev_hop	num_of_hops	fstfwdhop – address2	Now	RREQ packet id

Node 3:

6	5	6	2	50006	Now	xyz

We also assumed that node 3 has an available route in its routing table to node 1 (Requested destination) in 2 hops, where node 2 is the first next hop as shown in Figure 4.17. Therefore, it has prepared a RREP packet as described previously in Figure 4.16.

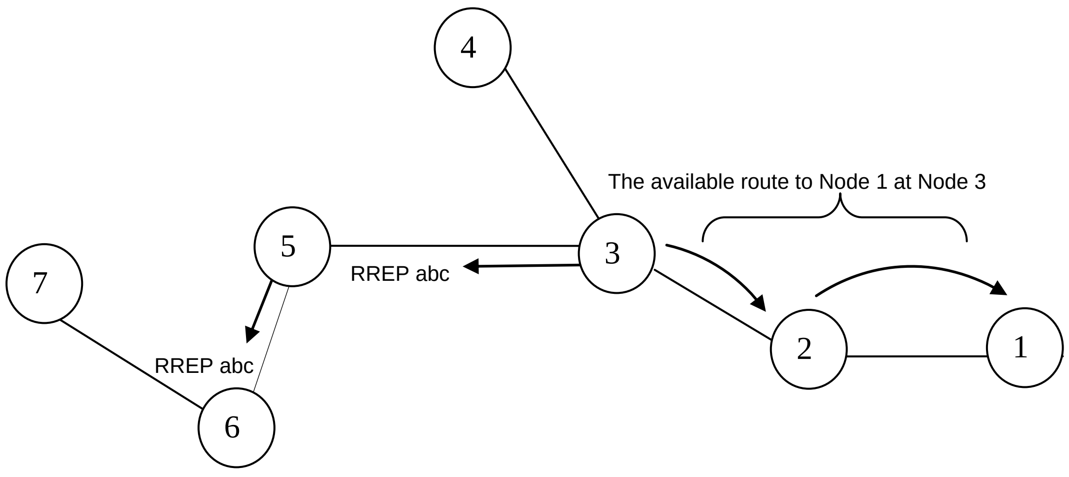


Figure 4.17: A simple example of RREP propagation

The RREP packet is sent to node 6 *Requester* via node 5, which has forwarded that RREP packet to node 6. Node 5 and 6 are both getting a route to node 1 *Requested destination* and a route to the previous node if it is unknown using the following simple lines in Figure 4.18 which are a part of the RREP packet receive algorithm for creating a new route to the requested destination. Tables 5 and 6 show the new routes that node 5 and 6 have got.

```
dst  (Destination) =  destination being requested;     // Address 1
f_nxt_hop  (First next hop) = prev_node;                // previous node
s_nxt_hop  (Second next hop ) = p_prev_node;            // pre previous node
metric     (Metric) = num_forwards_;                    // Number forwarded

If (fst_fwd_hop_==initial)
      link_num = myaddress * 10000+ Requester;               // creating link number
Else
     link_num = fst_fwd_hop_ * 10000+Requester;

changed_at = now;    // Time at which this route is created
```

Figure 4.18: Creating a new route to RREQ packet sender algorithm

Table 4.5: A new route to requested destination at an intermediate node 5

Dst	f_nxt_hop	s_nxt_hop	Metric	Link_id	Change_at	Packet_id
Address1	prev_hop	p_prev_hop	num_of_hop s	fstfwdhop – address2	Now	RREP packet id

Node 5:

1	3	2	3	50003	Now	abc

Table 4.6: A new route to requested destination at the requester node 6

Dst	f_nxt_hop	s_nxt_hop	Metric	Link_id	Change_at	Packet_id
Address1	prev_hop	p_prev_hop	num_of_hop s	fstfwdhop – address2	Now	RREP packet id

Node 6:

1	5	3	4	60005	Now	abc

4.7.6 MDVZRP Implementation

In the following few sections we will present and explain in details the implementation of MDVZRP routing protocol. This routing protocol is implemented using C++ under Ns2 Network simulator 2, where the scenarios are described using Tcl scripts in the simulation time. We firstly created a new directory called mdvzrp to allocate our code inside the NS2 base directory. Then, we created the protocol "*physical*" structure, by creating the following files there:

mdvzrp/mdvzrp.h This is the header file where we define all necessary timers and routing agents which perform protocol's functionality.

mdvzrp/mdvzrp.cc In this file, all timers, routing agent and Tcl hooks are actually implemented

.

common/packet.h here all packets of MDVZRP protocol that need to exchange among the nodes in the MANETs are declared.

mdvzrp/mdvzrp_rtable.h Header file where the routing table of our own protocol is declared.

mdvzrp/mdvzrp_rtable.h Routing table implementation.

Secondly, we created the protocol "*logical*" structure (classes), by creating an agent which is inherited from Agent class. These classes are representing the endpoints where the networks-layer packets are created and consumed, and also are used in the protocols implementation at various layers. In addition, by offering a linkage with Tcl interface. This class gave us the ability to control the protocol in Tcl scripts simulation time.

The routing table is a collection of entries or routes gathered by a node to most of the destinations in the network. The routing agent maintains a routing table (which is not continuously needed) and an internal state, which can be represented as attributes collection or a new class inside the routing agent itself. We will utilise routing table as a new class, *mdvzrp_rtable.*

MDVZRP protocol defined some new control packets. These packet types are defined and located in the *common/packet.h* header file. It is counted on a Time class any control packet needs to periodically send or after the occurrence of an event. There are many purposes for using such Timers. For example, when the MDVZRP protocol has some information and needs to store it for a specific period of time, (must be erased after while at a certain time). In such case the best way is to create a custom timer which has the capability to do such a job.

The MDVZRP protocol also uses a timer called *changed_at* to specify timelife for each entry in the agent’s (Node) routing table. In general, the protocol uses a timer whenever it needs to schedule a task at a given time.

It is useful for the protocol performance evaluation to save *dumping* the information in a trace file, about what happened during the simulation in each script scenario. This file is called a *Trace* or a *Log* file as we mentioned in chapter 2 section 2.4.2.6, and the trace class is the class that responsible for creating and writing this file.

4.7.7 Packet Implementation

The main purpose of using the packets is to exchange the information between the objects (nodes) during the simulation time. The MDVZRP protocol has two types of packets, the control packets (Beacon, FRIP, RSUP, RREQ and RREP), and the data packets (Tcp, Cbr and Ack). We made sure by adding these packets that the nodes in the MDVZRP protocol have the ability to send and receive them.

For dealing with any received packet we have to access and read the packet header first, which has been stored by packet class, by using an array of unsigned characters where packets fields are stored. We have put all data structures, constants and macros related to the protocol packet(s) type in *common/packet.h* header file as mentioned in the previous section, and shown in Figure 4.19. (This structure is common in most of Ad Hoc protocols, except the new fields that we have added for MDVZRP).

common/packet.h

```
#define HDR_CMN(p) hdr_cmn::access(p)) //macro to access hdr_cmn
.
#define HDR_IP(p)   (hdr_ip::access(p)) //macro to access hdr_ip
.
.

struct hdr_cmn
{
    enum dir_t { DOWN= -1, NONE= 0, UP= 1 };
    packet_t ptype_;        // packet type
    int      size_;         // simulated packet size
    int      uid_;          // unique id
    int      error_;        // error flag
    int      iface_;        // receiving interface (label)
    dir_t direction_;       // direction: 0=none, 1=up, -1=down

    //MDVZRP extn begins
    nsaddr_t prev_hop_;       // IP addr of forwarding hop
    nsaddr_t next_hop_;       // next hop for this packet
    nsaddr_t fst_fwd_hop_;    // first hop of Route Request (MDVZRP)
    nsaddr_t p_prev_hop_;     // pre-previous hop of Route Request(MDVZRP)
    int num_forwards_;        // how many times this pkt was forwarded
.
.

static int offset_;     // offset for this header
inline static int& offset() { return offset_; }
inline static hdr_cmn* access(const Packet* p) {
return (hdr_cmn*) p->access(offset_);}
}
```

Figure 4.19: MDVZRP Packet CMN Header, based on DSDV CMN header

The *common header* structure for this protocol packet has been declared in lines 468-518. For accessing a packet header we need to set a static offset, a member function to give us the ability to access that offset, and another function to return that given packet header as shown in lines 515 – 518. While in line 53 the macro which uses the function has been declared. In line 63 another macro has been declared which is use the function that returns the ip header of a given packet.

The *ip header* structure for this protocol's packet has been declared in Figure 4.20(A). In the two first lines we defined *ip* header length by 20 bytes, and time to live (TTL) in 32 hops, these hops are decreased by one each time the packet is forwarded, this is to prevent the packet from going far away, when any node receives that packet and finds its TTL is equal to zero it will drop it immediately.

common/ip.h

```
47: #define IP_HDR_LEN      20 Bytes
48: #define IP_DEF_TTL      32 Bytes
.
60: struct hdr_ip {
    /* common to IPv{4,6} */
61:   ns_addr_t   src_;
62:   ns_addr_t   dst_;
63:   int         ttl_;
      /* IPv6 */
71:   int         fid_; /* flow id */
72:   int         prio_;

74:   static int offset_;
75:   inline static int& offset() { return offset_; }
76:   inline static hdr_ip* access(const Packet* p) {
77:         return (hdr_ip*) p->access(offset_);
      }

     /* per-field member acces functions */
81:   ns_addr_t& src() { return (src_); }
82:   nsaddr_t& saddr() { return (src_.addr_); }
83:   int32_t& sport() { return src_.port_;}

84:   ns_addr_t& dst() { return (dst_); }
85:   nsaddr_t& daddr() { return (dst_.addr_); }
86:   nt32_t& dport() { return dst_.port_;}
87:   int& ttl() { return (ttl_); }
      /* ipv6 fields */
89:   int& flowid() { return (fid_); }
90:   int& prio() { return (prio_); }
};
```

Figure 4.20(A): MDVZRP Packet IP Header, based on DSDV ip.h

Lines 61-72 show the *ip* header fields, the 1st one is a packet *Source* (src_ the ip address of the node that generated the packet), the 2nd field is a *destination* (dst_ the ip address of the node that the packet is send to), both fields are *ns_addr_t* type (an integer 32 bits Ns2 simulator address type), the 3rd field is for *TTL*.

The last two fields are for Internet Protocol version 6 (IPv6), which is an Internet Layer protocol for packet-switched internetworks, and was designed by IETF as the successor to the previous version IPv4, which was designed for internet general use. Lines 74-77 show a static offset setting for accessing the packet ip header and two member functions for accessing that offset and returning the given packet ip header. While lines 81-90 show functions that return the value of each ip header field.

common/packet.h

```
enum packet_t
 {
       PT_TCP,  // Packet  0   DATA
       PT_UDP,
       PT_CBR,  // Packet 2 A traffic type where the amount of traffic is constant per unit of time
        .
        .
       PT_ACK,  // Packet 5   DATA Acknowledgment
        .
        .
        .
       PT_BEACON,                // Packet 20 Hello  MDVZRP
       PT_FRIP,                  // Packet 22 Full routing information  MDVZRP

       PT_ROUTE_UPDATE_PACKET,   // Packet 23 Partial routing information MDVZRP
        .
        .
       PT_ROUTE_REQUEST,         // Packet 26 Route Request Packet  MDVZRP

       PT_ROUTE_REPLY,           // Packet 27 Route Reply Packet for  MDVZRP
        .
        .
        .
       PT_TELNET,    // not needed: telnet use TCP
       PT_FTP,
       PT_HTTP,
        .
        .
       PT_BLTRACE,  // Bell Labs Traffic Trace Type (PackMime OL)
        .
       // insert new packet types here
       PT_NTYPE  // This MUST be the LAST one
};
```

Figure 4.20(B): MDVZRP Packets Numbers at packet.h

4.7.8 Routing Agent Implementation

We assume that MDVZRP is a hybrid routing protocol that periodically requires some control packets such as BEACON and RSUP (Routes Update Packets) to be sent out. This protocol needs timers to do that job in scheduling, and *RREQ* when a node needs to send data to a specific destination where it does not have a route to that destination in its routing table.

This section discusses the MDVZRP agent class, which contains all the member functions and attributes that carry out the protocol functionality together.

The agent base class is defined and placed in an *mdvzrp/mdvzrp.h* file, while the packet's IP header and class are defined and placed in the following file *common/ip.h.*

For sending a periodical control packet such as BEACON and RSUP, the *scheduler.h* has been implemented where a *Handler* base class which is used for creating the protocol timers is defined. The *trace.h* defines the *Trace* class, used for writing simulation results out to a trace file. The protocol's routing table and its required attributes are defined in *MDVZRP_rtable.h* file.

At the beginning of the Agent we have defined the maximum queue length for the dropped packets, the router port address, where each node is provided by an address classifier and a port classifier, the port address is used by the port classifier object. This is responsible for giving the received packets to the corresponding agent, while the address classifier diverts and guides the received packets either to a suitable link or to pass them (packets) to the port classifier, which will carry them to the right upper layer agent.

Two jitters also are defined in the implementation of the agent; the first one is used for events that should be effectively instantaneous to prevent synchronisation in between, while the second one is a jitter for all broadcast packets. In the following subsections we will clarify the agent's member functions in more details. The agent's member functions are listed in Table 4.7 according to the MDVZRP mechanism and its packet's action sequence:

Table 4.7: MDVZRP Agent's Member Functions

Function Name	Function type	Pass parameters
Start Up function	`void`	`void`
Helper Call Back Function	`void`	`Event * e`
Make a Beacon	`Packet *`	`void`
Make FRIP	`Packet *`	`An integer and nsaddr_t`
Make RSUP	`Packet *`	`An integer`
Send Out Broadcast Packet	`void`	`Packet *`
Mac Call Back Function	`Static void`	`Packet * p, void *arg`
Lost Link Function	`void`	`Packet *`
Receive Function	`void`	`Packet * p, Handler *`
Receive beacon function	`void`	`Packet * p`
Receive a FRIP function	`void`	`Packet * p`
Receive a RSUP function	`void`	`Packet * p`
Process Function	`void`	`Packet * p`
Forward Function	`void`	`Packet * p`
Make a RREQ Function	`Packet *`	`nsaddr_t, nsaddr_t`
Make RREP Function	`Packet *`	`four nsaddr_t, and two integers`
Error message function	`void`	`five integer a route information`

4.7.9 Start Up function

A start-up is a function automatically called once at each node in a network at the beginning of each scenario. The pseudocode in Figure 4.21 clarifies this function, where at the beginning each node creates a new entry for itself by setting destination (dst) and first next hop (f_nxt_hop) fields to its IP address (my address). While the 2nd next hop (s_nxt_hop) is set to null (no next hop for this entry), and number of hops (metric) is reset to zero. The link_num field carries a unique number indicates the link between two nodes, in this case the node needs to generate a link between it and itself (i.e. link_num of node 3 is 30003, while link_num between 3 and 4 could be 30004 or 40003).

Setting the advertise field to false means that entry belongs to that node only. Hence no need to advertise it. In addition, there is no need to buffer any packet in that entry's queue by resetting the q field (q=Null). The *timeout_event* field (a neighbour's interval time of next expected beacon) is set only when an entry for a new neighbour is being created, while for a node itself and all other nodes it is simply set to NULL (Do not time out our local host). The node needs to add the simulation time at which that entry has been created before saves that entry into its routing table (RT) by setting *changed_at* filed to now. Finally, it prepares to broadcast a first beacon in other words kick off periodic advertisements by setting and scheduling a new event or beacon.

```
Void MDVZRP_Agent::startUp()
{ // Start of startUp function
   Define and set now to Scheduler time; //define now variable to double

   // each node creates a new entry for itself
    define rte to a routing table entry;   // rte is a new entry (array)
     rte dst = myaddress;  // a new route destination
     rte f_nxt_hop = myaddress;  // next hop (1st hop)
     rte s_nxt_hop = Null; // 2st hop for this entry is NULL=-99
     rte metric = Null;  // distance is zero (number of hops)
     rte link_num = myaddress*10000+myaddress; //link number
     rte advertise = false; // do not advertise this route
     rte q = Null;  // don't buffer pkts for self

   // Don't time out our local host, only for adjacent nodes,
   // while for all other nodes it is simply set to NULL
     Set rte entry timeout_event to Null;
     Set rte entry changed_at to now;   // time of getting this route
   Add rte entry into my Routing Table; // save this new entry in my RT
   Schedule a first beacon time; // kick off periodic advertisements
} // End of startUp function
```

Figure 4.21: Start up Function pseudocode

4.7.10 Helper Call Back Function

The helper_callback is the function that receives an event either *a beacon or a time out*, and it is only effected by the scheduler whenever it is a node time for broadcasting a *Beacon* or reporting a *Broken link*, or by the *lost_link* function whenever it has received back non delivered data packets from its mac layer, because of a non reachable first next hop and is reporting a lost link.

The pseudocode in Figure 4.22 shows that a *helper_callback* function receives only an event (Event e), which could be a beacon time, or time out event (a non reachable adjacent). At the beginning of the function, the node that received the event, reads a scheduler time and saves it in a *now* variable for later use. It also, defines two arrays (*prte*) and (*pr2*) of routing table entry type (*MDVZRP_rtable_ent*) and a new packet *Packet p*. The *num_of_advertise_entry* and *immediately_update* are two flags; we will explain them later in this section.

We have divided the pseudocode in two stages, the 1st stage when the received event is a beacon time, the node that received the event goes through its routing table to check if there is any entry / entries need to advertise (where advertise field is true). If it is the case, the node includes and broadcasts that entry / entries in one RSUP packet. Otherwise, the node broadcasts an empty beacon packet. Each time the node that broadcast a beacon packet needs to set and reschedule its next beacon time.

The 2nd stage when the received a *Time out* event, the node that received the event goes through its routing table to check which entry (route) has time out event equal to the received one (entry timeout_event = received e) to assign it as a broken route (metric=Infinity), sets *immediately _update* flag to true for broadcasting an immediately update packet RSUP, and saves that entry *broken* in an array *prte*. Hence, breaks the loop as shown in the 1st loop (for loop) in the pseudocode in Figure 4.22.

Once the broken route is found and saved in an array, the node goes through its routing table again from the beginning. This is in order to assign each entry *route* where the first next hop *f_nxt_hop* is equal to the destination *dst* of the broken route *prte* to infinity too, as shown in the 2nd loop (for loop) in the function's pseudocode in Figure 4.22. When the node reaches the broken route again in the 2nd loop, it sets it's advertise field to true (advertise this route), for broadcasting later in the next RSUP as a broken route instead of sending a separate Error packet.

```
MDVZRP_Agent::helper_callback (Event * e)
{  // Start of helper call back function
   Define and set now to Scheduler time; //define now variable to double
   // Define two new routing table entries;// 2 arrays
    Set prte to routing table entry;      //prte is a new entry (array)
    Set pr2 to routing table entry;       //pr2 is a new entry (array)
    Locate and set p as a new packet;     // Packet *p
    num_of_advertise_entry = Null;  //set a flag to zero
    immediatly_update = Null;       //set a flag to zero

Stage1:
************************************************************************
// Check for a periodic call back (Beacon)
If (the received Event is a periodic Beacon)  // is it a Beacon
{
  // set a loop to read RT entry by entre
  For (not end of routing table, set prte to the current route)
    If (prte advertise field is true) //entry advertise field
         // Count only entries that need to advertise
         Increment num_of_advertise_entry;
 If (num_of_advertise_entry = Null) //no advertise
   p = make_BeaconMessage; // prepare a Beacon packet
 Else // Otherwise, prepare a RSUP
   p = makeRSUP(num_of_advertise_entry);

 If (there is certain information)
   Dump the packet's available information into the trace file

If (p) // if the packet (Beacon / RSUP) is ready
    //broadcast the packet (p) and place a periodic beacon time back
   // onto the scheduler queue for next time...
 Call sendOutBCastPkt1 (p);  // broadcast the packet (p)
 Return;
}

Stage2:
************************************************************************
  // Check for timeout event (BROKEN LINK)
For(not end of routing table, set prte to the current route)
  If (current route (prte) timeout = received Event)
   {
     immediatly_update = true;    // set a flag true
     Break; // ok break the loop, that is the route we are looking for
   }
   // Set a loop for my routing table, and assign as infinity any
     // entry that has first next hop = the prte destination
     // of the broken route no need to advertise it

   For (not end of routing table, set pr2 to the current route)
   {
    If (current route (pr2) first next hop = prte destination)
    {
     current route (pr2) metric = BIG;       // Big =infinity
     current route (pr2) changed_at = now;  // route update time
     If (it is a direct route)
        advertise field = true; // advertise it as a broken link
     Else
        advertise field = false; // no need to advertise it
      }
     Reset timeout_event; // an expired timeout, free it
    } //End of For

If (immediatly_update) //broadcast an immediately routing information
 {
   For (not end of routing table, set prte to the current route)
     //  Include any route where advertise field is true.
    If (advertise)
      //  Count entries that need to advertise, where advertise
      // field is true
     Increment num_of_advertise_entry;
     p = makeRSUP(num_of_advertise_entry); // make Routes update packet
If (p) //If the packet (RSUP) is ready
      // broadcast it (RSUP takes the role of beacon this time)
      Call sendOutBCastPkt1 (p);// broadcast the packet (p)
 }
} //End of helper call back function
```

Figure 4.22: Helper call back Function pseudocode

4.7.11 Make a Beacon Message

The *make_Beacon* Message is a function periodically called through the *helper_callback* function whenever a node needs to broadcast a *Beacon*.

The function's main goal is to prepare and return a *beacon* packet for broadcasting. This is carried out by setting the following header fields: the *packet type* header command field to *BEACON*, the packet's *source address* to the *IP* address of the node that needs to broadcast a packet, a first node and pre previous node that forwarded that packet to null, without an address by setting these two fields *fst_fwd_hop_* and *p_prev_hop_* to *null*. We have used -99 as a null instead of 0, because sometimes 0 is used as a node address as shown in Figure 4.23. The function returns a beacon packet for broadcasting using the *broadcastpkt1* function which we explain in *Send Out Broadcast Packet* in section 4.7.14.

```
DVZRP_Agent::make_BeaconMessage()
{// Start of make Beacon message function
   Define and allocate a Packet (p);
   Access the packet’s ip header;
   Access the packet’s hdrc header common;

   Define and set now to Scheduler time; //define now variable to double
   Set hdrc packet type to PT_BEACON; //Packet 20 Hello MDVZRP
   Set ip saddr to my address; // source ip address to my address
   hdrc fst_fwd_hop_ field = Null;  // first forward hop = -99
   hdrc p_prev_hop_  field = Null; // pre-previous hop field = -99

   //Add the header to the packet
   Set hdrc packet size field to IP header length;
   Return the packet (p);     // return p for broadcasting

}// End of make Beacon message function
```

Figure 4.23: Make a Beacon Function pseudocode

4.7.12 Make a FRIP Function

The makeFRIP (make Full Routing Information Packet) function is called whenever a node receives a broadcast packet (BEACON / RSUP) or a forward packet (RREP /RREQ) from a new neighbour.

The FRIP packet informs the new neighbour that it can access all the destinations included in that FRIP packet using the sender of the FRIP packet as the first hop towards each destination listed in the FRIP packet.

The *makeFRIP* function takes two parameters and returns a FRIP packet. The two parameters are an integer *new_or_broken* and the address of the node that sent a packet *dst*. The 1st one *new_or_broken* has two values (0, 1). A 0 means that destination *dst* in the 2nd parameter is an old non reachable node because of a broken link. While 1 means that the destination *dst* is a new node, therefore the receiver node needs to add a new entry into its routing table to that destination *dst* by calling *new_entry_intialisation* function as shown in the pseudocode in Figure 4.24.

In the 1st stage of *makeFRIP* function, the receiver node counts the number of routes that needs to include in a FRIP packet, excluding any broken links, only one route to each destination needs to be included in securing the best route (shortest).

The FRIP packet is not a broadcast packet but is a unicast or 'send to a specific destination', as shown in Figure 4.11. Therefore, we have set the next hop *next_hop_* and destination's address *iph daddr* both to the new node's address *dst*. The source's IP address (*iph saddr*) is set to the address of the node that goes to the unicast the FRIP packet (*myaddress*).

Because this packet is still under construction and this will be its first transmission, the *prev_hop_* field is set to *myaddress*, *fst_fwd_hop_* field and *p_prev_hop_* field are both set to null.

A FRIP packet is designed to carry 17 bytes for each included route as shown in the following format in table 4.8. The first 4 bytes are for destination, the 2nd and 3rd 8 bytes are for first next hop and second next hop towards that destination in respectively, while the last five bytes are 1 byte for metric and 4 bytes for link_num.

Table 4.8: Format of the included routes (entries) in a FRIP packet

4 Bytes	4 Bytes	4 Bytes	1 Byte	4 Bytes
dst	f_nxt_hop	s_nxt_hop	metric	link_num

The total FRIP packet size in bytes is 17 × number of included entries +1Byte (num of included route) + IP's header length.

In the 2nd stage of the *makeFRIP* function, the receiver node starts to include 17 bytes for each selected entry in the FRIP packet. When all the selected routes are included it returns the packet for unicasting.

```
MDVZRP_Agent::makeFRIP(int new_or_broken, nsaddr_t dst)
{   // Start of make full routing information packet function

    Defined and allocate a Packet(p);
    Access the packet's ip header;
    Access the packet's hdrc header common;
```

```
   Define and set now to Scheduler time; // define now variable to double
   Set prte to routing table entry; // prte is a new entry (array)

   If (a new neighbour) // new_or_broken is set to Null
    Call new_entry_intialization(dst); // add a new entry in my RT

    num_of_advertise_entry = 0; // set some variables (change_count)

Stage1:
*********************************************************************
// set a loop to read RT entry by entre and get only 1 entry
// (shortest) to each destination, don't count any broken entry
   For (not end of routing table, set prte to the current route)
    {
     If (prte not a broken rout && the shortest)
     Increment num_of_advertise_entry; // change_count++
    // Prepare a packet to be unicast to dst (the new neighbour)
     hdrc prev_hop_ = myaddress; // I'm the previous
     hdrc next_hop_ = dst; // my new neighbour is the next hop
     hdrc fst_fwd_hop_ = Null; // not forwarded before
     hdrc p_prev_hop_  = Null; // no pre previous hop before
    hdrc packet ptype = PT_FRIP;   // Packet 22 Full routing information  MDVZRP

     iph daddr = dst; // destination IP address=my new neighbour
     iph destination port = ROUTER_PORT;
     iph saddr = myaddress; // Source IP address = my address
     iph sport = ROUTER_PORT;
     number of bytes to be included in this packet;
     // num of entries multiply by 17 + 1,where 17 = 4B for dst, 4B
     // f_nxt_hop, 4B s_nxt_hop, 1B for metric and 4B for link_num +
     // 1Byte for num of entries that included, the packet's size
     // MDVZRP + IP
     packet size = num_of_advertise_entry * 17 + IP_HDR_LEN;


Stage2:
*********************************************************************
 // set a loop to read RT entry by entry and include only 1
 // entry (shortest) to each destination, into FRIP packet
     For (not end of routing table, set prte to the current route)
      {
       If (prte not a broken rout && the shortest)
       {
         Include destination into the packet;                 // 4 bytes
         Include first next hop (f_nxt_hop) into the packet;  // 4 bytes
         Include second next hop (s_nxt_hop) into the packet; // 4 bytes
         Include metric into the packet;                      // 1 bytes
         Include link_num into the packet;                    // 4 bytes
        }
       Decrement num_of_advertise_entry;  // Change_count--
       advertise = false; // no need to advertise this entry again
      }
   }
Assert num_of_advertise_entry = 0; // abort if not zero (Error)
Return the packet; // return FRIP packet for unicasting

} // End of make full routing information packet function
```

Figure 4.24: Make a FRIP Packet Function pseudocode

4.7.13 Make a RSUP Function

The *makeRSUP* (make Routes Update Packet) function is called whenever a node needs to broadcast a periodic beacon packet, and there is a change in the network such as a new or broken route/routes needing to advertise (advertise field is true). In this case that node has to call the *makeRSUP* function to broadcast an update packet (RSUP) instead of calling the *make_Beacon* function to broadcast an empty one (BEACON).

The *makeRSUP* function receives a one parameter, which is the number of entries needed to advertise and return a RSUP packet as shown in Figure 4.9. The RSUP packet is designed to be a dual function by taking the beacon packet role, plus its main goal, which is an update routes packet as we mentioned in the previous section of the *helper_callback* function.

The mechanism of the *makeRSUP* function is same as the *makeFRIP* function, but the only difference between them is the method of selecting and propagating the update routes, where the *makeRSUP* is a function that creates and includes only the latest update routes that a node has got in its routing table (advertise field is true), excluding the broken routes in a broadcast update packet *RSUP* as shown in stage 1 in Figure 4.25. While the *makeFRIP* function creates and includes a single route for each destination that the node has got in its routing table within a unicast update packet *FRIP* as shown previously in Figure 4.24.

```
MDVZRP_Agent::makeRSUP(int change_count)
{ // Start  of make routes update packet function
   Defined and allocate a Packet(p);
   Access the packet’s ip header;
   Access the packet’s hdrc header common;
   Set prte to routing table entry;  // prte is a new entry (array)

   //Prepare a packet to be broadcast to all 1 hop neighbours
    hdrc ptype = PT_ROUTE_UPDATE_PACKET; //Packet 23 MDVZRP
   iph saddr = myaddress;    // source’s address is my address
   hdrc fst_fwd_hop_  = Null; // not forwarded before
   hdrc p_prev_hop_  = Null;  // no pre previous hop before
   //the rest fields will fill in sendOutBCastPkt1 function

   Set number of bytes to be included in this packet;
   //number of entries multiply by 17 + 1, where 17 = 4B for dst,
     // 4B f_nxt_hop, 4B s_nxt_hop, 1B for metric and 4B for link_num +
     // 1Byte for number of entries
   packet size = change_count * 17 + IP_HDR_LEN;
   //change_count is the number of entries need to advertise, it is
     // a pass parameter to makeRSUP function
Stage1:
**********************************************************************
//Filling the Routes Update Packet
   For (not end of routing table, set prte to the current route)
     {
      //Include any route where advertise field is true.
      If (current route prte advertise is true) {
      Include destination into the packet;                      // 4 bytes
      Include first next hop (f_nxt_hop)  into the packet; // 4 bytes
      Include second next hop (s_nxt_hop) into the packet; // 4 bytes
      Include metric into the packet;                          // 1 bytes
      Include link_num into the packet;                          // 4 bytes
      Decrement Change_count;
      advertise = false; // Reset, no need to advertise this entry again}
     }
   Assert change_count = 0; // abort if not zero (Error)
   Return the packet;
} // End of make routes update packet function
```

Figure 4.25: Make a RSUP Packet Function pseudocode

4.7.14 Send Out Broadcast Packet

Any broadcast packet must be passed to the *sendOutBCastPkt1* function, which puts the received packet in the scheduler queue after completing the rest of the packet header fields for broadcasting as shown in the pseudocode in Figure 4.26, where the function reads that packet's *ip* and *common* headers and sets the header's fields as follows:

The previous hop field *prev_hop_* to the node address that broadcasts that packet (*my address*), the next hop *next_hop_* and destination address *dadd* fields both to IP_BROADCAST (send to all nodes that are within one hop distance, within node's transmission range), and the packet's direction field to DOWN.

We have used a jitter to avoid synchronisation in case more than one node needs to broadcast a packet at the same time. Finally, if the broadcast packet is not a beacon it needs to cancel the next periodic beacon time from the scheduler, hence it puts a new periodic beacon time onto the scheduler as such as a periodic beacon do.

```
MDVZRP_Agent::sendOutBCastPkt1(Packet *p)
{   // Start of send out a broadcast packet function
   Access and read the packet's(p) ip header;
   Access and read the packet's(p) hdrc header common;
   Define and set now to Scheduler time;   // define now variable to double
   hdrc prev_hop_ = my address; // previous hop  = myaddr
   hdrc next_hop_ = Broadcast;  // next destination
```

```
    ip daddr = Broadcast;          // destination IP address
    ip dport = ROUTER_PORT;        // IP destination port
    ip sport = ROUTER_PORT;        // IP source
    hdrc direction to DOWN;
    //jitter to avoid synchronisation
    Put the packet onto the scheduler queue;
    // s.schedule (target_, p, jitter(MDVZRP_BROADCAST_JITTER, be_random_));
    If (not PT_BEACON)      // not a Beacon
    Cancel the next periodic Beacon from the scheduler;
    // s.cancel(periodic_Beacon_); //ns2 standard command
    Put a new periodic Beacon time onto the scheduler;
    // s.schedule (helper_,periodic_Beacon_,perup_*(0.75 +
    //jitter(0.25,be_random_)); // ns2 standard command
    Return;
} // End of send out a broadcast packet function
```

Figure 4.26: Send Out Broadcast Packet Function pseudocode

4.7.15 Mac Call Back Function

The data link layer is one of OSI model layers (layer number 2). It consists of two sub-layers, the Logic Link Control (LLC) and the Media Access Control (MAC) (Roy, 2011). The MAC sub-layer is responsible for transferring data packets to and from one device network interface card (NIC) to another in same network using the available media (wired / wireless). Therefore it is also known as a data communication protocol sub-layer. The *mac_callback* is a member function that is called whenever the MAC layer returns back none delivered packets (the packets that the MAC layer has failed to deliver).

Mobility is the biggest issue we face in Ad Hoc networks, where nodes can move freely at any time to any place. Therefore, during the data packets transmission (forwarding) if the next hop does not exist anymore (non reachable) the *mac_callback* is activated and receives the packets that have returned back from the MAC layer to report a lost link to any destination via that next hop by calling *lost_link* function, as shown in the *mac_callback* function pseudocode in Figure 4.27.

```
mac_callback (Packet * p)
{ // Start of mac_caback  function
    Access and read the packet's(p) ip header;
    Access and read the packet's(p) hdrc header common;
    // Send back non delivered packet to lost link function
    Call lost_link (p);
} // End of  mac_callback function
```

Figure 4.27: Mac Call Back Function pseudocode

4.7.16 Lost Link Function

The lost link and mac_callback are two related functions that complete each other's jobs. The *lost_link* function is called each time through the *mac_callback* whenever it needs to report a non reachable next hop *f_nxt_hop*, by passing the non delivered packet to the *lost_link* function to delete that non reachable node (next hop field in CMN header). It also deletes any accessible destination via that node by passing the non reachable node*'s timeout_event* to *helper_callback* function, which has been described in the previous sections and shown in Figure 4.22.

The node that receives the non delivered packet has to check its routing table to find an alternative route to that destination via another next hop. Otherwise, if there is no route to that destination, it has to drop that packet as shown in the *lost_link* function pseudocode in Figure 4.28.

```
MDVZRP_Agent::lost_link (Packet *p)
{ // Start of lost link  function
   Define and set now to Scheduler time; // define now variable to double
   Access and read the packet's ip header;
   Access and read the packet's hdrc header common;
   int src = the packet source IP address;
   int dst = the packet destination IP address;
   Define prte to routing table entry type;// prte is a new entry (array)
   // fined a direct route to the non reachable node (next hop)
   Prte = Call GetEntry1 (next_hop,next_hop,myaddress);
   If (the packet is returned by mac)
   {
     If (prte && prte timeout_event) // is timeout_event
     {
     // there is a direct route and it has the same timeout_event
     Cancel that timeout_event from the Scheduler;
     Call helper_callback (timeout_event); // report a lost link
     }
      // Try to find an alternative route to send this packet by
        // Calling GetEntry2 function
           prte = GetEntry2(dst,-99,&myaddress); // get the best route to dst
     If (no route available or the available one is broken)
         // drop the packet because no an alternative route available
           Call Drop (p, DROP_RTR_MAC_CALLBACK); // drop the packet, reason
     Else
         // the Node found an alternative route to send that packet
     If (there is certain information)
          Dump the packet's available information into the trace file
         Assert (no packet failure by mac_callback); // abort if an Error
        Goto send (p);  // jump to send procedure in the forward function
     }
     Return;
} // End of lost link function
```

Figure 4.28: Lost Link Function pseudocode

4.7.17 Receive Function

The recv function is called whenever a packet is received from the same agent (Upper layer), or from a different agent (Another node). When a packet is received the node that received that packet checks if it is that packet's original sender (the number of forwards =0). This means that, it has received a packet that originated from it. Therefore, it needs to add its IP header to that packet and hence forward it. Otherwise, it received a packet from itself (Probably a routing loop). Therefore, it should drop that packet as shown in stage 1 in Figure 4.29.

If the received packet sent by another source, the receiver node checks if it is the destination, and calls the process function to process that packet, we clarify the process function in the next section. Otherwise, the received packet belongs to another node. Therefore, the receiver node needs to see if it has an available route to forward that packet to the right next hop by calling the *forwardpacket* function as shown in stage 2 in Figure 4.29. We also explain the *forwardpacket* member function in the next sections.

```
recv (Packet * Packet p, Handler)
  {  // Start of receive function

     Access and read the packet's (p) ip header;
     Access and read the packet's (p) hdrc header common;
Stage1:
************************************************************************
// see if I'm that packet (p) original sender

     If (packet sender = myaddress && num of forwards is Null)
      Add the IP Header;     // I'm the original sender
     Else if (packet sender = myaddress)
      // A node received a packet that it is sent probably a
       // routing loop
      Call drop packet (p); // drop received packet
     Return;
Stage2:
************************************************************************
// see if that packet (p) is sent to me, otherwise let us forward it

     If ((packet ip sender not myaddress)&&(not ROUTER_PORT))
       Call process (p); // process this packet
     Else
       // That packet(P) is belongs to another node,
         // let us see if we can forward it to its target.
       Call forwardpacket (P);

  } // End of receive function
```

Figure 4.29: The Receive Function

4.7.18 Process Function

Process is a function called whenever a node receives a packet, that packet could be a BEACON, RSUP, FRIP, RREQ or RREP. The aim of the *process* function is to select the right function depending on the received packet type as shown in the function's pseudocode in Figure 4.30. We are going to clarify each function called *recv* function in detail in the following sections.

```
Process (Packet * Packet p)
{  // Start of process function

   Access and read the packet's (P) ip header;
   Access and read the packet's (p) hdrc header common;

    If (the packet type is a BEACON)  // if a beacon (Hello) packet
     Call receive_Beacon function;

    If (the packet type is a FRIP)    // if full routing information packet
     Call receive_FRIP function;

    If (the packet type is a RSUP)  // if a Route Update packet
     Call receive_RSUP function;

    If (the packet type is a RREQ_PACKET)  // if a ROUTE REQUIST packet
     Call receive_RREQ function;

    If (the packet type is a RREP_PACKET)  // if a ROUTE REPLY  packet
     Call receive_RREP function;

}  // End of process function
```

Figure 4.30: Process Function pseudocode

4.7.19 Receive Beacon Function

The *receive_Beacon* function is called whenever a node receives a beacon. Each node expects to periodically receive a beacon packet from each adjacent within each node's beacon interval by setting the *timout-event* field. It is an event used to schedule timeout action, in other words nodes that are within transmission range, while for all other nodes it is simply set to NULL. Otherwise, it considers the neighbour that did not broadcast a beacon during its expected beacon interval to be a non reachable neighbour. This is indicated by setting the metric field to infinity in the entry with non reachable node as destination.

The pseudo code for receive beacon is listed in Figure 4.31 shows the *receive_Beacon* function mechanism. The node that receives a beacon packet reads the packet headers and checks its routing table to ascertain whether that packet sender is a known neighbour or not using *checkNeighbour* (check a neighbour) function. If it is a known neighbour, the beacon packet receiver node updates the next periodic beacon interval time of that neighbour (beacon packet sender). Otherwise, it considers that node as a new neighbour and sets its next periodic beacon interval time.

```
receive_Beacon (Packet * p)
{   // Start of receive Beacon function

    Access and read the packet's (p) ip header;
    Access and read the packet's (p) hdrc header common;
    Get the packet's (p) previous hop ip address; // Packet's previous_hop
    // check if previous hop is a neighbour
    Call checkNeighbour(previous_hop ip address);
    Return;

} // End of receive Beacon function
```

Figure 4.31: Receive Beacon Function pseudocode

A BEACON is a periodic broadcast packet the same as an RSUP, where the packet sender IP address, a previous hop IP address are the same and the packet's TTL is 1 (it has only one hop). So, we created a function called *checkNeighbour* to check if the previous node that forwarded that packet is a known neighbour or not, instead of checking a packet sender itself, because packet sender is not always an adjacent, such as the case in RREQ and RREP packets. Therefore, we have passed the previous node IP address to the *checkNeighbour* function instead of the packet sender IP address.

4.7.20 Check Neighbour Function

The checkNeighbour (check the received packet's previous hop) is a member function called whenever a node receives a control packet BEACON, RSUP, FRIP, RREQ or RREP to check if that packet previous hop (the last node that forwarded that packet) is a known neighbour, and still exists, to reschedule its next expected periodic beacon time *timout-event* field as shown in stage 1 in Figure 4.32, a known non reachable neighbour *broken link* as shown in stage 2 in Figure 4.32 or a new neighbour to set and schedule its next expected periodic beacon time, hence it unicasts a new FRIP packet to that node (*previous hop*) by calling a *make_FRIP* function immediately as shown in stage 3 in

Figure 4.32. Finally, as shown in stage 4 the function checks if there are any packets that are queued that it can send off to that new neighbour.

```
CheckNeighbour (nsaddr_t src)
 {  // Start of check Neighbour function
   Define prte to routing table entry type; //prte is a new entry (array)
   Locate and define p as a packet; // Packet *p
   Set prte to NULL; // initialise prte array (entry)
   // get the direct route if available for that destination (src)
   prte callGetEntry1 (src, src,&myaddress);  // Get the direct route
   int is_it_a_neighbor = Null; // set a flag to false
Stage1:
**********************************************************************
// see if the source (src) is a known neighbour
     If (prte route && prte metric is 1)  // it is a Known neighbour
      {
       // This is a known neighbour, cancel its next expected
       // periodic beacon time from the scheduler
       Cancel Scheduler prte timeout_event;
        // Reschedule its next expected periodic beacon time, where
        // jitter, min_update_periods_ and perup_ are 45, 15 and 3
        //respectively such as in DSDV
       Call schedule (prte timeout_event, min_update_periods_* perup_);
       Set is_it_a_neighbor field to true; //set a flag to true
      }
     Else
Stage2:
**********************************************************************
//  see if the source (src) is a known and non reachable neighbour
      If (a prte route && broken) // a known non reachable neighbour (old)
       {
        //received a beacon from an old non reachable neighbour
        prte metric =1; // repair broken route
        prte timeout_event = new Event;// a new neighbour
        prte advertise = true; // advertise this route
        prte changed_at = now; // set route's time field to now

       //Inform that neighbour it can access all the destinations in
       // FRIP packet through me (FRIP sender) as a next hop
        p = makeFRIP(1, source (src)); //make an immediately update packet
        If (p)
         {
          Assert (no failure in cmn header); // abort in case of failure
          Unicast that packet (p) to my neighbour; // destination = source
         }
        Reschedule a next expected periodic beacon time;
        is_it_a_neighbor = true;
       }
Stage3:
**********************************************************************
// source (src) is a new neighbour
      Else
       {
         // received a beacon from a new neighbour
         p = makeFRIP(0, src); // make an immediately update packet
         If (p)
         {
           Assert (no failure in cmn header); // abort in case of failure
           Unicast that packet (p) to my neighbour; // destination = source
         }
         is_it_a_neighbor = true;
       }
        If (there is certain information)
         Dump the packet's available information into the trace file
Stage4:
*************************************************************************
// see if we can send off any packets we've queued for this neighbour
       If (prte route)
         If (prte q) // is there any packet in the queue
          {
            Set prte to routing table entry; //prte is a new entry (array)
            Set prte to Null;  // Initialisation of new array (entry)
            While (prte queue)
              Give the packets to ourselves to forward;
          // reset the queue, of that destination's route (entry)
          Delete rte entry queue;
          SET rte entry queue to NULL;
          // Copy rte's fields to where prte pointing in my RT
          Copy back rte entry onto prte entry; // write new rte in my RT
           }
Return (is_it_a_neighbor); // return the flag value

} // The end of check Neighbour function
```

Figure 4.32: Check Neighbour Function pseudocode

4.7.21 Receive a FRIP Function

The receive_FRIP is a member function called whenever a node receives a full routing information packet FRIP form an adjacent. As we mentioned previously a FRIP packet gives the receiver node ability to access all the destinations included within that packet via the FRIP packet sender, a neighbour, by unicasting all the routes that it has in one FRIP packet excluding any broken route.

The *receive_FRIP* function reads the received packet's headers and calls the *checkNeighbor* function to check if the previous node that forwarded that packet (FRIP sender) is a known neighbour. Otherwise it considers that node to be a new neighbour by adding a new entry in its routing table to that node and sending a new FRIP to it as we mentioned in the previous section.

The mechanism of the *receive_FRIP* function is divided into three stages as shown in the function's pseudocode in Figure 4.33. In the 1st stage it extracts the data packet and starts to deal with the included entries one by one in a loop form to see if it can acquire any new routes by calling the updateRoute function in the 2nd stage each time it prepares an entry where it excludes longer, known unless better than the existing one, and in loop form routes. While in the 3rd stage it checks if can send off any packets it has got queued for that destination in the direct route. Packets are queued in the direct routes only.

```
void  MDVZRP_Agent::receive_FRIP(Packet * p)
{   // Start of receive FRIP function

   Access and read the packet's (p) ip header;
   Access and read the packet's (p) hdrc header common;

   // it's a MDVZRP packet
   Set rte  to routing table entry;  //rte  is a new entry (array)
   Set prte to routing table entry;  //prte is a new entry (array)
   Set pr2  to routing table entry; // pr2  is a new entry (array)

   Set FRIP_sender to iph previous hop address;
   dst = nsaddr_t;          // ns2 address format
   change_count = Null;    // change_count=0
   modify_rt = Null;       // flag for updating RT

   // Check if this packet sender (previous hop) is a neighbour
    yes_it_is_a_neighbr = Call checkNeighbor(hdrc prev_hop_);

Stage1:
**********************************************************************//
// the Node is going to deal with FRIP packet entry by entry
   For (not end of num of entries included)
   {
     Initialise rte entry as an array;
// extracting data (included entries) from the FRIP update Packet..
rte dst = destination included in the packet;     // 4 bytes
rte f_nxt_hop = f_nxt_hop included in the packet; // 4 bytes
rte s_nxt_hop = s_nxt_hop included in the packet; // 4 bytes
rte metric = metric included in the packet;       // 1 bytes
rte link_num = link_num included in the packet;   // 4 bytes
  If (myaddress = rte entry dst)
   {  // this entry belongs to me
     rte dst = FRIP_sender;
     If (rte entry metric > 1)
       // don't consider this entry because it is same as a loop,
        // access dst(FRIP_sender) through myself…!!
     Continue;
   }
 Else
  {
   If (rte entry metric != Null)  // Null =0
    {
      // this entry belongs to the FRIP_sender destination (dst)
     Set rte entry s_nxt_hop to Null;// a direct route 2nd hop should be -99
      // Create a new link number instead of the one that received. It
     // should be between the receiver node (myself) and the FRIP
     // packet sender
     rte entry link_num = myaddress * 10000 + rte entry dst;
    }
   Else
    rte s_nxt_hop (2nd hop)= rte entry f_nxt_hop (1st hop);
      If (rte not a broken route) // not a broken link
       metric++; Increment rte entry metric
   }
 rte f_nxt_hop = FRIP_sender;
 rte advertise = true; // I'd like to advertise this route
 rte changed_at = now;

Stage2:
***********************************************************************
  // Decide whether to update our routing table
  // Check if you have a route to this destination (dst) through the
  // first next hop (f_nxt_hop)

 prte = Call GetEntry1 (rte entry dst, FRIP_sender,&myaddress);
 If (not prte) // no route to that dst
  {
   // we've heard about a brand new destination
   If (rte metric = 1 && rte dst = rte f_nxt_hop)
    {
         // This entry belongs to a new neighbour
         // Consider this destination (dst) as a new neighbour;
      Set rte entry timeout_event to new Event;
      Put this neighbour expected periodic time in the scheduler;
    }
        // Try to add this entry in its routing table (receiver node)
     int xx =1;  // a flag
     int write = call updateRoute(NULL, &rte,xx);
     If (write) // a flag to see if the new entry is added or not
      {
        modify_rt++; // my Routing Table (RT) is modified
        Call updateRoute(NULL, &rte);
      }
     Else if (rte entry metric better than prte entry metric)
      {
       // The route is found ... choose the best
       If (rte entry dst = rte entry f_nxt_hop) //direct route (adjacent)
          // Received an entry belongs to the sender in one hop so,
          // consider it as a neighbour and set its timeout
         Set rte entry timeout_event to new Event;
         int xx = 2;
         int write = updateRoute(prte, &rte,xx);
        If (write)
           modify_rt++; // my Routing Table (RT) is modified
      }
     Else
      {
          // Ignore this entry, because the receiver node has same or
          // better route to that destination (dst)
       Continue; // ignore the longer route
       }

Stage3:
***********************************************************************
  // See if we can send off any packets we've got queued for this
  // destination
  Reset prte entry; // prte = NULL
 If (rte entry dst != rte entry f_nxt_hop)
 {
    // the route we are dealing with is indirect route
    // let us see if we have a direct route
   prte entry = GetEntry1 (rte entry dst, rte entry dst,&myaddress);
    If (prte)
      Copy prte entry onto rte entry;
   }
   // because we got a new route to that destination (dst) MULTIPATH,
   // where we can use it to send off any queued packets!
     If (prte)
      {
        // check if we have a queued packets in this direct route
        // even this direct route is broken
        If (rte entry queue)
              Give the packets to ourselves to forward;
      // reset the queue of that destination's route (entry)
      Delete rte entry queue;
      SET rte entry queue to NULL;
      // copy all rte fields to where prte pointing in my RT
      Copy back rte entry onto prte entry; // write new rte in my RT
      }
     } // end of the FOR loop that reads FRIP (back to the next entry)

   If (modify_rt != Null)
      // the Routing table has been updated. // for debug
   Else
      // the Routing table has'nt updated. // for debug
} // End of receive FRIP function
```

Figure 4.33: Receive a FRIP Function pseudocode

4.7.22 Receive a RSUP Function

The *receive_RSUP* function is called whenever a node receives an update packet (Routes Update Packet) from an adjacent. The RSUP packet allows the receiving neighbour to update its routing table regarding the latest update from that node (the node that broadcast RSUP) and accessibility to any new destination included within the RSUP packet via the RSUP packet sender, which became a one hop neighbour. The RSUP packet is same as the FRIP function, where the only difference is that the RSUP is a broadcast packet. Therefore, it receives by all the adjacent nodes within the sender transmission range as shown in RSUP packet format in Figure 4.9.

The *receive_RSUP* function reads the received packet's headers and calls the *checkNeighbor* function to check if the previous node that forwarded that packet (RSUP packet sender) is a known neighbour. Otherwise, it should add a new entry to that node and send to it a new FRIP as we mentioned in the *receive_FRIP* function section. The mechanism of the *receive_RSUP* function is same as the *receive_FRIP* function which is divided into three stages. In the 1st stage as shown in Figure 4.34, it extracts the data packet and starts to deal with the included entries one by one in turn.

In the 2nd stage it checks if the received route (entry) belongs to a new destination by calling the *updateRoute* function each time it prepares an entry, where it excludes longer, known, unless it is better than the existing one and in loop form route. If the received route *entry* is broken and its destination is the RSUP packet receiver *myaddre_* where that is the same as a node telling an adjacent that the link between them is broken, while it is still in that node transmission range where it received its broadcasting. In this case that node (RSUP receiver) unicasts a new FRIP packet to the RSUP packet sender to update its routing table. Otherwise, it calls *Error_Msg* function in case the destination is another node; we are going to explain this function in the next sections.

While the 3rd stage it checks if it can send off any packets it has got queued for that new destination in its direct route, packets are queued in the direct routes only, as shown in the following pseudocode in Figure 4.34.

```
void MDVZRP_Agent::receive_RSUP (Packet * p)
{  // Start of receive RSUP function
   Access and read the packet’s (p) ip header;
   Access and read the packet’s (p) hdrc header common;
   // set a pointer (d) to the beginning of the data packet the 1st
   // byte of the data packet (which is the change count)
   Set unsigned char *d to the 1st data byte; // a pointer

   // set a pointer (w) to the 2nd byte of data packet (destination)
   Set unsigned char *w to d + 1;     // *w=d+1
   Set rte to routing table entry;    //rte is a new entry (array)
   Set prte to routing table entry;   //prte is a new entry (array)
   int xx; // a flag
   Initialise rte entry as an array;
   int modify_rt = Null;
   int change_count =  *d; // number of entries included in this packet
   int kk = change_count; // a counter for number of routes
   packet_sender = iph previous hop address;
   If (myaddress = previous hop) // previous hop is the packet sender
    { // drop the RSUP packet that I have sent (loop)
      Drop (p, DROP_RTR_ROUTE_LOOP);
      Return; }
   // Check if this packet sender (previous hop) is a neighbour
   yes_it_is_a_neighbr = checkNeighbor(hdrc prev_hop_);

Stage1:
***********************************************************************
 // the Node is going to deal with RSUP packet entry by entry
   While (change_count != Null) // num of entries included != 0
    {
     // Extracting data packet contents, byte by byte starting from
       //  1st byte where the w pointer points, by moving the w to a
       //  next byte each time.
      Set int destination to the 4 bytes contents where w points;
      // The pointer (w) is incremented 4 times, by 1 each byte.
      int first_hop = the next 4 bytes contents;  // w increments by 4
      int second_hop = the next 4 bytes contents; // w increments by 4
      int number_of_hops = the next bytes;        // w increments by 1
      int link_num = the next 4 bytes contents;   // w increments by 4
       If ( metric = Infinity) // a broken route
      If (the broken link is between me and a neighbour) // destination
      {
        //this neighbour is telling me, the link between us is
          // broke! I've to send it a FRIP, because I'm stil in its
          // transmission range by receiving this broadcast packet RSUP
        Set p = call makeFRIP(1, packet_sender)// Set new packet p to FRIP
        If (p)
         {
          Assert (no failure); // abort in case of failure
          Unicast that packet (p) to my neighbour; // destination
          }
        }
    Else
         //this means that the node received a broken route included within
          // the received RSUP instead of sending a separate Error packet
      If (second_hop = Null)   // Null= -99, a direct route
        Call Error_Msg(destination,packet_sender,destination, link_num);
      Else
        Call Error_Msg(destination, -1 , -1 , link_num);
  change_count--;
  modify_rt++; // my Routing Table (RT) is modified
  Continue;
  }
If (myaddress = first_hop OR myaddress = second_hop) // loop
 {
   //discard this entry because of loop
   change_count--; Decrement
   Continue;
 }

Stage2:
*********************************************************************
//Decide whether to update our routing table
//save the RT entry fields into rte array (a new entry)
      rte dst = destination;
      rte f_nxt_hop = packet_sender;
      rte s_nxt_hop = first_hop;
      rte metric = number_of_hops+1;
      rte link_num = link_num;
      rte advertise = true; // advertise this route later
      rte changed_at = now;
    If (myaddress != rte entry's destination (dst))
     { xx = 1;
        int write = call updateRoute(NULL, &rte,xx);
       If (write)
        modify_rt++; // my Routing Table (RT) is modified }

Stage3:
*********************************************************************
   //See if we can send off any packets we've got queued for that
  //  destination (dst), in the direct route (packets are queued in
  //  direct routes only)
Reset prte entry to NULL; // prte = NULL
 If (rte entry dst != rte entry hop) // indirect route
 {
  //the route we are dealing with is indirect route let us see if
    // we have a direct route
  prte = Call GetEntry1 (rte entry dst,rte entry dst,&myaddress);
 If (prte)
   Copy prte entry onto rte entry;
 }
//because we got a new route to this destination (dst) MULTIPATH,
   // where we can use it to send off any queued packets!
 If (prte)
 {
 //check if we have a queued packets in this direct route even
 // this direct route is broken
 If (rte entry queue) // is there any packet queued
  Give the packets to ourselves to forward;
 //reset the queue, of that destination's route (entry)
 Delete rte entry queue;
 rte entry = NULL;

 //copy all rte fields over where prte pointing in my RT
 Copy back rte entry onto prte entry; // write the new entry (rte) in my RT
 }
    change_count--; // number of included entries
   } // end of the FOR loop that reads RSUP (back to the next entry)
} // End of receive RSUP function
```

Figure 4.34: Receive a RSUP Function pseudocode

4.7.23 ForwardPacket Function

The *forwardPacket* function is calling to forward a packet to its correct destination. Whenever a node needs to send or receive a packet belonging to another node, it has to find the best route to that destination in its routing table for that packet. If the route is available and usable, it calls the send procedure to forward that packet to the right next hop as shown in the stage 1 in the function's pseudocode in Figure 4.35.

When the intermediate node forwarding the packet is not the packet original sender and the available route is broken, this means that packet's original sender and the previous node that forwarded the packet as well, if both are not same, have no idea that the route to that destination is broken, that destination is not accessible through me (the intermediate node). Therefore, the intermediate node (forwarder) broadcasts an immediately ERROR regarding the broken link and drops that packet.

For reducing the overhead we have included the broken route within the update packet (RSUP) instead of sending a separate Error packet as we mentioned previously in the RSUP section, and as shown in stage 2 of the function's pseudocode in Figure 4.35. When the packet's original sender has no route to that destination or the available route is broken, in this case it needs to check if it has a direct broken route to that destination. Otherwise, it creates a direct broken route (unusable), where metric = infinity to save that packet in that route’s queue.

The node should always check the queue in case it is full and drops a packet, depending on the queue type and its technique, before it queues another non forwarded packet. Hence, it broadcasts a RREQ packet for that destination as shown in stage 3 in the pseudocode in Figure 4.35.

```
Void forwardPacket(Packet * Packet p, Handler)
{  // Start of forward packet function
     Access and read the packet’s ip header;
     Access and read the packet’s hdrc header common;
     Define and set now to Scheduler time; //define now variable to double
     Define and Set prte to routing table entry;//prte is a new entry
     Define and Set int source (src) to iph source address;
     Define and Set int dst to iph destination address;
     Define and Set int change_count to null;
    //to forward a packet, we have to set packet’s cmn header/ field
    // To down

    Set packet’s hdrc header common direction field to DOWN;

Stage1:
**********************************************************************
// in case of the route is available, whatever the Packet original
// sender get the best route (shortest) to the destination (dst)
// through any next hop (-99)
      prte = call GetEntry2 (dst,-99,&myaddress);  //-99 means any next hop
      If (a route and not broken)
       Call send procedure;   // Jump to send procedure (stage 4)
//over here, no route available or the available one is broken

Stage2:
**********************************************************************
  //In case of I'm not this Packet original sender, the original
  // sender is a different node
  If (route and broken && source != my address)
   {
     //found a broken route. I'm not this packet original sender
     // the previous hop that forwarded this packet, has no idea
     // that this route is broken, therefore, need to broadcast a
     // route Update Packet immediately (Error Pkt).
     For (not end of routing table)
       {
       //include any route where advertise field is true
       If (route && advertise)
        change_count++;
       }
      If (change_count > 0)
        Packet *p1 =call makeRSUP(change_count);
        If (p1)
         Call sendOutBCastPkt1(p1);  // broadcast an Update packet
      //drop the packet, with the reason
      Drop (p, DROP_RTR_NO_ROUTE);
      Return;
   }
  Else // I'm this packet original sender.
    {

Stage3:
*************************************************************************
  //in case of Packet original sender has no route or the available
  // route is broken
  If (not a direct route)
   {
     Set prte to NULL; //initialise prte array
     //get the direct route if available
     prte = GetEntry1 (dst, dst,&myaddress);
   }
  If (prte entry is a direct broken route)
   {
     //found a direct broken route, Queue the packet
      If (no queue available in this entry)
       prte q = new PacketQueue()  ; //Create a new queue

     Save the Packet in that route's queue;
      If (there is certain information)
       Dump the packet's available information into the trace file

     If (the queue is full)   // MAX_QUEUE_LENGTH
     Drop (a packet from the queue);   // the queue is full

     //make a Route request for that destination
     If (first RREQ or last RREQ time + the delay < now)
      {
       //make a RREQ for that destination (dst), because this is the
         // first RREQ for that dentition, or the last RREQ was since
         // this period (changed_at + 0.5) 0.5 is a delay.
       Packet *p = call makeRRQ(myaddress,dst);
       If (p)
       {
        Access and read the packet's (p) ip header;
        Access and read the packet's (p) hdrc header common;
       //a node may receive multiple copies of the same route
       // request packet from various neighbours. Therefore, we need
       //to control number of RREQ to the same destination
        prte entry packet_id = packet unique id;
        prte entry changed_at = now; //entry last update time
        Assert no failure in the cmn header;   // abort in case of failure
        Call sendOutBCastPkt1(p); // broadcast a RREQ for that destination
        }
      }
      Return;
    } // end of is this entry prte is a direct broken route
     Else
      {
      //brand new destination, create a new broken entry (no usable)
      Define rte as a new routing table entry type;// rte is a new entry
      Define and set now to Scheduler time; // define now to double
      Set rte array (entry) to zeros; //Initialise the new route
      rte entry dst = dst;
      rte entry f_nxt_hop = dst;
      rte entry s_nxt_hop = Null;
      rte entry metric = infinity;
```

```
       rte entry packet_id = Null;    // initial value
       rte entry link_num = myaddress*10000+dst;
       rte entry advertise = false; // don't advertise this route
       rte entry changed_at = now;
       rte entry expected beacon time = null;
       Create a new queue;

        // save the Packet in that route's queue
        Set rte entry q to the packet;
         If (there is certain information)
          Dump the packet's available information into the trace file
          // make a RREQ for that destination (dst) for 1st time
          Packet *p = call makeRRQ(myaddress,dst);
          If (p){
            prte entry packet_id = packet unique id;
            Assert no failure in the cmn header;
            Call sendOutBCastPkt1(p);  // broadcast a RREQ}
           // add this new entry (rte) in my routing table RT
           Call AddEntry (rte, &myaddress);
          Return;
        } // end of Brand new destination
     }

Stage4:
***********************************************************************
// this is the send procedure part
Send:
   Set packet's hdrc prev_hop_ field to myaddress; // header common
     If (prte entry metric > 1)
       hdrc next_hop_ = hop;
     Else
       hdrc next_hop_ = dst;
   If (there is certain information)
    Dump that information into the trace file;
    Assert (no failure in the cmn header); // abort in case of failure

   // give this packet to the next hop
   Call recv(p, (Handler *)0);
 Return;
} // End of forward packet function
```

Figure 4.35: Forward Function pseudocode

4.7.24 Make a Route Request Function

The *makeRRQ* is called by the *forwardPacket* function whenever a node needs to forward a packet to a specific destination, and has no route available in its routing table (RT) to that destination.

The *makeRREQ* function takes two parameters, the requester node IP address and the requested destination IP address and returns a *RREQ* packet, the RREQ packet is shown in Figure (4.13).

The mechanism of *makeRREQ* is simply to create a broadcast packet, that includes the requested destination and the requester node as shown in the make route request pseudocode in Figure 4.36.

```
Packet *
MDVZRP_Agent::makeRRQ(nsaddr_t requester, nsaddr_t destination)
 { // Start of make Route Request

   Defined and allocate a Packet (p);
   Access the packet’s (p) ip header;
   Access and read the packet’s (p) hdrc header common;
   Define and set now to Scheduler time; //define now variable to double

   // The node (my address) going to create and broadcasts a RRQ
   // looking for a route to a specific node (destination), because
   // it wants to forward a packet to it and has no route available

   //Prepare the packet to be broadcast to all one hop neighbours.

   hdrc packet type = PT_ROUTE_REQUEST;// Packet 26 Route Request Packet MDVZRP
   iph source address = myaddress;      // source’s address is my address
   hdrc fst_fwd_hop_  = Null;           // not forwarded before
   hdrc p_prev_hop_   = Null;           // no pre previous hop before

   //The packet’s rest fields will fill in sendOutBCastPkt function

   Set number of bytes to be included in this packet;
   //the number of included bytes are 8. Where 4Bytes for requester
   //  (source) and 4B for the requested node (destination)

   packet size = 8 + IP_HDR_LEN; // MDVZRP + IP

   // Filling the Routes Update Packet
     Include requester node address into the packet;        // 4 bytes
     Include requested destination address into the packet; // 4 bytes

  //return RREQ packet for broadcasting by sendOutBCastPkt1 function
     Return the packet
} // End of make Route Request
```

Figure 4.36: Make Route Request Function pseudocode

4.7.25 Make a Route Reply Function

The *makeRRP* is calling through the *receive_RREQ* function whenever a node receives a *RREQ* and it is the requested destination, or has a route to the requested destination in its routing table.

The *makeRRP* function receives six parameters, which are the *requester* node *IP address*, the requested *destination IP address*, *f_nxt_hop*, *s_nxt_hop*, *metric*, *link_num* and returns a route reply *RREP* packet, the RREP packet is shown in Figure (4.16).

The mechanism of *makeRRP* simply creates a packet including a route to the requested destination, where it unicasts that packet to the requester node as shown in the make route reply pseudocode in Figure 4.37.

```
Packet * MDVZRP_Agent::makeRRP(nsaddr_t sende_to,nsaddr_t
dst,nsaddr_t f_nxt_hop,nsaddr_t s_nxn_hop,int metric,int
link_num)
 { // Start of make Route Reply function
   Defined and allocate a Packet (p);
   Access the packet's (p) ip header;
   Access and read the packet's (p) hdrc header common;
   Define and set now to Scheduler time; // define now variable to double

   // The node (myaddress) going to create and unicast a RRP

   // prepare the packet to be unicast to the requester node
    iph destination address = sende_to; // destination's address
    iph destination port = ROUTER_PORT; // destination port
    cmn prev_hop_ field = myaddress;
    packet type = PT_ROUTE_REPLY; //Packet 27 Route Reply Packet for  MDVZRP
    int change_count = 1; // num of entries to be included in the packet

   int Number_of_bytes = 17;
   //number of bytes to be included in this packet = num of entries
   // multiply by 17 Bytes + 1 Byte ,where
   // 17 = 4 Bytes for dst, 4B f_nxt_hop, 4B s_nxt_hop, 1B for metric
   // and 4 B for link_num + 1Byte for num of entries that included

   //The packet's size MDVZRP + IP
   packet size = change_count * 17 + IP_HDR_LEN;

   // Filling the Route Reply Packet (RREP)
   Include destination into the packet;                      // 4 bytes
   Include first next hop (f_nxt_hop)  into the packet;      // 4 bytes
   Include second next hop (s_nxt_hop) into the packet;      // 4 bytes
   Include metric into the packet;                           // 1 bytes
   Include link_num into the packet;                         // 4 bytes

   // return RREP packet for unicasting to that destination requester
   Return the packet;
} // End of make Route Reply function
```

Figure 4.37: Make Route Reply Function pseudocode

4.7.26 Receive a RREQ Function

The *receive_RREQ* function is called whenever a node receives a route request packet. The function reads the received packet's headers, and makes the receiver node drops that packet if it's time to live has expired, or it has been received because of loop forwarding. Otherwise, it calls the *checkNeighbor* function to check if the previous node that forwarded that packet *RREQ* is a known neighbour. Otherwise, it considers that node as a new neighbour by adding a new entry in its routing table to that node and sending a new FRIP to it as we mentioned in the previous sections.

The mechanism of the *receive_RREQ* function is divided into three stages as shown in Figure 4.38. In the 1st stage; extracts the data packet and deals with the included information which is a route to the requested destination. While in the 2nd stage; the receiver node ignores that RREQ packet if it is received the same RREQ before. Otherwise, prepares for getting a new route to that packet sender *Requester* if it is an unknown node. In the 3rd stage, the receiver node unicasts a RREP to the destination requester (RREQ original sender), if it has an available workable route to that destination. Otherwise, it rebroadcasts the same RREQ packet again as shown in the RREQ function's pseudocode in Figure 4.38.

```
void  MDVZRP_Agent::receive_RREQ (Packet * p)
{   // Start of receive RREQ function

   Access and read the packet's (p) ip header;
   Access and read the packet's (p) hdrc header common;
   Define and set now to Scheduler time; //define now variable to double

   // set a pointer (d) to the beginning of the data packet the 1st
     // byte of the data packet (change count)
   Set unsigned char *d to the 1st data byte; // a pointer

   // set a pointer (w) to the 2nd byte of data packet (destination)
   Set unsigned char *w to d + 1;     // *w=d+1
   Set rte  to routing table entry;  //rte  is a new entry (array)
   Set prte to routing table entry;  //prte is a new entry (array)
   Set pr2  to routing table entry;  //pr2  is a new entry (array)
   Initialise rte entry as a new array;
   Initialise prte entry as a new array;
   prte = Null;

   int sender = packet saddr; // source address
   int previous_node = packet prev_hop_;
   int p_previous_node = packet p_prev_hop_ ;
   int first_hop_node = packet fst_fwd_hop_ ;
   int flag = Null;
   int Found_before = Null;
```

```
Stage1:
************************************************************************
   // extracting data from the RREQ packet....
   // get(Read) destination requester from the data packet
   int packet_requester  =  packet's 1st 4 bytes;

   // Get (Read) requested destination from the data packet
   int destination = packet's 2nd 4 bytes;

   // Get (Read) number of forwards from the packet's header
   int expected_hops = cmn header num_forwards_;

   If (there is certain information)
    Dump that information into the trace file;
   If (packet's Time to live = 0) // ttl_=0
   {
     now = Scheduler time;
     Drop (p, DROP_RTR_TTL); // drop that packet with reason
     Return;
   }
   If (myaddress is the packet sender or I've rebroadcast this RREQ)
   {
     // I am the sender or I've rebroadcast this RREQ packet. So, I
       // have to drop it because of loop
     Drop (p, DROP_RTR_ROUTE_LOOP);  // do nothing and free the packet
     Return;
   }

   Yes_it_is_a_neighbor = call checkNeighbor(prev_hop_);

Stage2:
************************************************************************
// the receiver node Checks if this RREQ is received before
   For (not end of routing table, set pr2 to the current route)
   {
     // Go through my RT and set prt2 to the current route
     If (current route (pr2) dst = packet_requester
      {
       // Found a Route to that Packet Requester (the packet sender)
       If (current route packet_id = hdrc packet unique id)
        {
         // I've received this RREQ Before
         Found_before = 1; // ignore this RREQ
         Break;
        }
       }
   } //end of FOR (loop)

   // Prepare for getting a new route for this packet sender
     //  (routing information)
   rte dst = packet_requester;
   rte f_nxt_hop = previous_node;
   rte s_nxt_hop = p_previous_node;
   rte metric = expected_hops;

   If (first_hop_node = Null) // Null =99
    rte link_num = myaddress*10000+packet_requester;
   Else
    rte link_num = first_hop_node*10000+packet_requester;
    rte packet_id = hdrc packet unique id; // Packet id
    rte advertise = true; // I'd like to advertise this route
    rte changed_at = now;
    nsaddr_t = &myaddress; // my address

    // Variables initialisation
    int f_nxt_hop;// first next hop
    int s_nxt_hop;// second next hop
    int metric;
    int link_num;

    // Add rte entry into my RT
    Call AddEntry (rte,&myaddress); // don't add if available before
    Set prte to NULL; //initialise prte array (entry)
    prte = call GetEntry2 (packet_requester,-99, &myaddress);
    If (prte && prte packet_id != hdrc packet unique id)
    prte packet_id = hdrc packet unique id;

Stage3:
***********************************************************************
  // the receiver node (myaddress) unicasts a RREP if it knows the
  // requested destination. Otherwise, rebroadcasts the same RREQ again
     int  Need_to_rebroadcast_the_same_RREQ = 1 // flag

    If (destination = myaddress )// am I the requested destination
      {
       // I've received a route request from *** packet_requester
       // looking for me ... (Destination is my address)
       Need_to_rebroadcast_the_same_RREQ = Null; // Need a RREP
       f_nxt_hop = Null;  // Null=-99
       s_nxt_hop = Null;  // Null=-99
       metric = Null;     // Null=0
       link_num = myaddress*10000+previous_node;
      }
     Else
       Set prte to NULL; // Initialise prte array
       prte = Call GetEntry2 (destination,-99,&myaddr)// get the best route
         If (a route prte && not broken)
          {
           Set Need_to_rebroadcast_the_same_RREQ = 1;
           // I found a route = that destination in my routing table
           f_nxt_hop = route prte f_nxt_hop;
           s_nxt_hop = route prte s_nxt_hop;
           metric = route prte metric;
           link_num = route prte link_num;
          }
      If (!Need_to_rebroadcast_the_same_RREQ && !Found_before)
       {
         // I need to rebroadcast this packet again, because I didn't
         // find a route to the requested destination
            If (hdrc fst_fwd_hop_ = Null) // first rebroadcast
               hdrc fst_fwd_hop_ = myaddress;

            hdrc p_prev_hop_ = header common prev_hop_ ;
            Call sendOutBCastPkt1(p);  //broadcast the RREQ packet again
             If (there is certain information)
              Dump that information into the trace file;
        }
      If (Need_to_rebroadcast_the_same_RREQ && !Found_before)
       {
        // I need to unicast a Route reply, because I found a route
```

```
          // to the requested destination
         Locate and set a new Packet p1;
         p1 = makeRRP(packet_requester,destination,f_nxt_hop,s_nxt_hop
         ,metric,link_num);
         If (p1)
          Call forwardPacket(p1);
         Else
          // There is an Error
          Abort ();
         // Free the RREQ Packet because, it sent a ROUTE REPLY
         Free the RREQ Packet (p);
       }
  Return;
} // End of receive RREQ function
```

Figure 4.38: Receive Route Request Function pseudocode

4.7.27 Receive a RREP Function

The *receive_RREP* function is called whenever a node receives a route reply packet. The function reads the received packet's headers, and makes the receiver node drop that packet if it is time to live (TTL) has expired. Otherwise, it calls the *checkNeighbor* function to check if the previous node that forwarded that packet (RREP) is a known neighbour. Otherwise, it considers that node as a new neighbour by adding a new entry in its routing table to that node and sends to it a new FRIP as we mentioned in the previous sections.

The mechanism of *receive_RREP* function is divided into three stages as well. In the 1st stage; as shown in Figure 4.39 extracts the data packet and deals with the included information (a route to the requested destination). While in the 2nd stage; the RREP packet receiver caches (gathers) routing information from the received packet (RREP) for getting a new route to that packet sender. In the 3rd stage; the receiver node checks if it can send off any packets it has got queued for that destination as shown in the *RREQ* function pseudocode in Figure 4.39.

```
Void MDVZRP_Agent::receive_RREP (Packet * p)
{  // Start of receive RREP function

   Access and read the packet's(p) ip header;
   Access and read the packet's(p) hdrc header common;
   Define and set now to Scheduler time; // define now variable to double

   // set a pointer (d) to the beginning of the data packet the 1st
    // byte of the data packet (which is the change count)
   Define and Set unsigned char *d to the 1st data byte; // a pointer
   // set a pointer (w) to the 2nd byte of data packet (destination)
   Define and Set unsigned char *w to d + 1; // *w=d+1

   Set rte to routing table entry;   //rte  is a new entry (array)
   Set prte to routing table entry;  //prte is a new entry (array)
   Set pr2 to routing table entry;   //pr2  is a new entry (array)

   Initialise rte entry as a new array;
   Set prte to Null;

   int sender = packet iph saddr; // source address
   int sent_to = packet iph daddr; // destination
   int previous_node = packet prev_hop_ ;

Stage1:
**********************************************************************
// extracting data from the RREP packet....

   //get (Read) requested destination from the data packet
   Set int destination to data packet's 1st 4 bytes;

   //get next hop towards that destination from the data packet
   Set int f_nxt_hop to data packet's 2nd 4 bytes;

   //get second hop towards that destination from data packet
   Set int s_nxt_hop to 3rd 4 bytes of data packet;

   //get metric towards that destination from the data packet
   Set int metric to data packet's byte number 13; //one byte

   //get link_num towards that destination from the data packet
   Set int metric to data packet's byte number 14; //one byte

   If (there is certain information)
   Dump that information into the trace file;

   If (packet's Time to live is Null) // ttl_= 0
    {
      now = Scheduler time;
      Drop (p, DROP_RTR_TTL); // drop that packet with reason
      Return;
    }
   Yes_it_is_a_neighbor = call checkNeighbor(prev_hop_);

Stage2:
**********************************************************************
// caching (gathering) routing information from the received packet
// (RREP). Prepare for getting a new route for this packet sender
 // (routing information)
   rte dst = destination;
   rte f_nxt_hop = destination;
   rte s_nxt_hop = f_nxt_hop;
   rte metric = metric +1;
   rte link_num = link_num;
   rte q = Null;
   rte advertise = true; // I'd like to advertise this route
   rte changed_at = now; // time of getting the destination (dst) route
   If (rte route belongs to a neighbour)
   {
    //considere the destination (dst) as a new neighbour
    Set rte timeout_event to new Event();
```

```
     //schedule its next expected periodic beacon (timeout)
     Set schedule (timeout_event, min_update_periods_ * perup_);
     }
   //any node should get routing information from any packet pass
     // through it

   If (sent_to != myaddress ) // I'm not the packet destination
     {
      // because this Packet is sent to another Node, I need to make
      // and forward a new RREP to the same destination node

      //caching routing information before free the received RREP
      Locate and set a new Packet p1;
      p1= Call makeRRP(sent_to,rte.dst,rte.f_nxt_hop,rte.s_nxt_hop,
          rte.metric,rte.link_num);
      If (p1)
       Call forwardPacket(p1);
      Else
       // There is an Error
       Abort;
      }

Stage3:
************************************************************************
//see if we can send off any packets we've got queued for this dest.
     Set prte array (entry) to Null;
     prte = call GetEntry1 (rte.dst, rte.dst,&myaddress);

   //see if we have a direct route
   If (prte)
     //copy all the fields where rte pointes into prte pointer
     bcopy(prte, &rte);
     If (prte)
       {
       //check if we have a queued packets in this direct route
          // even this direct route is broken
       If (prte entry queue)  // is there any packet queued
          Give the packets to ourselves to forward;

       //reset the queue, of that destination's route (entry)
       Delete prte entry queue;
       SET prte entry queue to NULL;

       //copy all prte fields over where rte pointes in my RT
       Copy back prte entry onto rte entry; // write prte in my RT
       }
   Call free (p); // free the received packet any way

} // End of receive RREP function
```

Figure 4.39: Receive Route Reply Function pseudocode

4.7.28 Error Message Function

The *Error_Msg* is a function called through the *receive_RSUP* function whenever a node receives a broken route in a route update packet (RSUP). In the current version of MDVZRP as we mentioned previously, that we have included the *Error Packet* within the update packet (RSUP) for reducing the overall number of control packets.

The *Error_Mgs* function receives five parameters regarding a broken route (*non reachable destination*), which are a destination IP address *dest*, next first hop towards that destination *f_nxt_hop*, second next hop towards that destination *s_nxt_hop* and the broken route *link_num* .

The node that received the RSUP and found a broken route within the data packet (one of the included entries) calls *Error_Mgs* function to look through its routing table for any entry that has the same number of the broken link to assign it as an infinity (broken link), as shown in the make route reply pseudocode in Figure 4.40.

```
Void MDVZRP_Agent::Error_Msg(int dest, int f_nxt_hop , int
s_nxt_hop ,int link_num)
{  // Start of Error message function
  now = Scheduler time;
  Set pr2 to routing table entry;  //pr2 is a new entry (array)

  For ( !end of my routing table, set pr2 to current entry)
  {
   If (current entry (pr2) link_num = the received link_num)
    // I reach that destination through the broken link,(link_num)
    If (there is certain information)
     Dump that information into the trace file;

     If (current entry (pr2) belongs to a neighbour) // metric =1
      // no need to tell my, that link between me and my neighbour
      // is broken, I've to discover it myself
        Continue; // ignore the error message
     Else
      // assign the current route (pr2) as a bad route (infinity)
      Set Current entry (pr2) metric to Infinity // BIG=250 (infinity)
      Set Current entry (pr2) changed_at to now;

      If (pr2 link_num = link_num in the error message)
       //advertise the broken route with the same link_num only
        Set pr2 advertise to true // advertise this route
   }
}  // End of Error message function
```

Figure 4.40: Error message Function pseudocode

4.8 Summary

MDVZRP came cross many stages of design and improvements over a period of time into what it is today. Some of the improvements were in the code itself and others were in the protocol's algorithm technique. In this chapter we presented and described MDVZRP in more detail, where firstly, the pre-MDVZRP design *version 1.00* was described. Subsequently the updated version of MDVZRP *version 2.00* was outlined in detail. Before the chapter discussed the MDVZRP v2.0 description and implementation, it listed and highlighted the most important changes that were made to v1.0 and the main differences between MDVZRP v1.0 and MDVVZRP v2.0.

To perform its functionality in the best possible manner, this routing protocol includes one routing table which lists a number of multi-paths (backup) to each destination of the network. The routing table is used to transmit packets through the Ad Hoc network. The MDVZRP includes two types of packets, the control packets (Beacon, FRIP, RSUP, RREQ and RREP), and the data packets (TCP, CBR and ACK). The main purpose of using the packets is to exchange the information between the objects (nodes) during the simulation time.

MDVZRP's packets, agent and functions all are implemented using C++ under Ns2 (Network Simulator 2), which were presented and described in detail by pseudocode in this chapter. We have firstly created a new directory called MDVZRP to allocate our code inside the NS2 base directory. Then, we have created the protocol "physical" structure by creating the header files and routing agent and Tcl hooks main file, secondly; we created the protocol "logical" structure (classes).

MDVZRP's functionality testing and performance comparison are briefly explained and described graphically in the next two chapters respectively.

CHAPTER 5: FUNCTIONALITY TESTING

5.1 Introduction

In the absence of formal mathematical proof of the correctness of the protocol we need an extensive set of tests (verification) to confirm that the protocol meets the intended specifications and it is fully functional, and works in a wide range of environments. It is easy sometimes to convince yourself that it works, only to find that in the random radio environment it fails. One of the most common methods is functionality testing, where the new protocol's results are checked to see if they meet expected results based on previously known results, from other protocols already tested and verified. Providing a product for example a routing protocol with bug-free or a minimum amount of issues is also important and every developer's goal. Therefore, functionality testing helps to achieve such targets.

Once the implementation of MDVZRP has been completely tested, and porting to run in the Ns2, its functionality has been verified. Several tested are carried out regarding using different scenarios; some of them are already supplied with Ns2 version 2.30, as well as standalone scenarios. The output data results are compared with expected results specified by each scenario's documentation. For more accurate checks and verification all the used scenarios were run using different standard protocols (AODV, DSDV, DSR and TORA) and their results were compared with MDVZRP's results.

5.2 General Setup

Figure 5.1 shows a general setup of the all script files that were used in MDVZRP functionality testing. We kept the same scenario configuration and setup, the only thing we have changed is the routing agent setting to MDVZRP as shown in line in 11. The scenarios use the default setting of the wireless network. All the components that are listed in figure 5.1 are mainly common in most of the used scenarios, which come with certain default settings.

The following parameters are the wireless default settings that have been used in all the tcl scenarios, i.e., a *Channel/WirelessChannel* is the wireless channel, a *Propagation/TwoRayGround* is the propagation model used, which indicates that it is the two-ray ground reflection radio propagation model, a *Phy/WirelessPhy* is the wireless network interface type with 2 Mbps bandwidth, the *Mac/802_11* is the MAC layer setting, which uses 2Mbps packet data rate for both broadcast/unicast, the interface queue type is also defined as a priority queue *Queue/DropTail/PriQueue*, where the maximum packet in the interface queue is defined by 50 packets, *LL* is the link layer type, and finally, an antenna model, which is defined as an Omni-directional antenna *Antenna/OmniAntenna*. We ran more than 1500 scenarios during MDVZRP implementation, improvement, and performance comparison stages using these parameters. The trace file format that that has been used in all the scenarios is the new format as shown in line 20 of Figure 5.1, which has been described in chapter 2, section 2.4.3.6 in more detail.

```
1: # Define options
2: set val(chan)  Channel/WirelessChannel   ;# channel type
3: set val(prop)  Propagation/TwoRayGround  ;# radio-propagation model
4: set val(netif) Phy/WirelessPhy           ;# network interface type
5: set val(mac)   Mac/802_11                ;# MAC type
6: set val(ifq)   Queue/DropTail/PriQueue   ;# interface queue type
7: set val(ll)    LL                        ;# link layer type
8: set val(ant)   Antenna/OmniAntenna       ;# antenna model
9: set val(ifqlen)  50                      ;# max packet in ifq
10:set val(nn)      n                       ;# number of mobilenodes
.
.
 16:set val(rp)   MDVZRP                    ;# routing protocol
.
.
19:$ns trace-all $tracefd       ;#in all formats always keep this line
20:$ns use-newtrace             ;# trace file new format
.
.
# configure the nodes
.
# Provide initial location of mobilenodes
.
# Generation of movements
.
#Set a TCP connection between node_(x) and node_(y)
.
# Telling nodes when the simulation ends
.
# ending nam and the simulation
.
.
```

Figure 5.1: Default wireless scenarios settings

5.3 Simple Scenarios for Functionality Testing

This section shows simple scenarios that have been used in the functionality testing of MDVZRP. They create a simple network configuration under the Ns2 network simulator and animator. All the scenarios are OTcl scripts, therefore, we will explain what each scenario does, and show the animator results rather than listing the script itself.

5.3.1 Example1.tcl Scenario Overview

In this scenario, a simple wireless network for three nodes out of transmission range of each other at the beginning as shown in figure 5.3 is presented for testing a communication between two mobile nodes.

5.3.1.1 Scenario Setup

The scenario places 3 nodes connected to each other via the wireless channel, but at the beginning, they are out of each other's radio range as shown in figure 5.3 in a 500x400 m flat grid area. In this scenario we assume that node (1) *Destination* receives any incoming TCP traffic. Therefore, it has a TCP sink agent attached to it. The other node (0) has an FTP agent connected to its TCP agent, simulating FTP traffic *Source*, as shown in Figure 5.2.

```
# Example1.tcl
# A 3-nodes example for Ad Hoc simulation with MDVZRP for 150 sec.
# Some options define
.
set val(nn)       3                     ;# mobile nodes Number
 set val(rp)      MDVZRP                ;# routing protocol
set val(x)        500                   ;# topography X dimension
set val(y)        400                   ;# topography y dimension
.
set val(stop)     150                   ;# end simulation time
.
set ns             [new Simulator]
set tracefd        [open Example1.tr w]
set windowVsTime2  [open Example1.tr w]
set namtrace       [open Example1.nam w]
.
# Generation of movements
$ns at 10.0 "$node_(0) setdest 250.0 250.0 3.0"
$ns at 15.0 "$node_(1) setdest 45.0 285.0 5.0"
```

```
$ns at 110.0 "$node_(0)setdest 480.0 300.0 5.0"
.
.
##Set a TCP connection between node_(0) and node_(1)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $node_(0) $tcp    ;# set node 0 as a source node
$ns attach-agent $node_(1) $sink   ;# set node 1 as a destination node
$ns connect $tcp $sink             ;# set a connection between them
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 12.0 "$ftp start"           ;# start traffic flow
.
.
```

Figure 5.2: Example1.tcl scenario setup

5.3.1.2 Scenario description and results

As we mentioned previously, at the beginning of this scenario, the three nodes are out of each other's radio range as shown in Figure 5.3, The FTP traffic (data sending) is started at time 12 sec **$ns at 12.0 "$ftp start"**. However, at this time no route is available at the source to the destination node as shown in Table 5.1, where both node (0) and node (1) are too far for any data transfer. Therefore, node (0) broadcast an on demand route request, as shown in Table 5.2.

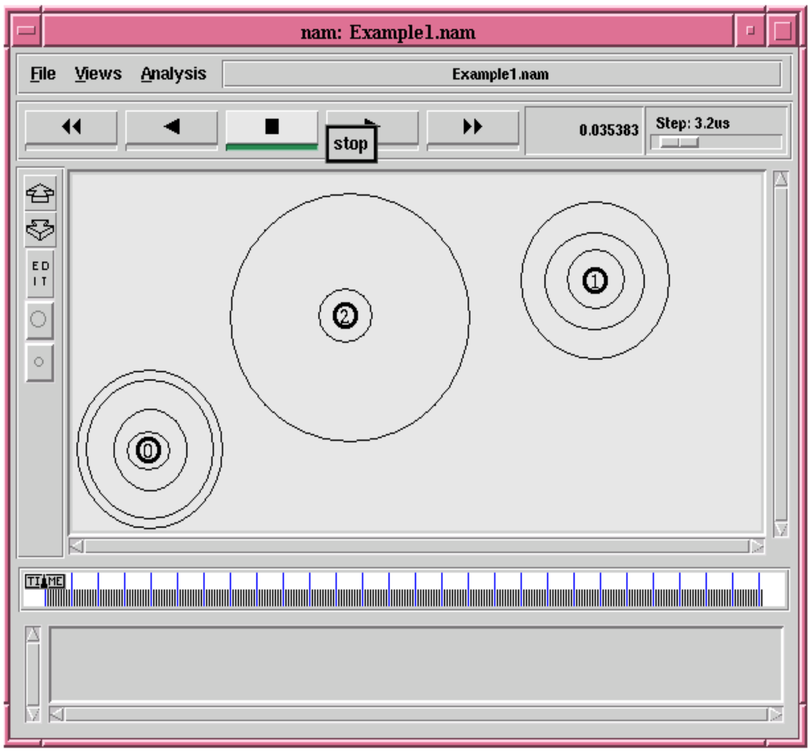


Figure 5.3: A simple network of three nodes

Table 5.1 : Initial state

```
***** In Start-up ... Routing table of Node ( 0 )
-----------------------------------------------------------------------------------------------------------
 Dst    1-hop  2-hop    metric    link_num  change at    timeout_event  Queed-Data    Need Advertise  Sent_RREQ
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
 0      0      -99       0         0         0.00000000     (nil)           (nil)          (nil)            0
-----------------------------------------------------------------------------------------------------------
```

Table 5.2 : Initiating a RREQ

```
***** Time: 50.9321   Routing table of Node ( 0 )
-----------------------------------------------------------------------------------------------------------
 Dst    1-hop  2-hop    metric    link_num  change at    timeout_event  Queed-Data    Need Advertise  Sent_RREQ
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
0        0     -99       0         0         0.00000000  (nil)            (nil)            (nil)          0
1        1     -99      250        1        12.00000000  (nil)            0x92d6ea0        (nil)          14
2        2     -99       1         2        43.21629950  0x92c6210        (nil)             0x1           0
-----------------------------------------------------------------------------------------------------------
```

At times of 10 and 15 seconds respectively, the source node (node with the FTP) node (0), and the destination node (node with TCP sink) node (1) start moving towards each other. At 53 seconds, both nodes are close enough to begin exchanging routing control messages via node2, when both nodes (source and destination) are in its transmission range, and have available routes to both of them in its routing table. Therefore, it has replied with a RREP message to the source node. Therefore, the TCP traffic starts to flow via node (2) from the source node (0) to the destination (1) as shown in Figure 5.4.

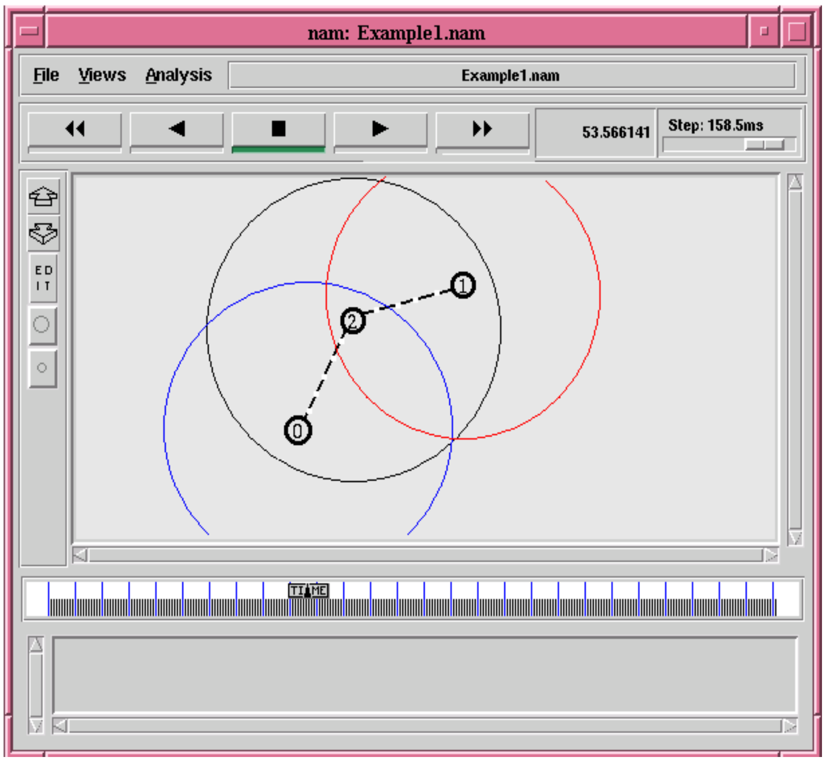


Figure 5.4: Traffic via an intermediate node

Since both nodes are moving toward each other, at 64.77 seconds as shown in table 5.3 and Figure 5.5, the two nodes became close enough to establish a direct flow of traffic. In spite of the continuous movement of the two nodes, the TCP traffic still flows directly from the source node to the destination as shown if Figure 5.6, because both of them (the source and destination) are still in each other’s transmission range.

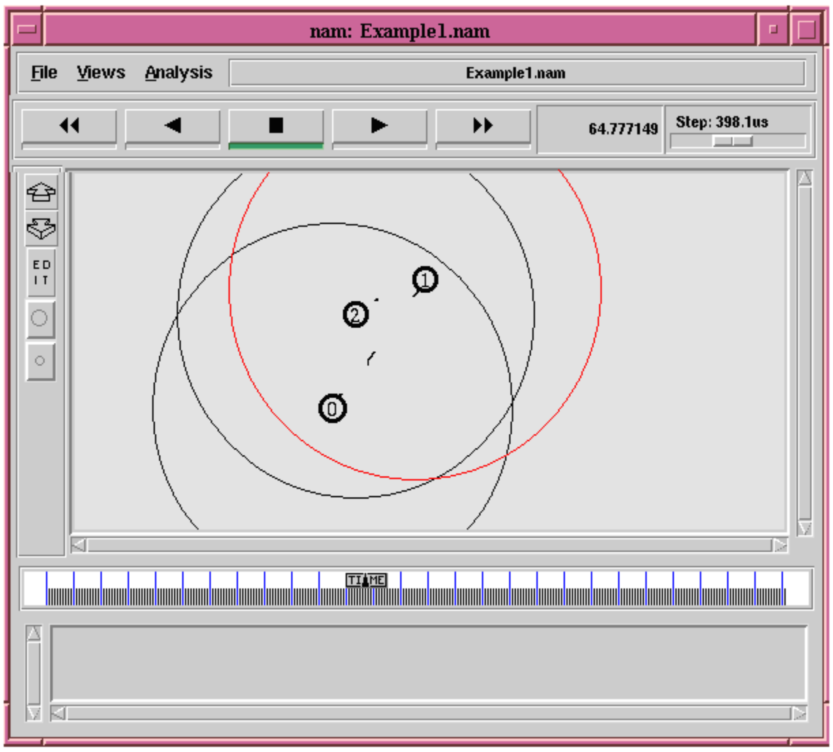


Figure 5.5: A direct Traffic connection

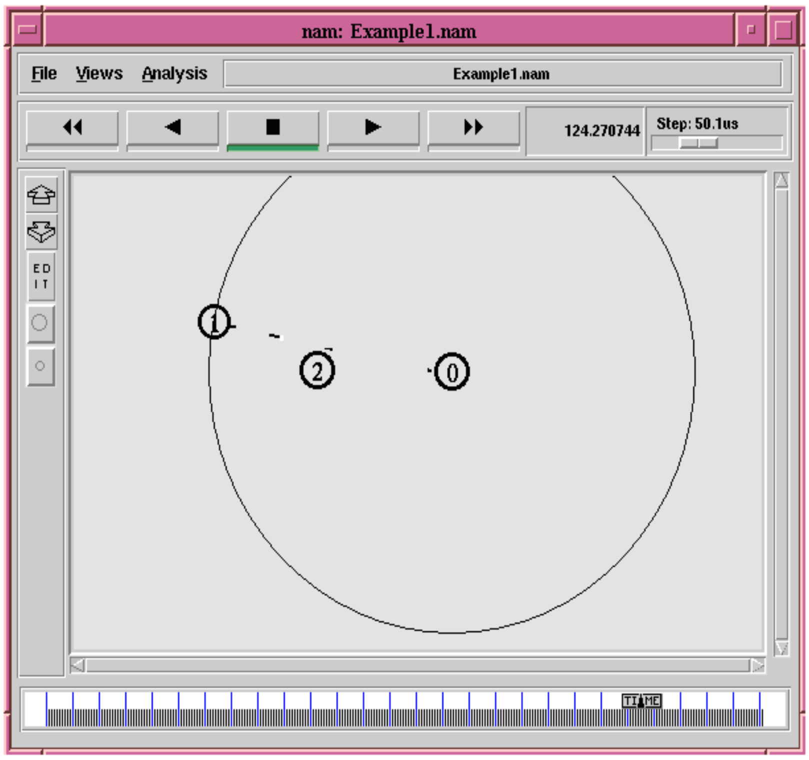


Figure 5.6: The destination still in the source transmission range

Table 5.3 : Getting Multipath

```
***** Time: 84.4411   Routing table of Node ( 0 )
-----------------------------------------------------------------------------------------------------------
Dst    1-hop  2-hop   metric   link_num  change at    timeout_event  Queed-Data  Need Advertise  Sent_RREQ
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
 0      0     -99       0          0      0.00000000  (nil)            (nil)           (nil)          0
 1      1     -99       1          1     64.77785751  0x9344b70        (nil)           (nil)          0

 1      2       1       2      20001     55.01021039  (nil)            (nil)           (nil)          0

 2      2     -99       1          2     55.01021039  0x933ab00        (nil)           (nil)          0
-----------------------------------------------------------------------------------------------------------
```

Table 5.4 : A Direct link Failure

```
***** Time:140.164   Routing table of Node ( 0 )
-----------------------------------------------------------------------------------------------------------
Dst    1-hop  2-hop   metric   link_num  change at    timeout_event  Queed-Data  Need Advertise  Sent_RREQ
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
0        0     -99       0          0     0.00000000   (nil)            (nil)     (nil)               0
1        1     -99     250          1     125.54052745 (nil)            (nil)     (nil)               0
1        2       1       2      20001     55.01021039  (nil)            (nil)     (nil)               0
2        2     -99       1          2     55.01021039  0x933ab00        (nil)     (nil)               0
-----------------------------------------------------------------------------------------------------------
```

At 123 seconds the FTP source moved away from the TCP sink that caused the link between them to break, as shown in Table 5.4 and dropped a few packets. Node (0) assigned the direct link to node (1) to infinity (250) as a broken link, and immediately used an alternative (back up) route, to establish the same connection to node (1) again via node (2) as shown in Figure 5.7.

Finally, the simulation time successfully ended at 150 seconds, and we have verified that the routing tables of the three nodes are sorted and set up correctly, by viewing and analysing their routing table logging file as shown in Tables 5.1-5.4 of node (0). Please note we have omitted the routing tables of node (1) and (2), and displayed the routing table of node (0) in 4 different times only for clarity.

The simulator trace log analysis as well as the MDVZRP log files shows that the scenario works as expected using MDVZRP protocol.

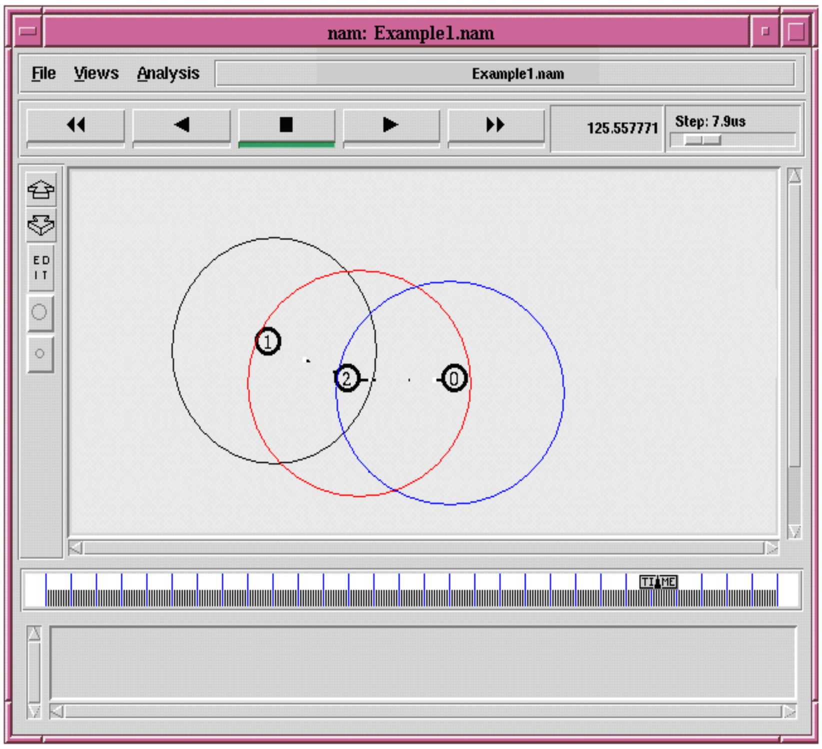


Figure 5.7: A direct link failure

5.3.2 Example2.tcl Scenario Overview

We present a test for the protocol capability for obtaining an alternative route here, if one or more of the intermediate nodes gradually moved away during the traffic transmission causing an active broken link. In Example2.tcl, seven mobile nodes [0 – 6] are placed in such a way that the two nodes need to establish a communication via couple of intermediate nodes to reach each other.

5.3.2.1 Scenario Setup

The scenario places 7 nodes which are connected to each other via a wireless channel in a 700x800 m rectangular grid area as described in tcl format in Figure 5.8. Node (6) *Destination* receives any incoming TCP traffic. Therefore, it has a TCP sink agent attached to it while node (1) has an FTP agent connected to its TCP agent, simulating FTP traffic *Source*.

```
# Example2.tcl
# A 7-nodes example for Ad Hoc simulation with MDVZRP for 150 sec.
# Some options define
.
set val(nn)       7                    ;# mobile nodes number
 set val(rp)      MDVZRP               ;# current routing protocol
set val(x)        700                  ;# topography X dimension
set val(y)        800                  ;# topography Y dimension

.
set val(stop)     150                  ;# End simulation time

.
set ns             [new Simulator]
set tracefd        [open Example2.tr w]
set windowVsTime2  [open Example2.tr w]
set namtrace       [open Example2.nam w]

.
.
# Generation of movements
$ns at 70.0 "$node_(2) setdest 250.0 700.0" ;# at 70 sec node 2 moves
                                            ;# to position 250,700

#Set a TCP connection between node_(1) and node_(6)
set tcp [new Agent/TCP/Newreno]
set sink [new Agent/TCPSink]
$ns attach-agent $node_(1) $tcp    ;# set node 1 as a source node
$ns attach-agent $node_(6) $sink   ;# set node 6 as a destination node
$ns connect $tcp $sink             ;# set a connection between them
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 5.0 "$ftp start"            ;# start traffic flow
.
```

Figure 5.8: Some lines of Example1.tcl Scenario

5.3.2.2 Scenario description and results

Figure 5.9 shows a simple network of seven nodes connected through a wireless channel, they still discover each other, and each node has only one route to itself created at the start up stage as shown in Table 5.5.

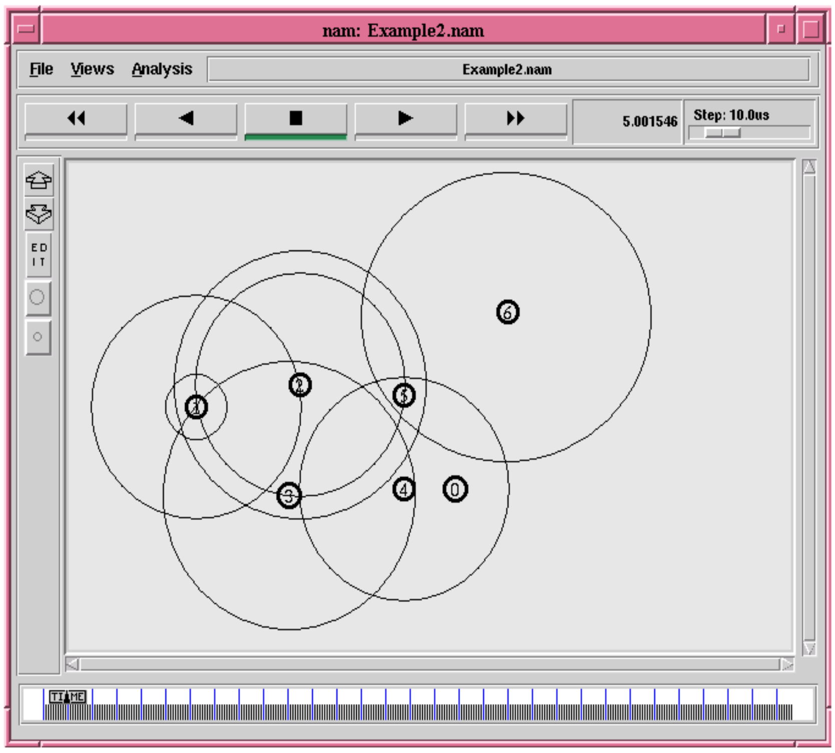


Figure 5.9: A simple network of seven nodes

Table 5.5: Initial state of some nodes

```
***** In Start-up ... Routing table of Node ( 1 )
-----------------------------------------------------------------------------------------------------------
 Dst   1-hop  2-hop   metric   link_num  change at   timeout_event  Queed-Data   Need Advertise  Sent_RREQ
-----------------------------------------------------------------------------------------------------------
1      1     -99      0        10001    0.00000000       (nil)        (nil)          (nil)          0
-----------------------------------------------------------------------------------------------------------
```

```
***** In Start-up ... Routing table of Node ( 2 )
-----------------------------------------------------------------------------------------------------------
 Dst   1-hop  2-hop   metric   link_num  change at   timeout_event  Queed-Data   Need Advertise  Sent_RREQ
-----------------------------------------------------------------------------------------------------------
2       2     -99      0        20002   0.00000000       (nil)         (nil)        (nil)            0
-----------------------------------------------------------------------------------------------------------
```

```
***** In Start-up ... Routing table of Node ( 5 )
-----------------------------------------------------------------------------------------------------------
 Dst   1-hop  2-hop   metric   link_num  change at   timeout_event  Queed-Data   Need Advertise  Sent_RREQ
-----------------------------------------------------------------------------------------------------------
5       5     -99      0        50005   0.00000000       (nil)         (nil)        (nil)            0
-----------------------------------------------------------------------------------------------------------
```

At 5.0 seconds FTP traffic flow is started. The source node (1) and the destination node (6) are too far from each other to establish a direct communication as shown in the source's node routing Table 5.6. However, as shown in Figure 5.10, FTP traffic flow is established between node (1) and (6) via intermediate nodes (2 and 5), which have routes to the destination in 2 and 1 hops as shown in their routing Tables 5.7 and 5.8 respectively.

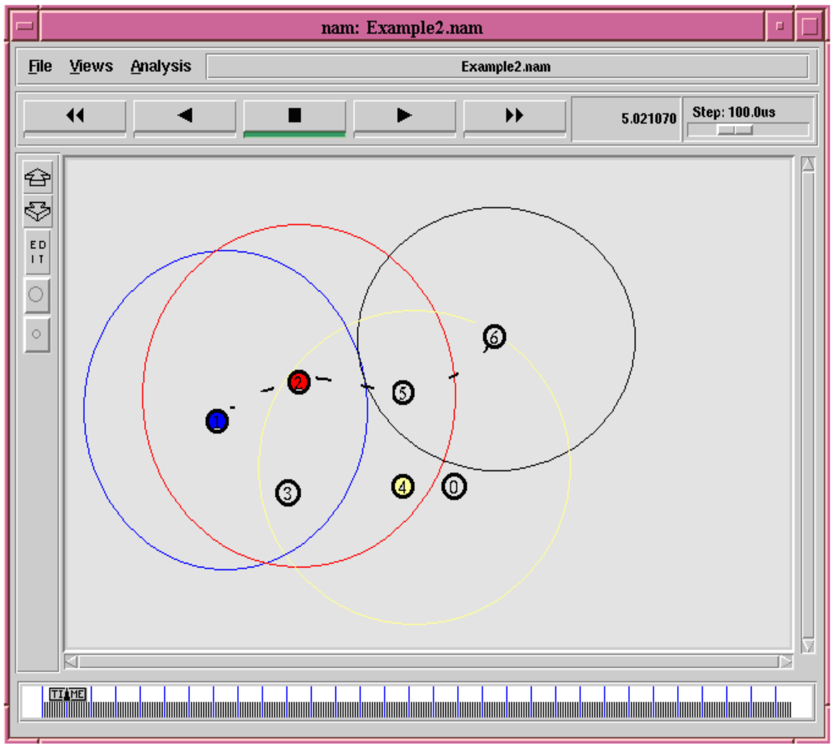


Figure 5.10: The TCP traffic starts to flow

Table 5.6: A route to the destination node (6) in 3 hops

```
 *****  At  17.7658... Routing table of Node ( 1 )
-----------------------------------------------------------------------------------------------------------
 Dst    1-hop  2-hop   metric   link_num  change at   timeout_event  Queed-Data   Need Advertise  Sent_RREQ
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
   0      3      4       3       40000    16.49139686     (nil)        (nil)          0x1              0
   1      1    -99       0       10001     0.00000000     (nil)        (nil)         (nil)             0
   2      2    -99       1       10002     0.17885948   0x95b61a0      (nil)          0x1              0
   2      3      2       2       30002     0.89172185     (nil)        (nil)          0x1              0
   3      3    -99       1       10003     0.89172185   0x95cd020      (nil)          0x1              0
   4      3      4       2       30004    16.49139686     (nil)        (nil)          0x1              0
   5      3      4       3       40005    16.49139686     (nil)        (nil)          0x1              0
   6      2      5       3       50006     5.02007043     (nil)        (nil)          0x1              0
   6      6    -99     250       10006     5.00000000     (nil)        (nil)         (nil)             0
-----------------------------------------------------------------------------------------------------------
```

Table 5.7: A route to the destination in 2 hops at an intermediate node

```
 *****  At  19.5945... Routing table of Node ( 2 )
-----------------------------------------------------------------------------------------------------------
 Dst    1-hop  2-hop   metric   link_num  change at   timeout_event  Queed-Data   Need Advertise  Sent_RREQ
-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------
   0       3      4       3      40000    16.49139673    (nil)         (nil)          0x1              0
   0       5      0       2      50000    13.47310439    (nil)         (nil)          0x1              0
   1       1    -99       1      20001     0.03169202  0x95b5c48       (nil)         (nil)             26
   1       3      1       2      30001     0.89172172     (nil)        (nil)          0x1              0
   2       2    -99       0      20002     0.00000000     (nil)        (nil)         (nil)             0
   3       1      3       2      10003    17.79612613     (nil)        (nil)          0x1              0
   3       3    -99       1      20003     0.89172172   0x95ccf30      (nil)          0x1              0
   3       5      4       3      40003    13.47310439     (nil)        (nil)          0x1              0
   4       3      4       2      30004    16.49139673     (nil)        (nil)          0x1              0
   4       5      4       2      50004    13.47310439     (nil)        (nil)          0x1              0
   5       3      4       3      40005    16.49139673     (nil)        (nil)          0x1              0
   5       5    -99       1      20005     0.63594949   0x95b5c78      (nil)          0x1              0
   6       5      6       2      50006     5.01605867     (nil)        (nil)          0x1              0
-----------------------------------------------------------------------------------------------------------
```

Table 5.8: A route to the destination in 1 hop at an intermediate node

```
*****  At  27.502... Routing table of Node ( 5 )
--------------------------------------------------------------------------------
 Dst     1-hop  2-hop   metric  link_num   change at   timeout_event  Queed-Data   Need Advertise  Sent_RREQ
--------------------------------------------------------------------------------
  0        0     -99       1     50000    1.11821258    0x95d28a0      (nil)         (nil)          0
  0        4       0       2     40000    1.60883373     (nil)         (nil)         (nil)          0
  1        2       1       2     20001    0.17885947     (nil)         (nil)         (nil)         26
  1        4       3       3     30001    1.60883373     (nil)         (nil)         (nil)          0
  2        2     -99       1     50002    0.17885947    0x95b6180      (nil)         (nil)          0
  2        4       3       3     30002    1.60883373     (nil)         (nil)         (nil)          0
  3        2       3       2     20003   19.61012102     (nil)         (nil)          0x1           0
  3        4       3       2     40003    1.60883373     (nil)         (nil)         (nil)          0
  4        0       4       2         4   14.12599291     (nil)         (nil)          0x1           0
  4        2       3       3     30004   19.61012102     (nil)         (nil)          0x1           0
  4        4     -99       1     50004    1.60883373    0x95b5258      (nil)         (nil)          0
  5        5     -99       0     50005    0.00000000     (nil)         (nil)         (nil)          0
  6        6     -99       1     50006    1.64095043    0x95d29c0      (nil)         (nil)       2343
--------------------------------------------------------------------------------
```

Table 5.9: Routing table of an isolated node

```
*****  At  129.842… Routing table of Node ( 2 )
 Dst   1-hop  2-hop   metric   link_num  change at     timeout_event  Queed-Data   Need Advertise  Sent_RREQ
  0      3      4      250      40000    116.28358926    (nil)          (nil)         (nil)          0
  0      5      0      250      50000    114.64453168    (nil)          (nil)         (nil)          0
  1      1    -99      250      20001    110.07397720    (nil)          (nil)         (nil)       5652
  1      3      1      250      30001    116.28358926    (nil)          (nil)         (nil)          0
  2      2    -99        0      20002    0.00000000      (nil)          (nil)         (nil)          0
  3      1      3      250      10003    110.07397720    (nil)          (nil)         (nil)          0
  3      3    -99      250      20003    116.28358926    (nil)          (nil)         (nil)          0
  3      5      4      250      40003    114.64453168    (nil)          (nil)         (nil)          0
  4      3      4      250      30004    116.28358926    (nil)          (nil)         (nil)          0
  4      5      4      250      50004    114.64453168    (nil)          (nil)         (nil)          0
  5      3      4      250      40005    116.28358926    (nil)          (nil)         (nil)          0
  5      5    -99      250      20005    114.64453168    (nil)          (nil)         (nil)          0
  6      5      6      250      50006    114.64453168    (nil)          (nil)         (nil)          0
```

At 70.0 seconds node 2, the first hop used by the source node (1) to reach to the destination node (6) in 3 hops, starts to move gradually away from the source node transmission range. However, the transmission (traffic flow) still takes place in spite of the first hop (intermediate node) movement away from the source node (1) and the second hop node (5) as shown in Figure 5.11.

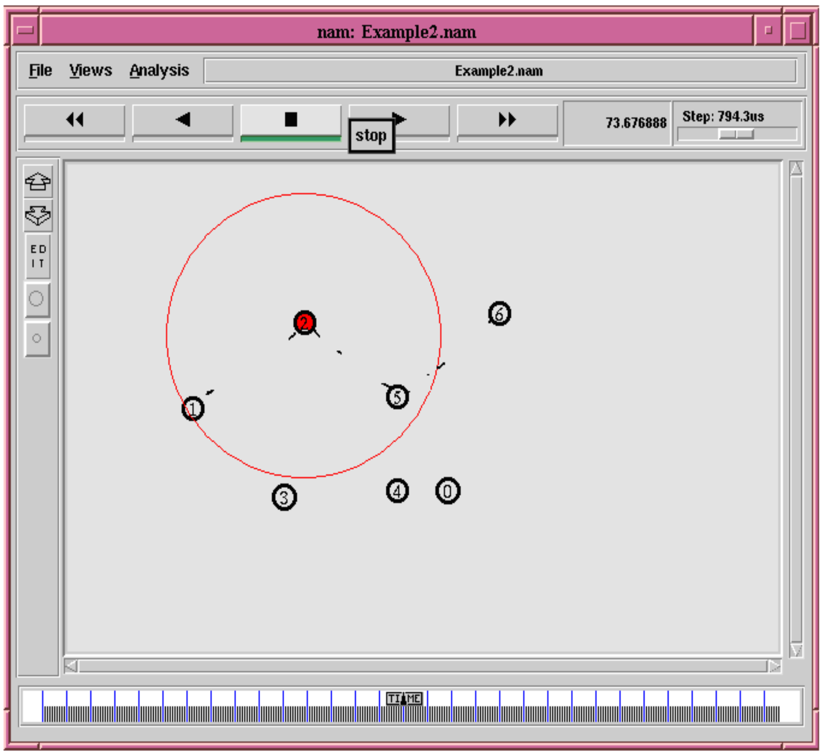
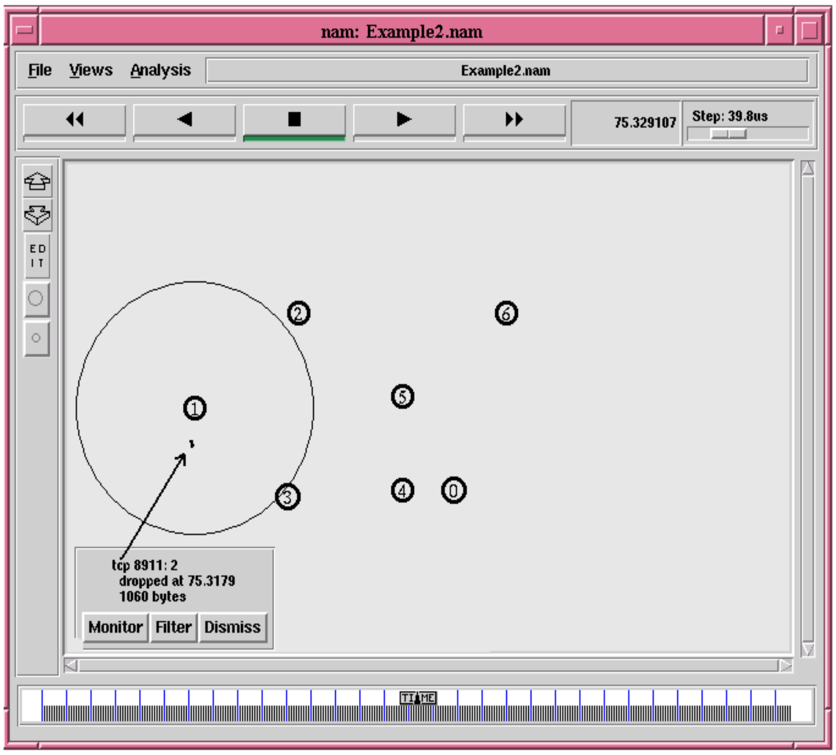


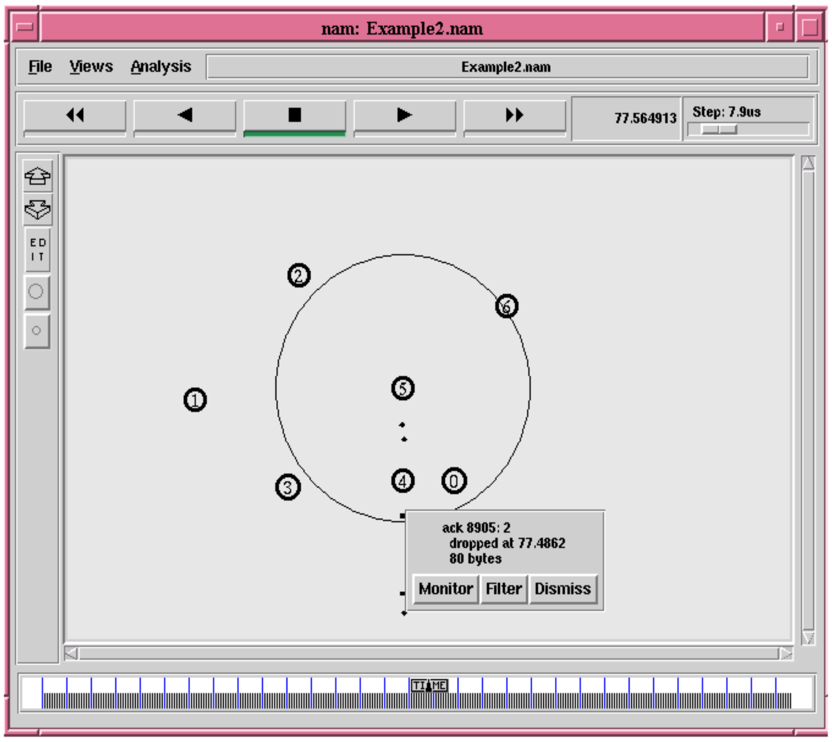
Figure 5.11: The TCP flow in spite of intermediate node movement

Figure 5.12 and 5.13 show that at 75.0 seconds node 2 became completely isolated and was outside of the radio transmission range not only of the source node (1) and the second hop (5) but entire the network as shown in its routing Table 5.9, causing broken links and the loss of few TCP packets.



tcp 8911: 2
dropped at 75.3179
1060 bytes

Figure 5.12: A broken link causing TCP packet lost at the source node



ack 8905: 2
dropped at 77.4862
80 bytes

Figure 5.13: A few control packets lost at an intermediate node

At 80.0 seconds the source node (1) used an alternative route in 4 hops instead of the broken route of 3 hops to establish the same connection to the destination node (6) via node (3) as a first hop, node (4) and node (5) as a second and third hop respectively as shown in Figure 5.14 and their routing tables are shown in Table 5.10 and Table 5.11.

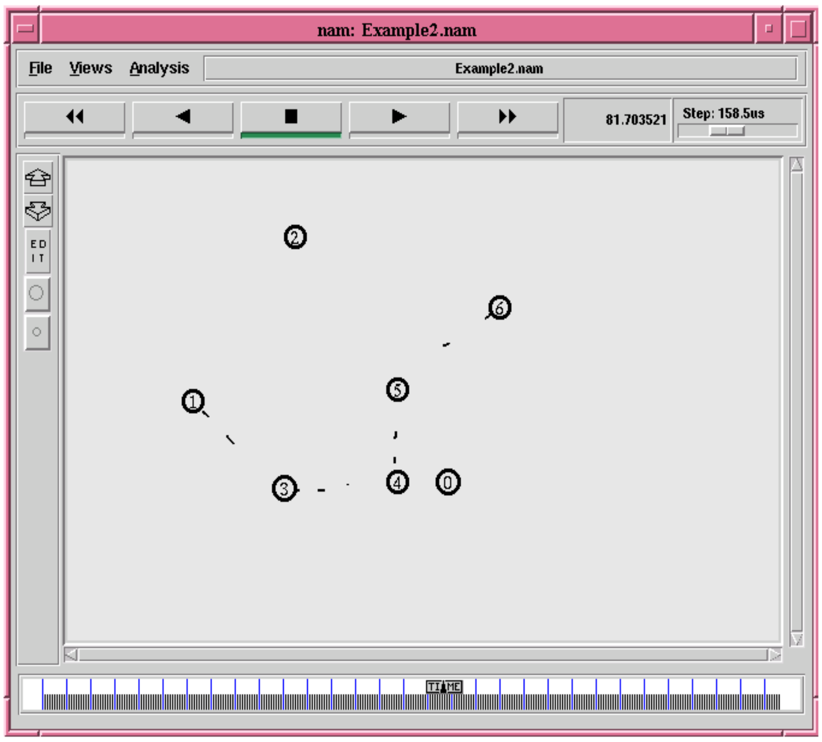


Figure 5.14: The TCP traffic back to flow again via an alternative route

At 150 seconds the simulation time ended successfully, and we have verified that the routing tables of the seven nodes are sorted and set up correctly, by viewing and analysing their routing table logging files. Tables 5.5-5.12 show some samples of the routing tables of nodes (1), (2), and node (5) at different times. The simulator trace log analysis as well as the MDVZRP log files shows that the scenario works as expected.

Table 5.10: A new route to the destination node in 4 hops, and old broken routes

```
*****  At  89.7138… Routing table of Node ( 1 )
-----------------------------------------------------------------------------------------------------
Dst    1-hop  2-hop   metric   link_num   change at   timeout_event  Queed-Data   Need Advertise  Sent_RREQ
-----------------------------------------------------------------------------------------------------
 0       2       5     250      50000    50.96021651      (nil)         (nil)        (nil)           0
 0       3       4       3      40000    16.49139686      (nil)         (nil)        (nil)           0
 1       1     -99       0      10001    0.00000000       (nil)         (nil)        (nil)           0
 2       2     -99     250      10002    75.31791565      (nil)         (nil)        (nil)           0
 2       3       2       2      30002    0.89172185       (nil)         (nil)        (nil)           0
 3       2       3     250      20003    50.96021651      (nil)         (nil)        (nil)           0
 3       3     -99       1      10003    0.89172185    0x95cd020        (nil)        (nil)           0
 4       2       5     250      50004    50.96021651      (nil)         (nil)        (nil)           0
 4       3       4       2      30004    16.49139686      (nil)         (nil)        (nil)           0
 5       2       5     250      20005    50.96021651      (nil)         (nil)        (nil)           0
 5       3       4       3      40005    16.49139686      (nil)         (nil)        (nil)           0
 6       3       4       4      50006    77.40399911      (nil)         (nil)         0x1            0
 6       6     -99     250      10006    77.40087065      (nil)         (nil)        (nil)           0
-----------------------------------------------------------------------------------------------------
```

Table 5.11: A new route and old broken routes at an intermediate node

```
*****  At  136.956... Routing table of Node ( 5 )
-----------------------------------------------------------------------------------------------------
Dst    1-hop  2-hop   metric   link_num   change at   timeout_event  Queed-Data   Need Advertise  Sent_RREQ
-----------------------------------------------------------------------------------------------------
 0       0     -99      1     50000    1.11821258    0x95d28a0      (nil)       (nil)          0
 0       4       0    250     40000    77.55085802     (nil)        (nil)       (nil)          0
 1       2       1    250     20001    77.47317802     (nil)        (nil)       (nil)         26
 1       4       3      3     30001    81.66525136     (nil)        (nil)       (nil)          0
 2       2     -99    250     50002    77.47317802     (nil)        (nil)       (nil)          0
 2       4       3    250     30002    77.55085802     (nil)        (nil)       (nil)          0
 3       2       3    250     20003    77.47317802     (nil)        (nil)       (nil)          0
 4       0       4      2     40000    14.12599291     (nil)        (nil)       (nil)          0
 4       2       3    250     30004    77.47317802     (nil)        (nil)       (nil)          0
 4       4     -99      1     50004    78.2566526    0x9631e78      (nil)       (nil)          0
 5       5     -99      0     50005    0.00000000      (nil)        (nil)       (nil)          0
 6       6     -99      1     50006    1.64095043    0x95d29c0      (nil)       (nil)      14790
-----------------------------------------------------------------------------------------------------
```

5.3.3 Example 3.tcl Scenario Overview

In this example we present a simple test for the protocol capability for sending data from one network to another. In Example 3.tcl, thirteen mobile nodes [0 – 12] are placed in such a way that they are divided into two separate groups, each group represents a separate network, where initially each group establishes communication between its members. Later communication between two mobile nodes in separate groups is established and tested.

5.3.3.1 Scenario Setup

The scenario places 7 nodes connected to each other via a wireless channel in a 1200x800 m rectangular grid area as shown in Figure 5.15. Nodes (1) and (8) are two d*estinations* each in a separate group ready to receive any incoming TCP traffic. While nodes (6) and (11) both have an FTP agent, simulating FTP traffic *Sources*.

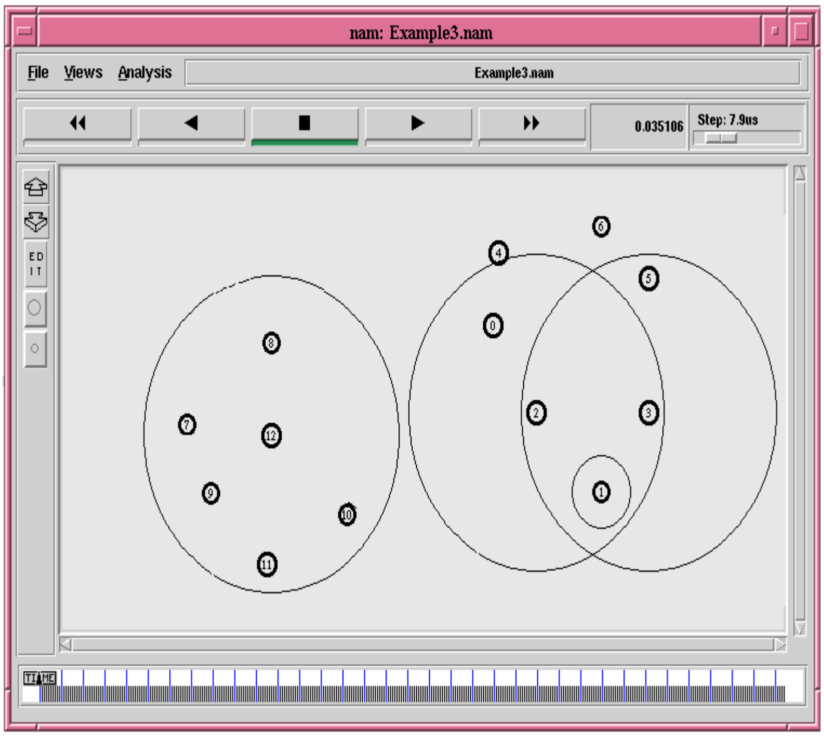


Figure 5.15: Two separate groups

5.3.3.2 Scenario description and results

Figure 5.16 shows a simple network of thirteen mobile nodes that are connected through a wireless channel. They are divided into two separate groups, where each group established its own communication between its members.

At 2.0 seconds the TCP traffic starts, in the first group on the right hand side we consider that node (6) is a source node and node (1) is a destination, TCP traffic flows in three hops distance (6-0-2-1) in between, the control packets flow in an alternative path in 3 hops also (1-3-5-6). While in the second group on the left hand side, node (11) sends TCP directly to the destination node (8).

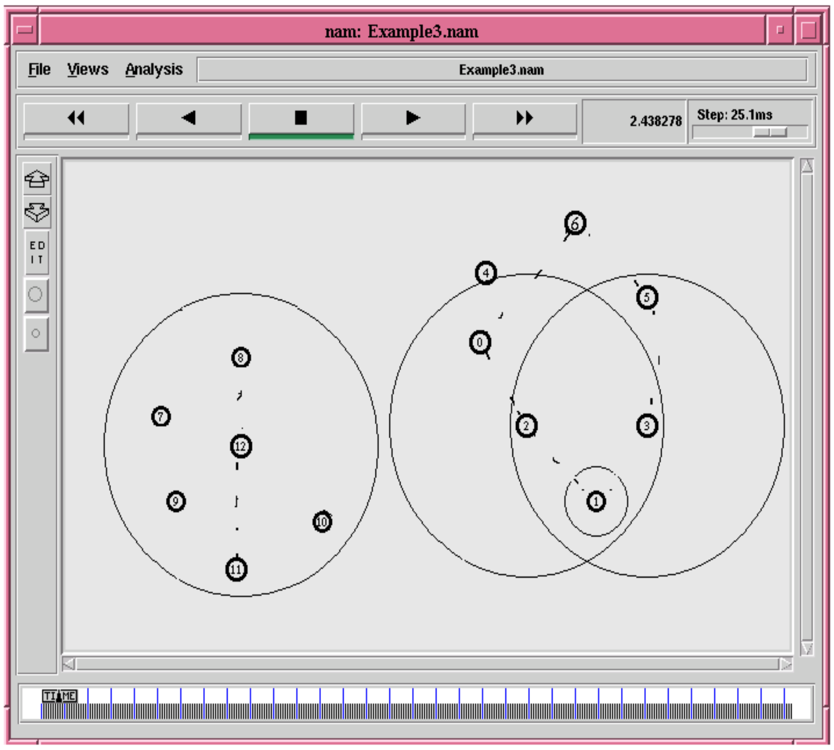


Figure 5.16: Two TCP traffic each in a separate group

At 5.0 seconds TCP connection between node (6) in this first group and node (9) in the second group is established. However, no TCP traffic took a place till 20.0 sec when nodes (12) and (10) start moving towards node (2) transmission range as shown in Figure 5.17 and Figure 5.18.

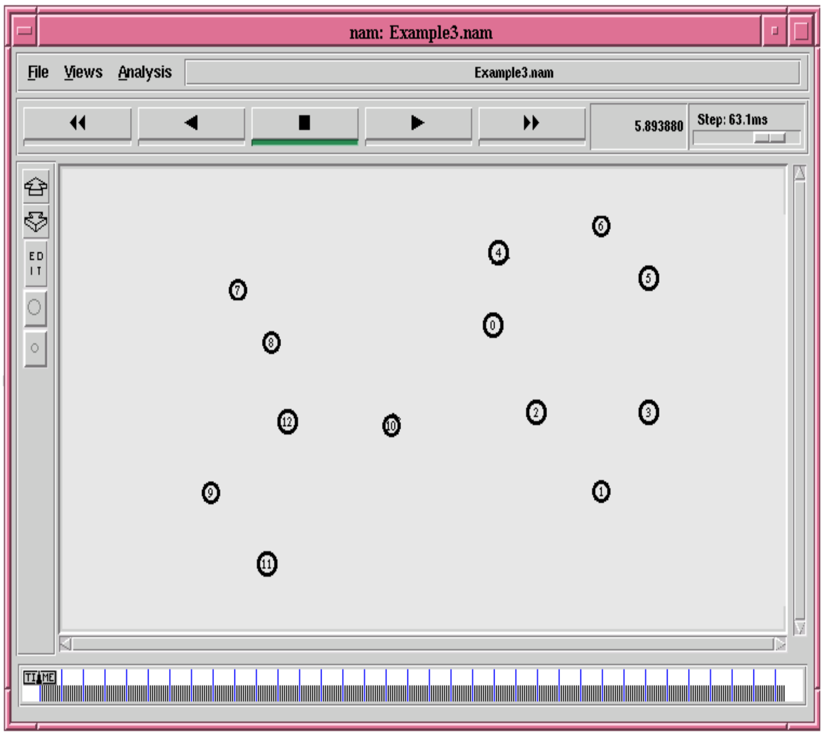


Figure 5.17: Nodes (12) and (10) movement

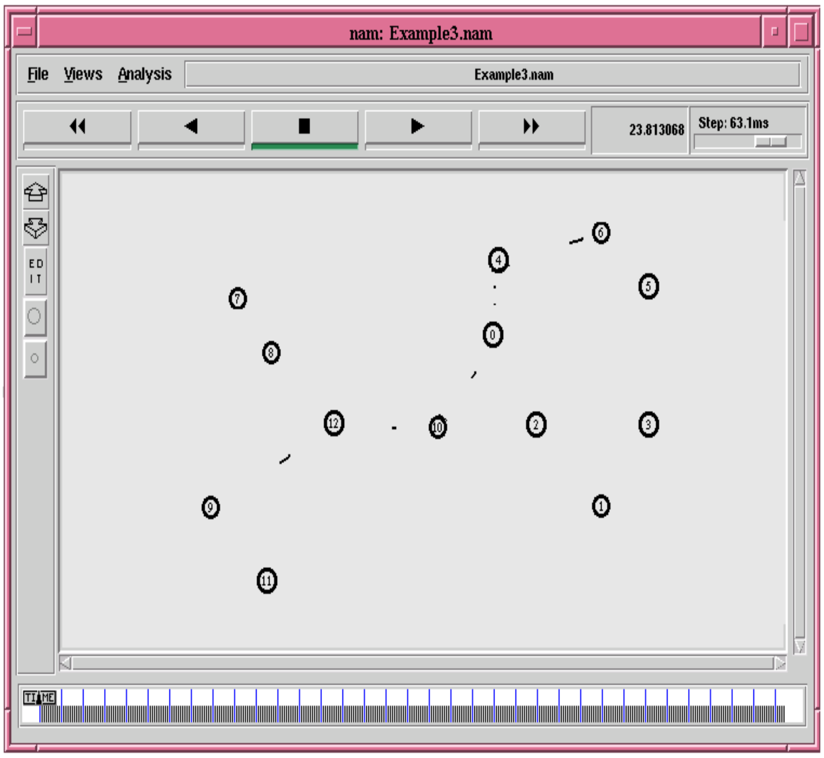


Figure 5.18: TCP traffic between node (6) and node (9)

5.3.4 Random.tcl Scenario Overview

In this example we have tested MDVZRP's ability to read the movement and traffic patterns from separate files.

5.3.4.1 Random Scenario Setup

The scenario places 100 nodes moving with a maximum speed of 15 m/s, connected to each other via a wireless channel, with a pause time of 2s, for a maximum simulation time of 150s within a topology (area) boundary of 800 x 700. The Nodes movement model (RWP) is loaded from a separate file (Mov-scn-N100-x800-y700). This scenario file has been generated by typing the following command in the `/ns-allinone-2.30/ns-2.30/indep-utils/cmu-scen-gen/setdest` directory:

```
/setdest -v 1 -n 100 -p 2 -M 15 -t 150 -x 800 -y 700> Mov-scn-N100-x800-y700
```

The connections are established between 3 pairs of nodes with data rate of 4 packets per sec. We specified this connection pattern in a separate file (cbr-scn-N100-1-C3-4cbr-2) by typing the following command in `/ns-allinone-2.30/ns-2.30/indep-utils/cmu-scen-gen/` directory:

```
ns cbrgen.tcl -type cbr -nn 3 -seed 1.0 -mc 1 -rate 4> cbr-scn-N100-1-C3-4cbr-2
```

5.3.4.2 Random Scenario description and results

Figure 5.19 shows a layout of a complicated random network of 100 mobile nodes that are connected through a wireless channel, within a topology boundary of 800 x 700, for a simulation time of 150 seconds. Figure 5.20 shows nodes random movement and CBR traffics flow. The scenario's results show that the protocol has successfully read the movement and traffic pattern from separate files. The visualisation of the NAM trace files by the network animator in Figures 5.19 and 5.20 also shows that the random-waypoint during the duration of the simulation (150 seconds) was clearly correct.

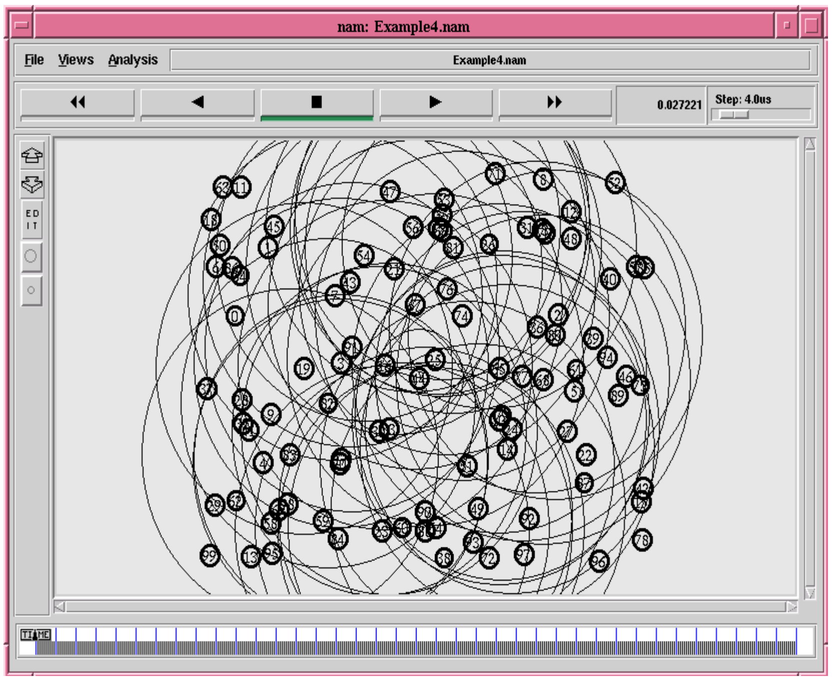


Figure 5.19: The layout of a random scenario of 100 nodes

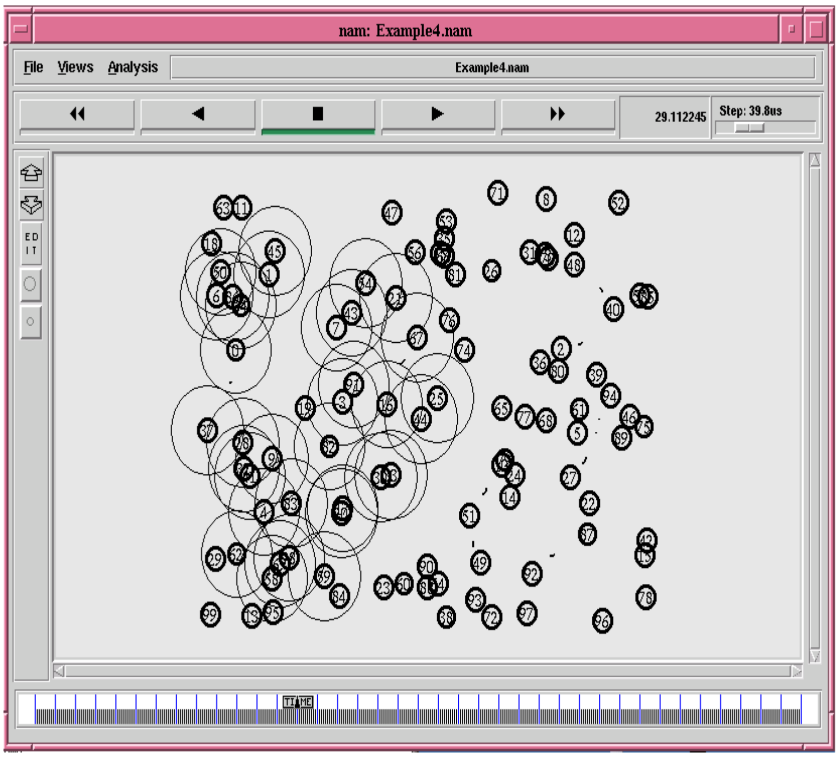


Figure 5.20: The random-waypoint movement with some traffic

5.4 Trace File Analysis for NS-2

The log or trace files would gather (cache) the required information that could be visualised such as in the NAM trace file (Kurkowski, 2005), or could be used in a network and protocols performance study, for example the amount of packets transferred, delay, and packet loss. These events log files contain packets being sent received or only dropped, so called Packet Traces. In fact, the trace files are just a text format in other words they are human readable, where we can access them using some form of offline software. In order to ease the process of extracting data for performance studies, many Ns2 Trace Analysers have been proposed (Salleh, 2006). In this thesis we have used our own programmes using AWK (Robbins, 2001), TCL and C++ to analyse the Ns2 trace files that have been generated from the scenarios which were used in the testing of our protocol. Ns2 offers three different formats of the trace file *Old*, *New* and *Tagged* trace format as mentioned in chapter 2. Since we have used the *New* trace format in our protocol study, we will give an overview of only that format which has been generated by Ns2 simulations.

5.4.1 Sample of Wireless Trace file

```
s -t 0.008749440 -Hs 0 -Hd -1 -Ni 0 -Nx 120.19 -Ny 17.57 -Nz 0.00 -Ne
-1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 0.255 -Id -1.255
-It BEACON -Il 20 -If 0 -Ii 0 -Iv 32

r -t 0.009645566 -Hs 10 -Hd -1 -Ni 10 -Nx 137.33 -Ny 51.10 -Nz 0.00 -
Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 0 -Mt 800 -Is
0.255 -Id -1.255 -It BEACON -Il 20 -If 0 -Ii 0 -Iv 32

s -t 0.033595809 -Hs 2 -Hd -1 -Ni 2 -Nx 86.44 -Ny 135.34 -Nz 0.00 -Ne
-1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 2.255 -Id -1.255
-It RSUP -Il 37 -If 0 -Ii 9 -Iv 32

r -t 0.034587863 -Hs 17 -Hd -1 -Ni 17 -Nx 96.62 -Ny 122.64 -Nz 0.00 -
Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md ffffffff -Ms 2 -Mt 800 -Is
2.255 -Id -1.255 -It RSUP -Il 37 -If 0 -Ii 9 -Iv 32

s -t 53.425636044 -Hs 1 -Hd -2 -Ni 1 -Nx 162.44 -Ny 262.52 -Nz 0.00 -
Ne -1.000000 -Nl AGT -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 1.0 -Id 2.0 -
It cbr -Il 512 -If 0 -Ii 641 -Iv 32 -Pn cbr -Pi 203 -Pf 0 -Po 2

r -t 53.425636044 -Hs 1 -Hd -2 -Ni 1 -Nx 162.44 -Ny 262.52 -Nz 0.00 -
Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 1.0 -Id 2.0 -
It cbr -Il 512 -If 0 -Ii 641 -Iv 32 -Pn cbr -Pi 203 -Pf 0 -Po 2

f -t 53.431526201 -Hs 18 -Hd 2 -Ni 18 -Nx 260.19 -Ny 70.21 -Nz 0.00 -
Ne -1.000000 -Nl RTR -Nw --- -Ma 13a -Md 12 -Ms 1 -Mt 800 -Is 1.0 -Id
2.0 -It cbr -Il 532 -If 0 -Ii 641 -Iv 31 -Pn cbr -Pi 203 -Pf 1 -Po 2
```

Figure 5.21: The new trace format

Figure 5.21 shows a sample from a wireless trace file of 7 events in a new trace format, which includes the following fields (Nikos, 2010):

Event type: It is the first field encoded as one small letter, and describes the node's event type which can be one of the four characters:

s: send

r: receive

d: drop

f: forward

General tag: The second field is the event's time, based on the simulation time, starting with "-t" which stands for time or global setting

-t time

-t * (global setting)

Hop information: The third and fourth fields, they respectively show the id of the current node and the next hop towards the destination and the tag starts with a leading "-H" as follow:

-Hs: ID for this node

-Hd: ID for next hop towards the destination.

Node property tags: These tags are encoded as two letters, the first letter is always a capital letter "-N", which indicates the first letter in the word of "Node", while the second small letter describes the current node's properties such as:

-Ni: Node's id

-Nx: Node's x-coordinate

-Ny: Node's y-coordinate

-Nz: Node's z-coordinate

-Ne: Node energy level

-Nl: The level at which tracing is being done like agent AGT, router RTR or MAC.

-Nw: Reason of the event, e.g. the following are some of the reasons for dropping a packet:

"END" DROP_END_OF_SIMULATION

"LOOP" DROP_RTR_ROUTE_LOOP i.e. there is a routing loop

"ERR" DROP_MAC_PACKET_ERROR

"TTL" DROP_RTR_TTL i.e. TTL has reached zero

"IFQ" DROP_IFQ_QFULL i.e. no buffer space in IFQ

"CBK" DROP_RTR_MAC_CALLBACK

Packet information (at IP level)**:** This field tags start with a leading "-I", the following is a list for these tags along with their explanations.

-Is: Source address.port number

-Id: Destination address.port number

-It: Packet type `(message, Beacon, RSUP)`

-Il: Packet size

-If: Flow id

-Ii: Unique id

-Iv: TTL value e.g. 32

Packet info at MAC level: MAC layer information field always starts with a leading "-M" and are listed along with their explanations as following:

-Ma: Duration.

-Md: Destination MAC address

-Ms: Source MAC address

-Mt: Ethernet type (800=IP, 806=ARP)

Packet info (information) **at Application level:** This field tags start with a leading "-P". It consists of the application type (ARP, TCP) and routing protocol (e.g. MDVZRP, DSDV, or AODV) being traced. Here is a list of tags for various applications:

-P arp: Address Resolution Protocol.

-Pm: Source MAC address

-Ps: Source address

-Po: ARP Request/Reply

-Pa: Destination MAC address

-Pd: Destination address

-P cbr: Information about the CBR application is represented by the following tags:

-Pi: Sequence number

-Pf: How many times this packet was forwarded

-Po: Optimal number of forwards

-P tcp: Information about TCP flow is given by the following sub tags:

-Ps: Sequence number

-Pa: Acknowledge number

5.5 Summary

We have run more than 1500 scenarios during MDVZRP design, implementation and testing. In this chapter we presented 4 different scenarios, each scenario tests a specific situation. The results of all tested scenarios show that the MDVZRP in Ns-2 works as expected and its functions perform their jobs as designed and implemented. In addition, the analysis of the log files for both the animator (NAM) and trace that were produced by the network simulator as well as the MDVZRP log files for the mobile nodes routing tables all show a good indication that our protocol is performing its functionality as we expected. Finally, we have presented the new trace format which is used to calculate measures including packet delivery ratio, end-to-end delay, and throughput and packet drop ratio.

CHAPTER 6: PERFORMANCE EVALUATION

6.1 Introduction

In this chapter, the performance of MDVZRP v2.0 is evaluated and compared to the performance of both standard proactive and reactive protocols. We have also showed and explained the performance improvements of MDVZRP v2.0 compared to MDVZRP v 1.0. From now by MDVZRP we mean MDVZRP v2.0.

MDVZRP was evaluated using the discrete event Ns2 (Kevin, 2010) version 2.30. The Ns2 simulator includes both proactive and reactive protocols. Some of them such as DSDV, AODV, DSR and TORA are already included within the download version as standard protocols, while other protocols are available as separate downloads. This simulator is designed especially for both wired and wireless network research. It is an open source supported by a large global user base, for this reason researchers and developers of Ad Hoc protocols have designed and implemented their protocols under Ns2.

6.2 Performance Evaluation Metrics

The primary metrics we considered evaluating the performance of MDVZRP with were Packet Delivery Fraction (PDF), End to End Delay (EED), Throughput, Normalised Routing Load (NRL) and Overhead (OH). These are defined as follow:

Packet Delivery Fraction: is the ratio of received packets by CBR sink at destination over sent packets by constant bit rate source (CBR,"application layer"). This metric actually tells us how reliable the protocol is.

$$\text{PDF} = \frac{\sum CBR\ \text{Re}ceived\,Packets\,by\ \ CBR\ \ destination}{\sum CBR\,Sent\,Packets\ \ by\ \ CBR\,sources} x100 \qquad (6.1)$$

End-to-End Delay: is the delay in seconds that could be caused by buffering during route discovery, queuing delays at interface queues, retransmission delays at the MAC, and propagation and transfer times.

$$\text{EED} = \frac{1}{N}\sum_{n=1}^{N}(r_n - s_n) \qquad \text{sec.} \qquad (6.2)$$

s_n = Time that data packet n was sent

r_n = Time that data packet n was received

N = Total number of data packets received

Normalised Routing Load: is the number of routing packets sent per one data packet delivered at the destination. Each routing packet hop–wise is counted as one transmission.

$$\text{NRL} = \frac{Routing\ \ Control\ \ Packets}{Data\ \ Packets\ \ Delivered} \qquad (6.3)$$

Throughput: is the amount of data (bits) transferred from the destination node to the source node during a specified amount of time (s).

$$\text{Throughput} = \frac{Size\ of\ \text{Re}ceived\ Data}{Transmission\ StopTime - Start\ Time}\ x\ \frac{8}{1000} \qquad (6.4)$$

Routing Overhead: is the sum of all the routing control packets sent during the simulation time. The control packets include all MDVZRP’s routing packets such as Beacon, RSUP, RREQ, and RREP, which we reviewed in the previous chapter. For all the forwarded packets over multiple hops, each packet transmitted over multi hops counts as one transmission.

$$\text{OH} = \sum Transmissions\ Of\ Routing\,Packets \qquad (6.5)$$

This metric is important to compare the performance of routing protocols over scalability and power consumption (less routing packets sending is less power consumption). The probability of packet collision is also increased with the rise in sending routing packets where that may increase the delay data packets to be sent or waiting in queues. Table 6.1 shows the characters that specify the packet process action.

Table 6.1: Packet Process Actions

Packet Process Action		
1	s	send
2	f	forward
3	r	receive
4	d	drop

6.3 Simulation Models

In our study and evaluation for the three protocols DSDV, AODV and MDVZRP, we have used the radio propagation simulation model based on Ns2 (v2.30). The IEE802.11 protocol (DCF) Distributed Coordination function (IEEE Standards Department., 1997) is used as the MAC layer protocol. The mobility mode is a Random Way Point RWP in a rectangular field. Two fields are used in this simulation study 550x500 and 750x600 m with 20, 60 and 100 nodes, where each node at the start of the simulation time remains fixed for an instance of time from 0-100 seconds (Pause time), then choose an arbitrary destination (Random) from its location and starts its journey towards it with a randomly selected speed, uniformly distributed between 0-20 m/sec, compared to the traffic speed inside the cities this is a fair speed for an Ad Hoc network.

Once the node reaches that desired destination, it stops for a pause time interval, and then another random destination is targeted with the same or different speed. The relative speeds of the mobiles are affected by varying the pause time, (which can be seen clearly in this thesis’s graphs). The simulation time of each scenario is run for 100 seconds. The traffic sources used are Constant Bit Rate (CBR) with 512 byte data packets in 4 packets per second per source, where each pair of source and destination is randomly distributed over the network.

For accuracy, across the three protocols, identical scenarios in mobility and traffic are used, and repeated 10 times. Therefore, each data point in this thesis's graphs represents an average of 10 iterations. In this simulation model we also assume that each node has sufficient power to function properly throughout the simulation time.

6.4 Simulation System Environment

For accuracy, we ran more than 1500 scenarios in both Phases using the Network Simulator 2 version 2.30, and the same machine for the three protocols at the same time based on the same parameters listed in Table 6.3 and Table 6.4. Table 6.2 shows the machine specification used in MDVZRP design, simulation and analysis.

Table 6.2: Simulation Environment Specification

Machine Specification					
Model	CPU	CPU's Speed (Ghz)	Memory Size (GB)	Memory speed (Hz)	Operating System
HP Compaq	Intel Pentium 4	3.40	0.99	2.78	Linux

6.5 Evaluation Methodology

This thesis's experimental work was divided into the following three phases:

Phase 0: is concerned with the performance comparison of MDVZRP v1.0 with MDVZRP v2.0, where we show this graphically with a brief explanation of how the performance developed to the 2nd version.

Phase 1: is concerned with determining optimum values of MDVZRP's two parameters: Route request threshold *RREQ_Threshold* and route request time out delay *RREQ_TimeOut*, which we mentioned in the previous chapters.

Phase 2: is concerned with results that we have obtained from phase1 to study and investigate the performance of MDVZRP in different network scenarios. As we mentioned previously, the performance results were compared to other two standard protocols DSDV and AODV, and were also simulated using the same scenarios and under identical conditions. In the following sections, we are going to describe each phase in detail.

6.5.1 Phase 0

In this Phase, we have focused on the development and improvement to MDVZRP v1.0. These were not the only improvements for MDVZRP, but there are other improvements carried in the other Phases, and we are going to explain them in detail in each Phase.

MDVZRP v2.0 shows better results during the simulation time in Phase 0, after the improvements and development that were made to v1.0 which explained and clarified in details in the preceding chapter, we have shown that improvements based on the MDVZRP v2.0 performance which will be explained briefly and graphically in the Phase 0 discussion section later on in this chapter.

6.5.2 Phase 1

The MDVZRP protocol was reconfigured for Phase 2 by using simulating Ad Hoc network of 20, 40, 60, 80, 100, 120, 140, and 160 nodes in Phase 1 to determine the optimum operating parameters values depending on the network size where each node has a 250m transmission range, a Carrier sensing range of 550m, queue size of 50 packets and a data rate of 2Mb/s IEEE 802.11. Node movement scenarios and the traffic patterns were generated using the parameters in Table 6.3 and Table 6.4.

Table 6.3: Parameters Specifying Node Movement and Network Size Patterns

Node movement scenarios and Network size parameters						
Mobility model	Network Size (Node)	Topology Size (m)	Transm. Range (m)	Node’s Speed (ms^{-1})	Pause Time (Seconds)	Simulation Time (Seconds)
RWP	20 - 160	500x550	250	20	0-100	100

The simulation parameters listed in Table 6.3 and Table 6.4 were chosen based on some papers that studied performance comparison of TCP and CBR in both reactive and proactive Ad Hoc protocols (Marina, 2001). The main differences in the selection of these parameters are the number of nodes that were simulated, the number of connections, and network size.

Table 6.4: Parameters Specifying Traffic Patterns

Traffic patterns parameters				
Mobility model	Traffic Type	Packet Size (Byte)	Max. Connections	Sending Rate (Packet/s)
RWP	CBR	512	5-50	4

Random Waypoint (**RWP**) model (Bettstetter, 2003; Hyytia, 2005) is used very widely for mobility in Ad Hoc network scenarios. In this basic model, each node moves randomly without restrictions and independent in speed and direction of other nodes. Its movement and direction must be within a topology or domain given in X and Y coordination in uniform velocity distribution U (0 ms, Vmax). Once it reaches that destination, optionally the node waits for a period called *pause* or *thinking* time before continuing the same process to the next waypoint destination.

Our choice for RWP, despite the existence of other models such as Random Walk Mobility Model, Markovian Waypoint Model MWP, and Random Direction Mobility Model, was based on its simplicity in the use, implementation and observation of the process simulation, also; some performance measures could be computed directly. These basic properties made it one of the most used models in Ad Hoc network simulations at present.

The significance of the selection of MDVZRP's parameters optimum values of RREQ Threshold (RREQ_Threshold) and RREQ Time Out Delay (RREQ_TimeOut) came from the observation in simulation time of various scenarios using RWP technique, where we found that the optimum value of RREQ_Threshold is 2 for 500x500m topology, that is because each node in the network is equipped with an IEEE interface with a transmission range of 250m, that gives it the ability to cover the topology area within 500x500m in a maximum of 2 hops distance, while the optimum value of RREQ_TimeOut is 1.6 sec more or less increasing the control packets (Overhead), or end to end delay (EED). This means that the requester node waits for a period equal to the last RREQ time (change_at) + RREQ_TimeOut before rebroadcasting the same RREQ again in case of no RREP received.

10, 20 and 50 iterations of the simulation scenarios were carried out separately, and the results were averaged. Each scenario utilised varying randomly generated mobility and traffic patterns, with the parameters listed in Table 6.3, and Table 6.4. The total number of simulations carried out in Phase 1 is the product of the number of iterations, the number of different network sizes simulated, the number of values of RREQ_Threshold and RREQ_TimeOut.

Figure 6.1 shows the directory structure of Phase1 that was created based on the simulation results, for couple values of RREQ_Threshold =1, 2 …32. As the figure illustrates, for RREQ_Threshold's values, a number of RREQ_TimeOut were investigated. The results we have got from 5x3x10 simulation scenarios show that the value of the Overhead increases gradually as RREQ_Threshold increases and RREQ_TimeOut decreases, while the EED increases gradually as both RREQ_Threshold and RREQ_TimeOut increase as shown in Figure 6.1.

Number of simulations = 5(Networks) x 3(Number of protocols) x 10(Iterations)

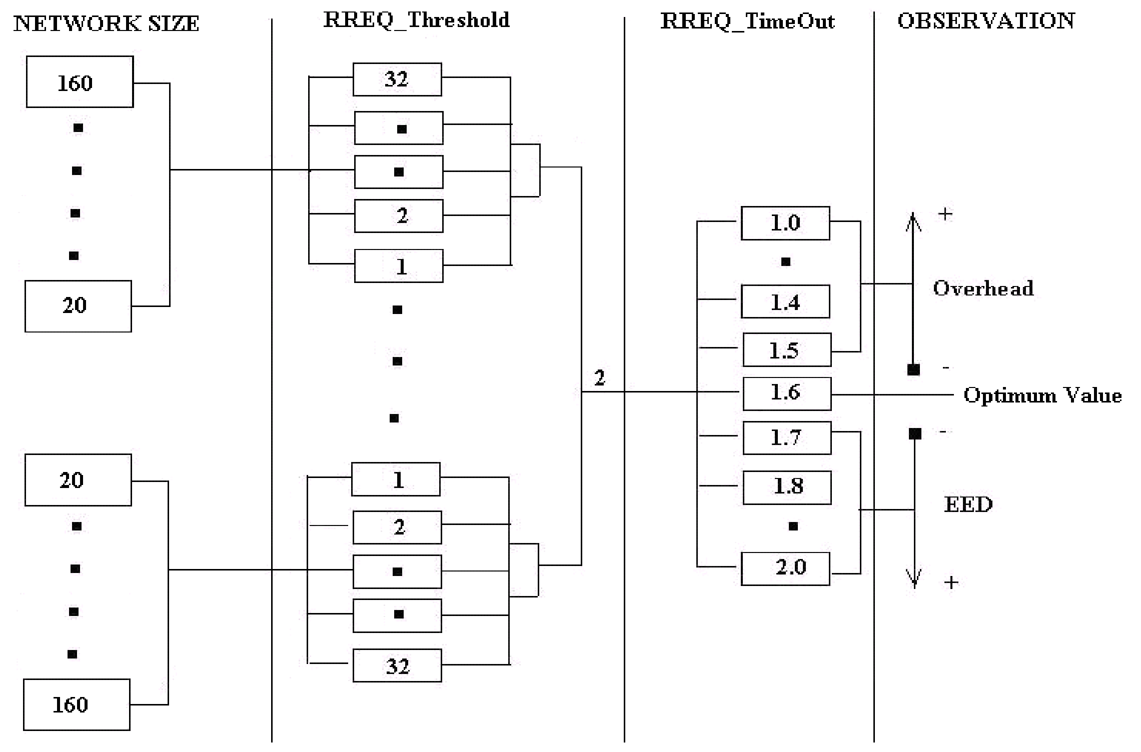


Figure 6.1: Phase 1 Directory Structure

6.5.3 Optimum Values of MDVZRP

Increasing and decreasing the RREQ_Threshold or RREQ_TimeOut values rather than their optimum ones affects the MDVZRP's performance as we can see in Figures (6.7-12) in Phase1 results discussion section. Table 6.5 shows the optimum values of RREQ_Threshold and RREQ_TimeOut based on the network's topology size.

Table 6.5: MDVZRP's Optimum Values

MDVZRP's Optimum Values			
Network Size X,Y (meter)	Minimum and Maximum distance	RREQ_Threshold	RREQ_TimeOut sec
250 x 250	$(X,Y) \leq 250$	1	1.6
500 x 500	$250 < (X,Y) \leq 500$	2	1.6
750 x 750	$500 < (X,Y) \leq 750$	3	1.6
1000 x 1000	$750 < (X,Y) \leq 1000$	4	1.6

6.5.4 Phase 2

MDVZRP shows better results during the simulation time in Phase 2, where the optimum values of RREQ_Threshold and RREQ_TimeOut from Phase 1 were used to reconfigure the simulator, as we going to focus on that in a brief discussion in the following sections. Network size and traffic parameters were maintained from Phase 1. However, this time, instead of testing against varying values of MDVZRP's two operating parameters, each simulation was executed with respect to the degree of mobility in the network, measured in terms of nodes *pause time* and *movement speed*.

Figure 6.2 illustrates the structure of the work that carried out in Phase 2, showing the varying degrees of mobility that MDVZRP was evaluated against in terms of pause time and node movement speed, which is considered to be a good indication of how dynamic a network is. A total of 10 iterations of each simulation were carried out, and the results averaged. Each time for 10 iterations, we have used different traffic patterns and a set of random mobility scenarios.

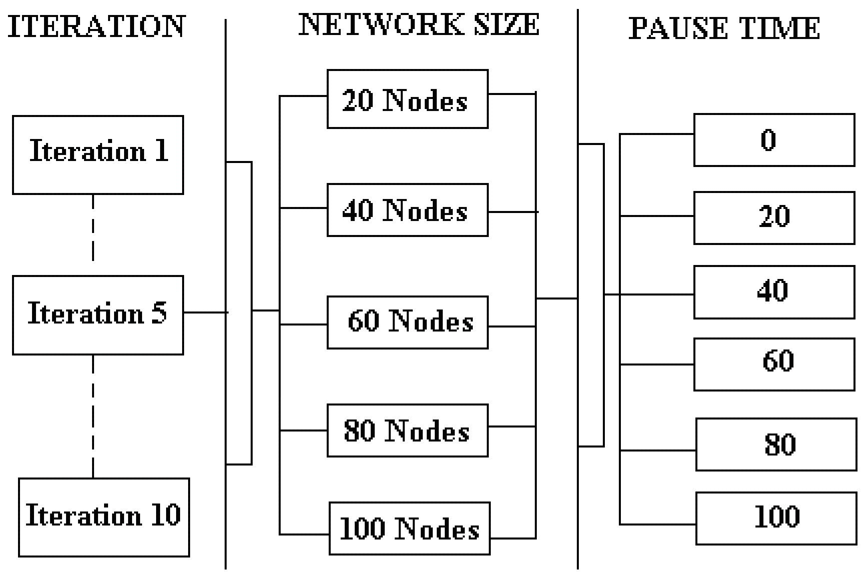


Figure 6.2: Phase 2 Directory Structure

To find out the strengths and weakness of MDVZRP, the same sets of simulations that were carried out in Phase 2 were also conducted for the other two protocols AODV and DSDV to evaluate their performance under identical conditions as MDVZRP.

The total number of simulations carried out in this part of Phase 2 is the product of the number of iterations (10), the number of different networks (5), number of different pause times (6), and number of protocols(3).

Number of simulations = 10x5x6x3 = 900

The total number of trace files that have been generated are the product of the number of iterations (10), the number of different networks (5), number of different pause times (6), number of protocols(3), number of connections (4), and number of speeds (4).

Number of trace files = 10x5x6x3x4x4 = 14400 files

The total number of files that have been produced by the analyser program is the product of the number of different networks (5), number of connections (4), and number of speeds (4).

Number of analysis files = 5x4x4 = 100 files

91.1 GB space needed in the hard disk to store all that files without the performance evaluation graphs.

6.6 Phase 0 Results Discussion

This Phase is concerned with the performance comparison of MDVZRP v2.0 with MDVZRP v1.0. The following figures (6.3-6) show the effect of the changes that were made on MDVZRP v1.00 compared to the developed version v2.00. Where, Figure 6.3 shows that the Normalised Routing Load NRL as a function of pause time in v2.00 is better than in v1.00.

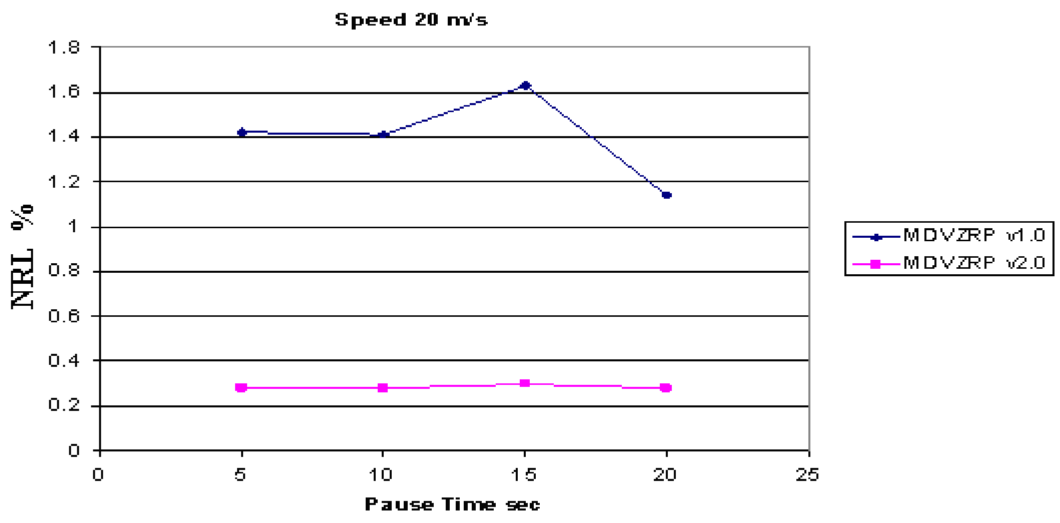


Figure 6.3: NRL Over MDVZRP v1.00 and v2.0, Speed 20m/s

Figure 6.4 shows the increasing of Packet Delivery Fraction as a function in both pause time and speed over MDVZRP v2.00 compared to v1.00 especially at pause time 15 sec.

Increasing speed with a short pause time increases the possibility of link breakage. Due to using a combination of proactive and reactive techniques, increasing the PDF in v2.0, while in v1.00, the PDF increases as the network attend to be stationary with a slow speed and long period of pause time.

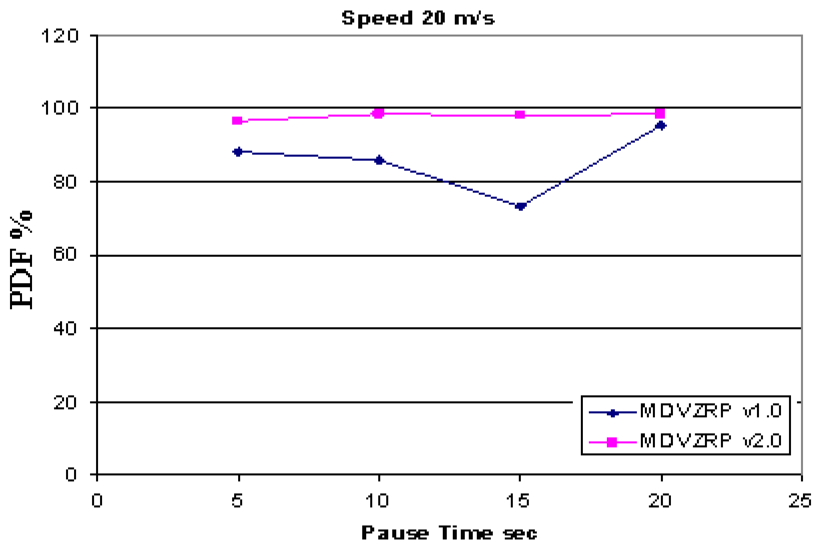


Figure 6.4: PDF over MDVZRP v1.00 and v2.0, Speed 20m/s

Figure 6.5 shows a better reading of End to End Delay EED as a function in both pause time and speed over MDVZRP v2.00 compared to v1.00. EED decreased in v2.0 by about 60.75 % compared to v1.0 as an average, as the network's nodes speed increased.

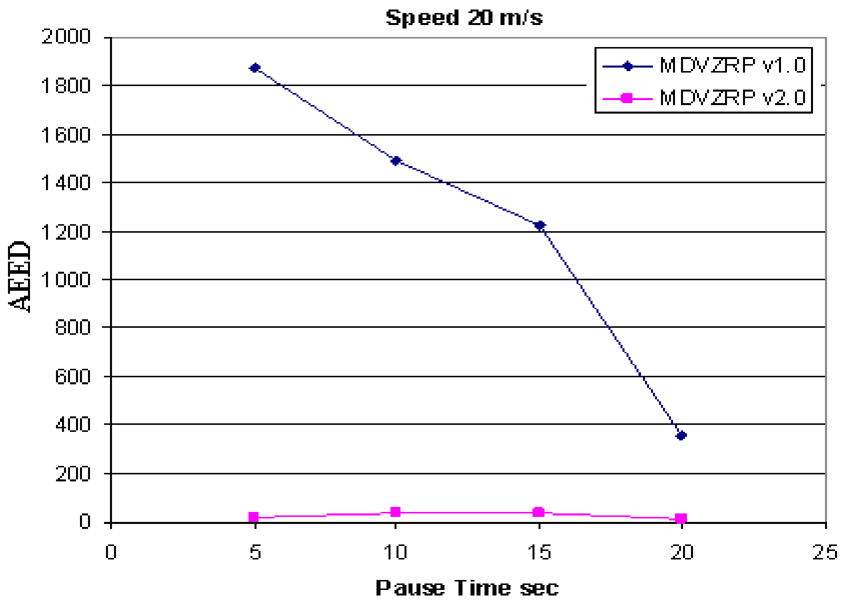


Figure 6.5: EED over MDVZRP v1.00 and v2.0, Speed 20m/s

Finally, we can confirm from the preceding figures that the changes made in MDVZRP's version 1.0 for obtaining version 2.0, show that the performance as an overall average of MDVZRP v2.0 became better than that in v1.0 by about 90.5%, we can also touch that in the MDVZRP's v2.0 overhead in the Figure 6.6 where v2.0 shows less overhead than v1.0 by about 94.5%.

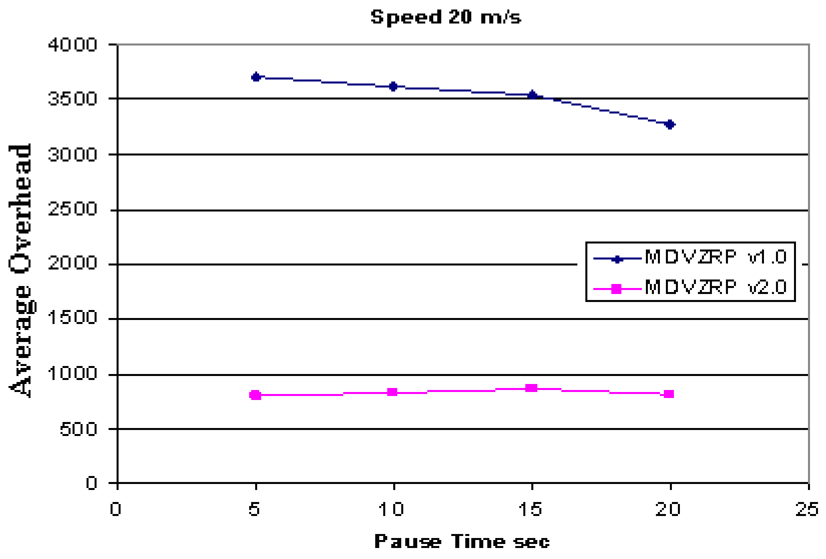


Figure 6.6: Overhead over MDVZRP v1.00 and v2.0, Speed 20m/s

6.7 Phase 1 Results Discussion

Scenarios testing during Phase 1 show that choosing the right RREQ_Threshold parameter for each network size had a direct effect on the results of the simulations especially in the amount of sent control packets *Overhead* and end to end delay *EED* metrics, and hence effected the amount of data transferred in other words Throughput. This is due to fact that the maximum extent to which control packets may travel is affected by the size of the zone radius and RREQ_Threshold value. Therefore, increasing or decreasing of RREQ_Threshold to be a bigger or smaller topology area affects the protocol's performance.

For example, if the network topology area despite the nodes density is 500 x 500 more or less, then the node that lay on boundary is the farthest node from the centre of the network in maximum of 250m, and maximum distance between any two nodes lay on corners is 500m. While each node's transmission range is 250m, which covers that area for a maximum of 2 hops, the optimum RREQ_Threshold in this case is 2. Therefore, each node has to cancel (drop) any RREQ packet after its TTL becomes zero, which is equal to RREQ_Threshold value (2 hops) at the RREQ initiating, to reduce the number of forwarded RREQ and RREP packets and hence reduces overheads in general.

Increasing the RREQ_Threshold more than the optimum values as shown in Table 6.5, may make some control packets such as RREQ travel to the maximum extent where that increases the amount of Overhead. On the other hand, decreasing the RREQ_Threshold than the optimum value increases EED due to the lack of RREQ, and thence no RREP packets back received.

The following Figures show the effect of RREQ_Threshold on the MDVZRP's performance in case of choosing a non optimum value of RREQ_Threshold. Figure 6.7 shows that the MDVZRP's EED increased gradually with scalability till it became higher than AODV when RREQ_Threshold was 4 for a network size of 500x550, while Figure 6.8 shows that the MDVZRP's EED increased gradually with scalability but was still better than AODV. EED decreased when RREQ_Threshold was 2 (Optimum value) compared to when it was 4.

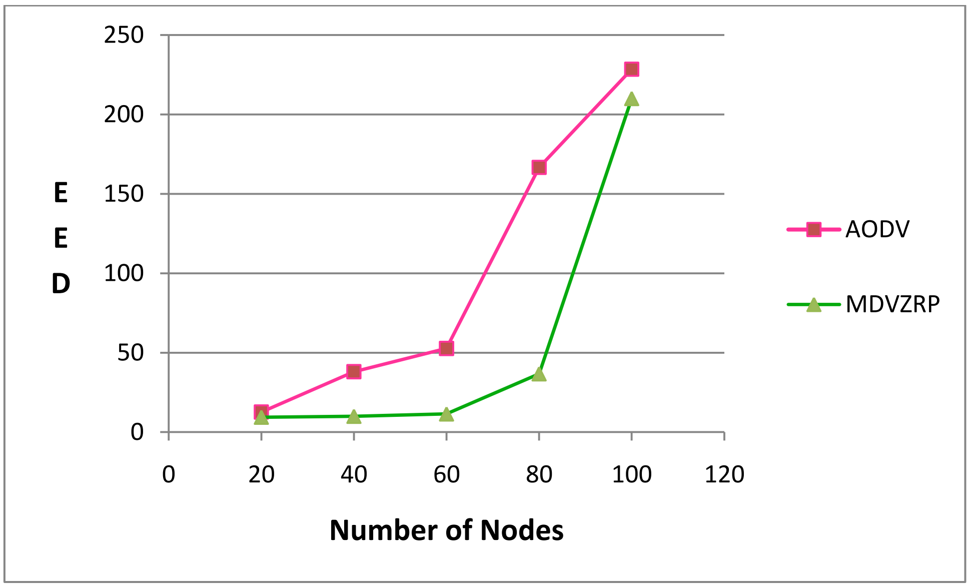


Figure 6.7: EED for **RREQ_Threshold** = 4

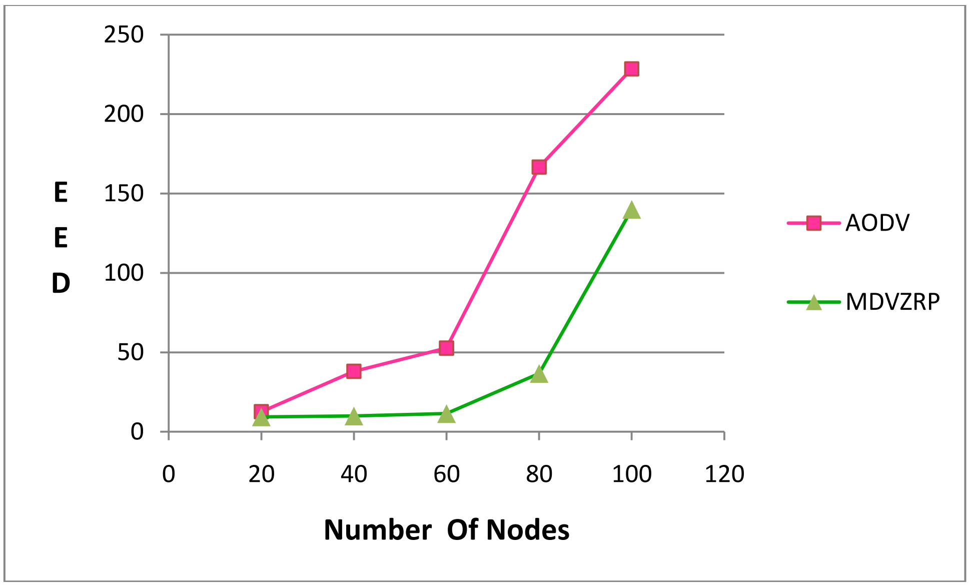


Figure 6.8: EED for **RREQ_Threshold** = 2

Figures 6.9 and 6.10 show the effect of RREQ_Threshold on the average Overhead, where it is decreased with the optimum value of RREQ_Threshold = 2. While Figures 6.11 and 6.12 show that the MDVZRP's average number of dropped packets is decreased with optimum value of RREQ_Threshold.

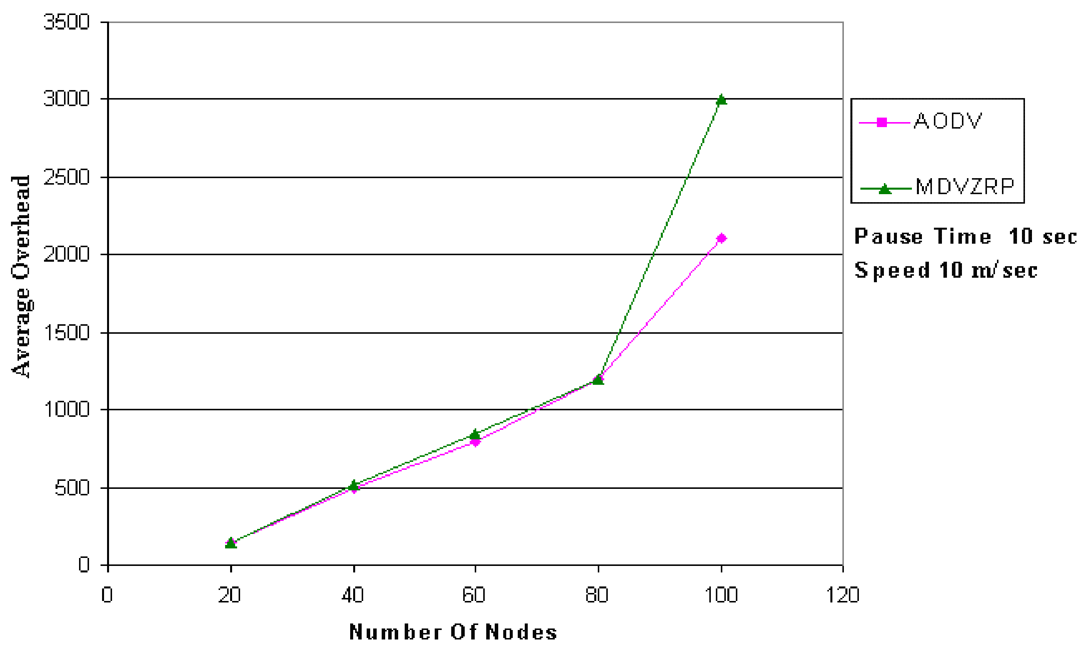


Figure 6.9: Average Overhead for **RREQ_Threshold** = 4

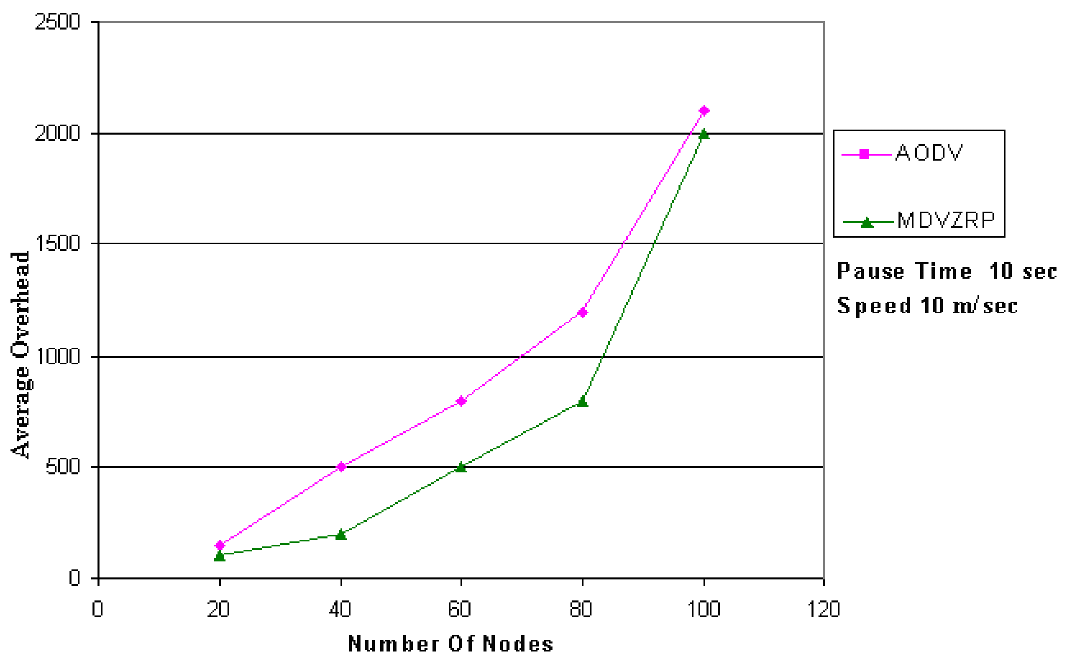


Figure 6.10: Average Overhead for **RREQ_Threshold** = 2

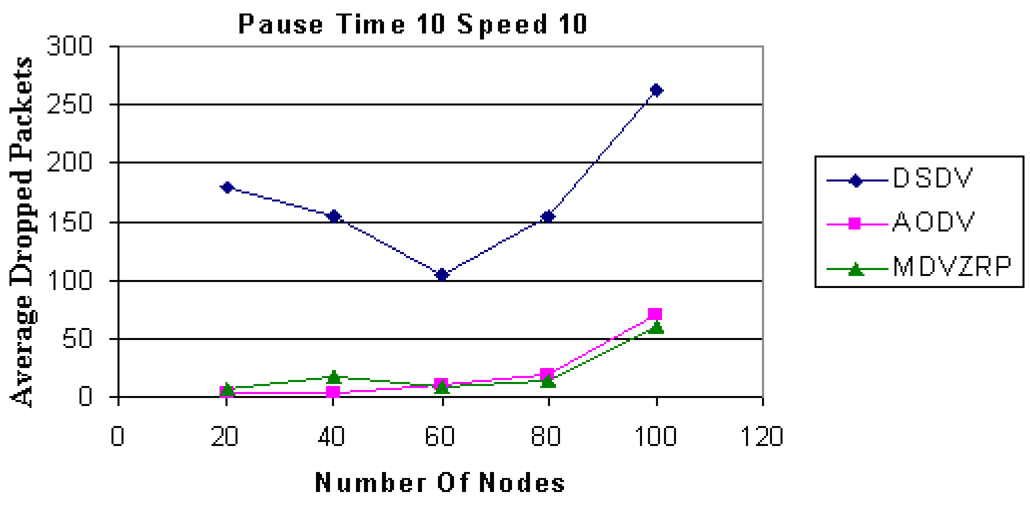


Figure 6.11: Average Dropped Packets for **RREQ_Threshold** = 4

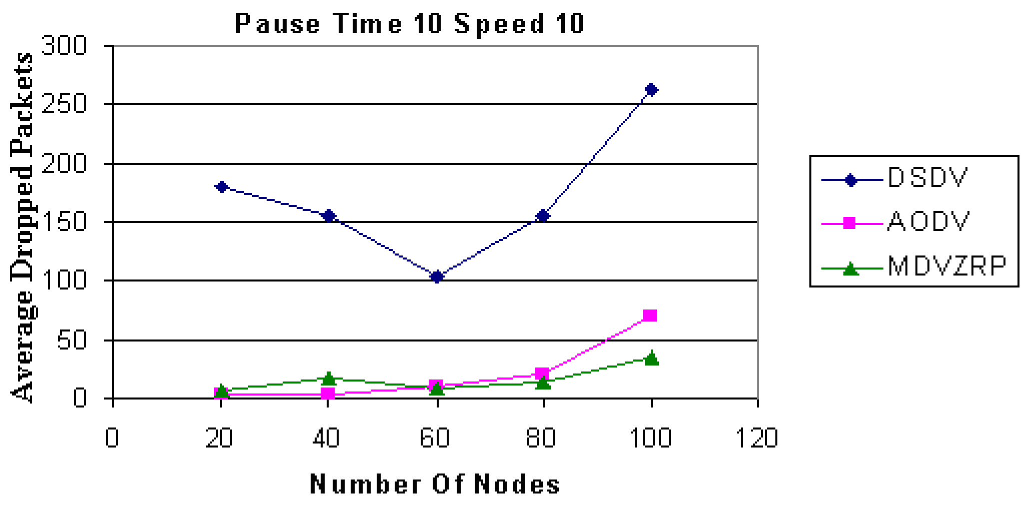


Figure 6.12: Average Dropped Packets for **RREQ_Threshold** = 2

6.8 Phase 2 Results Discussion

In this section, we present and discuss the results that we have got in Phase 2, which are the performance comparison of MDVZRP with DSDV and AODV. These performance analysis, study and discussion are based on the effect of scalability, mobility and congestion. The results of Phase 2 that are presented show the performance of the three investigated protocols over several of networks with only CRB traffic.

We have divided phase 2 results and discussion into three sets of experiments; the 1st set studies the performance of the three protocols over a small number of nodes (20 nodes) with 10, 15 and 20 traffic sources, while the 2nd set is 60 nodes with 20, 30 and 40 traffic sources and a packet rate of 4 packets/s. The 3rd set is 100 nodes with 20, 40 and 50 sources. Please note, for 40 and 50 sources we kept the same packet rate (4 packets /sec) because we would like to see the affect of the congestion on the three protocols' performance as well

6.8.1 The Graphs and How to Read Them

Figures (6.14-64) show the performance of MDVZRP compared to the most known MANET standard protocols DSDV and AODV. Each network size has 6 graphs (PDF, NRL, Throughput, EED, OH and Average Dropped Packets) by 3 different connections. Each point on the graph is an average of the results obtained from the 10 iterations performed. This study shows the spread in the data collected. 95% a sample of confidence intervals for PDF are given in Table 6.6, and shown in Figure 6.13. Confidence intervals for the other metrics are of similar magnitude, but are omitted from the graphs for clarity.

Table 6.6: A sample of PDF confidence interval and error bars

Parameters	DSDV		AODV		MDVZRP	
Num. of Nods	PDF %	Confidence Interval	PDF %	Confidence Interval	PDF %	Confidence Interval
20	85.31	2.48	99.6	2.48	98.9	2.49
60	86.23	4.36	98.85	4.40	98.87	4.43
100	83.97	6.14	92.87	6.81	97.09	6.85

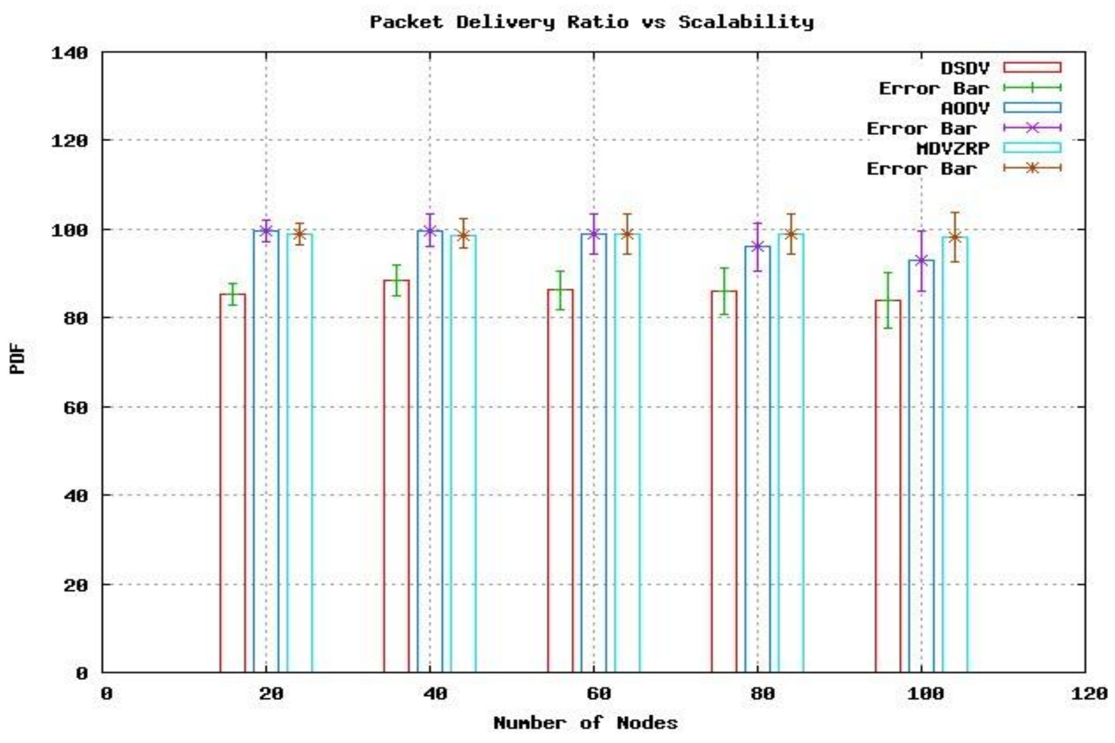


Figure 6.13: PDF 95% confidence interval and error bars

6.8.2 The 1st Set of Experiments

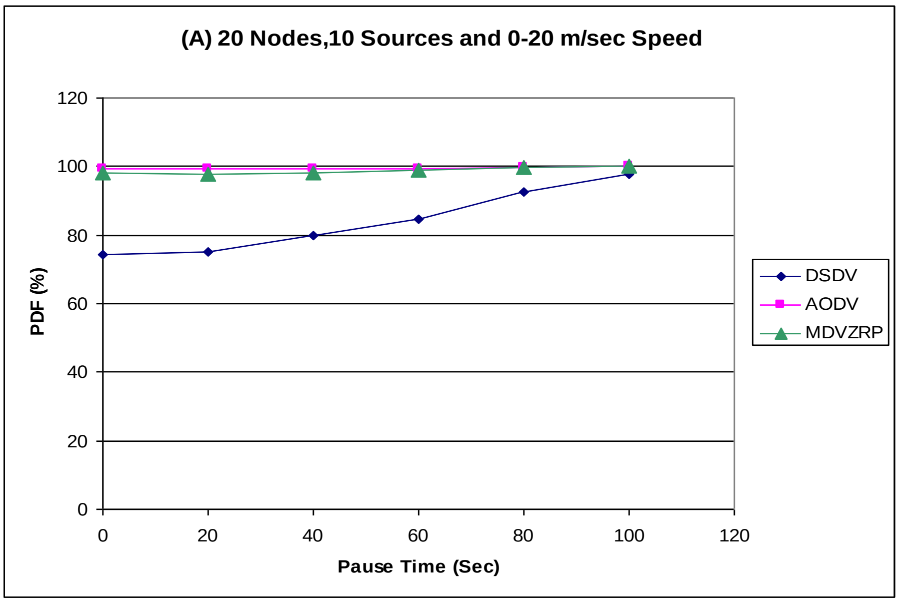


Figure 6.14: PDF for the 20 nodes model with 10 sources

The Packet Delivery Fraction Figures (6.14-16), for both AODV and MDVZRP are very similar with 10, 15 and 20 sources in the first set of 20 nodes network, while DSDV has a lower performance at lower pause times (high mobility scenarios).

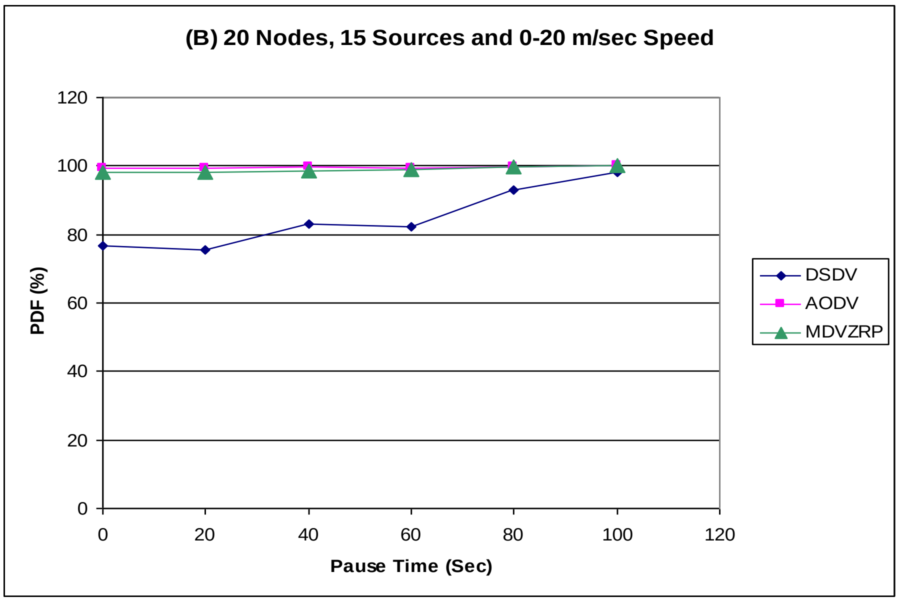


Figure 6.15: PDF for the 20 nodes model with 15 sources

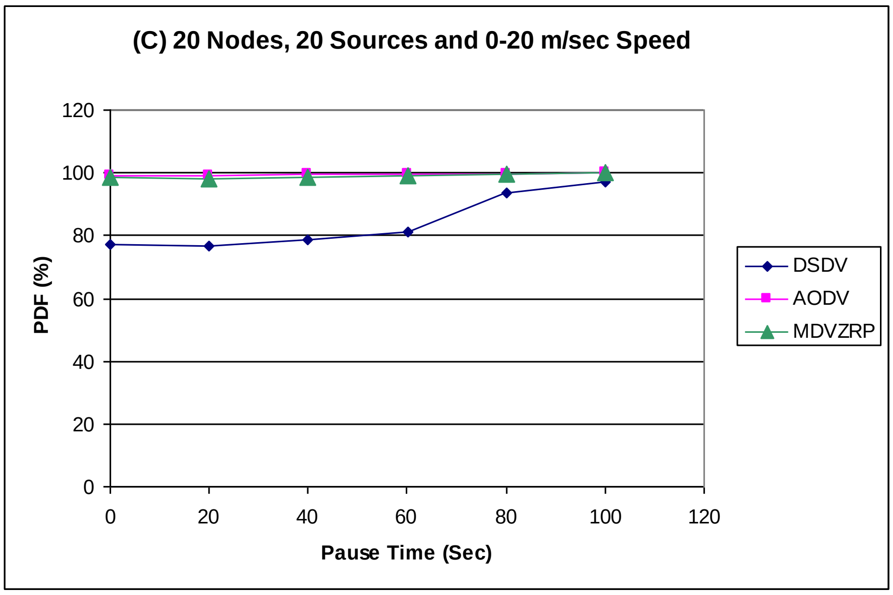


Figure 6.16: PDF for the 20 nodes model with 20 sources

Both AODV and MDVZRP are also show high similarity in the Throughput metric with 10, 15 and 20 sources in the first set of 20 nodes network as well, while DSDV shows a lower throughput performance at high mobility scenarios as shown in Figures (6.17-19).

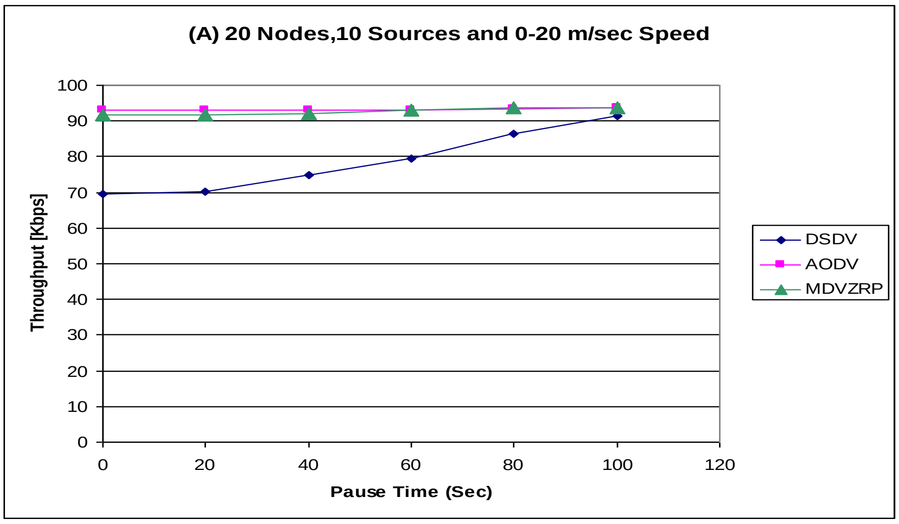


Figure 6.17: Throughput for the 20 nodes model with 10 sources

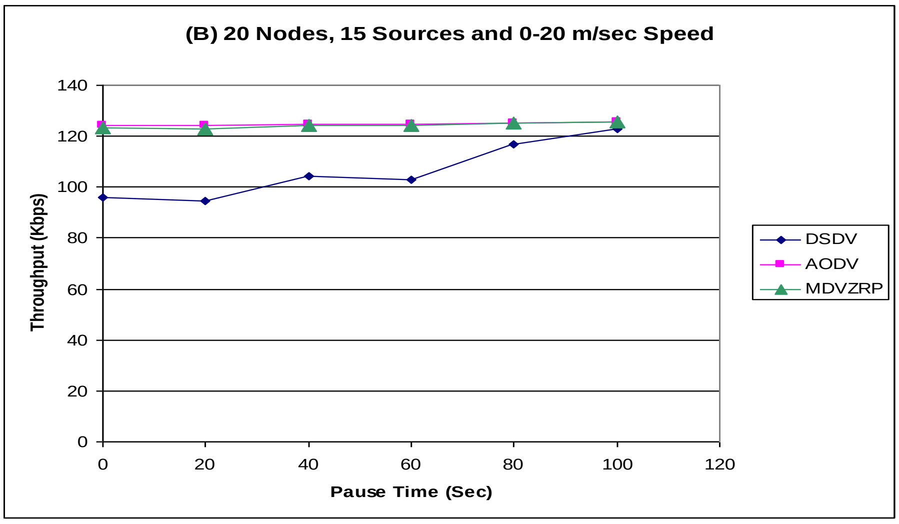


Figure 6.18: Throughput for the 20 nodes model with 15 sources

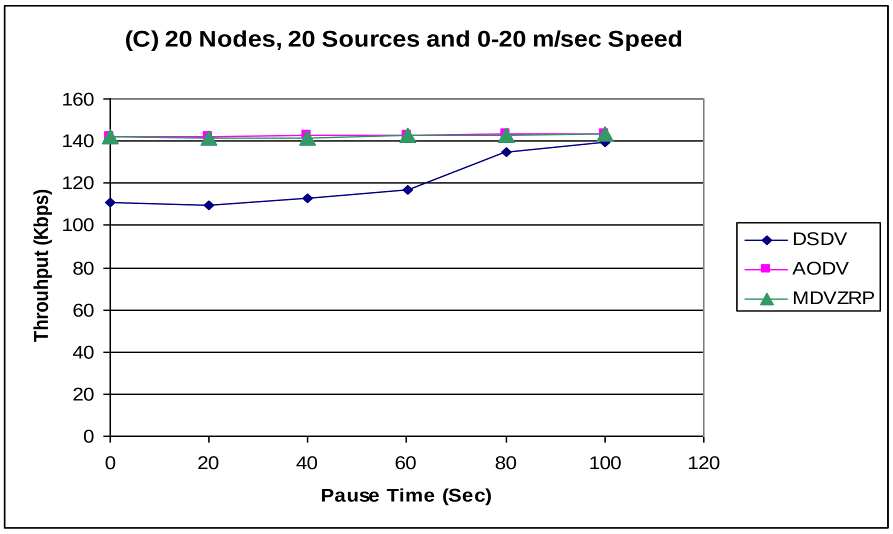


Figure 6.19: Throughput for the 20 nodes model with 20 sources

MDVZRP shows a best NRL in all cases, while AODV show the highest NRL at the high mobility times, decreasing gradually as the network tends to be a stationary as shown in Figures (6.20-22), because both AODV and DSDV compared to MDVZRP generate more packets per data packet, which made both AODV and DSDV produce a higher overhead than MDVZRP in all scenarios as shown in Figures (6.23-25).

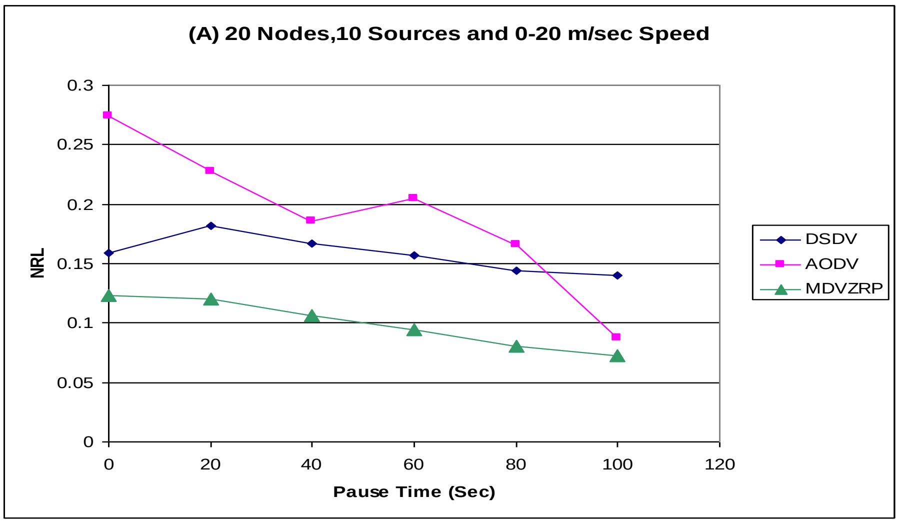


Figure 6.20: NRL for the 20 nodes model with 10 sources

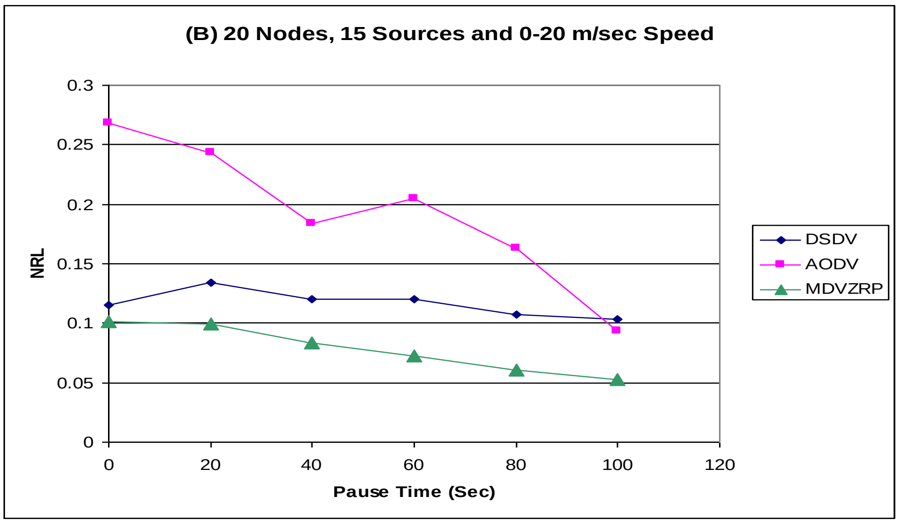


Figure 6.21: NRL for the 20 nodes model with 15 sources

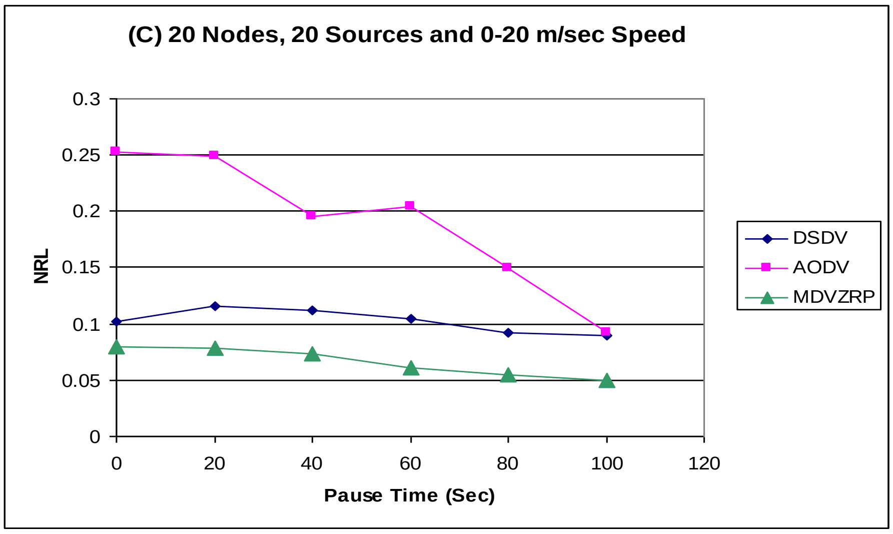


Figure 6.22: NRL for the 20 nodes model with 20 sources

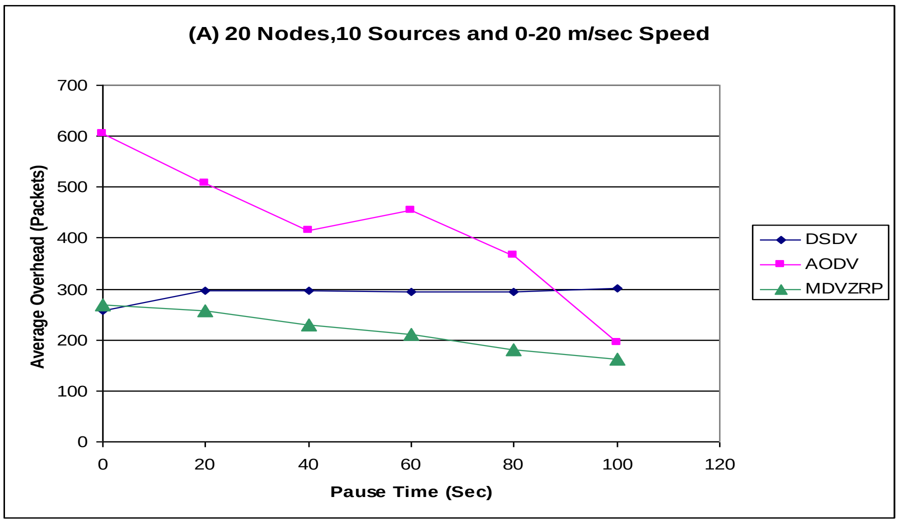


Figure 6.23: Average Overhead for the 20 nodes model with 10 sources

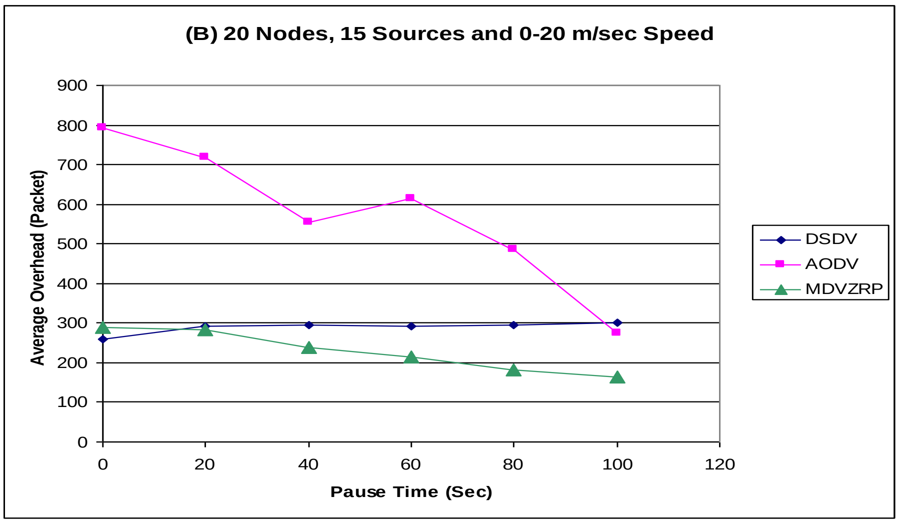


Figure 6.24: Average Overhead for the 20 nodes model with 15 sources

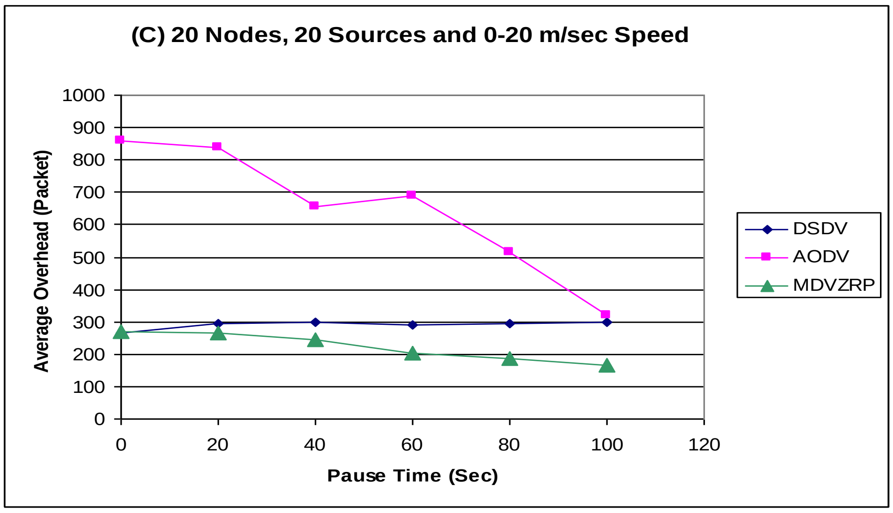


Figure 6.25: Average Overhead for the 20 nodes model with 20 sources

The relative performance of the three protocols with respect to EED is similar as the network size increases and tends to have low mobility where it can be seen clearly in Figure (6.28). MDVZRP shows a better EED than AODV at low and medium mobility as shown in Figures (6.26-28).

MDVZRP shows a higher EED than AODV and DSDV, that refers to data queued for longer in MDVZRP because of RREQ and RREP mechanism, which takes longer in MDVZRP than AODV, because each intermediate node checks the whole of its routing table to see if there is an available route to the required destination during the RREQ receive mechanism, and some of the intermediate nodes do the same thing again to obtain routing information during the RREP receive mechanism.

Increasing the speed with a short pause time increases the possibility of link breakage and hence that increases the possibility of using routes on demand where requests from MDVZRP to call RREQ/RREP many times, which negatively effects in EED as shown in Figures (6.26-28), where we can see clearly at pause time 20 sec.

Regarding average dropped packets performance, the MDVZRP shows a better performance compared to DSDV, and an outstanding performance compared to AODV in all scenarios of the 1st set of experiments (20 nodes) as shown in Figures (6.29-31).

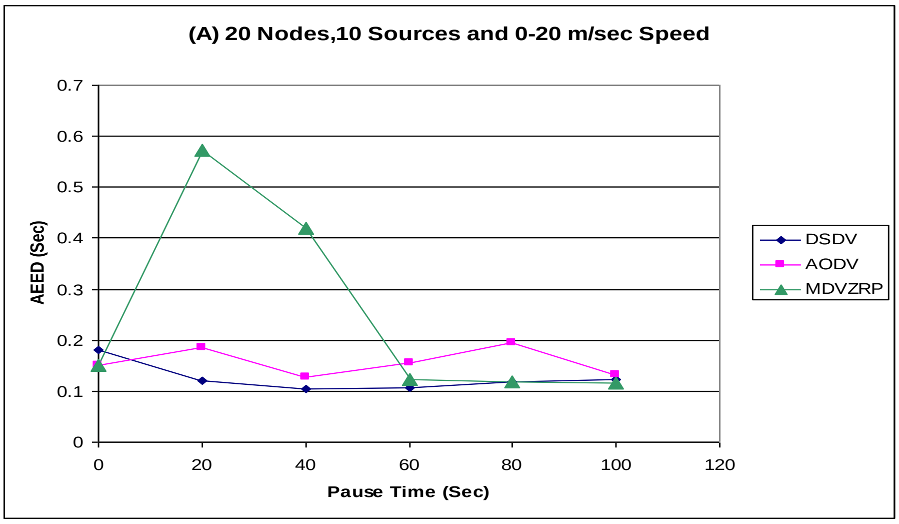


Figure 6.26: EED for the 20 nodes model with 10 sources

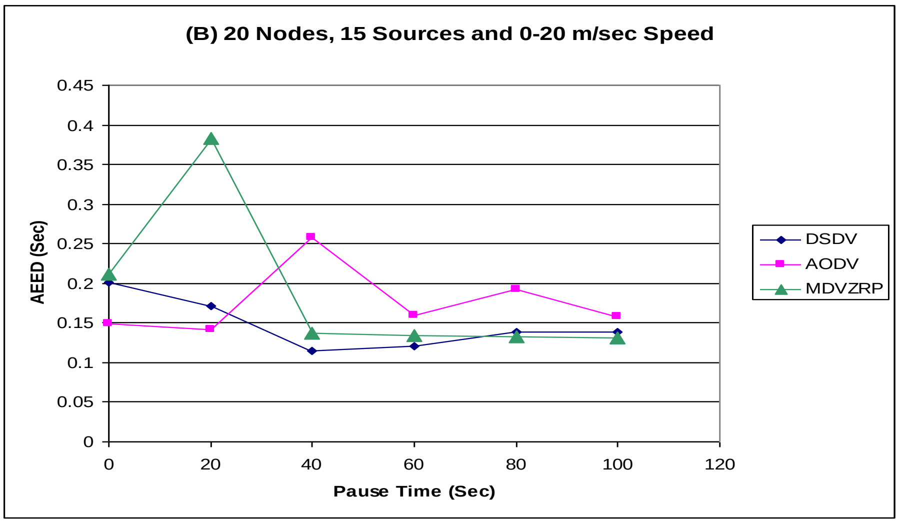


Figure 6.27: EED for the 20 nodes model with 15 sources

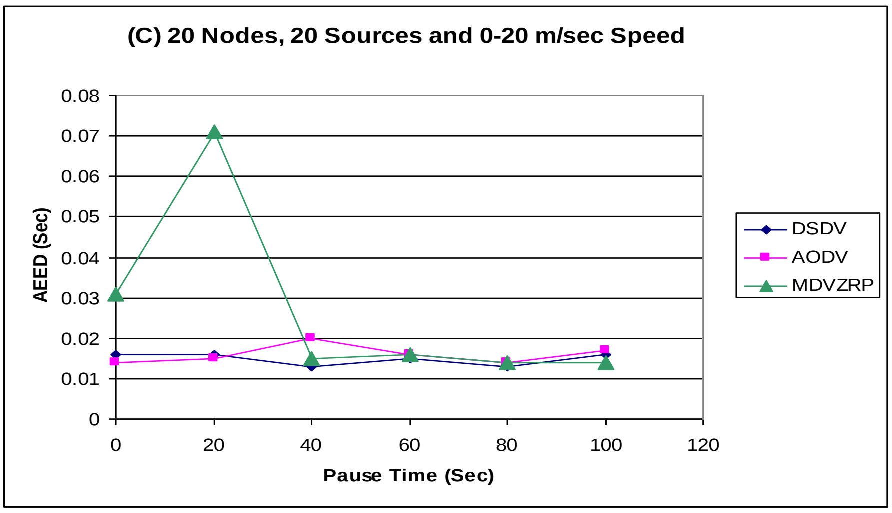


Figure 6.28: EED for the 20 nodes model with 20 sources

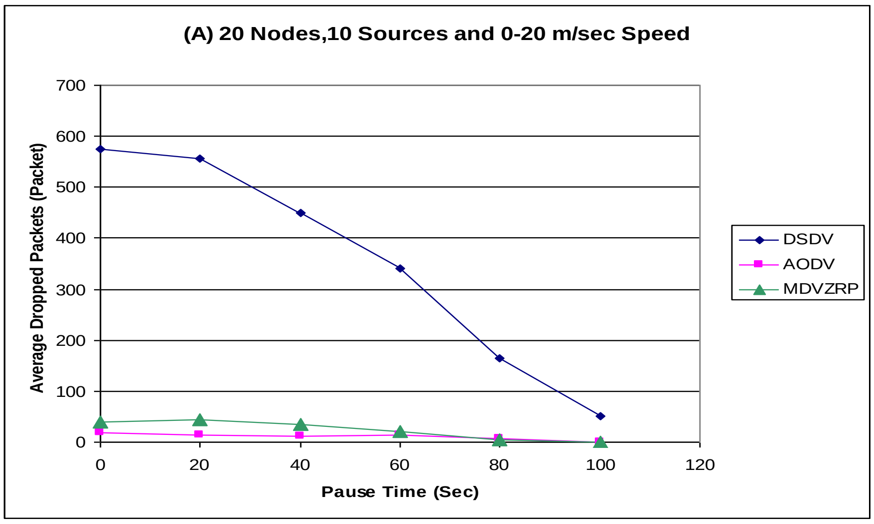


Figure 6.29: Average Dropped Packets for the 20 nodes model with 10 sources

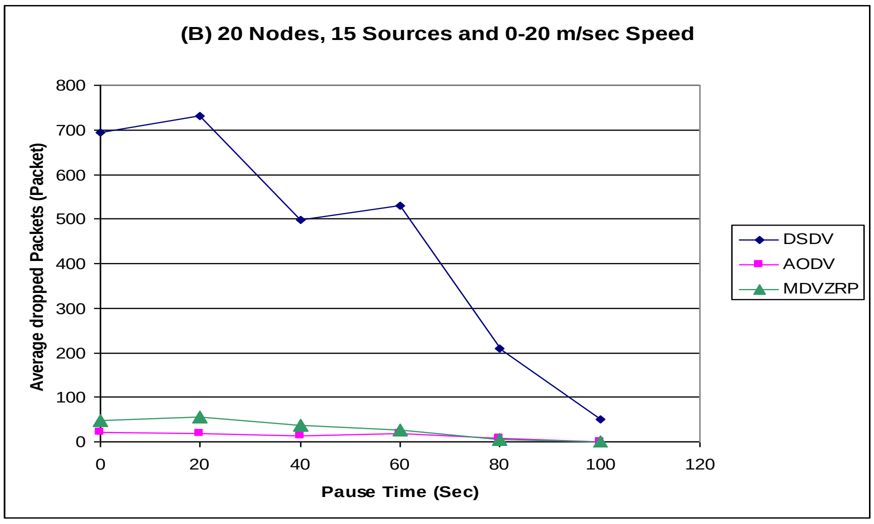


Figure 6.30: Average Dropped Packets for the 20 nodes model with 15 sources

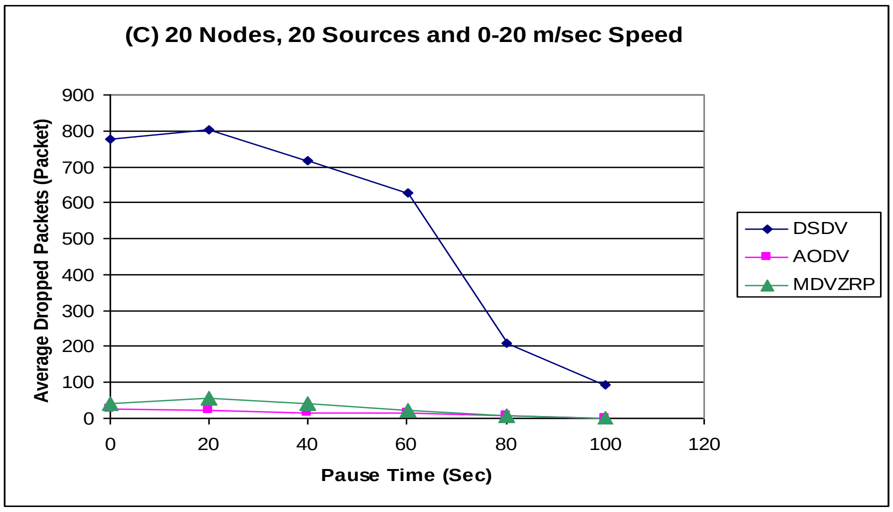


Figure 6.31: Average Dropped Packets for the 20 nodes model with 20 sources

6.8.3 The 2nd Set of Experiments

Networks of 60 nodes with 20, 30 and 40 sources and fixed data rate 4 packets / sec are used for the three protocols' performance evaluation in this set of experiments. Figure (6.32) shows that MDVZRP has similar PDF to AODV for 20 sources, while AODV shows a slightly better performance as the mobility increased with 30 and 40 sources as shown in Figures (6.33, 34).

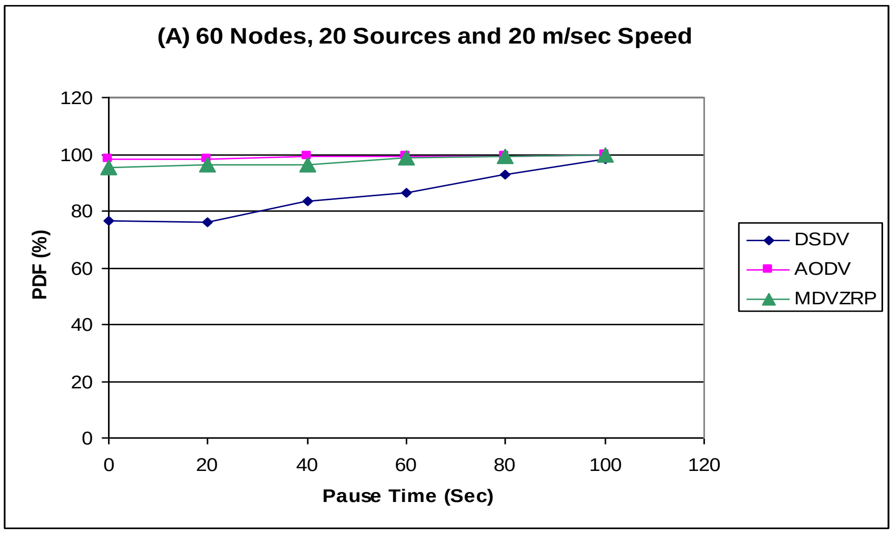


Figure 6.32: PDF for the 60 nodes with 20 sources

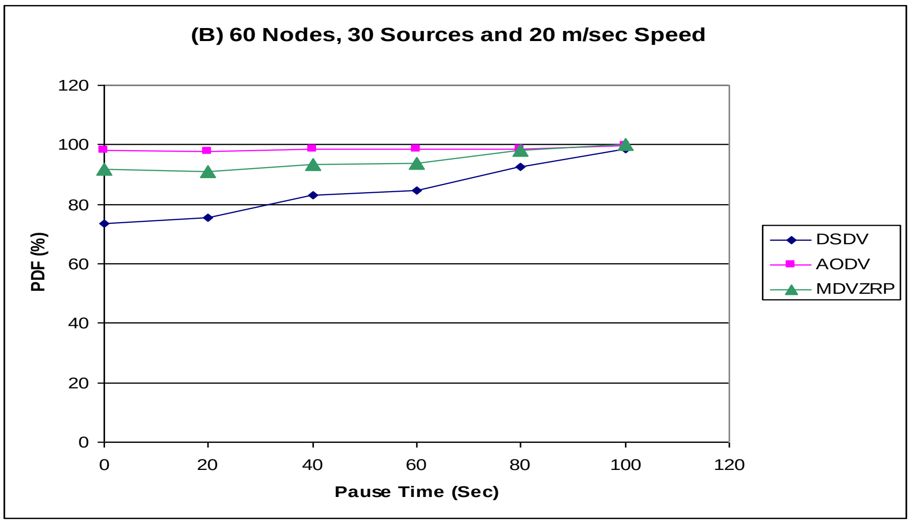


Figure 6.33: PDF for the 60 nodes with 30 sources

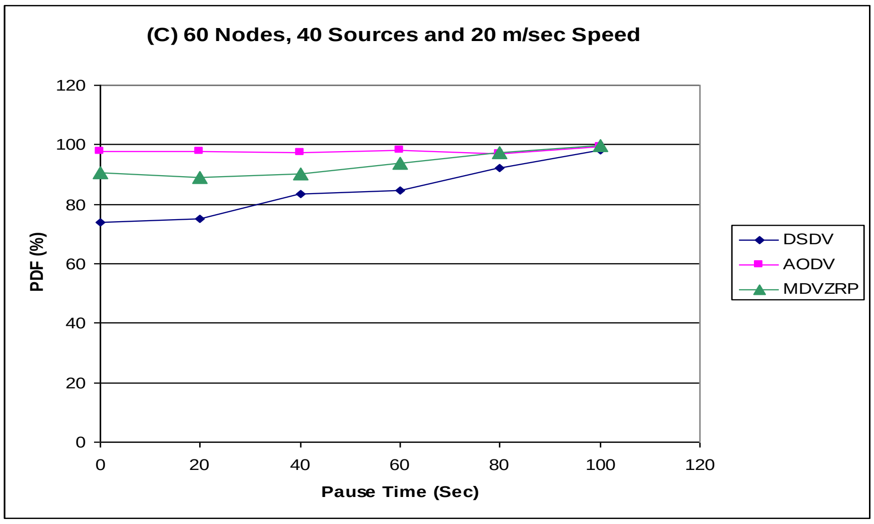


Figure 6.34: PDF for the 60 nodes with 40 sources

Because fewer control packets broadcast for each data packet, MDVZRP still achieves better NRL and overhead in all scenarios, especially in high mobility network scenarios. The difference in both NRL Figures (6.35-37) and overhead control packets (6.41-43) are 5 times more in AODV for 60 nodes than in the 20 nodes network.

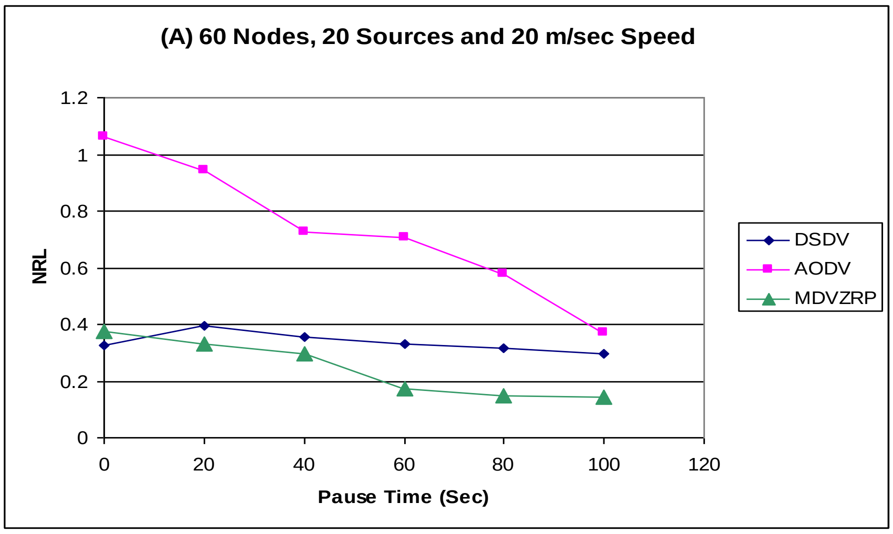


Figure 6.35: NRL for the 60 nodes with 20 sources

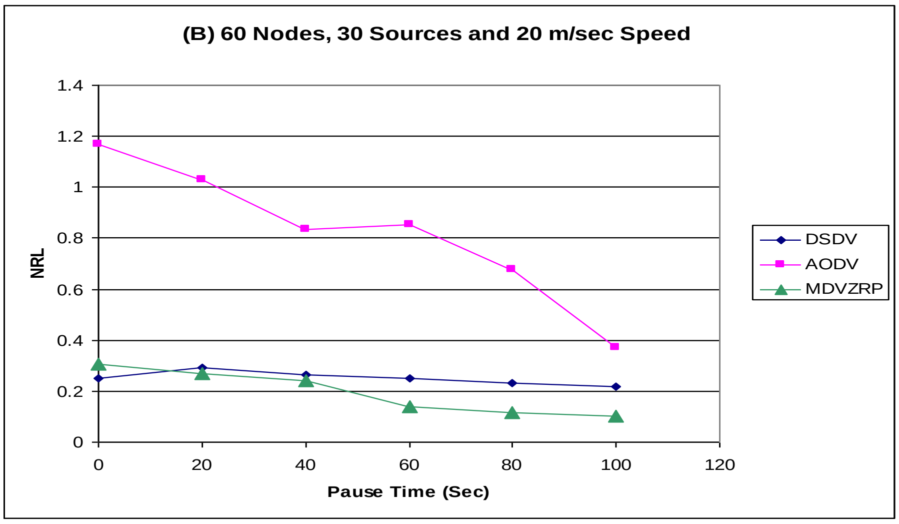


Figure 6.36: NRL for the 60 nodes with 30 sources

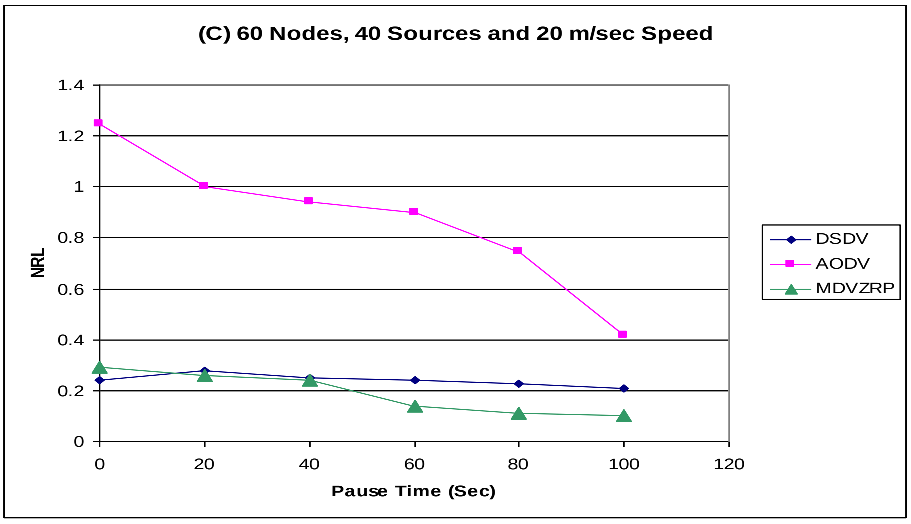


Figure 6.37: NRL for the 60 nodes with 40 sources

However, MDVZRP’s delay is still higher than both AODV and DSDV at high mobility, but shows a better delay performance at low mobility than the other two protocols and reasonable EED at medium mobility as shown in the EED Figures (6.38-40).

Compared to DSDV, the average dropped packets performance of MDVZRP in the 2nd set of experiments shows a better performance as shown in Figures (6.44-46), but the same figures show that MDVZRP has a lower performance compared to AODV at high mobility scenarios and the best performance at low mobility.

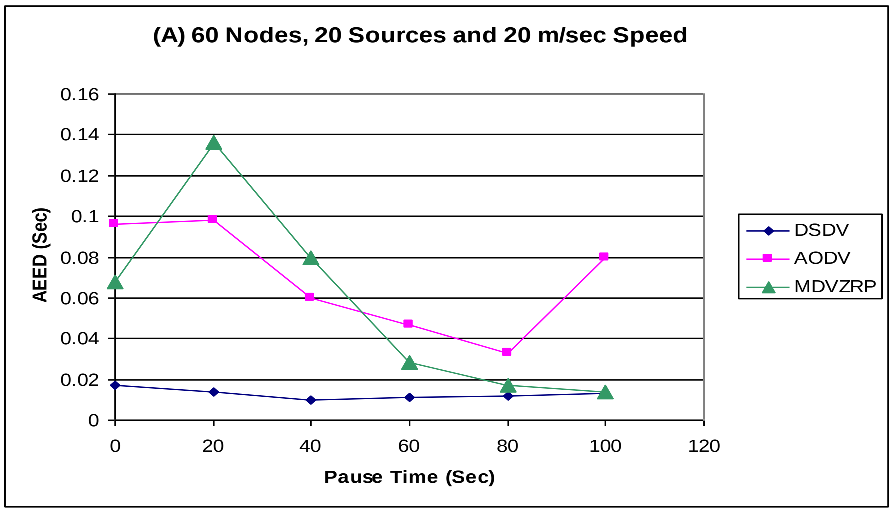


Figure 6.38: EED for the 60 nodes model with 20 sources

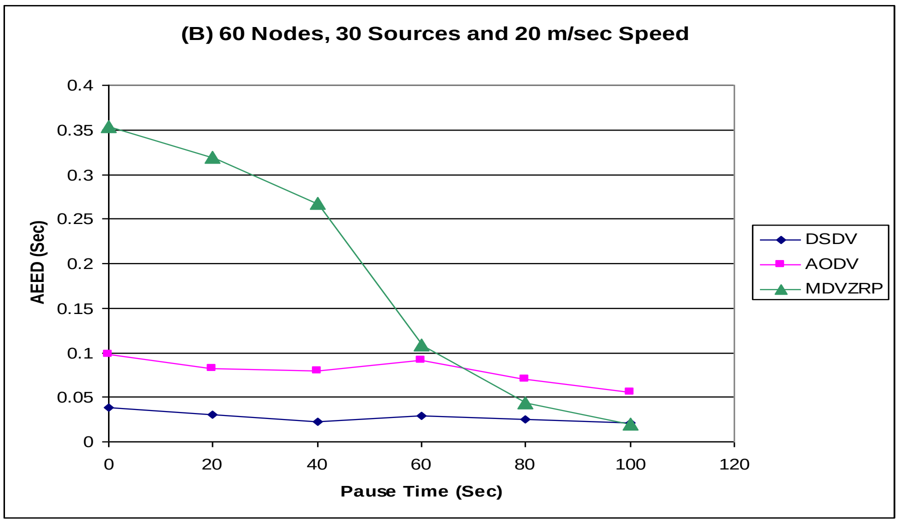


Figure 6.39: EED for the 60 nodes model with 30 sources

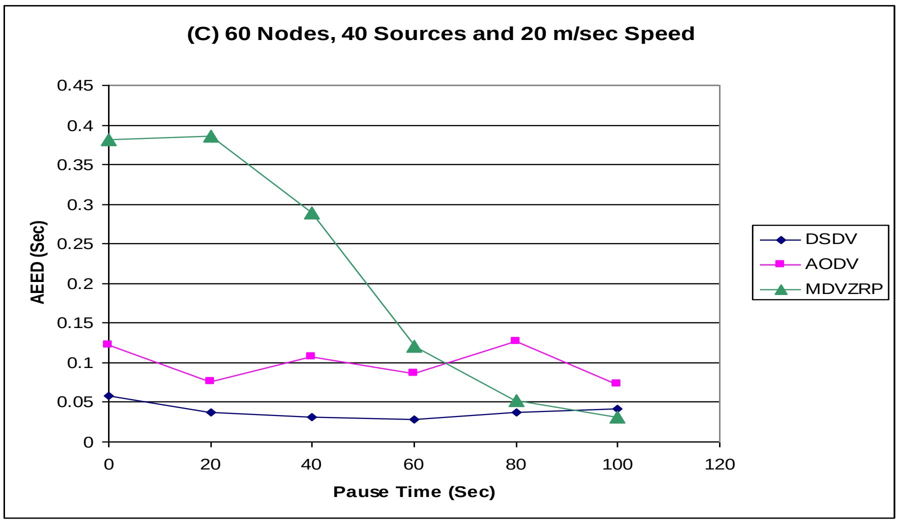


Figure 6.40: EED for the 60 nodes model with 40 sources

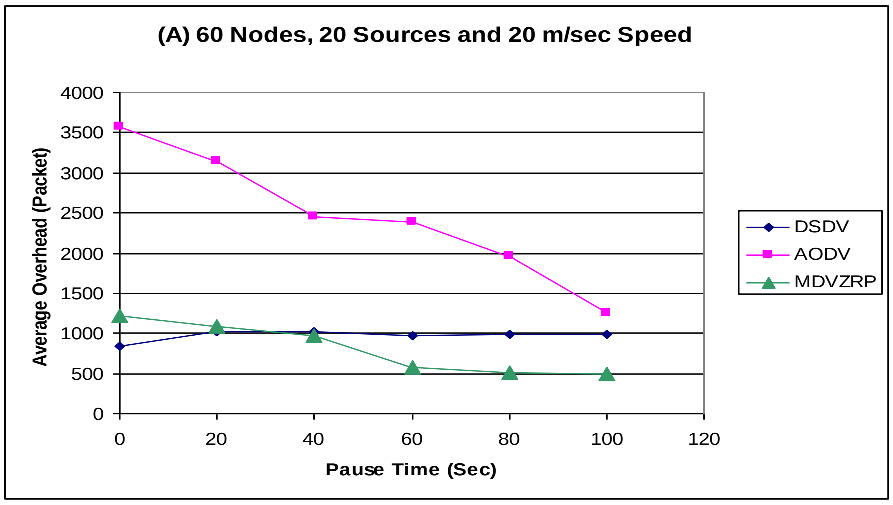


Figure 6.41: Average Overhead for the 60 nodes model with 20 sources

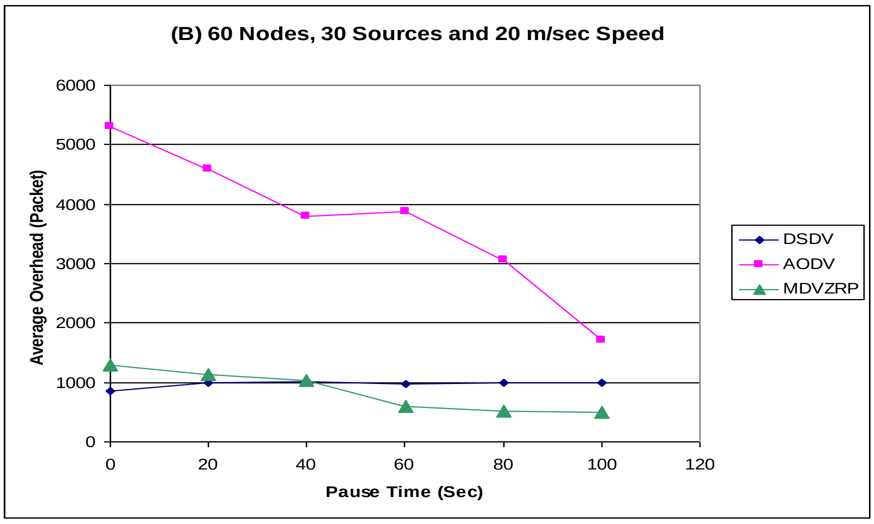


Figure 6.42: Average Overhead for the 60 nodes model with 30 sources

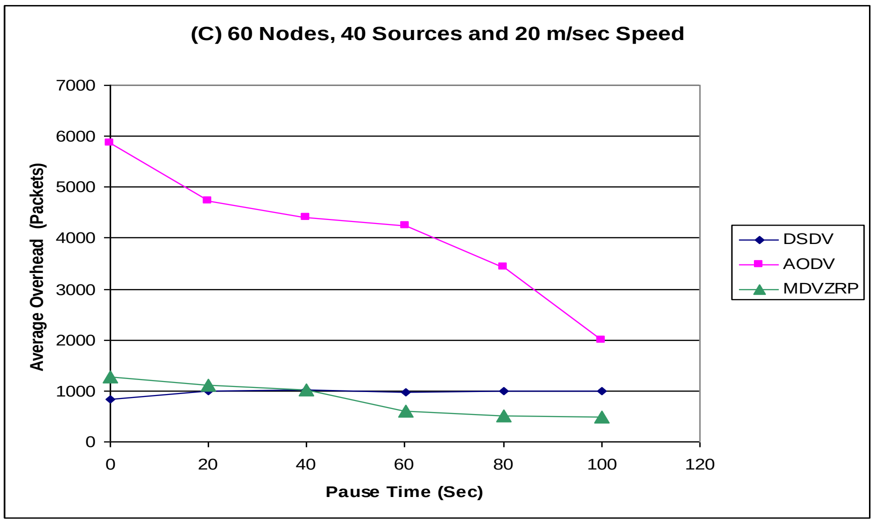


Figure 6.43: Average Overhead for the 60 nodes model with 40 sources

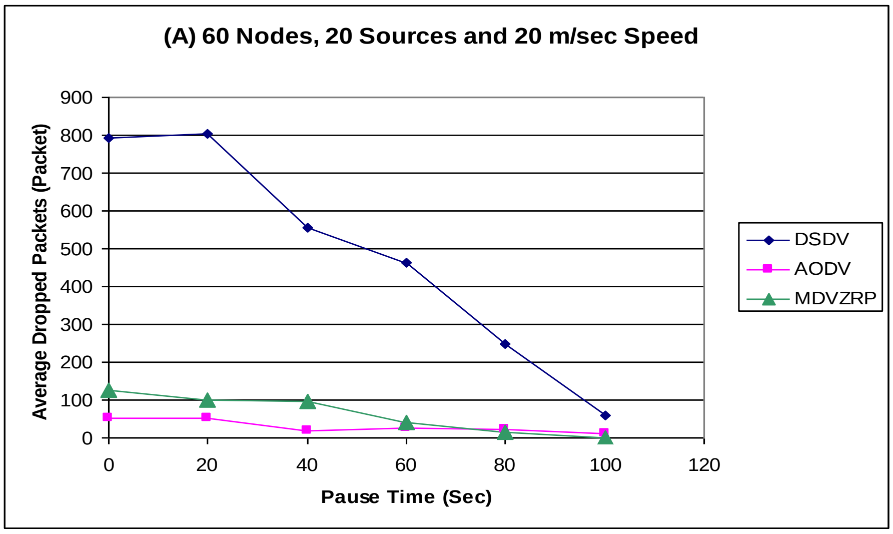


Figure 6.44: Average Dropped Packets for the 60 nodes model with 20 sources

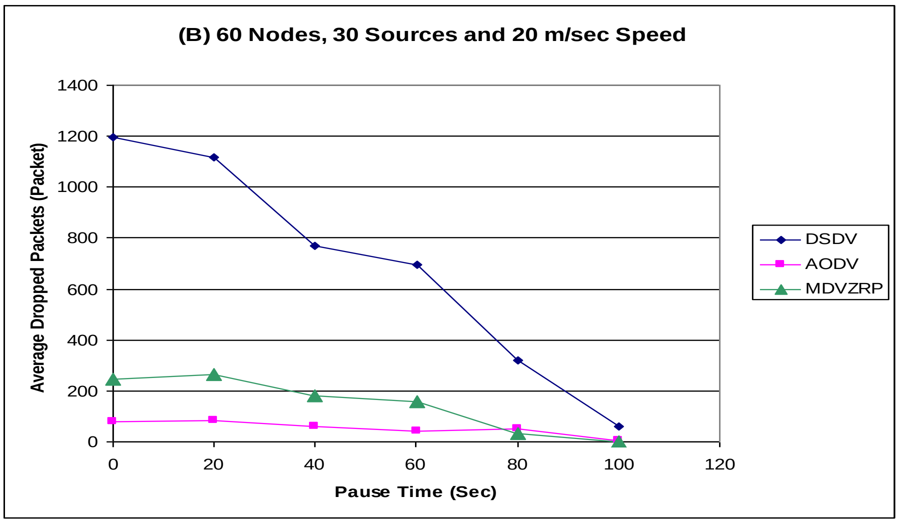


Figure 6.45: Average Dropped Packets for the 60 nodes model with 30 sources

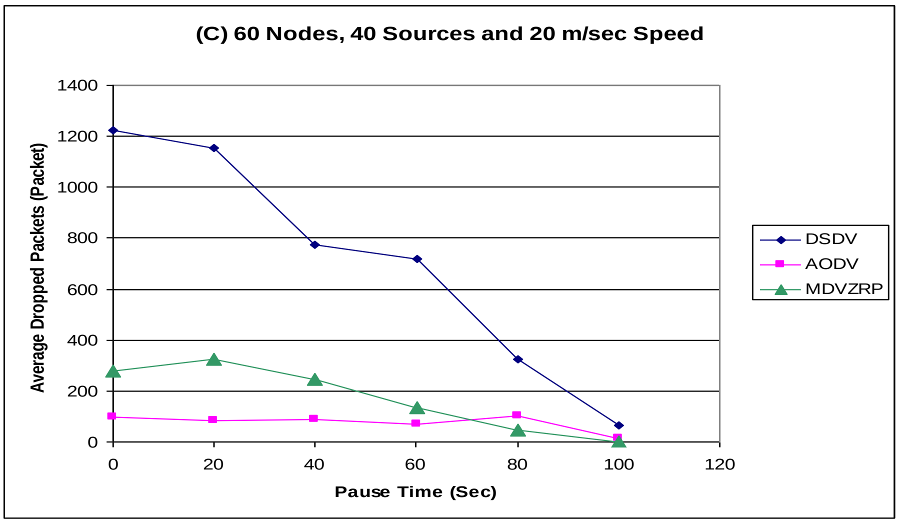


Figure 6.46: Average Dropped Packets for the 60 nodes model with 40 sources

6.8.4 The 3rd Set of Experiments

For the 3rd set of experiments, we have used 100 nodes with 20, 40 and 50 sources and 4 packets / sec data transfer rate as well. Figures (6.47-64) show that, the three protocols have same performance results as the 2nd set of experiments with some slightly differences. Generally, MDVZRP shows best PDF and Throughput as the scenarios tend to low mobility comparing to both DSDV and AODV as shown in Figures (6.47-49) and Figures (6.56-58) respectively.

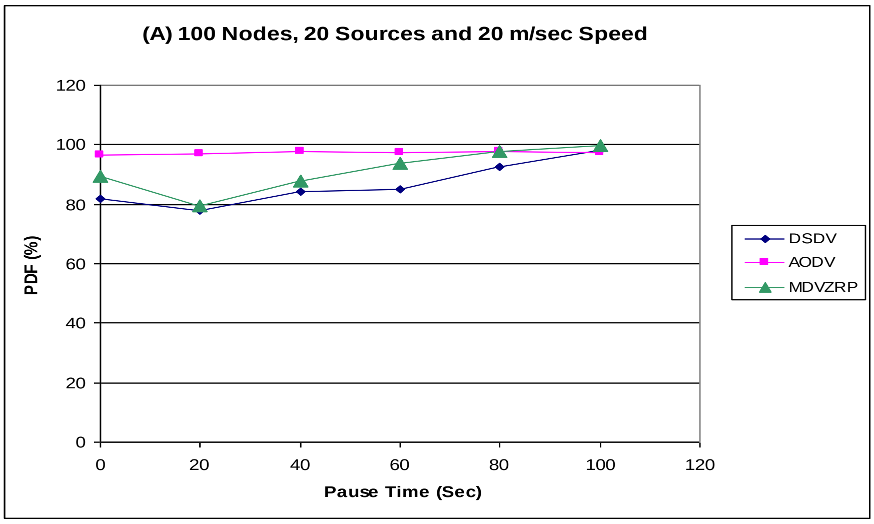


Figure 6.47: PDF for the 100 nodes with 20 sources

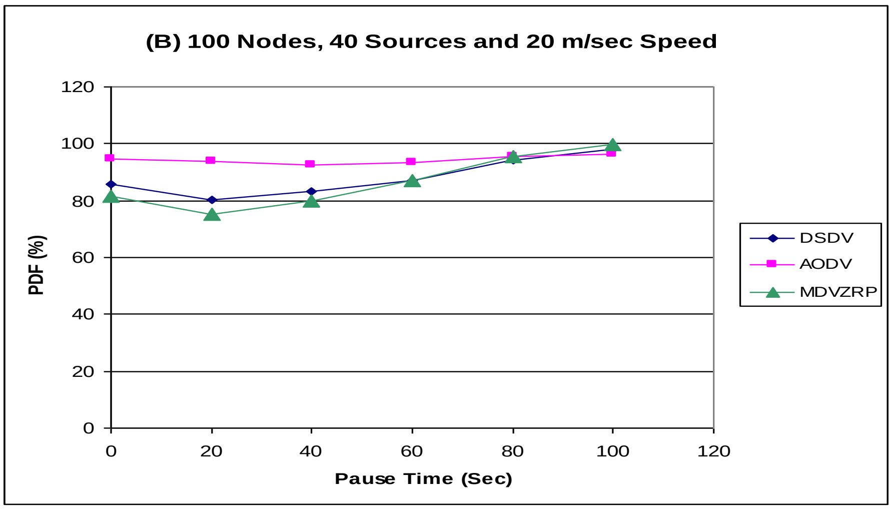


Figure 6.48: PDF for the 100 nodes with 40 sources

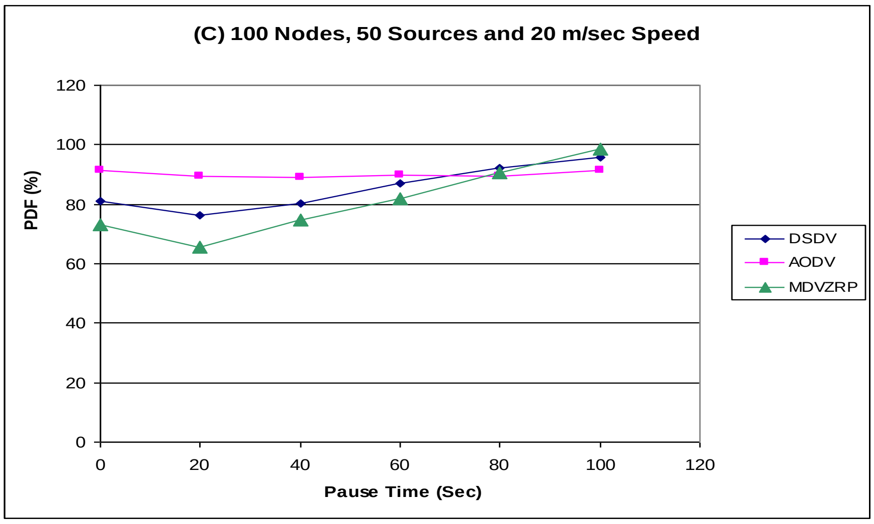


Figure 6.49: PDF for the 100 nodes with 50 sources

DSDV shows a slightly better performance than MDVZRP in high mobility for 40 and 50 sources only for first time, while AODV has its best performance at high mobility scenarios. However, the difference in average NRL and Overhead loading in AODV comparing to MDVZRP is tremendous as both the network size and mobility are increased as shown in Figures (6.50-52) and Figures (6.59-61) respectively.

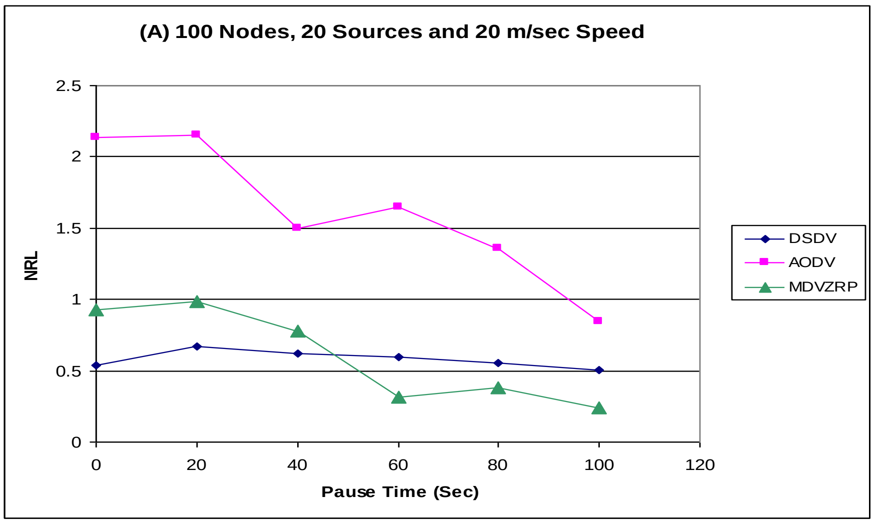


Figure 6.50: NRL for the 100 nodes with 20 sources

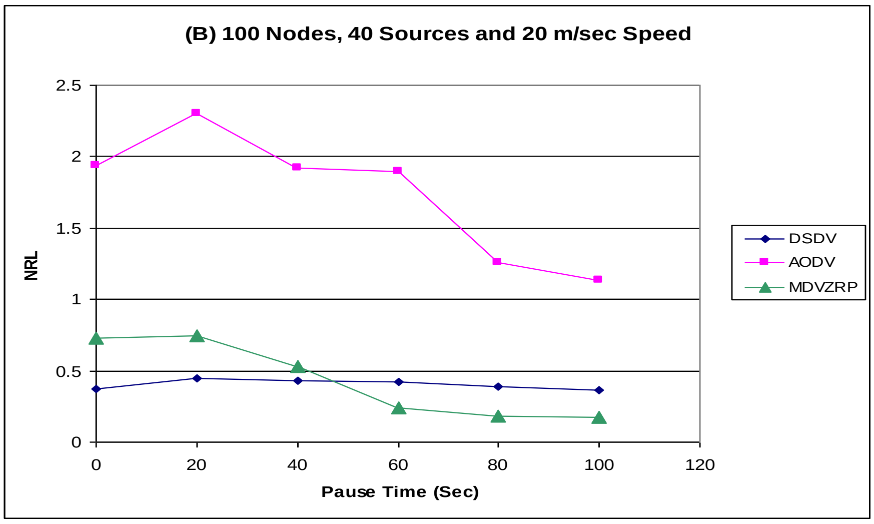


Figure 6.51: NRL for the 100 nodes with 40 sources

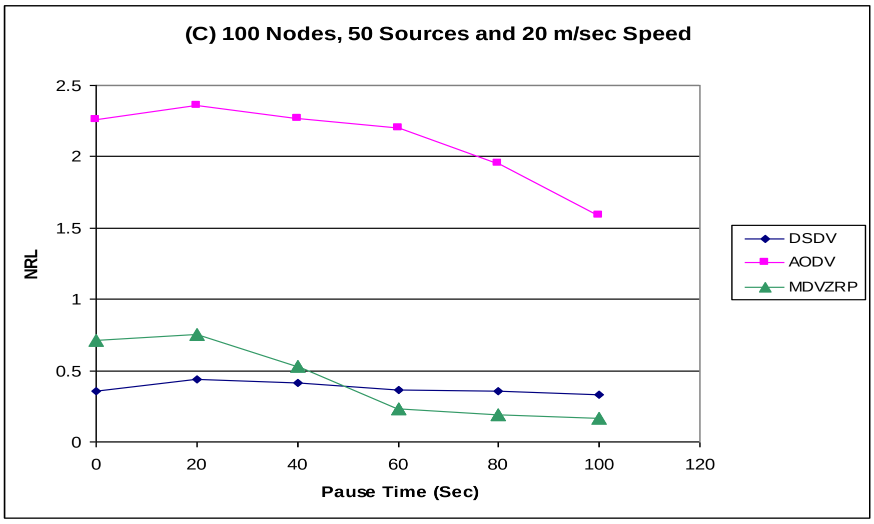


Figure 6.52: NRL for the 100 nodes with 50 sources

Figures (6.53-55) show that, the three protocols have the same performance results as the 2nd set of experiments respect to EED, where MDVZRP's delay is still higher than both AODV and DSDV at high mobility, but shows a better delay performance at low mobility than the other two protocols and reasonable EED at medium mobility as the case in the 2nd set of experiments.

Comparing to DSDV, average dropped packets performance of MDVZRP in the 3rd set of experiments shows a better performance as shown in Figures (6.62-64), but the same figures show that MDVZRP has less performance compared to AODV at high mobility scenarios and the best performance at low mobility as the case in 2nd set of experiments.

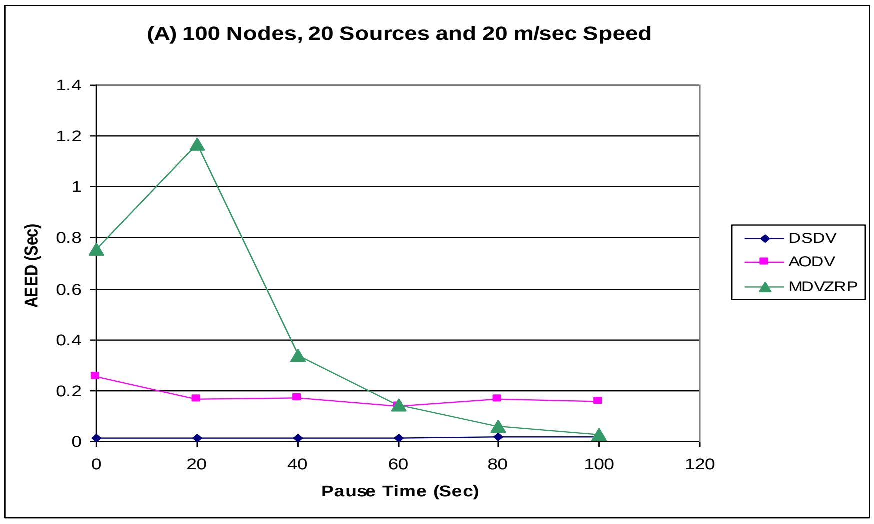


Figure 6.53: EED for the 100 nodes with 20 sources

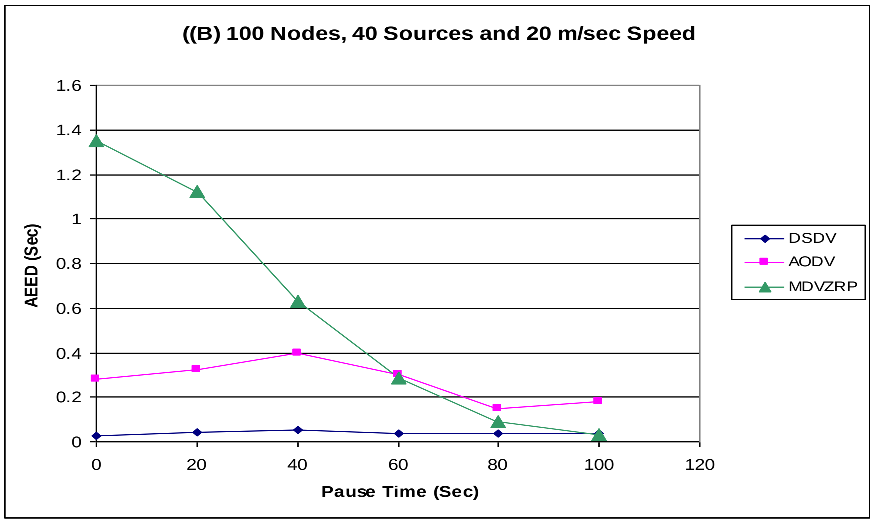


Figure 6.54: EED for the 100 nodes with 40 sources

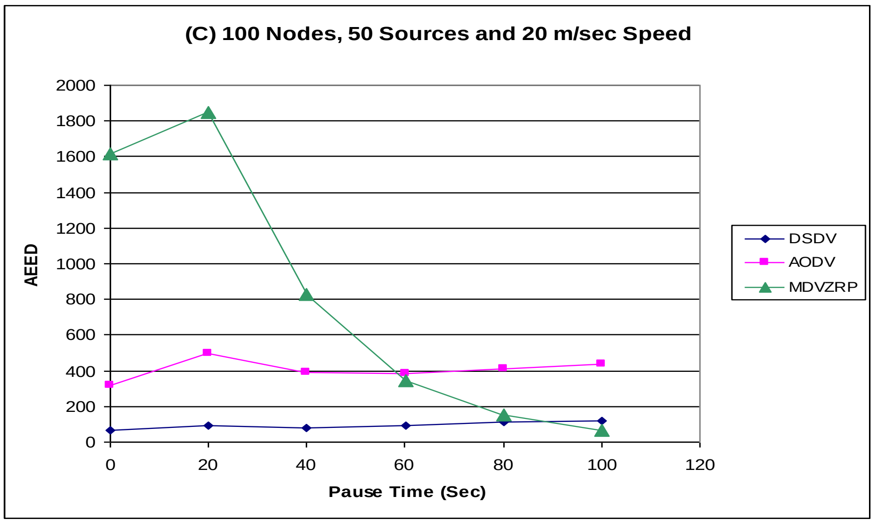


Figure 6.55: EED for the 100 nodes with 50 sources

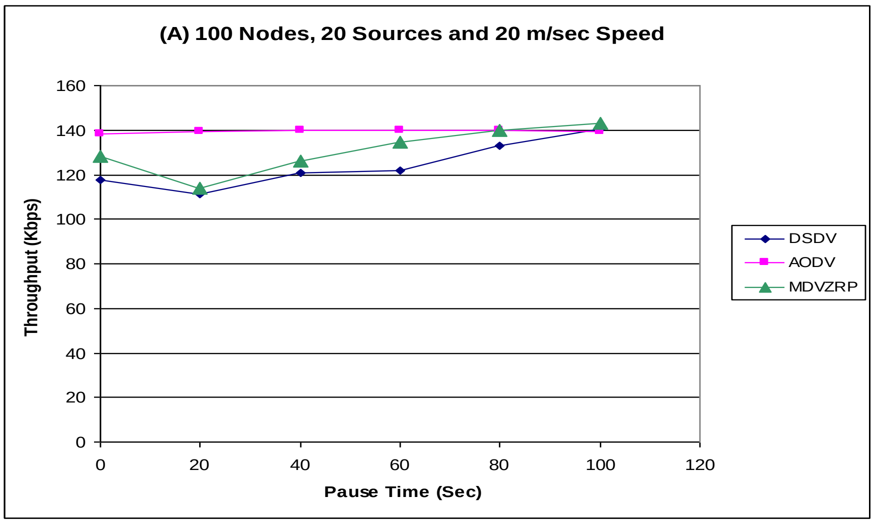


Figure 6.56: Throughput for the 100 nodes with 20 sources

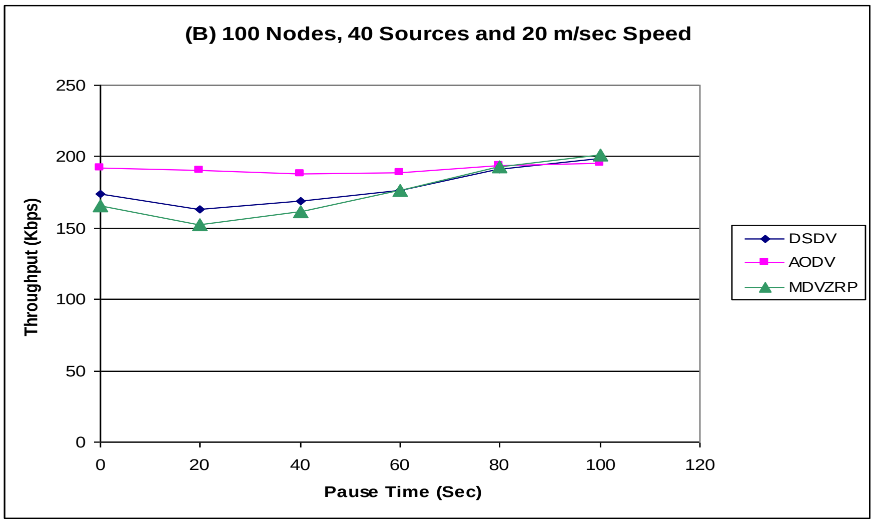


Figure 6.57: Throughput for the 100 nodes with 40 sources

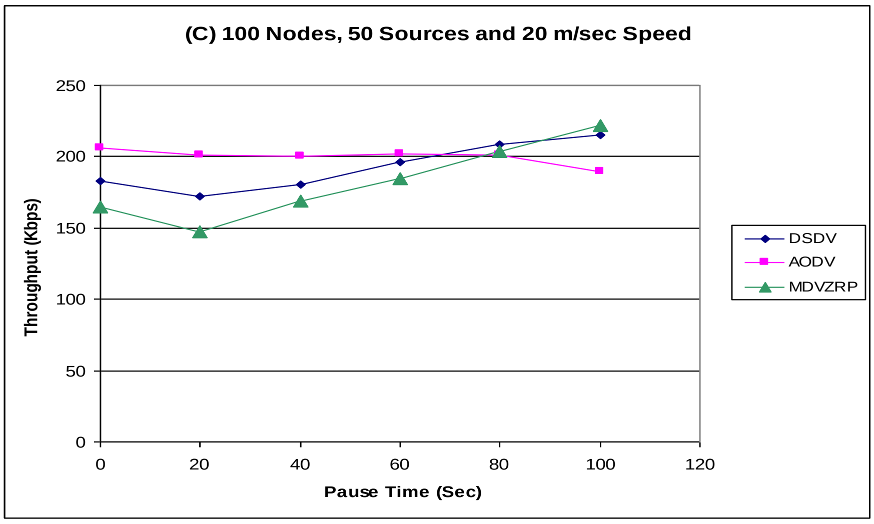


Figure 6.58: Throughput for the 100 nodes with 50 sources

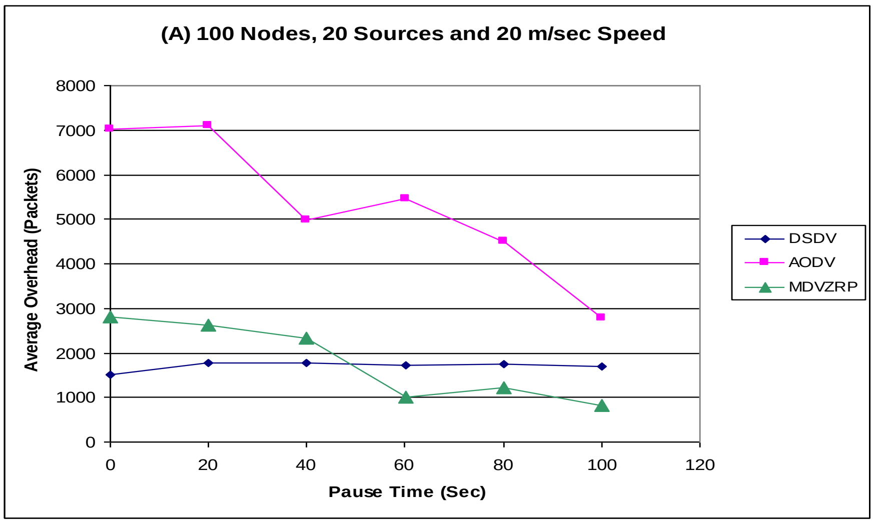


Figure 6.59: Average Overhead for the 100 nodes with 20 sources

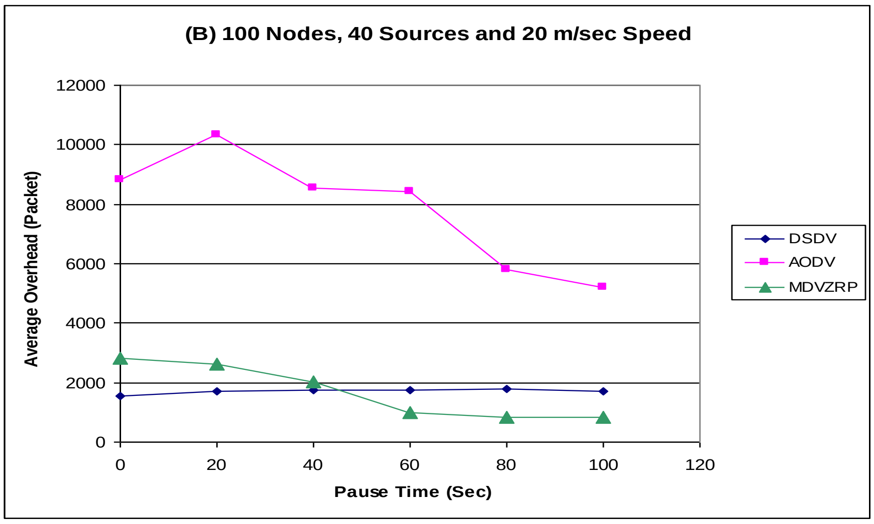


Figure 6.60: Average Overhead for the 100 nodes with 40 sources

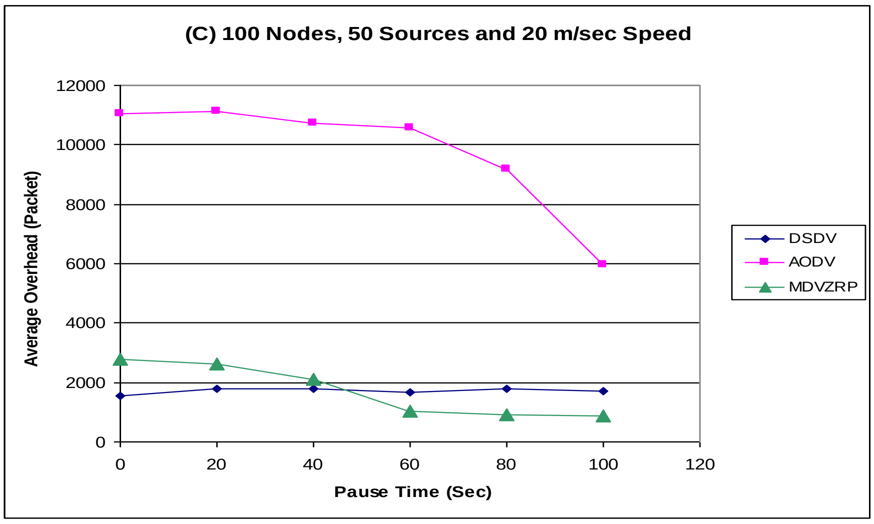


Figure 6.61: Average Overhead for the 100 nodes with 50 sources

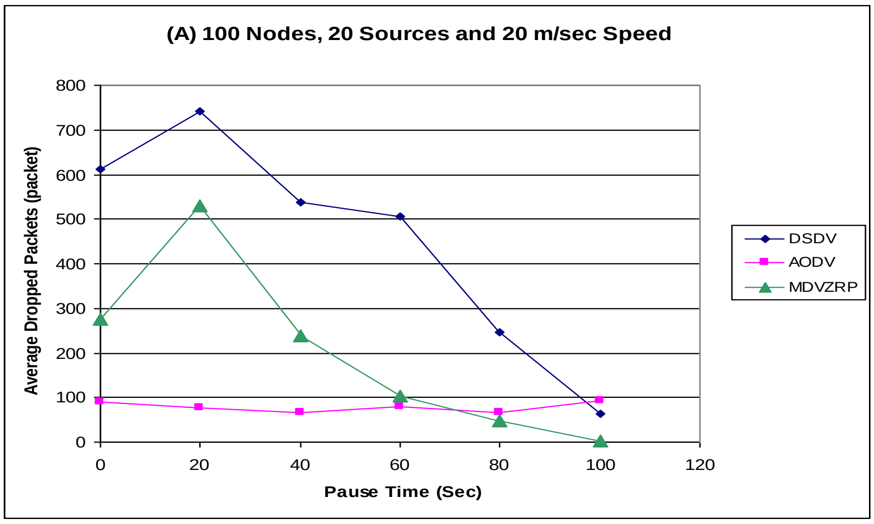


Figure 6.62: Average Dropped Packets for the 100 nodes with 20 sources

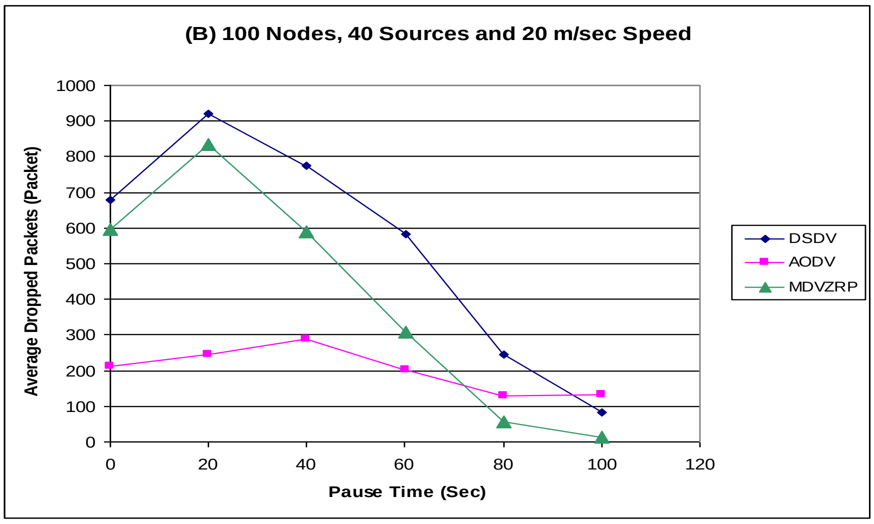


Figure 6.63: Average Dropped Packets for the 100 nodes with 40 sources

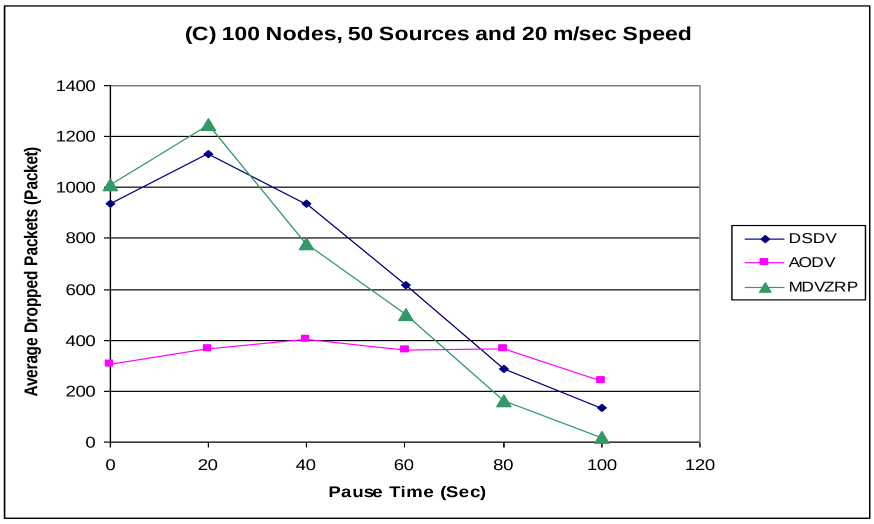


Figure 6.64: Average Dropped Packets for the 100 nodes with 50 sources

6.9 Summary

We have compared the performance of CBR in our hybrid dynamic routing protocol MDVZRP with two standard dynamic routing protocols DSDV and AODV. Both are single path routing protocols, where DSDV is a proactive protocol while AODV is a reactive on demand routing protocol. Both are using routing tables to save one route per destination and a destination sequence number to refresh the routing tables.

The three protocols deliver a large percentage of the offered data packets when there is little node mobility (i.e. at large pause time), converging to 100% delivery when there is no node motion. AODV and MDVZRP perform particularly well, delivering over 98% of the data packets regardless of mobility rate especially at small or medium network load (number of sources). Also, both protocols show the same throughput, while DSDV shows fewer throughputs as the pause time decreased.

The three protocols impose vastly different amount of overhead. Nearly an order of magnitude separates MDVZRP, which has the least overhead and normalized routing load from AODV which shows the most overhead irrespective of the node mobility or density. The basic character of each protocol is demonstrated in the shape of its overhead curve in all the graphs. AODV is a reactive routing protocol, while MDVZRP is a hybrid routing protocol. So, both protocols are share in on-demand route requests, therefore their overhead and normalized routing load drop as the mobility rate drops. DSDV is a largely periodic routing protocol; its overhead is nearly constant with respect to mobility rate. At low pause time (higher rates of mobility), DSDV does poorly, dropping to a packet delivery ratio of 70% or less as the network load increases. Nearly all of the dropped packets are lost because stale routes within the routing table entry direct them to be forwarded over broken links. Additionally, MDVZRP has highest end to end delay especially at high mobility rates.

The general observation in most scenarios is that MDVZRP has outperformed DSDV. When MDVZRP is compared to AODV it has outperformed it in low mobility situations despite the node density and network load. MDVZRP and AODV have almost the same performance for medium mobility networks.

Because Broch's results (Broch, 1998) demonstrate the performance of DSDV, TORA, DSR and AODV, and our results compare DSDV and AODV with MDVZRP, we can confirm that MDVZRP is effective in any size of network with low or medium mobility. Also, this makes MDVZRP a good choice for ad hoc networks because of the relationship between overhead and power consumption.

CHAPTER 7: CONCLUSION

7.1 Introduction

In this chapter, we summarise and present the outcome of the PhD thesis, using the aims of research (proposed goals) of section 3.2 as a reference point. The goals have been accomplished and achieved, except for that relating to end to end delay. Finally, those areas that require further investigation are proposed as future work.

7.2 Routing

The routing algorithm's main aim is to establish a route between a pair of mobile nodes correctly and efficiently, taking into account minimum overhead and bandwidth consumption.

In the conventional networks (wired), there are different distance vector and link state routing protocols, which were not designed to cope with a highly dynamic environment. Link-state protocols update their global routing information (global state) by broadcasting their local routing information to every other node, while distance-vector protocols exchange their local information with one hop neighbours only.

MANET has a changeable nature that leads to a dynamic topology in its links (paths) structure. As a consequence, routing is a complex and challenging issue, which is probably the most fundamental problem in MANETs. This is reflected in the large number of routing protocols (algorithms) for MANETs. In general, routing algorithms for MANETs may be divided into two broad classes: *proactive* and *reactive*, as we discussed in chapter 2.

Ideally, routing algorithm features for MANET have the same general features as the other routing algorithms plus taking into account the characteristics of a mobile environment in particular, limited bandwidth and energy and mobility.

7.3 Main Contribution

We presented a new routing protocol for MANET called MDVZRP *multi-path distance vector zone routing protocol* as an attempt to address the issues facing the two broad classes of MANET protocols (*proactive* and *reactive*). MDVZRP underwent many stages of design and improvement over a period of time into what it is today. Some of the improvements were in the code itself and others were in the protocol's algorithm technique.

MDVZRP is a hybrid routing protocol, where its control messages are either proactive or reactive. The proactive messages consist of beacon (Hello messages) and RSUP (Routes update packets), while the reactive ones are RREQ (Route request) and RREP (route reply). Packets in MDVZRP are forwarded using a routing table maintained by each node, with several routes to each destination.

A node in MDVZRP has a flat view over nearby network nodes, but not the entire network. When it joins the network it propagates a beacon packet / Hello message (Heart beat) for the first time. Unlike many other protocols, MDVZRP does not periodically broadcast control packets, only when there are significant changes in the network topology.

Beacons are the only messages that are sent periodically, when no other messages are being transmitted, so that a node informs others of its existence and maintains connectivity with its neighbours. When a change, such as a broken active route or a new node is detected, *RSUP*s are broadcast through the network by the new node's neighbours or the node that detected the broken route (Error) to inform other nodes.

The RSUP (Routes Update Packet) is a non periodical rebroadcasting message. This type of message is essentially to keep the routing tables of all the nodes up to date. This RSUP packet can serve as a *beacon* when broadcast as we mentioned previously, and is also used instead of the *ERROR* message as a part of the protocol early version development. Therefore, a node that broadcasts such packets would not broadcast beacons at that period of time (this period is known as the beacon Interval) to reduce the number of broadcast packets and hence reduce the overhead in general. Transmission of an update packet represents the proactive part mechanism in MDVZRP protocol.

As we mentioned, MDVZRP is a combination from the two extremes of proactive and reactive protocols that aims to provide the best of both techniques. There are a number of other hybrid protocols already existing and each of them has its own techniques. However, the hybrid approach used in MDVZRP is significantly different to the route request and zones approaches employed in other hybrid protocols. We assume in our routing protocol, that all the entries are operational (fresh not stale) in the routing table and there is no need to update them unless a node receives an error message regarding a specific route. In that case the node assigns that route as a broken link.

On demand requests can be more efficiently performed without querying nodes of the entire network, where all nodes proactively store local routing information. The reactive component of MDVZRP in the event that a route to a known node is unavailable, and a transmission needs to take place immediately, is responsible for initiating the route discovery process. This is through broadcasting a RREQ (Route Request) and unicasting a RREP (Route Reply) message respectively.

When an error message regarding a failed link or non reachable node is received, MDVZRP utilises an alternative path finding mechanism to obtain a suitable alternative one or Best metric among the multipath (backup) that is stored into the node's routing table, instead of issuing a route request every time or wasting time on route repairs.

7.4 Results and Discussion

MDVZRP is implemented and simulated in Ns2, and its performance is compared with AODV and DSDV in different network scenarios. The results of all tested scenarios indicate that MDVZRP in ns-2 works as expected.

Basically, MDVZRP always outperforms the packet delivery performance of the DSDV and often matches or exceeds AODV in low and medium mobility scenarios, making it a good choice for any network size with low or medium mobility. Although, the protocol shows a good performance in low and medium mobility with CBR traffic, it did not achieve the same performance especially, for end to end delay (EED) metric as well as expected for large or even medium networks at high mobility, this may due to these factors:

Firstly, the routing table size of MDVZRP is larger than that for AODV because of keeping all the active routes to each destination due to the multipath mechanism. Increasing speed with a short pause time increases the possibility of link breakage. *Secondly*, it appears that the caching of routing information from route request (RREQ) and route reply (RREP) mechanisms delays the RREP response, or RREQ rebroadcast in case of no route available to the required destination. , and the *last* reason is the lack of any mechanism that determines routes freshness when multipath choices are available.

Overall, we can confirm that our new routing protocol at this stage, guarantees the best performance in small and medium networks at low mobility. We also guarantee a reasonable performance in large networks and high mobility scenarios when we overcome the three issues we mentioned in the previous section.

7.5 Future Work

The investigations are continuing in to MDVZRP performance, especially in high mobility networks. Possible performance enhancements may be investigated by introducing better route request and route reply mechanisms, to reduce end to end delay, and hence that reduces the overheads and increases both packet delivery ratio and throughput especially in high mobility situations.

Additional, enhanced performance is expected by MDVZRP once the following issues are investigated and resolved:

1. Speed up the search for available routes, by reducing nodes routing table's size to keep only fresh and active routes to each known destination, especially in the large networks and at high speeds with short pause of time, which increases the possibility of links breakage and more stale not useful routes.

2. Enhance the existence mechanism to determine routes freshness when multipath choices are available.

3. Speed up routing information gathering mechanism at the intermediate nodes, especially during route reply mechanism RREP, would reduce the end to end delay.

Furthermore, the current MDVZRP specification states that the *RREQ_Threshold* parameter limit the extent to which all the control packets may propagate through the network, and hence reduced the average overhead as clarified in Phase 1 and Phase 2 results. Till this stage, the *RREQ_Threshold* sets manually, we hope to find an efficient mechanism to set its value automatic based on the network area size.

We also concluded that every node that receives RREQ either unicasts a RREP or rebroadcasts the same RREQ again in case if no information available, since the nodes are moving, some nodes that received that RREQ may be out of the RREQ sender or forwarder transmission range for a short period. Therefore, there is a need for an efficient mechanism to detect the node who expects itself to be out of its neighbour's or (the RREQ sender or forwarder) transmission range in a short period to neglect that RREQ. This leads to reduce both overhead and weak routes those affected by fast breakage, and hence increases the MDVZRP's performance.

Finally, MDVZRP is utilising multipath as a backup only at this stage, possible performance enhancements may be investigated by transferring data packets over the multipath simultaneously.

7.6 Summary

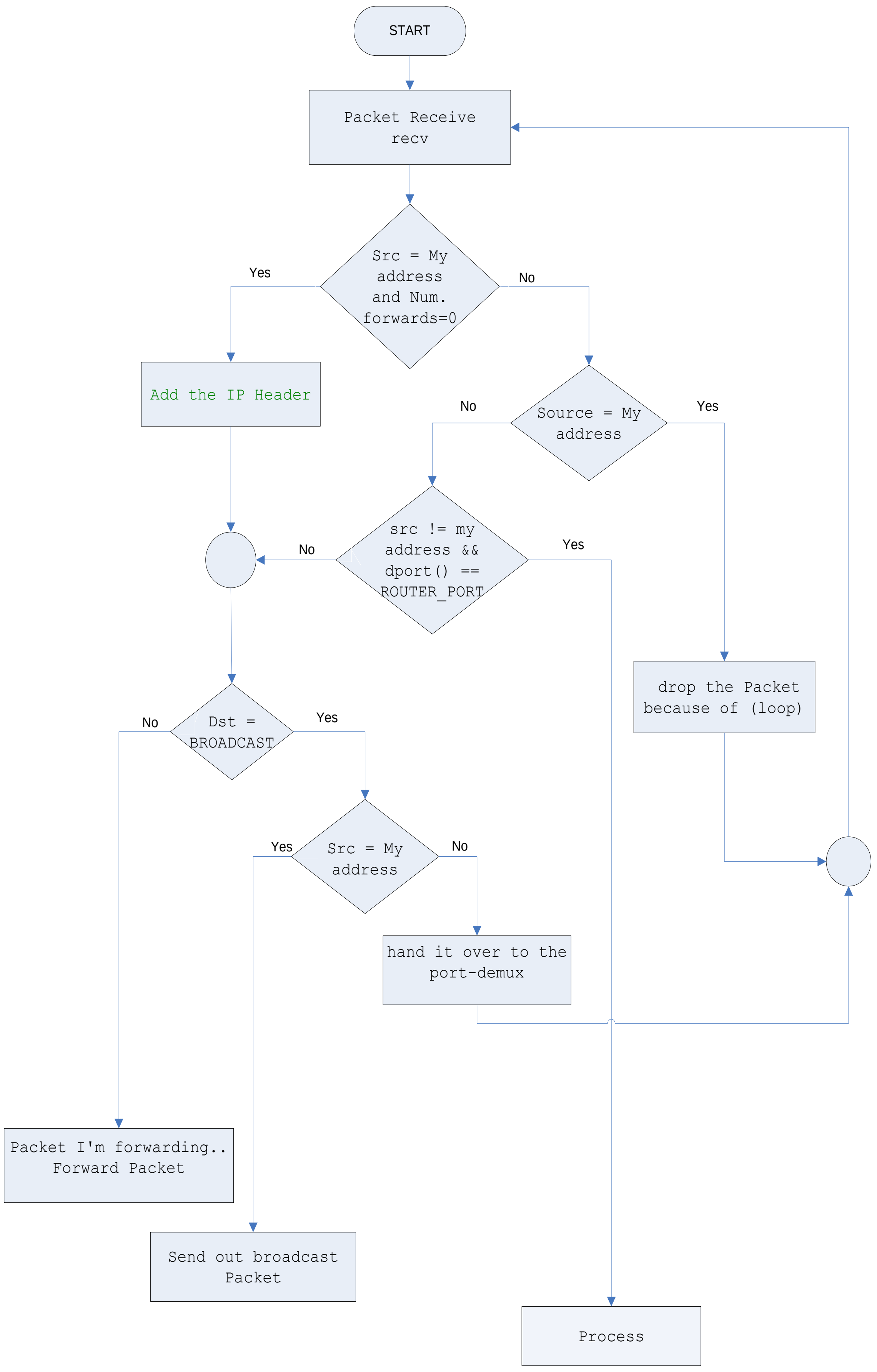
This work has been concerned with the development of a new hybrid routing protocol for MANETs based on AODV and DSDV structures and mechanisms as a starting point. The new protocol (MDVZRP) adapts both reactive and proactive mechanisms to perform a reliable connection between mobile nodes via multi backup paths.

MDVZRP came cross many stages of design and improvement over a period of time into what it is today. Some of the improvements were in the code itself and others were in the protocol's algorithm technique. MDVZRP's packets, agent and functions all are implemented using C++ under Ns2 (Network Simulator 2).

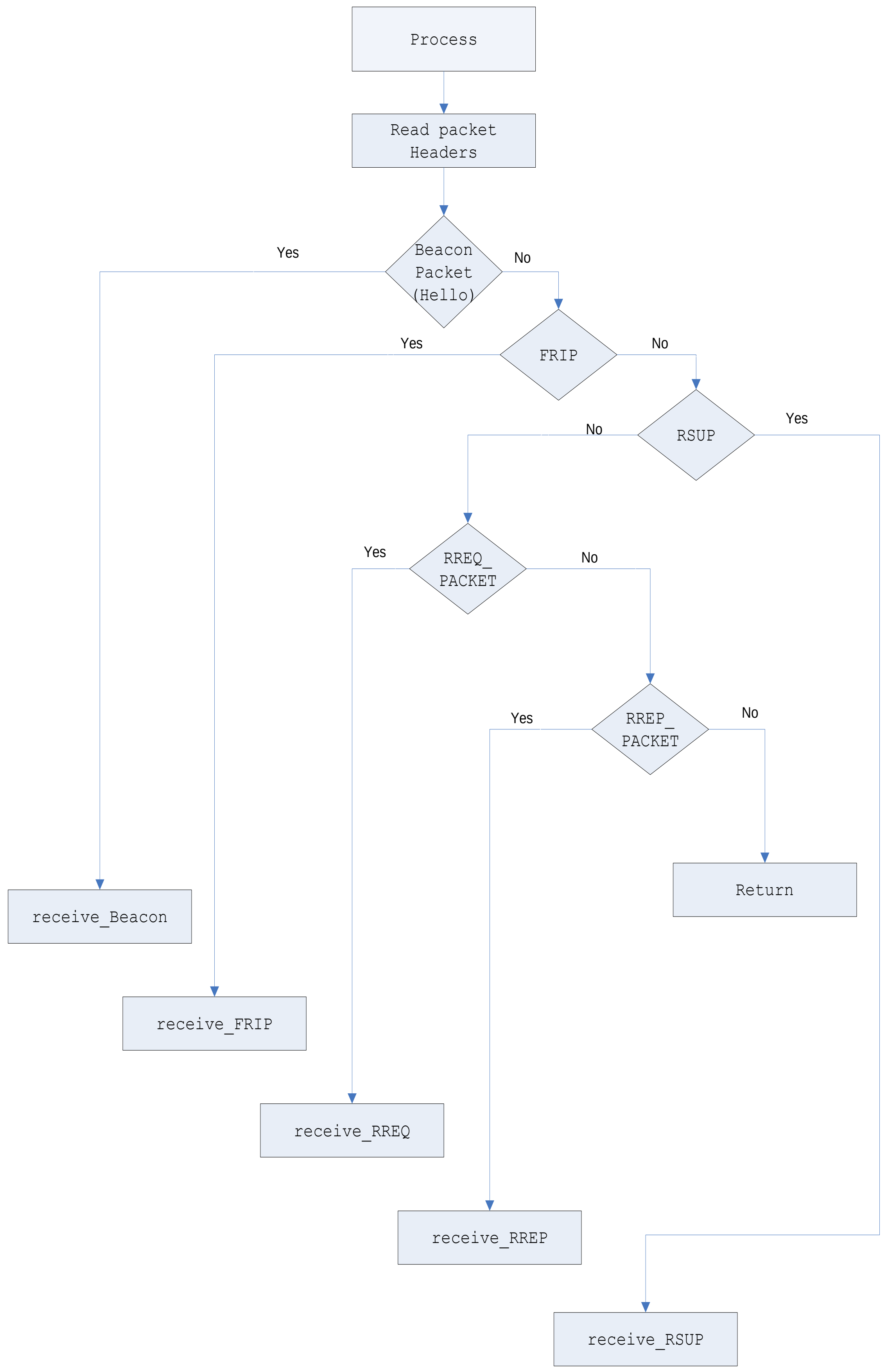
Since the protocol shows a good performance in CBR traffic for any network sizes with low or medium mobility, it did not meet the same performance as well as expected for high mobility ones. Preliminary results indicate a very promising start for MDVZRP, which is currently at version 2.00. Further work is likely to lead to additional improvements to its performance.

APPENDIX A

MDVZRP Packet Reception Algorithm


START
Packet Receive recv
Src = My address and Num. forwards=0
Yes
No
Add the IP Header
Source = My address
No
Yes
src != my address && dport() == ROUTER_PORT
No
Yes
drop the Packet because of (loop)
Dst = BROADCAST
No
Yes
Src = My address
Yes
No
hand it over to the port-demux
Packet I'm forwarding.. Forward Packet
Send out broadcast Packet
Process

APPENDIX A

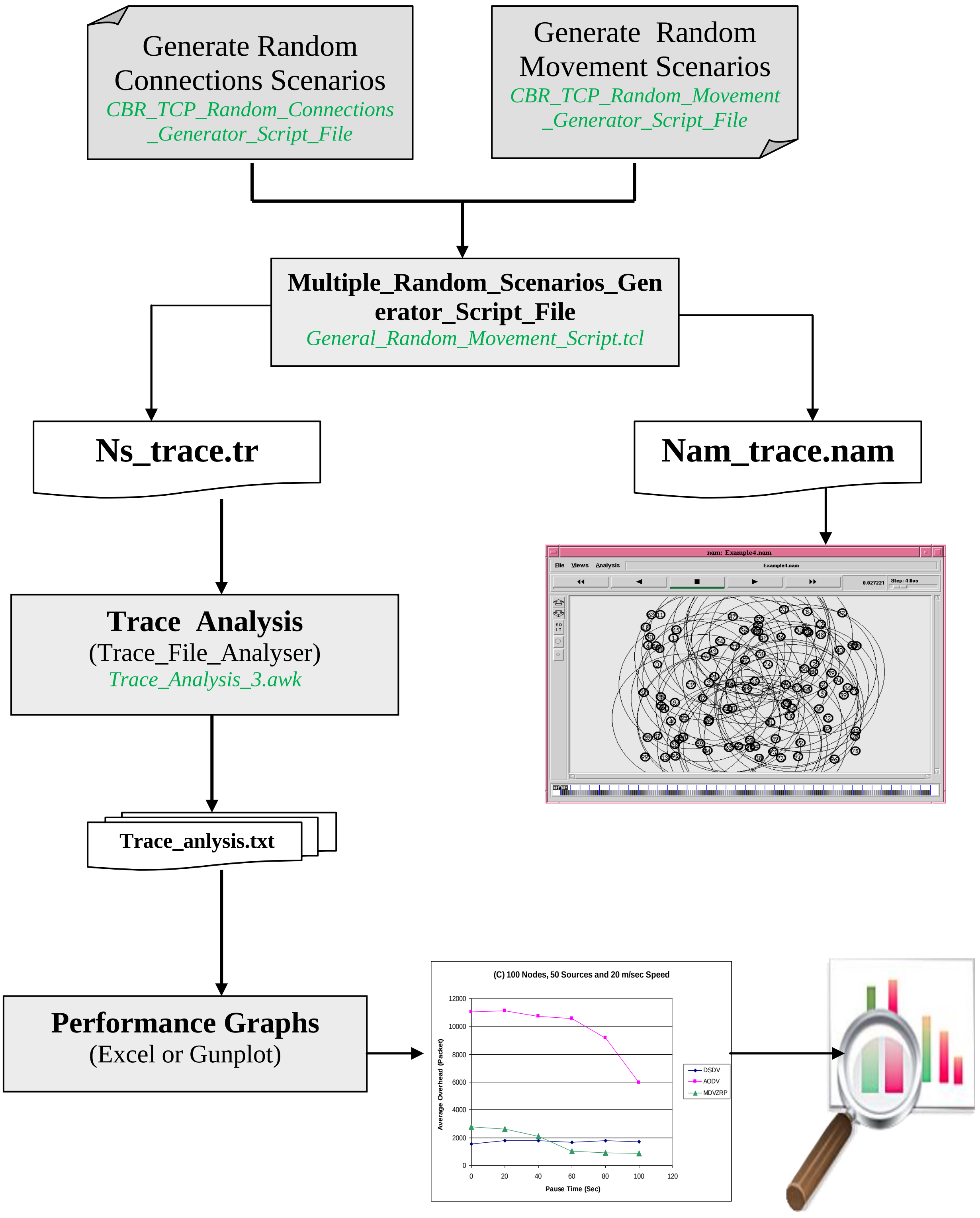

Process
Read packet Headers
Beacon Packet (Hello)
Yes
No
FRIP
Yes
No
RSUP
No
Yes
RREQ_ PACKET
Yes
No
RREP_ PACKET
Yes
No
Return
receive_Beacon
receive_FRIP
receive_RREQ
receive_RREP
receive_RSUP

APPENDIX B

Performance Analysis Flow Diagram

APPENDIX B

Performance Analysis Flow Diagram



Generate Random Connections Scenarios
CBR_TCP_Random_Connections_Generator_Script_File
Generate Random Movement Scenarios
CBR_TCP_Random_Movement_Generator_Script_File
Multiple_Random_Scenarios_Generator_Script_File
General_Random_Movement_Script.tcl
Ns_trace.tr
Nam_trace.nam
nam: Example4.nam
File Views Analysis
Example4.nam
Trace Analysis
(Trace_File_Analyser)
Trace_Analysis_3.awk
Trace_anlysis.txt
Performance Graphs
(Excel or Gunplot)
(C) 100 Nodes, 50 Sources and 20 m/sec Speed
Average Overhead (Packet)
Pause Time (Sec)
DSDV
AODV
MDVZRP

APPENDIX C

Some Script Files used for Performance Evaluation

CBR_TCP_Random_Connections_Generator

```
# ===================================================================
#  This file generates Number of TCP or CBR connection scenarios  according to
# N x CONN values , this  files for an example generates  1x4 files
# ===================================================================
#   Part 1 : Some Parameters and Argmentsts Declaration
# ===================================================================
clear                              # To clear the output screen [CRT]
X=500                              # Topology x size (width)
Y=500                              # Topology y size (high)
Time=100                           # Maximum simulation time
for N in  25 50 100                # Number of Nodes  25 50 100 150 200
                                   # 250 ..etc
do
for CONN in 5 10 15 20 60          # Num. of  Connections
do
       echo " "                    # print new line
       echo " "                    # print new line
      DATE="$(date '+%Y%m%d')"     # set date
      echo $DATE                   # print date
      date | awk '{ print $4 }'    # print time
      date | awk '{ print $1 $2 $3 $4 }'
# ===================================================================
#   Part 2 : To Create a TCP Random Connections Files
# ===================================================================
# echo " Creating  tcp_scen-N"$N-1-C$CONN-4
# ns  ~/ns-allinone-2.30/ns-2.30/indep-utils/cmu-scen-gen/cbrgen.tcl –
# type tcp -nn $N -seed 1.0 -mc $C -rate 4.0 > tcp_scen-N$N-1-C$C-4
#  echo " The File " TCP_Connections_Scen-N"$N-1-C$CONN ...OK Done!!”
#  echo “==============================================================”
# ===================================================================
#   Part 3 : To Create a CBR Random Connections Files
# ===================================================================
echo " Creating  cbr_scen-N"$N-1-C$CONN "Random Connection File
                                              .... Please Wait !!"
 ns  ~/ns-allinone-2.30/ns-2.30/indep-utils/cmu-scen-gen/cbrgen.tcl -type cbr -nn
$N -seed 1.0 -mc $CONN -rate 2.0 > CBR_Connections_Scen-N$N-1-C$CONN
echo " The File " CBR_Connections_Scen-N"$N-1-C$CONN .... OK Done!!”
echo "=============================================================="
 done
done
```

CBR_TCP_Random_Movement_Generator

```
# =================================================================
#  This file generates Random movement scenarios according CxNxMxP  times where C
#  for example 10 if written as (for C in 1 2 3 4 5 6 7 8  9 10), N is twice if it is written as
# (for N in 30 40) for an example, M is only one e.g (for M in 20), and P is 6 if it is  written
#  as (for P in 0 20 40 60 80 100). In this case this scripts generates [C x N x M x P]
# 10x2x1x6=120 random scenario files.
# =================================================================
#   Part 1 : Some Parameters and Argmentsts Declaration
# =================================================================
lear                              # To clear the output screen [CRT]
X=500                             # Topology x size (width)
Y=500                             # Topology y size (high)
Time=100                          # Maximum simulation time
for C in 1 2 3 4 5 6 7 8 9 10  # Number of random files to generate
do
 for N in 25 50 100               # Maximum number of Nodes in this network 20 40 60
 do
   for M in  20                  # Node's Speed [0-20] meter/sec  [eg. 0 1 5..etc]
   do
           #clear                # To clear the outpot CRT screen
           for P in 0 20 40 60 80 100    # Pause Time [1 2 3 4 5 ] /sec or 0 20 40 60 80 100
                                         # 0 means high mobility no pause time , if simulation
                                         #  Time =100  and P=100 it means no mobility.
           do
# =================================================================
#          Part 2 : Some Messages Printing
# =================================================================
# The next 3 lines just for printing a messages show that thescrip is running fine (Ok),
# echo means (Print).
echo  "File Scenario Program is  ..................... Running"
echo  "The File Scenario  Mov_Random_Scen-N"$N"-"$C"-P"$P"-M"$M"-t"$Time"-X"$X"-Y"$Y""
echo  # Print nothing just a space line
echo  "            Under Creation .....  Please Wait .....!!"
# =================================================================
#   Part 3 : To Create Random Movement Scenario Files
# =================================================================
# This is the main command to generate a random scenario according
# to the given parameters, see setdest command from ns2 manual.
~/ns-allinone-2.30/ns-2.30/indep-utils/cmu-scen-gen/setdest/setdest -v 1 -n $N -p $P
-M $M -t $Time -x $X -y $Y >Mov_Random_Scen-N$N-$C-P$P-M$M-t$Time-x$X-y$Y
  # Print a message to show that the file is correctly generated.
   echo  "The FileScenario  Mov_Random_Scen-N"$N"-"$C"-P"$P"-M"$M"-t"$Time"-x"
   $X"-y"$Y"  .. Ok ... Done"
           echo " "
           echo " "
           echo " "
    done
   done
  done
 done
exit
```

General_Random_Movement_Script.tcl

```
# ====================================================================
# This is the main script.tcl file, receives from the
# Multiple_Random_Scenarios_Generator_Script_File, some parameters, argments and
#the 2 following files :
# 1- Random scenario (Mov_Random_Scen-N..etc)
# 2- Random connections (CBR_Connections_Scen-N..etc), which generates
# two trace files (Ns_trace_'..etc'.tr' and 'Nam_trace_'..etc'.nam'),and saves them in a given path
# ====================================================================
# Set New Argments to That Argments Passed From
# Multi_Scripts_Generator_Script_File
# ====================================================================
set arg0 [lindex $argv 0]   ;# Protocol name DSDV, AODV or MDVZRP [$Protocol_nam]
set arg1 [lindex $argv 1]   ;# Pause time value  [$p]
set arg2 [lindex $argv 2]   ;# Speed value [$M]
set arg3 [lindex $argv 3]   ;# Ns trace file name and directory  [e.g  'Ns_trace_'..etc'.tr']
set arg4 [lindex $argv 4]   ;# Nam trace file and directory   [e.g 'Nam_trace_'..etc'.nam']
set arg5 [lindex $argv 5]   ;# name of movment file [e.g 'Mov_Random_Scen-N...etc]
set arg6 [lindex $argv 6]   ;# x  [$x]
set arg7 [lindex $argv 7]   ;# y  [$y]
set arg8 [lindex $argv 8]   ;# Time [ $Time]
set arg9 [lindex $argv 9]   ;# number of Nodes [$N]
set arg10 [lindex $argv 10] ;# name of connection file [e.g 'CBR_Connections_Scen-N..etc]
set arg11 [lindex $argv 11] ;# senario Number  [$C  ]
set arg12 [lindex $argv 12] ;# Num. of Sources ,Connection Number [$CONN]
# ====================================================================
#          Some Messages Printing
# ====================================================================
puts "===================================================================="
puts [format "%s   Is Runing ok........Senario(%d)   Pause Time= %d
   Speed= %d  Num. Of Connections= %d" $arg0 $arg11 $arg1 $arg2
   $arg12] ;#Print These Argments
puts [format "          Please Wait.....!" ] ;# Print a message
puts " "  ;# Print a line
puts " "  ;# Print a line
#exit
# ====================================================================
#          Define Simulator Network Options
# ====================================================================
set opt(chan)       Channel/WirelessChannel
set opt(prop)       Propagation/TwoRayGround
set opt(netif)      Phy/WirelessPhy
set opt(mac)        Mac/802_11
#set opt(ifq)       CMUPriQueue  ;# use this for DSR
set opt(ifq)        Queue/DropTail/PriQueue
set opt(ll)         LL
set opt(ant)        Antenna/OmniAntenna
set opt(x)          $arg6      ;# X dimension of the topography
set opt(y)          $arg7      ;# y dimension of the topography
set opt(ifqlen)     50         ;# max packet in ifq
set opt(seed)       0.0
set opt(tr)         $arg3      ;# set opt(tr)      x555.tr
set opt(adhocRouting)  $arg0   ;# set opt(adhocRouting)DSDV,AODV,MDVZRP....etc
set opt(nn)         $arg9      ;# how many nodes are simulated
set opt(sc)         $arg5      ;# The Random Movement File , For an Example
;# set opt(sc) "Mov_Random_Scen-N...etc"
set opt(cp)         $arg10     ;# The Random Connection File , For an Example
```

```
 ;# set opt(cp) "CBR_Connections_Scen-N50-1-C15-4"
set opt(stop)   $arg8       ;# Maximum Simulation Time
# =======================================================================
#           Main Program
# =======================================================================
# Initialize Global Variables
# create simulator instance
set ns_               [new Simulator]
# set wireless channel, radio-model and topography objects
set wtopo             [new Topography]
# =======================================================================
#           Define Ns  Trace File, and Set its Format
# =======================================================================
# create trace object for ns and nam
set tracefd           [open $opt(tr) w];# Ns trace File
# in all formats always keep this line
$ns_ trace-all $tracefd
# In case You want the Old Ns trace file format use this line
#$ns_ use-taggedtrace ;#-----> Old Ns Trace File Format (False)
# In case You want the New Ns trace file format use this line
$ns_ use-newtrace     ;# ----> New Ns Trace File Format (True)
# =======================================================================
#           Define Nam Trace File
# =======================================================================
set namtrace  [open $arg4 w] ;# Nam trace file
$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)
# =======================================================================
#           Define Topology
# =======================================================================
$wtopo load_flatgrid $opt(x) $opt(y)
# Create God
# puts "before god_....." ;#Print a message
set god_ [create-god $opt(nn)]
# puts "After god_....."  ;#Print a message
# =======================================================================
#           Define How Node Should Be Created
# =======================================================================
#global node setting
$ns_ node-config -adhocRouting $opt(adhocRouting) \
                 -llType $opt(ll) \
                 -macType $opt(mac) \
                 -ifqType $opt(ifq) \
                 -ifqLen $opt(ifqlen) \
                 -antType $opt(ant) \
                 -propType $opt(prop) \
                 -phyType $opt(netif) \
                 -channelType $opt(chan) \
                 -topoInstance $wtopo \
                 -agentTrace ON \
                -routerTrace ON \
                -macTrace OFF
# =======================================================================
#   Create The Specified Number Of Nodes and Attach Them To Channel1
# =======================================================================
for {set i 0} {$i < $opt(nn) } {incr i} {
   #puts "Starting the node creation......$i....." ;# Print a messag
   set node_($i) [$ns_ node]
```

```
    #puts "motion----------------"          ;# Print a message
    $node_($i) random-motion 0       ;# Disable Random Motion
}
# ====================================================================
#           Define Node Movement Model
# ====================================================================
puts "Loading connection pattern..." ;# Print a Message
source $opt(cp) ;# Read (load) Connection from  The Random Connection File
# ====================================================================
#           Define Traffic Model
# ====================================================================
puts "Loading scenario file..."          ;# Print a Message
source $opt(sc) ;# Read (load) Movement from  The Random Scenario File
# ====================================================================
#           Define Node Initial Position In Nam
# ====================================================================
for {set i 0} {$i < $opt(nn)} {incr i} {
    # 30 defines the node size in nam, must adjust it according to
      your scenario
    # The function must be called after mobility model is defined
   $ns_ initial_node_pos $node_($i) 30
}
# ====================================================================
#           Tell Nodes When The Simulation Ends
# ====================================================================
for {set i 0} {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).000000001 "$node_($i) reset";
}
# ====================================================================
#           Tell Nam When The Simulation Stop Time
# ====================================================================
#The next line was as a comment...
$ns_ at  $opt(stop)           "$ns_ nam-end-wireless $opt(stop)"
#The next line is added...
$ns_ at $opt(stop) "stop"

puts " "  ;#Print a line (space)
puts " "  ;#Print a line (space)

$ns_ at  $opt(stop).000000001 "puts \"NS EXITING...\" ; $ns_ halt"
puts " "  ;#Print line (space)
proc stop {} {
    global ns_ tracefd namtrace
    $ns_ flush-trace
    close $tracefd    ;# Close Trace File
    close $namtrace   ;# Close Nam File
}
puts "Starting Simulation  * * *    Please Wait....."  ;#Print a message
$ns_ run
```

Trace_File_Analyser

```
# ==================================================================
#  This is a script file defines the network topology parameters in  part 1, prints some
# messages in part 2 and then calles an AWK  program called Trace_Analysis_3 to carry
# some calculations on a  gaiven trace file
# ==================================================================
#   Part 1 : Some Parameters and Argmentsts Declaration
# ==================================================================
clear           # To clear the output screen [CRT]
X=500           # Topology x size (width)
Y=500           # Topology y size (high)
Time=100        # Maximum simulation time
N=100           # Number of Nodes
CONN=60         # Num. of Connections(Sources)
# ==================================================================
#   Part 1 : Printing Some Messages into trace_analysis.txt File
# ==================================================================
echo  " " >> trace_anlysis.txt
echo "    T       E       C       H       N       I       C       A       L
          D       A       T       A               S       H       E       E       T  " >>
trace_anlysis.txt
echo "    Performance Evaluation of MDVZRP Ver2.00 v DSDV and AODV Ad Hoc Standard Protocols
          ">> trace_anlysis.txt
#date | awk '{ print $0 $1 $2 $3 $4 $5}' >> trace_anlysis.txt       #print time
echo  " " >> trace_anlysis.txt
echo  " " >> trace_anlysis.txt
date | awk '{ print "Date: "$1" " $3 "-"$2"-"$6 }' >> trace_anlysis.txt
date | awk '{ print "Time: "$4" " $5 }' >> trace_anlysis.txt
echo  " " >> trace_anlysis.txt
echo  "Senario Parametrs: ">> trace_anlysis.txt
echo  "Nodes:"$N >> trace_anlysis.txt
echo  "Simulation time:"$Time >> trace_anlysis.txt
echo  "Area Size: X"$x "Y"$y   >> trace_anlysis.txt
echo  "Num. of Connections(Sources):" $CONN >> trace_anlysis.txt
echo  " " >> trace_anlysis.txt
echo " ========================================">> trace_anlysis.txt
for P in 0 20 40 60 80 100                      #Pause Time [0 1 2 3 4 5 ....etc or 0 20 30.....etc ]  m.sec
do
   echo " Number of Nodes:" $N                  # Number of Nodes
   echo " Number of Sources:" $CONN             # Num. of Connections(Sources)
# ==================================================================
  echo  "            Senario  Pause   Speed   StTime  SpTime  PDF    NRL    EED    [kbps]
          O/H         Send    Recive  Drop" >> trace_anlysis.txt
# ==================================================================
  for Protocol_name in 'DSDV' 'AODV' 'MDVZRP'
  do
   for C in 1 2 3 4 5 6 7 8 9 10       #Number of times  2 3 4 5 6 7 8 9 10
   do
    for M in 20                        # Node's Speed [5 10 15 ... etc] 5 10 15 sec
    do
      echo  "Reading " $Protocol_name  "trace file please wait ...... Senario No. "$C  "Pause "$P   "Speed
"$M
# ==================================================================
#   Part 3 : Calling the AWK 'Trace_Analysis_3.awk' and passing to it Some Parameters
#   and the trace file to be analysis
# ==================================================================
awk -v p_name=$Protocol_name -v Senario_No=$C -v pause=$P -v speed=$M -f Trace_Analysis_3.awk
'Ns_trace_P'$P'_M'$M'_N'$N'_'$C'_CONN'$CONN'_'$Protocol_name'.tr' >> trace_anlysis.txt
echo " ">> trace_anlysis.txt
 done
done
echo "  -----------------------------------------">> trace_anlysis.txt
            echo " ">> trace_anlysis.txt
            echo " "
            done
            echo " "
   echo " ">> trace_anlysis.txt
   echo "  ====================================">> trace_anlysis.txt
done
```

APPENDIX D

Samples of Performance Evaluation Technical Data Sheet

```
     T    E    C    H    N    I    C    A    L         D    A    T    A         S    H    E    E    T

                        Performance Evaluation of MDVZRP Ver2.00 v DSDV and AODV Ad Hoc Standard Protocols

Date: Tue 2-Mar-2010
Time: 13:08:40 GMT
Senario Parametrs:
Nodes:100
Simulation time:100
Area Size: X500 Y500
Num. of Connections(Sources): 60
  ==========================================================================================
         Senario  Pause  Speed  StTime  SpTime    PDF    NRL    EED    [kbps]    O/H    Drop
  DSDV       1      0     20     2.56      100    78.8   0.54   0.06   109.84    1416   692
  DSDV       2      0     20     2.56      100    75.5   0.58   0.05   106.03    1475   814
  DSDV       3      0     20     2.56      100    73.5   0.61   0.02   102.65    1480   866
  DSDV       4      0     20     2.56      100    86.1   0.54   0.01   119.39    1536   449
  DSDV       5      0     20     2.56      100    82.4   0.55   0.01   114.80    1501   577
  DSDV       6      0     20     2.56      100    89.7   0.48   0.01   124.90    1424   336
  DSDV       7      0     20     2.56      100    77.7   0.57   0.02   108.41    1467   733
  DSDV       8      0     20     2.56      100    74.4   0.62   0.04   104.05    1536   841
  DSDV       9      0     20     2.56      100    82.6   0.55   0.01   115.47    1507   575
  DSDV      10      0     20     2.56      100    73.7   0.62   0.04   102.92    1517   866
  ----------------------------------------------------------------------------------------
  AODV       1      0     20     2.56      100    94.6   4.22   0.40   132.34    13287   169
  AODV       2      0     20     2.56      100    96.9   4.97   0.39   134.45    15874   77
  AODV       3      0     20     2.56      100    96.0   4.75   0.29   134.59    15191   99
  AODV       4      0     20     2.56      100    95.0   3.64   0.15   132.87    11510   147
  AODV       5      0     20     2.56      100    98.2   4.13   0.18   136.78    13442   51
  AODV       6      0     20     2.56      100    97.1   3.10   0.27   135.61    9991    78
  AODV       7      0     20     2.56      100    95.4   4.97   0.49   134.01    15856   144
  AODV       8      0     20     2.56      100    95.6   3.94   0.14   134.05    12551   127
  AODV       9      0     20     2.56      100    95.2   4.82   0.51   133.68    15342   130
  AODV      10      0     20     2.56      100    93.5   5.99   0.25   130.41    18580   195
  ----------------------------------------------------------------------------------------
  MDVZRP     1      0     20     2.56      100    85.2   1.01   0.58   119.33    2863   278
  MDVZRP     2      0     20     2.56      100    74.1   1.41   1.54   103.54    3477   413
  MDVZRP     3      0     20     2.56      100    76.3   1.37   1.88   107.19    3485   471
  MDVZRP     4      0     20     2.56      100    95.3   0.95   0.48   133.21    3004   118
  MDVZRP      5     0    20    2.56        100    92.8   0.92   0.67   129.90   2842  147
  MDVZRP      6     0    20    2.56        100    95.5   0.86   0.25   132.84   2717  145
  MDVZRP      7     0    20    2.56        100    85.7   1.04   0.68   119.77   2963  250
  MDVZRP      8     0    20    2.56        100    86.5   1.11   0.72   119.98   3180  159
  MDVZRP      9     0    20    2.56        100    85.2   1.11   0.93   117.84   3102  278
  MDVZRP     10     0    20    2.56        100    77.5   1.32   1.15   107.74   3389  439
  --------------------------------------------------------------------------------------


  ========================================================================================
         Senario  Pause  Speed  StTime  SpTime    PDF    NRL    EED    [kbps]   O/H   Drop
  DSDV        1    20    20    2.56        100    79.2   0.71   0.07   111.18   1890  690
  DSDV        2    20    20    2.56        100    70.0   0.74   0.02   97.43    1725  981
  DSDV        3    20    20    2.56        100    77.1   0.70   0.05   107.98   1790  753
  DSDV        4    20    20    2.56        100    80.4   0.64   0.03   111.95   1699  641
  DSDV        5    20    20    2.56        100    86.6   0.63   0.01   121.10   1803  439
  DSDV        6    20    20    2.56        100    78.8   0.65   0.01   110.49   1708  696
  DSDV        7    20    20    2.56        100    75.0   0.77   0.02   104.11   1898  818
  DSDV        8    20    20    2.56        100    81.0   0.64   0.01   113.05   1710  620
  DSDV        9    20    20    2.56        100    77.6   0.67   0.03   108.75   1724  738
  DSDV       10    20    20    2.56        100    75.8   0.68   0.02   106.01   1722  797
  --------------------------------------------------------------------------------------
  AODV        1    20    20    2.56        100    97.8   3.25   0.16   135.56   10467  71
  AODV        2    20    20    2.56        100    94.5   4.95   0.42   132.12   15567  150
  AODV        3    20    20    2.56        100    95.6   4.19   0.20   133.50   13313  128
  AODV        4    20    20    2.56        100    97.4   3.02   0.32   135.49   9723   86
  AODV        5    20    20    2.56        100    98.8   2.86   0.07   137.84   9377   21
  AODV        6    20    20    2.56        100    97.4   3.60   0.18   134.93   11563  73
  AODV        7    20    20    2.56        100    95.7   4.29   0.14   132.47   13505  117
  AODV        8    20    20    2.56        100    95.8   3.93   0.26   132.70   12416  135
  AODV        9    20    20    2.56        100    96.5   4.46   0.41   135.45   14379  114
  AODV       10    20    20    2.56        100    97.1   3.93   0.20   135.74   12700  86
  --------------------------------------------------------------------------------------
  MDVZRP      1    20    20    2.56        100    76.1   1.24   1.73   106.36   3127  422
  MDVZRP      2    20    20    2.56        100    65.6   1.44   0.81   91.18    3121  686
  MDVZRP      3    20    20    2.56        100    83.7   0.98   0.80   116.61   2727  315
  MDVZRP      4    20    20    2.56        100    97.4   0.78   0.51   135.32   2519  74
  MDVZRP      5    20    20    2.56        100    86.7   1.04   0.56   121.00   3001  250
  MDVZRP      6    20    20    2.56        100    83.7   0.95   1.09   115.24   2602  265
```

```
  MDVZRP      7     20     20     2.56          100     74.2    1.13    1.02    102.99    2775   373
  MDVZRP      8     20     20     2.56          100     80.1    0.95    1.04    110.98    2502   351
  MDVZRP      9     20     20     2.56          100     66.6    1.41    2.15    92.62     3117   576
  MDVZRP     10     20     20     2.56          100     79.0    1.02    1.66    110.44    2680   416
  -------------------------------------------------------------------------------------------------



=====================================================================================================
          Senario  Pause  Speed   StTime   SpTime     PDF     NRL     EED     [kbps]     O/H    Drop
  DSDV        1     40     20     2.56          100     73.9    0.71    0.02    102.74    1742   846
  DSDV        2     40     20     2.56          100     88.5    0.62    0.01    123.46    1817   373
  DSDV        3     40     20     2.56          100     84.4    0.65    0.02    117.82    1833   513
  DSDV        4     40     20     2.56          100     82.7    0.64    0.01    115.44    1768   567
  DSDV        5     40     20     2.56          100     90.0    0.60    0.01    126.42    1813   330
  DSDV        6     40     20     2.56          100     83.7    0.62    0.02    116.37    1717   537
  DSDV        7     40     20     2.56          100     94.3    0.55    0.01    131.19    1713   183
  DSDV        8     40     20     2.56          100     88.5    0.63    0.02    124.14    1847   377
  DSDV        9     40     20     2.56          100     80.1    0.65    0.02    112.75    1737   645
  DSDV       10     40     20     2.56          100     78.4    0.66    0.02    109.61    1709   706
  -------------------------------------------------------------------------------------------------
  AODV        1     40     20     2.56          100     95.5    4.27    0.37    134.40    13648   97
  AODV        2     40     20     2.56          100     95.8    2.55    0.16    134.14    8134    138
  AODV        3     40     20     2.56          100     96.0    3.64    0.24    134.75    11653   128
  AODV        4     40     20     2.56          100     95.2    3.82    0.34    132.17    12004   132
  AODV        5     40     20     2.56          100     99.2    2.34    0.11    138.21    7696    22
  AODV        6     40     20     2.56          100     98.0    3.10    0.27    137.37    10146   48
  AODV        7     40     20     2.56          100     96.0    2.72    0.18    134.07    8675    131
  AODV        8     40     20     2.56          100     97.7    3.06    0.10    136.24    9903    38
  AODV        9     40     20     2.56          100     95.9    3.23    0.13    133.42    10242   134
  AODV       10     40     20     2.56          100     98.7    2.95    0.06    137.07    9602    41
  -------------------------------------------------------------------------------------------------
  MDVZRP      1     40     20     2.56          100     82.4    0.90    0.22    115.22    2477   219
  MDVZRP      2     40     20     2.56          100     92.7    0.84    0.31    129.33    2596   50
  MDVZRP      3     40     20     2.56          100     79.5    0.90    0.61    111.11    2387   252
  MDVZRP      4     40     20     2.56          100     86.7    0.81    0.18    120.51    2322   179
  MDVZRP      5     40     20     2.56          100     90.5    0.83    0.30    126.19    2501   92
  MDVZRP      6     40     20     2.56          100     94.3    0.74    0.21    131.66    2313   52
  MDVZRP      7     40     20     2.56          100     92.7    0.78    0.22    127.74    2363   60
  MDVZRP      8     40     20     2.56          100     89.6    0.88    0.27    125.10    2606   90
  MDVZRP      9    40    20    2.56        100    87.2   0.89   0.22   121.30   2567  127
  MDVZRP     10    40    20    2.56        100    86.7   0.82   0.20   121.99   2371  238
  -------------------------------------------------------------------------------------
=========================================================================================
         Senario  Pause  Speed  StTime     SpTime  PDF    NRL    EED    [kbps]    O/H   Drop
  DSDV        1    60    20    2.56        100    80.2   0.68   0.02   111.85   1816  648
  DSDV        2    60    20    2.56        100    89.2   0.55   0.01   125.12   1647  357
  DSDV        3    60    20    2.56        100    89.3   0.60   0.02   124.72   1785  346
  DSDV        4    60    20    2.56        100    86.1   0.61   0.02   119.97   1728  454
  DSDV        5    60    20    2.56        100    85.5   0.57   0.02   118.76   1624  472
  DSDV        6    60    20    2.56        100    88.1   0.60   0.02   122.71   1752  384
  DSDV        7    60    20    2.56        100    87.1   0.58   0.02   121.74   1674  423
  DSDV        8    60    20    2.56        100    87.3   0.57   0.02   123.42   1669  426
  DSDV        9    60    20    2.56        100    85.1   0.59   0.02   118.96   1681  481
  DSDV       10    60    20    2.56        100    88.4   0.57   0.01   123.05   1661  380
  -------------------------------------------------------------------------------------
  AODV        1    60    20    2.56        100    96.8   3.55   0.09   134.72   11375  89
  AODV        2    60    20    2.56        100    94.2   3.44   0.27   132.16   10817  188
  AODV        3    60    20    2.56        100    97.5   3.11   0.13   136.79   10126  79
  AODV        4    60    20    2.56        100    95.4   2.74   0.20   134.41   8754   112
  AODV        5    60    20    2.56        100    96.3   3.20   0.33   134.30   10227  99
  AODV        6    60    20    2.56        100    97.9   2.83   0.39   136.61   9197   18
  AODV        7    60    20    2.56        100    94.9   3.83   0.33   130.94   11940  151
  AODV        8    60    20    2.56        100    98.3   3.18   0.10   137.30   10392  52
  AODV        9    60    20    2.56        100    98.8   2.79   0.15   139.77   9286   32
  AODV       10    60    20    2.56        100    97.9   3.54   0.13   135.61   11409  51
  -------------------------------------------------------------------------------------
  MDVZRP      1    60    20    2.56        100    83.0   0.39   0.50   115.63   1062  279
  MDVZRP      2    60    20    2.56        100    95.0   0.37   0.09   132.33   1166  20
  MDVZRP      3    60    20    2.56        100    93.2   0.35   0.22   130.36   1080  80
  MDVZRP      4    60    20    2.56        100    95.7   0.34   0.11   132.75   1067  72
  MDVZRP      5    60    20    2.56        100    90.7   0.39   0.12   125.74   1175  122
  MDVZRP      6    60    20    2.56        100    95.1   0.31   0.09   132.19   972   72
  MDVZRP      7    60    20    2.56        100    93.3   0.37   0.19   129.73   1140  81
  MDVZRP      8    60    20    2.56        100    95.1   0.34   0.11   133.21   1068  112
  MDVZRP      9    60    20    2.56        100    96.3   0.29   0.17   134.84   922   61
  MDVZRP     10    60    20    2.56        100    93.6   0.37   0.17   129.72   1152  153
  -------------------------------------------------------------------------------------
```

```
=========================================================================================
        Senario  Pause  Speed  StTime    SpTime   PDF    NRL    EED    [kbps]   O/H    Drop
  DSDV       1    80     20    2.56       100    91.3   0.56   0.02   127.51   1713   286
  DSDV       2    80     20    2.56       100    91.6   0.60   0.02   127.35   1807   273
  DSDV       3    80     20    2.56       100    94.5   0.57   0.02   131.41   1772   178
  DSDV       4    80     20    2.56       100    93.4   0.58   0.02   130.33   1798   213
  DSDV       5    80     20    2.56       100    94.1   0.55   0.01   131.87   1741   194
  DSDV       6    80     20    2.56       100    93.1   0.60   0.02   130.74   1865   225
  DSDV       7    80     20    2.56       100    86.9   0.59   0.03   121.37   1712   428
  DSDV       8    80     20    2.56       100    94.7   0.54   0.02   131.75   1693   173
  DSDV       9    80     20    2.56       100    92.7   0.61   0.02   129.92   1882   234
  DSDV      10    80     20    2.56       100    88.0   0.60   0.02   122.99   1741   390
  ---------------------------------------------------------------------------------------
  AODV       1    80     20    2.56       100    97.0   3.27   0.15   134.64   10458   92
  AODV       2    80     20    2.56       100    99.3   2.04   0.08   138.25   6724   21
  AODV       3    80     20    2.56       100    98.8   2.33   0.13   137.88   7628   19
  AODV       4    80     20    2.56       100    98.3   2.61   0.13   136.52   8479   31
  AODV       5    80     20    2.56       100    93.9   2.40   0.08   131.09   7484   90
  AODV       6    80     20    2.56       100    96.9   3.12   0.16   134.21   9944   85
  AODV       7    80     20    2.56       100    95.6   3.36   0.17   133.94   10701 105
  AODV       8    80     20    2.56       100    97.3   2.34   0.19   135.93   7575   57
  AODV       9    80     20    2.56       100    94.5   3.65   0.31   132.88   11522 127
  AODV      10    80     20    2.56       100    98.1   3.03   0.12   137.58   9933   29
  ---------------------------------------------------------------------------------------
  MDVZRP     1    80     20    2.56       100    95.6   0.28   0.11   132.96   890    56
  MDVZRP     2    80     20    2.56       100    96.3   0.32   0.05   134.69   1011   32
  MDVZRP     3    80     20    2.56       100    98.5   0.38   0.09   137.21   1248   9
  MDVZRP     4    80     20    2.56       100    96.0   0.28   0.08   133.90   876    43
  MDVZRP     5    80     20    2.56       100    97.4   0.28   0.05   136.66   901    14
  MDVZRP     6    80     20    2.56       100    98.4   0.30   0.04   137.58   989    14
  MDVZRP     7    80     20    2.56       100    92.9   0.34   0.08   129.60   1047   104
  MDVZRP     8    80     20    2.56       100    98.7   0.30   0.05   138.39   990    33
  MDVZRP     9    80     20    2.56       100    95.2   0.36   0.06   132.34   1132   34
  MDVZRP    10    80     20    2.56       100    95.9   0.36   0.04   133.17   1127   38
  ---------------------------------------------------------------------------------------
```

```
=========================================================================================
         Senario  Pause Speed  StTime     SpTime   PDF    NRL    EED    [kbps]   O/H    Drop
  DSDV         1    100   20    2.56        100    99.7   0.54   0.02   139.10   1802   8
  DSDV         2    100   20    2.56        100    98.9   0.51   0.02   138.56   1674   36
  DSDV         3    100   20    2.56        100    97.2   0.52   0.02   135.23   1678   88
  DSDV         4    100   20    2.56        100    99.6   0.52   0.02   139.60   1738   12
  DSDV         5    100   20    2.56        100    99.1   0.53   0.02   138.17   1749   30
  DSDV         6    100   20    2.56        100    99.9   0.48   0.02   139.64   1610   2
  DSDV         7    100   20    2.56        100    99.8   0.52   0.02   138.88   1718   6
  DSDV         8    100   20    2.56        100    98.9   0.52   0.02   139.43   1737   33
  DSDV         9    100   20    2.56        100    96.3   0.50   0.02   135.19   1623   121
  DSDV        10    100   20    2.56        100    98.3   0.54   0.01   137.80   1773   58
  ---------------------------------------------------------------------------------------
  AODV         1    100   20    2.56        100    97.9   1.65   0.16   135.29   5311   65
  AODV         2    100   20    2.56        100    97.4   1.82   0.27   135.52   5877   39
  AODV         3    100   20    2.56        100    99.4   1.84   0.12   139.40   6118   5
  AODV         4    100   20    2.56        100    99.7   1.55   0.09   139.35   5125   9
  AODV         5    100   20    2.56        100    99.6   1.38   0.03   139.60   4590   12
  AODV         6    100   20    2.56        100    98.2   1.89   0.19   137.75   6208   46
  AODV         7    100   20    2.56        100    99.7   1.58   0.10   139.27   5218   2
  AODV         8    100   20    2.56        100    98.1   1.79   0.20   137.43   5866   51
  AODV         9    100   20    2.56        100    99.8   1.34   0.06   139.86   4467   2
  AODV        10    100   20    2.56        100    99.8   1.48   0.08   139.14   4909   3
  ---------------------------------------------------------------------------------------
  MDVZRP       1    100   20    2.56        100    100.0   0.29  0.02   139.57   964    1
  MDVZRP       2    100   20    2.56        100    99.8   0.29   0.02   138.34   947    7
  MDVZRP       3    100   20    2.56        100    99.8   0.25   0.03   139.50   845    5
  MDVZRP       4    100   20    2.56        100    99.9   0.24   0.02   138.60   805    2
  MDVZRP       5    100   20    2.56        100    99.7   0.29   0.04   139.52   964    9
  MDVZRP       6    100   20    2.56        100    99.9   0.32   0.02   139.11   1065   2
  MDVZRP       7    100   20    2.56        100    99.8   0.31   0.03   138.67   1014   7
  MDVZRP       8    100   20    2.56        100    99.9   0.31   0.02   138.51   1036   2
  MDVZRP       9    100   20    2.56        100    99.7   0.34   0.03   139.56   1143   9
  MDVZRP      10    100   20    2.56        100    99.8   0.24   0.03   139.31   801    3
  ---------------------------------------------------------------------------------------
```

APPENDIX E

Publications Authored

[1] **Skloul, I. Ib**., King, P. J. B., and Pooley, R. J., “MANETs form Zones to Threshold”, The Fifth International Conference on Systems and Networks Communications (ICSNC 2010), IEEE Computer society, Nice, France, 22-27 August 2010,.

[2] **Skloul, I. Ib**., King, P. J. B., and Pooley, R. J., “Performance Evaluation of Routing Protocols for MANET”, The Fourth International Conference on Systems and Networks Communications (ICSNC 2009), IEEE Computer society,pp.,105-112, Porto, Portugal, 21-26 September 2009,

[3] **Skloul, I. Ib**, King, P. J. B., and Pooley, R. J., “Performance Comparison of CBR in MDVZRP with DSDV and AODV Multipath”, 25th UK Performance Engineering workshop, modelling and analysis of computer and telecommunication systems, pp., Available from: http://www.comp.leeds.ac.uk/ukpew09/papers/09.pdf, Leeds University 6–7 July 2009.

[4] **Skloul, I. Ib**, Etorban, A., and King, P. J. B., “Multipath Distance Vector Zone Routing Protocol for Asymmetric Mobile Ad-Hoc Networks”, 24th UK Performance Engineering workshop, modelling and analysis of computer and telecommunication systems, UKPEW 2008, pp.,271-284, http://ukpew.org/,Imperial College London, 3–4 July 2008.

[5] **Skloul, I. Ib**, Etorban, A., and King, P. J. B., “DVZRP Protocol for Symmtric Mobile Ad-Hoc Networks”, 9th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting,pp.,171-176, sponsored by the EPSOURCE (SRC), Liverpool John Moores University, 23-24 June 2008,

[6] **Skloul, I. Ib**, and King, P. J. B., “Wireless Networks Design and Issues”, PGR, Research in Information Communication Modelling, Machine and System Communication Session, ,pp.,22-23,Heriot Watt University,12 Jun 2008.

[7] King, P. J. B., Etorban, A., and **Skloul, I. Ib**., ”DSDV-based Multipath Routing Protocol for Mobile Ad-Hoc Networks”, 8th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting,pp.,93-98, sponsored by the EPSOURCE (SRC), Liverpool John Moores University, 28 - 29 June 2007.

APPENDIX F

Bibliography

Abolhasan, M., Wysocki, T., and Dutkiewicz, E., (2004), “A review of routing protocols for mobile ad hoc networks”, Ad Hoc Networks, vol. 2 (1), pp., 1–22.

Agarwal, S. and Ahuja, A., (2000), "Route-lifetime assessment based routing (RABR) protocol for mobile ad-hoc networks", IEEE, vol., (3), pp.,1697-1701, New Orleans.

Ai Hua Ho, Yao Hua Ho and Kien A. Hua, (2009), "Handling high mobility in next-generation wireless ad hoc networks", International Journal of Communication Systems, (2009), Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/dac.1056.

Bahk, S. and El-Zarki, W., (1992), "Dynamic Multi-path Routing and how it Compares with other Dynamic Routing Algorithms for High Speed Wide-area Networks", Proceeding of the ACM SIGCOM, 1992,pp.,53-64.

Barnett, B. L., (1993), "An Ethernet performance simulator for undergraduate networking", Twenty Forth SIGCSE Technical Symposium on Computer Education, ACM New York, NY, USA, vol., 25 (1), Miller, March 1993, pp., 145-150.

Basagni, S. and Chlamtac, I., (1998), "A distance routing effect algorithm for mobility" (DREAM), Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking Dallas, Texas, United States, pp., 76 - 84, ISBN:1-58113-035-X.

Bettsttter, C., Resta, G., and Santi, P., (2003), "The node distribution of the random waypoint mobility model for wireless ad hoc networks", IEEE, Transactions on Mobile Computing, vol.,(2), pp., 257–269, July–September 2003.

Black, F., (2009), “ Wireless Network Simulation Using Opnet“, Acadimic Edition, Part 1, Available from: <http://4blackfire.blogspot.com/2009_10_01_archive.html>, accessed on [April, 2011].

Bluetooth, (2007), "Official website", Available from: <http://www.bluetooth.com/Pages/Bluetooth-Home.aspx>, [Accessed April 201].

Blum, J. J. and Eskandarian, A., (2004), "Challenges of intervehicle ad hoc networks", IEEE Intelligent Transportation Systems Society , vol., 5 (4), Dec., 2004, pp., 347 - 351, Digital Object Identifier 10.1109/TITS.2004.838218.

Broch, J. and Maltz, D. A., (1998), "A performance comparison of multi-hop wireless ad hoc network routing protocols", Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking, Dallas, Texas, United States, pp., 85 - 97, ACM Press New York, NY, USA, ISBN:1-58113-035-X.

Boukerche, A., (2009), "Algorithms and Protocols For Wireless and Mobile Ad hoc Networks",University of Ottawa Ottawa, Canada, Published by John Wiley & Sons, Inc., Hoboken, New Jersey, ISBN 978-0-470-38358-2 .

Camp, T. and Boleng, J., (2002), "A survey of mobility models for ad hoc network research, Wireless Communications & Mobile Computing", (WCMC), Special issue on

Mobile Ad Hoc Networking, Research, Trends and Applications, vol., 2 (5), (2002), pp., 483-502.

Chang, X. (1999), "Network simulations with OPNET", Proceedings of the 31st conference on Winter simulation, Simulation a bridge to the future, pp., 307-314, December 05-08, 1999, Phoenix, Arizona, United States .

Chaudet, C., Dhoutaut, D. and Lassous, G., (2005), "Performance issues with IEEE 802.11 in ad hoc networking, IEEE Communication Magazine, vol. 43, no. 7, pp.110-116.

Clausen, T. and Jacquet, P., (2003), "Optimized Link State Routing Protocol", (OLSR), RFC Editor, United States, RFC3626. Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century. Proceedings. IEEE International,DigitalObject Identifier: 10.1109/INMIC.2001.995315,pp., 62 - 68, 2001.

Dhar, S. (2005), "MANET, Applications, Issues and Challenges", the Future, International Journal of Business Data Communications and Networking, vol., 1 (2), April - June 2005, pp.,66-92.

Dube, R., Rais, C., and Tripathi, S., (1997), "Signal Stability-Based Adaptive Routing for Ad-Hoc Mobile Networks", IEEE, Personal Communications Magazine, vol 5, no. 1, pp. 36-45.

Dublin-Research and Markets, (2010),"The Taiwanese WLAN Industry, 2Q 2009, 1Q 2010",< http://www.researchandmarkets.com/research/e41731/the_taiwanese_wlan>.

Egli, J., (1957), "Radio Propagation above 40 MC over Irregular Terrain", Proceedings of the IRE (IEEE) 45 (10), pp.,1383–1391.

Elsallabi, H., Vainikainen, M. and Vuokko, L. (2007), "An Additive Model as a Physical Basis for Shadow Fading", Vehicular Technology, IEEE Transactions on pp. 13 - 26 , 1(56), Jan. 2007.

Fairhurst, G., (2005), "Address Resolution Protocol", (ARP), Available from: <http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/arp.html>,Last Update, 1-Dec. -2005, Accessed [May 2009].

Friisø, T. B., Haaland, T., and Radziwill, P., (2003), "Security Challenges in Self-organizing Wireless", Telenor R&D R 17/2003, Project No TFPFAN, Program Peer-to-peer computing Security, no. pp., 31. , 9-Aug-2003, ISBN 82-423-0581-1.

Ganapathi, G. J., (2008), "Reference Point Group Mobility and Random Waypoint Models in Performance Evaluation of MANET Routing Protocols", Journal of Computer Systems, Networks, and Communications, vol., 2008, Article ID 860364, 10 pages,doi:10.1155/2008/860364 Research Article.

Gloss, B., Scharf, M., and Neubauer, D., (2005), "A More Realistic Random Direction Mobility Model", in Proceeding of COST 290 4th Management Committee Meeting, Wi-QoST, Germany, 2005.

Goldsmith, A., (2005), "Ad-Hoc Wireless Networks", Routing Wireless communications, Chapter 16, pp., 535-553, Cambridge University Press,ISBN-13: 9780521837163.

GSM Association, (2010)," Device Field and Lab Test Guidelines" . Available from: < http://www.gsmworld.com/documents/TS11_10_0.pdf>, 16 December 2010.

Green, E. R., (2004), "System architectures for high-rate ultra-wideband communication systems", A review of recent developments, Intel Corporation, Hillsbro, OR: Intel Labs, vol., (6), pp.,241.

Haas, J., (1997), "A new routing protocol for the reconfigurable wireless network", In Proceedings of the 1997, IEEE 6th International Conference on Universal Personal Communications, ICUPC'97,San Diego, CA, 12-16 October 1997, pp., 562-566.

Haas, Z. J., Pearlman, R., and Samar, P., (2001), "Interzone Routing Protocol "(IERP), Wireless Networks Laboratory, School of Electrical Engineering, Cornell University, Ithaca, NY 14853,United States of America, Available from: <http://tools.ietf.org/html/draft-ietf-manet-zone-ierp-01#page-i>, IETF, Internet Draft, draft-ietf-manet-ierp-01.txt.

Haas, Z. J., Pearlman, R., and Samar, P., (2001), "Intrazone Routing Protocol (IARP)", Wireless Networks Laboratory, School of Electrical Engineering, Cornell University, Ithaca, NY 14853,United States of America, Available from: http://tools.ietf.org/html/draft-ietf-manet-zone-iarp-01, IETF, Internet Draft, draft-ietf-manet-iarp-01.txt.

Haseeb, Z., David, H. and Ivan, A., (2007) ”On-demand Partial-Disjoint Multipath Routing with Low Density for Mobile Ad-Hoc Networks”, 8th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting,pp.,233-236, Liverpool John Moores University.

Hoang, L., Nguyen, U. and Trang, N., (2008), “A study of different types of attacks on multicast in mobile ad hoc networks”, Ad Hoc Networks, Volume 6, Issue 1, Pages 32-46, January 2008.

Hong, X., Pei, G., and Mario, G., (2000), "Landmark Routing for Large Scale Wireless Ad Hoc Networks with Group Mobility", Proceedings of IEEE/ACM MobiHOC 2000, Boston, MA, Aug. 2000.

Hyytia, E., Koskinen, H., and Virtamo, J., (2005), "Random Waypoint Model in Wireless Networks", In Proceedings of Networks and Algorithms, complexity in Physics and Computer Science, Helsinki, Finland, pp., 132-156 ACM Press, 2005.

Itoua, S. M., (2008), "Effect of Propagation Models on Ad Hoc Networks Routing Protocols", Sensor Technologies and Applications, SENSORCOMM 08, Second International Conference, pp., 752-757, ISBN: 978-0-7695-3330-8.

Jacquet, P., Mühlethaler, P., and Qayyum, A., (1998)," Optimized Link State Routing Protocol", IETF MANET Working Group, Published Online, Available from:< http://www.ietf.org/proceedings/98dec/I-D/draft-ietf-manet-olsr-00.txt>.

Jayakumar, G. and Gopinath, G., (2007), "Ad Hoc Mobile Wireless Networks Routing Protocols", A review, vol., (3), pp., 574-582, Journal of Computer Science, Science Publications, ISSN 1549-3636.

Jianfeng, M., Changguang, W. and Zhuo, M., (2009), “Security Access in Wireless Local Area Networks From Architecture and Protocols to Realization”, Springer ,ISBN 978-3-642-00940-2.

Johansson, P., Larsson, T. and Hedman, N., (1999), "Scenario-based performance analysis of routing protocols for mobile ad-hoc networks", Proceedings of the 5th annual ACM/IEEE, international conference on Mobile computing and networking, Seattle, Washington, United States, pp., 195 - 206, ISBN:1-58113-142-9.

Johnson, D. B., Hu, J., and Jetcheva, Y. C., (1999), "The CMU Monarch Project’s Wireless and Mobility Extensions to ns", Computer Science Department Carnegie Mellon University, August 1999, Available from: <http://www.monarch.cs.cmu.edu/>.

Khengar, P., (2003), "Design and Performance Evaluation of a New Routing Protocol for Mobile Ad Hoc Networks", Electrical and Electronic Engineering, Kings College London, University of London, PhD., Thesis.

Ko, Y. and Vaiday, N., (1998), "Location Aided Routing (LAR) in Mobile Ad Hoc Networks", Mobile Computing and Networking, MOBICOM'98, October 25-30, 1998, Dallas, Texas, USA, PP. 66-75.

Kuosmanen, P., (2002), "Classification of ad hoc routing protocols", Finnish Defence Forces, Naval Academy, Finland, Available from: <http://keskus.hut.fi/opetus/s38030/k02/Papers/12-Petteri.pdf>, Accessed [May 2009].

Kurkowski, S. C., Mushell, T., and Colagrosso, M., (2005), "A visualization and analysis tool for NS-2 wireless simulations", iNSpect, Proceedings of the 13th IEEE, International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp., 503-506.

Larsson, T. and Hedman, N., (1998), "Routing protocols in wireless ad-hoc networks", a simulation study, Computer Science and Electrical Engineering, Computer Communication, Luleå Tekniska University, 1998-12-18, ISSN 1402-1617,ISRN LTU-EX--98,362--SE, NR 1998:362, Master's Thesis.

Lee, S. J. and Gerla, M., (2001), "Split multipath routing with maximally disjoint paths in ad hoc networks", In Proceedings of IEEE, ICC 2001, Helsinki, Finland, June 2001, pp., 3201-3205.

Li, M., Zhang, L., and Li, V., (2005), "An Energy-Aware Multipath Routing Protocol for Mobile Ad Hoc Networks", Proceeding of ACM SIGCOMM Asia Workshop, April 2005, Beijing, China, pp.,166–174.

Luis, B., Rodolfo, O. and Paulo, P., (2008), “A Telephony Application for Manets: Voice over a MANET-Extended JXTA Virtual Overlay Network”, E-Business and Telecommunication Networks Communications in Computer and Information Science, 2008, Volume 9, IV, 347-358, DOI: 10.1007/978-3-540-70760-8_28 .

Maltz, D. A. and Johnson, D. B., (1996), "Dynamic Source Routing in Ad Hoc Wireless Networks", Mobile Computing by Tomasz Imielinski and Hank Korth, chapter 5, pp., 153-181, Kluwer Academic Publishers.

Marina, M. K., Perkins, C. E., Royer, E. M., and Das, S. R., (2001), "Performance comparison of two on-demand routing protocols for ad hoc networks", IEEE, Personal Communications, Feb., 2001, vol., 8(1), pp., 16-28.

Martin, S., (2011), "From GSM to LTE: An Introduction to Mobile Networks and Mobile Broadband", John Wiley & Sons, Ltd. Published 2011, ISBN: 0470667117 / 0-470-66711-7.

Michel, B. and Evangelos, K., (2007), “Principles of Ad Hoc Networking”, Carleton University, Canada, John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England, ISBN: 978-0-470-03290-9.

Mueller, S. T. and Ghosal, R. P., (2004), "Multipath routing in mobile ad hoc networks, Issues and challenges", Springer, 2965 Performance Tools and Applications to Networked Systems, pp., 209-234.

Murthy, S. and Garcia-Luna-Aceves, J. J., (1995), "Wireless Routing Protocol", A Routing Protocol for Packet Radio Networks, Proceeding of ACM International Conference on Mobile Computing and Networking, pp., 86-95, Nov., 1995, Springer.

Murthy, S. and Manoj, B.,(2004), "Ad Hoc Wireless Networks", Prentice Hall International, ISBN 0-13-147023-X,2004.

Nandi, S. and Sarma, N., (2007), "Route Stability based QoS Routing (RSQR) in MANETs," Proc of the 10 th International Symposium on Wireless Personal Multimedia Communications (WPMC-2007), pp. 770-3, Dec. 2007.

Nasipuri, A. and Das, S. R., (1999), "On-Demand Multi-path Routing for Mobile Ad Hoc Networks", Proceedings of IEEE ICCCN'99, Boston, MA, pp.,64-70,Oct. 1999.

Nasui, D., Oana, F., Sgarciu, V. and Oprea, B., (2010)," Vehicular Networks in a Computerized City Using Safe Mobile", The Fifth International Conference on Systems and Networks Communications (ICSNC 2010), France, Nice, IEEE Computer Society Publications, pp.,105-110. August 22-27, 2010.

Natarajan, M., (2010), “On the Time between Successive Multi-path Discoveries and Hop Count Per Multi-path for Zone-Disjoint Routing in Mobile Ad Hoc Networks”,Recent Trends in Wireless and Mobile Networks ,Communications in Computer and Information Science, Ankara, Turkey, 2010, Volume 84, Part 1, 254-265, DOI: 10.1007/978-3-642-14171-3_21, Springer.

Nikos, R. M., (2010), "Explanation of new trace format", Available from: <http://www.isi.edu/nsnam/ns/doc/node186.html> , Last Update 2009-01-06, Acessed[April 2010].

Nikos, R. M., (2010), "Packet Headers and Formats", Available from: <http://www.isi.edu/nsnam/ns/doc/node216.html>, Last update 2009-01-06, Acessed[April 2010].

Nikos, R. M., (2010), "Radio Propagation Models", Available from: <http://www.isi.edu/nsnam/ns/doc/node216.html>, Last Update 2009-01-06, Acessed[April 2010].

Nishu, G. and Mahapatra, R., (2009), “MANET Security Issues”, International Journal of Computer Science and Network Security, Volume.9, No.8, 2009.

NIST, (2001), "Wireless ad hoc networks", Retrieved April 28, 2009, Available from: <http://w3.antd.nist.gov/wahn_bkgnd.shtml>, Accessed [May 2009], Last update (June 04, 2008).

OMNeT++, (2009), "OMNeT++ Home Page", Available from: <http://www.omnetpp.org> ", Accessed [Jan 2009].

OPNET, (2007), "OPNET Technologies Inc.", Available from: <http://www.opnet.com/>, Accessed on [May, 2009], Retrieved May 15, 2009.

Osman, H. and Taylor, H.(2008), "Towards a reference model for m-commerce over ad hoc wireless networks", Proc. E-Activity and Leading Technologies (E-ALT) Conference, 2008, pp. 223-232.

Osman, H. and Taylor, H.(2011), " Identity Support in a Security and Trust Service for Ad Hoc M-Commerce Trading Systems”, Proceedings of 7th. International Symposium on Frontiers in Networking with Applications, Biopolis, Singapore, pp 285-290, IEEE.
OTcl, (1996), "OTcl-Object Tcl Extensions", Available from: <http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl/>, Accessed[Oct 2008].

Pearlman, M. R., Haas, Z. J., and Zygmunt, J., (1999), "Determining the Optimal Configuration for the Zone Routing Protocol", IEEE Journal on Selected Areas in Communications, vol., 17(8), pp., 1395-1414, August 1999.

Pearlman, M. R., Haas, Z. J., Sholander, P., and Tabrizi, S. S., (2000), "On the Impact of Alternate Path Routing for Load Balancing in Mobile Ad Hoc Networks", in Proceeding of IEEE MobiHoc 2000, Boston, MA, pp., 3-10, Aug. 2000,

Pearlman, M. R. and Haas, Z. J., (2000), "Alternate path routing in mobile ad hoc networks", in the Proceedings of IEEE MILCOM 2000, Los Angeles, CA, pp.,501-506,Oct. 2000,

Pei, G., Mario, G. and Tsu-Wei, C., (2000), “Fisheye State Routing in Mobile Ad Hoc Networks”,Proceedings of ICDCS Workshop on Wireless Networks and Mobile Computing, April 2000, Taipei, Taiwan,pp.D71-D78,available from http://www.cs.ucla.edu/NRL/wireless/PAPER/pei-wnmc00.ps.gz.

Perkins, C., (1994), "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers", ACM SIGCOMM, Computer Communication Review, vol., 24(4), pp., 234-244.

Perkins, C. and Royer, E., (2001), "Ad Hoc Networking", An Introduction, chapter 1, pp., 1-28, Addison-Wesley: ISBN 0-201-30976-9.

Perkins, C. and Royer, E., (2001), "Ad hoc Networking", The Ad Hoc On-Demand Distance-Vector Protocol, chapter 6, pp., 173-219, Addison-Wesley: ISBN 0-201-30976-9.

Ramanarayana, K. and Lillykutty, J., (2007), “Secure Routing in Integrated Mobile Ad hoc Network (MANET)-Internet”. Third International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing, Pages 19-24, 2007.

Redi, R. A., (2002), "A Brief Overview of Ad Hoc Networks Challenges and Directions", IEEE Communications Magazine, 50th Anniversary Commemorative Issue, May 2002, vol., 40 (5), pp., 20-22.

Rice, L., Longley, A., Norton, G. and Barsis, A., (1967), "Transmission loss predictions for tropospheric communication circuits," U.S. Government Printing Office, Washington, DC, NBS Tech. Note 101, issued May 1965; revised May 1966 and Jan. 1967Simulation Overview.

Robbins, A., (2001), "Effective awk programming", Available from: <http://goanna.cs.rmit.edu.au/~milad/gawk.pdf>, last update February 1997, Accessed[May 2009].

Rodig, Utz; Sreenan, Cormac J. (Eds.),(2009),"Wireless Sensor Networks", 6th European Conference, EWSN 2009 Cork, Ireland, February 11-13, 2009, Proceedings Series: Lecture Notes in Computer Science, Vol. 5432, , pp. 375, Subseries: Computer Communication Networks and Telecommunications.

Roy, R., (2011),"Handbook of Mobile Ad Hoc Networks for Mobility Models", DOI 10.1007/978-1-4419-6050-4_1, Springer Science+Business Media, LLC 2011.

Royer, E. M. and Toh, C-K., (1999), "A Review of Current Routing Protocols for Ad Hoc Wireless Networks", IEEE Personal Communication, vol.(6), pp., 46-55. April 1999.

Salleh, A. U. I., Din, Z., and Jamaludin, N. M., (2006), "Trace Analyzer for NS-2", 4th Student Conference on Research and Development, (SCOReD 2006).

Samir, R. D., Robert, C., Jiangtao, Y., and Rimli, S., (1998), "Comparative Performance Evaluation of Routing Protocols for Mobile Ad hoc Networks", pp.,153, Seventh International Conference on Computer Communications and Networks (ICCCN '98), 1998.

Seybold, S., (2005), "Introduction to RF propagation", John Wiley and Sons. pp. 144–146, ISBN: 978-0-471-65596-1.

Sharvani, G. S., Cauvery, N. K. and Rangaswamy, T. , (2009), “ADAPTIVE ROUTING ALGORITHM FOR MANET”, International Journal of Next-Generation Networks (IJNGN),Vol.1, No.1, December 2009.

Sinha, P., Sivakumar, R. and Bharghavan, V., (1999)," CEDAR: Core extraction distributed ad hoc routing", Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM '99, New York, NY, USA,pp.,202-209.

Skloul, I. Ib, Etorban, A., and King, P. J. B., (2008), "Multipath Distance Vector Zone Routing Protocol for Asymmetric Mobile Ad-Hoc Networks", 24th UK Performance Engineering workshop, modelling and analysis of computer and telecommunication systems, UKPEW 2008, pp.,271-284, http://ukpew.org/,Imperial College London, 3–4 July 2008.

SSFNet, (1999), "Scalable Simulation Framework", Retrieved May 15, 2009, Available from: <http://www.ssfnet.org/homePage.html>, Accessed [May 2009].

Stefano, B., Irnrich, C. and Violet, R.,(1998), "A Distance Routing Effect Algorithm for Mobility (DREAM)", Proceeding MobiCom '98 Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking ACM New York, NY, USA.

Tachtatzis, C. and Harle, D., (2008), "Performance evaluation of multi-path and single-path routing protocols for mobile ad-hoc networks", Proc. 2008 Int. Symp. on Performance Evaluation of Comput. and Telecommun. Systems (SPECTS), Edinburgh, June 2008.

Taruna, S. and. Purohit, G.N, (2011), "Scenario Based Performance Analysis of AODV and DSDV in Mobile Adhoc Network", Advances in Networks and Communications First International Conference on Computer Science and Information Technology, CCSIT 2011, Bangalore, India, January 2-4, 2011. Proceedings, Part II, CCIS 132, pp. 10–19, 2011.

Thomas, H., Cormen, R., Rivest, L., and Clifford, S. I., (2001), "The Bellman-Ford algorithm", M. P. a. McGraw-Hill, pp.,588–592, Problem 24-1, pp.,614–615.

Toh, C. K., (1997), "Associativity-based Routing for Ad Hoc Mobile Networks", Wireless Personal Communications, vol., 4 (2), pp., 103-139.

Toh, C. K., (2001), "Ad Hoc Mobile Wireless Networks: Protocols and Systems", Publisher: Prentice Hall 2001, 336 Pages,ISBN: 0130078174.

Tsirigos, A. and Haas, Z. J., (2001), "Multipath Routing in Mobile Ad Hoc Networks ,or how to route in the presence of frequent topology changes", in Proceedings of IEEE, MILCOM 2001, McLean, VA, Oct. 2001, pp., 878-883.

Kevin, F. and Kannan, V., (2010), "The ns Manual", (formerly ns Notes and Documentation), vol.,(1), Avalable from: <www.isi.ede/nsnam/ns/ns-documentation.html>, last accessed [April 2010].

Varga, A. (2001), "The OMNeT++ Discrete Event Simulation System", in the Proceedings of the European Simulation Multiconference, ESM2001, June 6-9, 2001, Prague, Czech Republic.

Vetriselvi, V. and Parthasarathi, R., (2007), "Trace Based Mobility Model for Ad Hoc Networks", in Proceedings of the Third IEEE, International Conference on Wireless and Mobile Computing, Networking and Communications, WIMOB, IEEE Computer Society, Washington, DC, pp.81-81, 2007.

VINT, (1996), "Virtual InterNetwork Testbed", A Collaboration among USC/ISI, Xerox PARC, LBNL, and UCB, Retrieved May 2009, Available from: <http://www.isi.edu/nsnam/vint/index.html>, Accessed [19, May 2009].

Wiberg, B., (2002), "Porting AODV-UU Implementation to Ns-2 and Enabling Trace-based Simulation", Information Technology Department Systems, Uppsala, Sweden, Uppsala University, Master's Thesis.

Wu, K. and Harms, J., (2001), "Performance study of a multipath routing method for wireless mobile ad hoc networks", in Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Cincinnati, OH, Aug. 2001, pp., 99 -107.

Wu, K. and Harms, J., (2006), "Performance Study of Proactive Flow Handoff for Mobile Ad Hoc Networks", ACM/Kluwer Wireless Networks Journal (ACM WINET), vol., 12(1), February 2006, pp., 119 - 135, ISSN:1022-0038.

Xiang, Li., (2008), "Wireless Ad Hoc and Sensor Networks: Theory and Applications", ISBN-13: 9780521865234, Pub. Date: June 2008, Cambridge University Press.

Yang, S. and John, S. B., (2002), "TORA, Correctness, Proofs and Model Checking", Available as Technical Report from the Institute for Systems Research, Electrical and Computer Engineering Department, University of Maryland, December 2002, Master's Thesis.

Yang, Guang-Zhong,(2006), "Body Sensor Networks", ISBN 978-1-84628-272-0.

Young-Bae, K. and Vaidya, N. H., (2000), "Location-Aided Routing (LAR) in mobile Ad Hoc Networks", Kluwer Academic Publishers, vol., 6 (4), pp., 307-321 ISSN:1022-0038 . Hingham, MA, USA .

Yousefi, S. M. and Fathy, M. S, (2006), "Vehicular Ad Hoc Networks (VANETs), Challenges and Perspectives", ITS, Telecommunications Proceedings, 6th International Conference on Publication, June 2006.

Zafar, H., Harle, D. and Ivan, A., (2007), "On-demand Partial-Disjoint Multipath Routing with Low Delay for Wireless Ad-hoc Networks",PG Net Preceedings, 8th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting,pp.,233-236, 28 - 29 June 2007.

Zafar, H., Harle, D., and Aonovic, I. and Khawaja, Y., (2009), "Performance evaluation of shortest multipath source routing scheme", Institution of Engineering and Technology Communications, IET, May 2009, 3(5),pp., 700 - 713 ,ISSN: 1751-8628.