

Jaypee University of Information Technology
Waknaghat, Distt. Solan (H.P.)

Learning Resource Center

CLASS NUM:

BOOK NUM.:

ACCESSION NO.: ~~SP08076~~ SP0812078

This book was issued is overdue due on the date stamped below. If the book is kept over due, a fine will be charged as per the library rules.

Due Date	Due Date	Due Date
<div data-bbox="502 593 758 840"><p>LRC Learning Resource Center 30 MAR 2013 CHECK OUT Waknaghat, Solan (H.P.)</p></div>		

Performance Analysis of DSDV and AODV Routing Algorithms in MANETs

Project Report submitted in partial fulfillment of the requirement for the
degree of

Bachelor of Technology.

in

Electronics and Communication Engineering

under the Supervision of

Dr. Davinder Singh Saini

By

SANCHITA AGARWAL (081122)

SURABHI BALI (081123)

NITESH KUMAR (081130)

to



Jaypee University of Information and Technology


Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled "Performance Analysis of DSDV and AODV Routing Algorithms in MANETs", submitted by **Sanchita Agarwal (081122)**, **Surabhi Bali (081123)** & **Nitesh Kumar (081130)** in partial fulfillment for the award of degree of Bachelor of Technology in Electronics and Communication Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date: 01-06-2012

Supervisor's Name  Dr. D.S. Saini

Designation Associate Professor

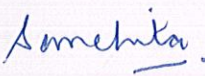
ACKNOWLEDGEMENT

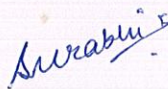
This project has been an outcome of sustained and continual efforts on part of every group member. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project.

We are deeply indebted to our project guide **Dr. Davinder Singh Saini** (ASSOCIATE PROFESSOR) **Electronics and Communication Department**) for his help, stimulating suggestions and encouragement which helped throughout the project. For his coherent guidance throughout the tenure of the project, we feel fortunate to be taught by him, who gave us his unwavering support.

Sincere thanks to senior lab technicians **Mr. Mohan** and **Mr. Kamlesh** for extensive cooperation.

Date: 01-06-2012


Sanchita Agarwal
081122


Surabhi Bali
081123

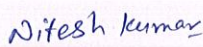

Nitesh Kumar
081130

TABLE OF CONTENTS

Topics	Page No.
ABSTRACT	
CHAPTER 1	
INTRODUCTION	1
1.1 Decentralized approach or Infrastructure less (ad-hoc) Networks	1
1.2 Routing Protocols	1
1.3 The Protocol Stack	2
1.3.1 Interworking	3
1.4 Proactive, Reactive and Hybrid Routing Protocol	4
 CHAPTER 2	
DESTINATION SEQUENCED DISTANCE VECTOR (DSDV)	6
2.1 Protocol Overview	6
2.2 Route Advertisements	6
2.3 Route Table Entry Structure	7
2.4 Responding to Topology Changes	7
2.5 Route Selection Criteria	8
 CHAPTER 3	
AD- HOC ON-DEMAND DISTANCE-VECTOR (AODV)	11
3.1 Protocol Overview	11
3.2 Unicast Route Establishment	12
3.3 Route Discovery	12
3.4 Forward Path Setup	13
3.5 Route Maintenance	14
3.6 MULTICAST GROUP ESTABLISHMENT	15
3.6.1 Route Discovery	15
3.6.2 Forward Path Setup	16
3.6.3 Leaving the Group	17
3.6.4 Multicast Tree Maintenance	17
3.6.5 Broadcast	18

CHAPTER 4

NETWORK SIMULATOR

19

4.1 Overview

19

4.2 Simple Simulation Example

22

4.3 Post Simulation

25

4.4 Trace Analysis Example

28

4.5 AWK Command

29

CHAPTER 5

SIMULATION GRAPHS

32

5.1 Scenario

32

CHAPTER 6

SIMULATION GRAPHS

33

6.1 Scenario 1

33

6.1.1 Without mobility

33

6.1.2 With mobility

37

6.2 Scenario 2

46

CHAPTER 7

CONCLUSIONS and FUTURE WORK

48

APPENDIX

50

REFERENCES

56

LIST OF ABBREVIATIONS

MANETS: Mobile Ad-hoc Networks
AODV: Ad-hoc On Demand Distance Vector Routing
DSDV: Destination Sequenced Distance Vector Routing
TORA: Temporally Ordered Routing Algorithm
PDR: Packet Delivery Ratio
NS: Network Simulator
NAM: Network Animator
OTcl: Object Tool Command Language
RREQ: Route Request
RERR: Route Error
RREP: Route Reply
FTP: File Transfer Protocol
TCP: Transmission Control Protocol
UDP: User Datagram Protocol
CBR: Constant Bit Rate
VBR: Variable Bit Rate
MAC: Medium Access Control
IP: Internet Protocol
ZRP: Zone Routing Protocol
OLSR: Optimized Link State Routing Protocol
m/s: meter per second
mbps: megabyte per second
ms: millisecond
sec: second
Kbyte: Kilobyte

LIST OF FIGURES

PAGE NO.

Figure 1.1: Classification of MANETs Routing Protocols	2
Figure 1.2: The OSI model, TCP/IP suite and MANET protocol stack	3
Figure 1.3: The protocol stacks used by mobile nodes, gateways and Internet nodes.	4
Figure 2.1: Example of DSDV in Operation	8
Figure 3.1: RREQ (Route Request) Broadcast Flood in AODV	13
Figure 3.2: RREP (Route Reply) Propagation in AODV	14
Figure 3.3: RERR (Route Error) Propagation in AODV	15
Figure 3.4: Route Discovery and Reply Generation for multicast group in AODV	16
Figure 3.5: Route Activation for multicast group in AODV	17
Figure 3.6: Pruning of Group Member in AODV	17
Figure 3.7: Multicast Tree after Prune in AODV	17
Figure 3.8: Link Breakage of multicast tree in AODV	18
Figure 3.9: Repaired Multicast Tree (AODV)	18
Figure 4.1: Simplified User's View of NS	19
Figure 4.2: C++ and OTcl: The Duality	21
Figure 4.3: Architectural View of NS	21
Figure 4.4: A Simple Network Topology and Simulation Scenario	22
Figure 4.5: Trace Format Example	29

LIST OF GRAPHS

PAGE NO.

Graph 6.1.1.1: PDR_20: Packet Delivery Ratio for 20 nodes, no mobility	33
Graph 6.1.1.2: PDR_50: Packet Delivery Ratio for 50 nodes, no mobility	34
Graph 6.1.1.3: Overhead_20: Overhead for 20 nodes, no mobility	34
Graph 6.1.1.4: Overhead_50: Overhead for 50 nodes, no mobility	35
Graph 6.1.1.5: E2ED_20: End to end delay for 20 nodes, no mobility	36
Graph 6.1.1.6: E2ED_50: End to end delay for 50 nodes, no mobility	36
Graph 6.1.2.1: PDR_20_1: Packet Delivery Ratio for 20 nodes, Mobility: 1m/s	37
Graph 6.1.2.2: PDR_20_5: Packet Delivery Ratio for 20 nodes, Mobility: 5m/s	37
Graph 6.1.2.3: PDR_20_10: Packet Delivery Ratio for 20 nodes, Mobility: 10m/s	38
Graph 6.1.2.4: PDR_50_1: Packet Delivery Ratio for 50 nodes, Mobility: 1m/s	38
Graph 6.1.2.5: PDR_50_5: Packet Delivery Ratio for 50 nodes, Mobility: 5m/s	39
Graph 6.1.2.6: PDR_50_10: Packet Delivery Ratio for 50 nodes, Mobility: 10m/s	39
Graph 6.1.2.7: Overhead_20_1: Overhead for 20 nodes, Mobility: 1m/s	40
Graph 6.1.2.8: Overhead_20_5: Overhead for 20 nodes, Mobility: 5m/s	40
Graph 6.1.2.9: Overhead_20_10: Overhead for 20 nodes, Mobility: 10m/s	41
Graph 6.1.2.10: Overhead_50_1: Overhead for 50 nodes, Mobility: 1m/s	41
Graph 6.1.2.11: Overhead_50_5: Overhead for 50 nodes, Mobility: 5m/s	42
Graph 6.1.2.12: Overhead_50_10: Overhead for 50 nodes, Mobility: 10m/s	42
Graph 6.1.2.13: E2ED_20_1: End to End Delay for 20 nodes, Mobility: 1m/s	43
Graph 6.1.2.14: E2ED_20_5: End to End Delay for 20 nodes, Mobility: 5m/s	43
Graph 6.1.2.15: E2ED_20_10: End to End Delay for 20 nodes, Mobility: 10m/s	44
Graph 6.1.2.16: E2ED_50_1: End to End Delay for 50 nodes, Mobility: 1m/s	44
Graph 6.1.2.17: E2ED_50_5: End to End Delay for 50 nodes, Mobility: 5m/s	45
Graph 6.1.2.18: E2ED_50_10: End to End Delay for 50 nodes, Mobility: 10m/s	45
Graph 6.2.1: PDR_DR1: Packet Delivery Ratio with data rate 1mbps	46
Graph 6.2.2: PDR_DR5: Packet Delivery Ratio with data rate 5mbps	47
Graph 6.2.1: PDR_DR10: Packet Delivery Ratio with data rate 10mbps	47

LIST OF TABLES

PAGE NO.

Table 2.1: Forwarding Table maintained at MH4

9

Table 2.2: New Routing Table Information

9

Table 5.1: Simulation Parameters

32

Abstract

Efficient routing protocols can provide significant benefits to mobile ad hoc networks in terms of both performances and reliability. Mobile Ad-hoc Network (MANET) is an infrastructure less and decentralized network which needs a robust dynamic routing protocol. Many routing protocols for such networks have been proposed so far. Amongst popular ones we have studied the following: Dynamic Source Routing (DSR), Ad-hoc On-demand Distance Vector (AODV), and Destination-Sequenced Distance Vector (DSDV) routing protocol. Nodes of these networks functions as routers which discovers and maintains the routes to other nodes in the network. In such networks, nodes are able to move and synchronize with their neighbors. Due to mobility, connections in the network can change dynamically and nodes can be added and removed at any time.

In our project, we have compared Mobile Ad-Hoc network routing protocols DSDV, AODV using network simulator NS2.34. We have compared the performance of two protocols together and individually. The performance matrix includes PDR (Packet Delivery Ratio), Average End to End Delay, Routing Overhead. We are comparing the performance of routing protocols in two scenarios. In the first one we have calculated PDR, Average End to End Delay, Routing Overhead in an area of $50 \times 50 \text{m}^2$ taking number of nodes 20 & 50, varying the mobility of node as 0, 1m/s, 5m/s and 10m/s for a simulation time of 50 seconds to 150 seconds. In the other scenario we measured the performance of PDR of AODV and DSDV in an area of $500 \times 500 \text{m}^2$ for 50 nodes changing the data rate as 1mbps, 5mbps, 10 mbps for a simulation time varying from 5minutes to 30 minutes.

CHAPTER 1

INTRODUCTION

A Mobile Ad-hoc Network (MANET) is a collection of wireless nodes that can dynamically be set up anywhere and anytime without using any pre-existing network infrastructure. It is an autonomous system in which mobile hosts connected by wireless links are free to move randomly and often act as routers at the same time. The topology of such networks is likely highly dynamic because each network node can freely move and no pre-installed base stations exist. Due to the limited wireless transmission range of each node, data packets then may be forwarded along multi-hops. Route construction should be done with a minimum of overhead and bandwidth consumption.

1.1 Decentralized approach or Infrastructure less (ad-hoc) Networks

In contrast to infrastructure based wireless network, in ad-hoc networks all nodes are mobile and can be connected dynamically in an arbitrary manner. A MANET is a collection of wireless mobile nodes forming a temporary network without using any existing infrastructure or any administrative support. The wireless ad-hoc networks are self-creating, self-organizing and self-administrating. The nodes in an ad-hoc network can be a laptop, cell phone, PDA (personal digital assistant) or any other device capable of communicating with those nodes located within its transmission range. The nodes can function as routers, which discover and maintain routes to other nodes. The ad-hoc network may be used in emergency search-and-rescue operations, battlefield operations and data acquisition in inhospitable terrain. In ad-hoc networks, dynamic routing protocol must be needed to keep the record of high degree of node mobility, which often changes the network topology dynamically and unpredictably.

1.2 Routing Protocols

The existing routing protocols in MANETs can be classified into three categories. Figure 1.1 shows the classification of existing MANETs protocols:

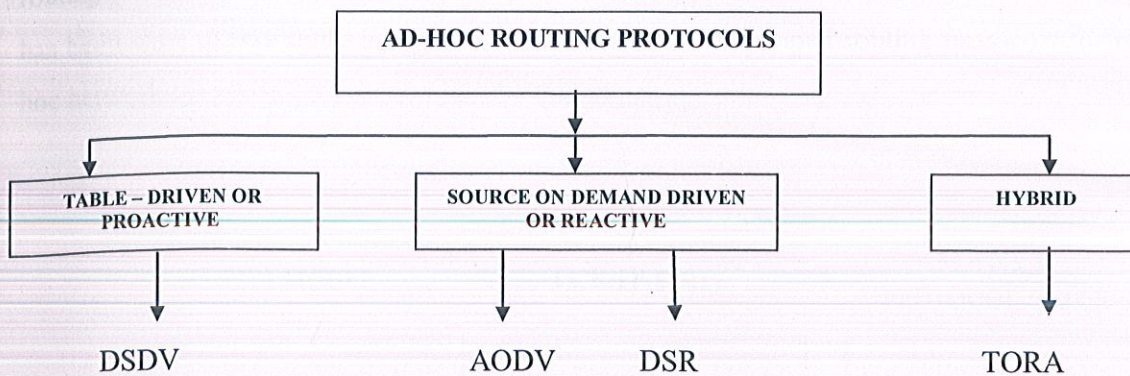


Figure 1.1: Classification of MANETs Routing Protocols

1.3 The Protocol Stack

In this section the protocol stack for mobile ad hoc networks is described. This gives a comprehensive picture of, and helps to better understand, mobile ad hoc networks. Figure 1.2 shows the protocol stack which consists of five layers: *physical layer*, *data link layer*, *network layer*, *transport layer* and *application layer*. It has similarities to the TCP/IP protocol suite. As can be seen the OSI layers for *session*, *presentation* and *application* are merged into one section, the application layer. On the left of Figure 1.2, the OSI model is shown. It is a layered framework for the design of network systems that allows for communication across all types of computer systems. In the middle of the figure, the TCP/IP suite is illustrated. Because it was designed before the OSI model, the layers in the TCP/IP suite do not correspond exactly to the OSI layers. The lower four layers are the same but the fifth layer in the TCP/IP suite (the application layer) is equivalent to the combined session, presentation and application layers of the OSI model. On the right, the MANET protocol stack - which is similar to the TCP/IP suite - is shown. The main difference between these two protocol stacks lies in the network layer. Mobile nodes (which are both hosts and routers) use an ad hoc routing protocol to route packets. In the physical and data link layer, mobile nodes run protocols that have been designed for wireless channels. Some options are the IEEE standard for wireless LANs, IEEE 802.11, the European ETSI standard for a high-speed wireless LAN, HIPERLAN 2, and finally an industry approach toward wireless personal area networks, i.e. wireless LANs at an even smaller range, Bluetooth. In the simulation tool used in this project, the standard IEEE 802.11 is used in these layers. This thesis focuses on ad hoc routing which is handled by the network layer. The network layer is divided into two parts: Network and Ad Hoc Routing.

The protocol used in the network part is Internet Protocol (IP) and the protocol used in the ad hoc routing part is Ad hoc On-Demand Distance Vector (AODV). One of the reasons to why AODV has been used in this study is that it is one of the most developed routing protocols for mobile ad hoc networks.

OSI MODEL	TCP/IP SUITE	MANET PROTOCOL STACK
APPLICATION		
PRESENTATION	APPLICATION	APPLICATION
SESSION		
TRANSPORT	TRANSPORT	TRANSPORT
NETWORK	NETWORK	NETWORK AD HOC ROUTING
DATA LINK	DATA LINK	DATA LINK
PHYSICAL	PHYSICAL	PHYSICAL

Figure 1.2: The OSI model, TCP/IP suite and MANET protocol stack

1.3.1 Interworking

Whenever a mobile node is to send packets to a fixed network, it must transmit the packets to a gateway. A gateway acts as a bridge between a MANET and the Internet. Therefore, it has to implement both the MANET protocol stack and the TCP/IP suite, as shown in the middle of Figure 1.3. Although the figures shows that all the layers are implemented for the gateway, it does not necessarily need all of the layers. The protocol stack used by the mobile node is the MANET protocol stack discussed previously and shown on the right of Figure 1.2. The fixed Internet node uses the TCP/IP suite. A gateway, that must be able to translate between these two “languages” ,must understand the both architectures.

MOBILE NODE

APPLICATION	
UDP	
IP	AODV
LLC 802.11 MAC	
802.11 PHY	

GATEWAY

APPLICATION		APPLICATION	
UDP		UDP	
IP	AODV	IP	
LLC 802.11 MAC		DATA LINK	
802.11 PHY		PHYSICAL	

INTERNET NODE

APPLICATION	
UDP	
IP	
DATA LINK	
PHYSICAL	

Figure 1.3: The protocol stacks used by mobile nodes, gateways and Internet nodes.

1.4 Proactive, Reactive and Hybrid Routing Protocols

Traditional distance-vector and link-state routing protocols are proactive in that they maintain routes to all nodes, including nodes to which no packets are sent. For that reason they require periodic control messages, which lead to scarce resources such as power and link bandwidth being used more frequently for control traffic as mobility increases. One example of a proactive routing protocol is Optimized Link State Routing Protocol (OLSR). OLSR, which has managed to reduce the utilization of bandwidth significantly. Reactive routing protocols, on the other hand, operate only when there is a need of communication between two nodes. This approach allows the nodes to focus either on routes that are being used or on routes that are in process of being set up. Examples of reactive routing protocols are Ad hoc On-Demand Distance Vector (AODV), and Dynamic Source Routing (DSR).

Both proactive and reactive routing have specific advantages and disadvantages that make them suitable for certain types of scenarios. Proactive routing protocols have their routing tables updated

at all times, thus the delay before sending a packet is minimal. However, routing tables that are always updated require periodic control messages that are flooded through the whole network - an operation that consumes a lot of time, bandwidth and energy. On the other hand, reactive routing protocols determine routes between nodes only when they are explicitly needed to route packets. However, whenever there is a need for sending a packet, the mobile node must first find the route if the route is not already known. This route discovery process may result in considerable delay. Combining the proactive and reactive approaches results in a hybrid routing protocol. A hybrid approach minimizes the disadvantages, but also the advantages of the two combined approaches. The Zone Routing Protocol (ZRP) is such a hybrid reactive/proactive routing protocol. Each mobile node proactively maintains routes within a local region (referred to as the routing zone). Mobile nodes residing outside the zone can be reached with reactive routing.

CHAPTER 2

DESTINATION SEQUENCED DISTANCE VECTOR (DSDV)

It is a table driven routing scheme developed for ad-hoc networks.

It was developed by C. Perkins and P. Bhagwat in 1994. The main contribution of the protocol was to solve the routing loop problem.

2.1 Protocol Overview

Packets are transmitted between the nodes of the network using route tables stored at each node. Each route table, at each of the nodes, lists all available destinations and the number of hops to each. Each route entry is tagged with a sequence number that is originated by the destination node. To maintain the consistency of route tables in a dynamically varying topology, each node periodically transmits updates, doing so immediately when new information is available. No assumption is made about the mobile nodes making any sort of time synchronisation and phase relationship of the update periods between the mobile hosts. These packets indicate which nodes are accessible from each node and the number of hops necessary to reach them, following the traditional distance-vector routing algorithms.

Routing information is advertised by broadcasting or multicasting the packets that are transmitted periodically and incrementally as topological changes are detected. Data is also kept about the length of time between the arrival of the first and the arrival of the best route for each particular destination. On the basis of this data, a decision may be made to delay advertising routes that are about to change, thus damping fluctuations of the route tables. The advertisement of possibly unstable routes is delayed to reduce the number of rebroadcasts of possible route entries that normally arrive with the same sequence number.

2.2 Route Advertisements

The DSDV protocol requires each mobile node to advertise, to each of its current neighbors, its own route table. The entries in the list may change fairly dynamically over time, so the advertisement

must be made often enough to ensure that every mobile computer can almost always locate every other mobile computer in the collection. In addition, each mobile computer agrees to relay data packets to other computers upon request. This helps to determine the shortest number of hops.

2.3 Route Table Entry Structure

The data broadcast by each mobile computer will contain its new sequence number and the following information for each new route:

- The destination's address
- The number of hops required to reach the destination
- The sequence number of the information received regarding that destination, as originally stamped by the destination

Within the headers of the packet, the transmitted route tables will also contain the hardware address and (if appropriate) the network address of the mobile computer transmitting them. The route tables will also include a sequence number created by the transmitter. Routes with more recent sequence numbers are always preferred as the basis for forwarding decisions, but they are not necessarily advertised. Of the paths with the same sequence number, those with the smallest metric will be used.

Routes received in the broadcasts are also advertised by the receiver when it subsequently broadcasts its routing information; the receiver adds an increment to the metric before advertising the route, as incoming packets will require one more hop to reach the destination.

2.4 Responding to Topology Changes

Mobile nodes cause broken links as they move from place to place. The broken link may be inferred if no broadcasts have been received for a while from a former neighbour. A broken link is described by a metric of infinity. When a link to a next hop has broken, any route through that next hop is immediately assigned an infinite metric and an updated sequence number. Such modified routes are immediately disclosed in a broadcast routing information packet. Sequence numbers generated to indicate infinite hops to a destination will be one greater than the last sequence number received from the destination. When a node receives an infinite metric and it has an equal or same sequence

number with a finite metric, it triggers a route update broadcast to disseminate the important news about that destination.

2.5 Route Selection Criteria

When a mobile node receives new routing information, that information is compared to the information already available from previous routing information packets. Any route with a more recent sequence number is used; routes with older sequence numbers are discarded. A route with a sequence number equal to an existing route is chosen if it has a better metric and the existing route is discarded or stored as less preferable. The metrics for routes chosen from the newly received broadcast information are each incremented by one hop. When a mobile node can determine that a route with a better metric is likely to show up then the advertisement of the new routes should be delayed. The route with later sequence number must be available for use, but it does not have to be advertised immediately unless it is a route to a previously unreachable destination. Thus, there will be two route tables at each node- one for use with forwarding packets and another to be advertised via incremental routing information packets. To determine the probability of imminent arrival of routing information showing a better metric, the mobile node has to keep a history of the weighted average time that routes fluctuate until the route with the best metric is received. Received routes with infinite metrics are not included in this computation of the settling time for route updates.

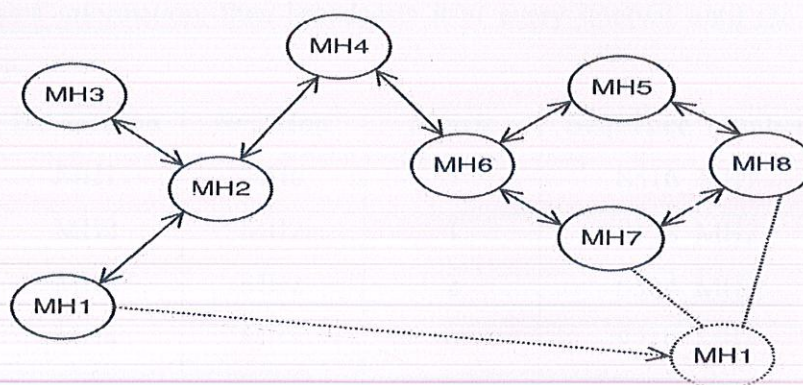


Figure 2.1: Example of DSDV in Operation

Consider MH4 in Figure 2.1. Table 2.1 shows a possible structure of the forwarding table maintained at MH4.

Destination	Next Hop	Metric	Sequence_number
MH1	MH2	2	S406_MH1
MH2	MH2	1	S128_MH2
MH3	MH2	2	S564_MH3
MH4	MH4	0	S710_MH4
MH5	MH6	2	S392_MH5
MH6	MH6	1	S076_MH6
MH7	MH6	2	S128_MH7
MH8	MH6	3	S050_MH8

Table 2.1: Forwarding Table maintained at MH4

The address of each mobile node is represented by MH_i and all sequence numbers are denoted by SNNN_MH_i where MH_i specified the computer that created the sequence number and SNNN is sequence number value.

Now suppose MH1 moves into the general vicinity of MH8 and MH7 and away from the others (especially MH2). The new internal forwarding table at MH4 might then appear as shown in table. Only the entry for MH1 shows a new metric. When MH1 moves into the vicinity of MH8 and MH7, it triggers an immediate incremental routing information update, which is then broadcast to MH6. MH6, having determined that significant new routing information has been received, also triggers an immediate update, which carries along the new routing information for MH1. MH4, upon receiving this information, then broadcasts it at every interval until the next new routing information dump.

Destination	NextHop	Metric	Sequence_number
MH1	MH6	3	S516_MH1
MH2	MH2	1	S128_MH2
MH3	MH2	2	S564_MH3
MH4	MH4	0	S710_MH4
MH5	MH6	2	S392_MH5
MH6	MH6	1	S076_MH6
MH7	MH6	2	S128_MH7
MH8	MH6	3	S050_MH8

Table 2.2: New Routing Information Table

In the incremental advertised routing table, the information for MH4 comes first since it is doing the advertisement. The information for MH1 comes next because it is the only one that has any significant route changes affecting it. As a general rule, routes with changed metrics are first included in each incremental packet. The remaining space is used to include those routes whose sequence numbers have changed.

CHAPTER 3

AD- HOC ON-DEMAND DISTANCE-VECTOR (AODV)

It provides quick and efficient route establishment between nodes desiring communication and was designed specifically for ad hoc wireless networks. Its goal is to reduce the need for system-wide broadcasts to the furthest extent possible as compared to DSDV which issues broadcasts to announce every change in the overall connectivity of the ad hoc network.

3.1 Protocol Overview

The protocol does not attempt to maintain routes from every node to every other node in the network. Routes are discovered on an as-needed basis and are maintained only as long as they are necessary. It is loop-free at all times, even while repairing broken links. This loop freedom is accomplished through the use of sequence numbers, which it increases each time it learns of a change in the topology of its neighbourhood. This sequence number ensures that the most recent route is selected whenever route discovery is executed. In addition, each multicast group has its own sequence number, which is maintained by the multicast group leader.

AODV is able to provide unicast, multicast and broadcast communication ability. A protocol that offers both unicast and multicast communication can be streamlined so that route information obtained while searching for a multicast route can also increase unicast routing knowledge and vice versa. AODV utilises both a route table (for unicast routes) and a multicast route table (for multicast routes). The route table is used to store the destination and next-hop IP addresses as well as the destination sequence number. Additionally, for each destination the node maintains a list of precursor nodes, which route through it in order to reach the destination. This list is maintained for the purpose of route maintenance if the link breaks. Also associated with each route table entry is a lifetime, which is updated whenever a route is used.

It also provides for the quick deletion of invalid routes through the use of a special route error message. It also responds to topological changes that affect active routes in a quick and timely manner. It builds routes with only a small amount of overhead from routing control messages and no additional network overhead. It requires nodes to maintain only next-hop information thereby decreasing the storage requirement at each of the mobile nodes.

3.2 Unicast Route Establishment

Route discovery is purely on demand and follows a route request / route reply discovery cycle. Requests are sent using a route discovery (RREQ) message. Information enabling the creation of a route is sent back in a route reply (RREP) message.

The basic outline of the route discovery process is as follows:

- When a node needs a route to a destination, it broadcasts a RREQ.
- Any node with a current route to that destination (including the destination itself) can unicast a RREP back to the source node.
- Route information is maintained by each node in its route table.
- Information obtained through RREQ and RREP messages is kept with other routing information in the route table.
- Sequence numbers are used to eliminate stale routes.
- Routes with old sequence numbers are aged out of the system.

3.3 Route Discovery

When a node wishes to send a packet to some destination node, it checks its route table to determine whether it has a current route to that node. If so, it forwards the packet to the appropriate next hop toward the destination. However, if the node does not have a valid route to the destination, it must initiate a route discovery process. To begin such a process, the node creates a RREQ packet. This packet contains the source node's IP address and current sequence number as well as the destination's IP address and last known sequence number. The RREQ also contains a broadcast ID, which is incremented each time the source node initiates a RREQ. After creating the RREQ, the source node broadcasts the packet and then sets a timer to wait for a reply.

When a node receives a RREQ, it first checks whether it has seen it before by noting the source IP address and broadcast ID pair. Each node maintains a record of the source IP address/ broadcast ID for each RREQ it receives, for a specified length of time. If it has already seen a RREQ with the same IP address / broadcast ID pair, it silently discards the packet. Otherwise, it records this information and then processes the packet.

To process the RREQ, the node sets up a reverse route entry for the source node in its route table. This reverse route entry contains the source node's IP address and sequence number as well as the number of hops to the source node and the IP address of the neighbour from which the RREQ was received. In this way, the node knows how to forward the RREQ to the source if one is received later. Associated with the reverse route entry is a lifetime. If this route entry is not used within the specified lifetime, the route information is deleted to prevent the stale routing information from lingering in the route table.

To respond to the RREQ, the node must have an unexpired entry for the destination in its route table. Furthermore, the sequence number associated with the destination must be at least as great as that indicated in the RREQ. This prevents the formation of routing loops by ensuring that the route returned is never old enough to point to a previous intermediate node.

If the RREQ is lost, the source node is allowed to retry the broadcast route discovery mechanism. After `rreq_retries` additional attempts, it is required to notify the application that the destination is unreachable.

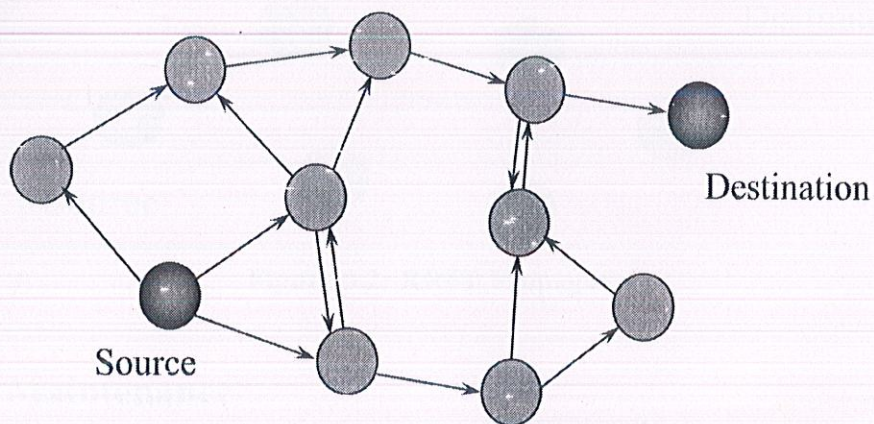


Figure 3.1: RREQ Broadcast Flood

3.4 Forward Path Setup

When a node determines that it has a route current enough to respond to the RREQ, it creates RREP. The RREP sent in response to the RREQ contains the IP address of both the source and destination. If the destination node is responding, it places its current sequence number in the packet, initializes the hop count to zero and places the length of time this route is valid in the RREP's lifetime field. However, if an intermediate node is responding, it places its records of the

destination's sequence number in the packet, sets the hop count equal to its distance from the destination and calculates the amount of time for which its route table entry for the destination will still be valid. It then unicasts the RREP toward the source node, using the node from which it received the RREQ as the next hop.

When an intermediate node receives the RREP, it sets up a forward path entry to the destination in its route table. To obtain its distance to the destination, the node increments the value in the hop count field by 1. Also associated with this entry is a lifetime which is updated each time the route is used. If the route is not used within the specified lifetime, it is deleted.

It is likely that a node will receive a RREP for a given destination from more than one neighbour. In this case, it forwards the first RREP it receives and forwards a later RREP only if that RREP contains a greater destination sequence number or a smaller hop count. Otherwise, the node discards the packet. This decreases the number of RREPs propagating toward the source while ensuring the most up-to-date and quickest routing information.

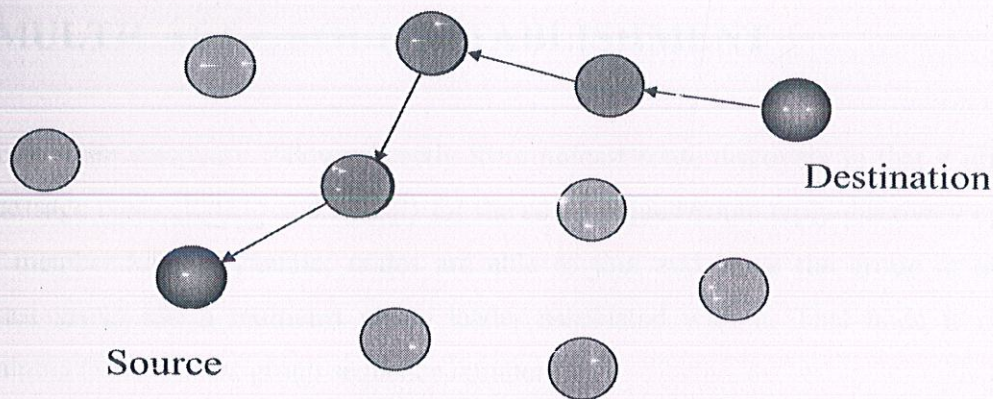


Figure 3.2: RREP Propagation

3.5 Route Maintenance

Once a route has been discovered for a given source/destination pair, it is maintained as long as needed by the source node. Movement of nodes within the ad hoc network affects only the routes containing those nodes; such a path is called an active path.

When either the destination or some intermediate node moves, a route error (RERR) message is sent to the affected source nodes. This RERR is initiated by the node upstream of the break. It lists each of the destinations that are now unreachable because of loss of the link. If the node upstream of the

break has one or more nodes listed as precursor node for the destination, it broadcasts the RERR to these neighbours. When the neighbours receive the RERR, they mark their route to the destination as invalid by setting the distance to the destination as infinity and in turn propagate the RERR to the precursor nodes, if any such nodes are listed for the destinations in their route tables. When a source node receives the RERR, it can reinitiate the route discovery if the route is still needed.

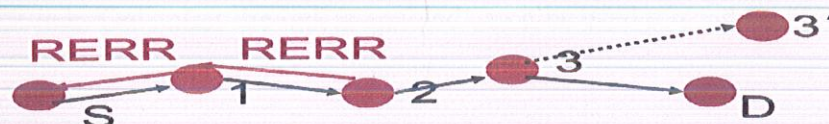


Figure 3.3: RERR Propagation

The link from node 3 to D breaks as node 3 moves to new position 3'. Node 2 sends RERR message to node 1 which further sends to node S. Node S initiates a route discovery if it still needs a route to D.

3.6 MULTICAST GROUP ESTABLISHMENT

Multicast route discovery follows directly from unicast route discovery in that it utilises the same two message types (RREQ and RREP) for the route request/route reply discovery cycle. Multicast group membership is dynamic; nodes are able to join and leave the group at any time. Each multicast group has a multicast group leader associated with it. That node is responsible for maintaining the multicast group sequence number.

3.6.1 Route Discovery

The route discovery begins when a node wishes to join a multicast group or when it has data to send to a multicast group and does not have a current route to it. This source node creates a RREQ with the destination address set to the IP address of the multicast group and that contains the group's last known sequence number. The node indicates in the RREQ whether it wishes to join the multicast group through join flag. It then broadcasts the RREQ to its neighbours.

If the RREQ is a join request, only a node that is a member of the desired multicast tree may respond. Otherwise, any node with a current route to the multicast group may reply. If a node receives a RREQ for a multicast group of which it is not a member or if it receives a RREQ and

does not have a route to that group, it creates a reverse route entry to the source and then broadcasts the RREQ to its neighbours.

When a node receives a join RREQ for a multicast group, it adds an inactivated entry for the source node in its multicast route table. Each next hop entry in the multicast route table has an associated activated flag. If this flag is false, the node does not forward any data packets for the multicast group along that link. Only after the link is enabled can it be used to send data packets.

3.6.2 Forward Path Setup

If a node receives a join RREQ for a multicast group, it may reply if it is a router for the multicast group's tree and if its recorded sequence number for the multicast group is at least as great as that contained in the RREQ. The group leader can always reply to a join RREQ for its multicast group. The responding node updates its multicast route table by placing the requesting node's next-hop information in the table and then generates a RREP. The node unicasts the RREP back to the node indicated in the RREQ. As nodes along the path to the source node receive the RREP, they set up a forward path entry for the multicast group in their multicast route table by adding the node from which they received the RREP as a next hop. Then they increment the hop count field and forward the RREP to the next node.

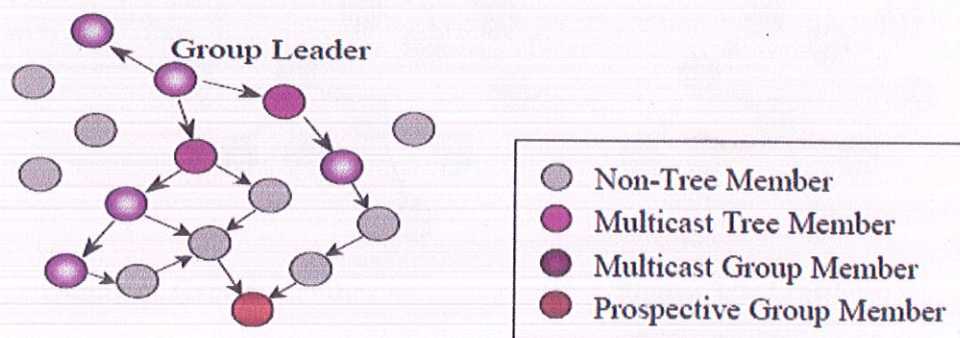


Figure 3.4: Route Discovery and Reply Generation for multicast group

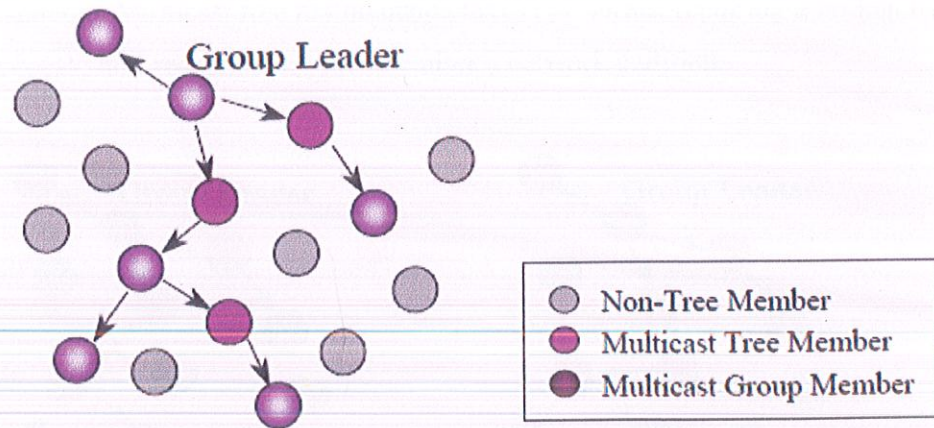


Figure 3.5: Route Activation for multicast group

3.6.3 Leaving the Group

The group nodes may revoke their member status at any time. Leaf nodes may prune themselves from the tree while non-leaf nodes should continue to be routers in the tree even while they are not active members of the multicast group.

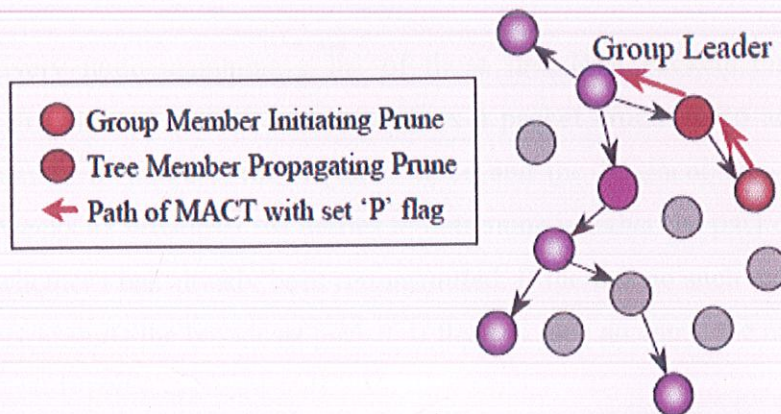


Figure 3.6: Pruning of Group Member

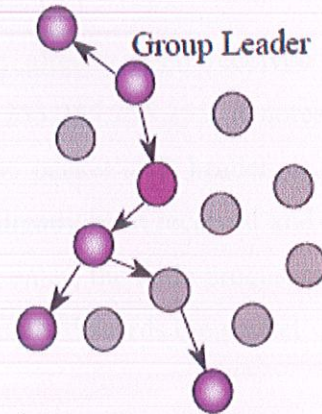


Figure 3.7: Multicast Tree after Prune

3.6.4 Multicast Tree Maintenance

The multicast tree must be maintained for the lifetime of the multicast group. Unlike in the unicast scenario, however, a link break due to changes in the topology necessarily triggers route reconstruction because the multicast group members must remain connected during the group's lifetime. Each multicast link requires ongoing route maintenance to ensure that other multicast tree

members are always reachable. Multicast tree maintenance takes two forms: repairing a broken tree branch following a link break, and reconnecting the tree after a network partition.

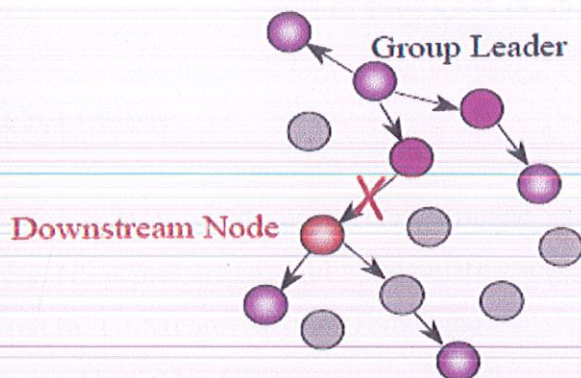


Figure 3.8: Link Breakage of multicast tree

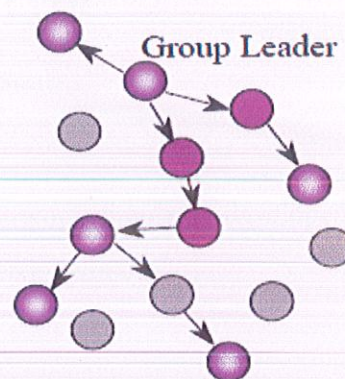


Figure 3.9: Repaired Multicast Tree

3.6.5 Broadcast

AODV specifies behaviour for transmitting broadcasts. When a node wishes to generate a broadcast, it sends the broadcast packet to the well known broadcast address 255.255.255.255.

Every node maintains a list of those broadcast packets that have already been received and retransmitted. When a node receives a packet broadcast to address 255.255.255.255, it notes the source IP address, the IP_ident value and the fragment effect of the packet's IP header. It then checks its broadcast list entries to determine whether the packet has already been received and thus whether it has already been retransmitted. If there is no such matching entry, the node processes and retransmits the broadcast packet. If there is such an entry, the node silently discards the packet.

CHAPTER 4

NETWORK SIMULATOR

4.1 Overview

NS (version 2) is an object-oriented, discrete event driven network simulator developed at UC Berkely. NS is primarily useful for simulating local and wide area networks.

Written in: C++ (core) , Python (bindings)

Platform: Unix, Mac OS X

It implements network protocols such as TCP and UDP, traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue management mechanism such as Drop Tail, RED and CBQ, routing algorithms such as Dijkstra, and more. NS also implements multicasting and some of the MAC layer protocols for LAN simulations.

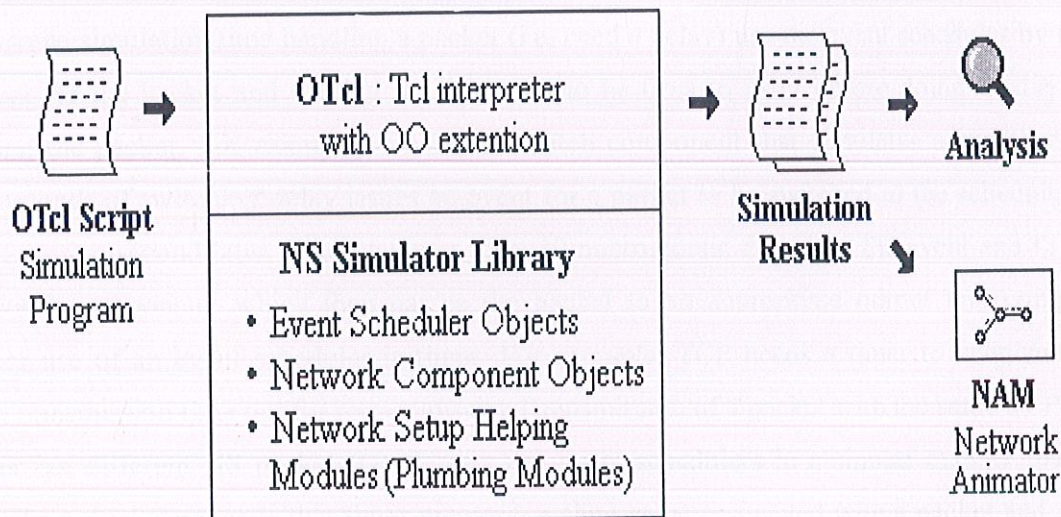


Figure 4.1: Simplified User's View of NS

As shown in Figure 4.1, in a simplified user's view, NS is Object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries, and network setup (plumbing) module libraries (actually, plumbing modules are implemented as member functions of the base simulator object). In other words, to use NS, you program in OTcl script language. To setup and run a simulation network, a user should write an OTcl script that initiates an event scheduler, sets up the network topology using the network objects and the

plumbing functions in the library, and tells traffic sources when to start and stop transmitting packets through the event scheduler. The term "plumbing" is used for a network setup, because setting up a network is plumbing possible data paths among network objects by setting the "neighbor" pointer of an object to the address of an appropriate object. When a user wants to make a new network object, he or she can easily make an object either by writing a new object or by making a compound object from the object library, and plumb the data path through the object. This may sound like complicated job, but the plumbing OTcl modules actually make the job very easy. The power of NS comes from this plumbing.

Another major component of NS beside network objects is the event scheduler. An event in NS is a packet ID that is unique for a packet with scheduled time and the pointer to an object that handles the event. In NS, an event scheduler keeps track of simulation time and fires all the events in the event queue scheduled for the current time by invoking appropriate network components, which usually are the ones who issued the events, and let them do the appropriate action associated with packet pointed by the event. Network components communicate with one another passing packet, however this does not consume actual simulation time. All the network components that need to spend some simulation time handling a packet (i.e. need a delay) use the event scheduler by issuing an event for the packet and waiting for the event to be fired to itself before doing further action handling the packet. For example, a network switch component that simulates a switch with 20 microseconds of switching delay issues an event for a packet to be switched to the scheduler as an event 20 microsecond later. The scheduler after 20 microsecond dequeues the event and fires it to the switch component, which then passes the packet to an appropriate output link component. Another use of an event scheduler is timer. For example, TCP needs a timer to keep track of a packet transmission time out for retransmission (transmission of a packet with the same TCP packet number but different NS packet ID). Timers use event schedulers in a similar manner that delay does. The only difference is that timer measures a time value associated with a packet and does an appropriate action related to that packet after a certain time goes by, and does not simulate a delay.

NS is written not only in OTcl but in C++ also. For efficiency reason, NS separates the data path implementation from control path implementations. In order to reduce packet and event processing time (not simulation time), the event scheduler and the basic network component objects in the data path are written and compiled using C++.. One thing to note in the figure is that for C++ objects that have an OTcl linkage forming a hierarchy, there is a matching OTcl object hierarchy very similar to that of C++.

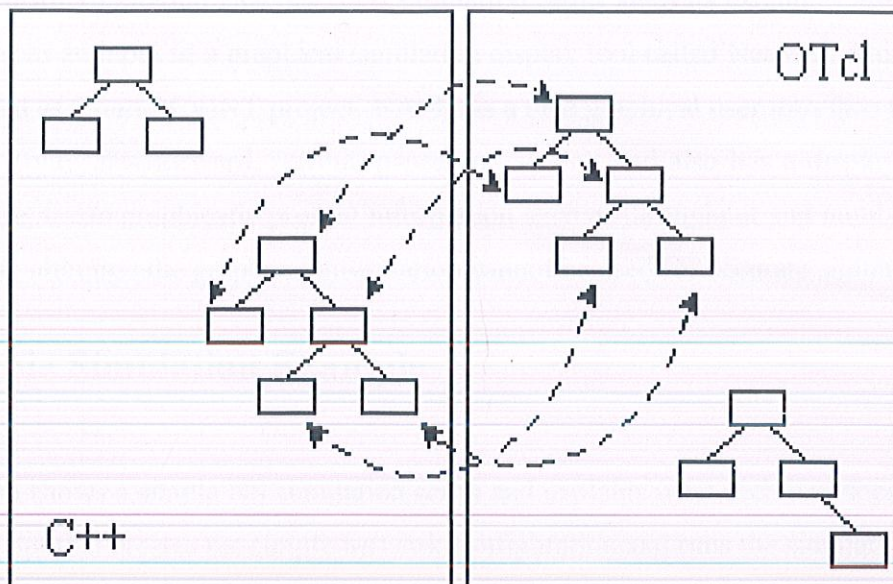


Figure 4.2: C++ and OTcl: The Duality

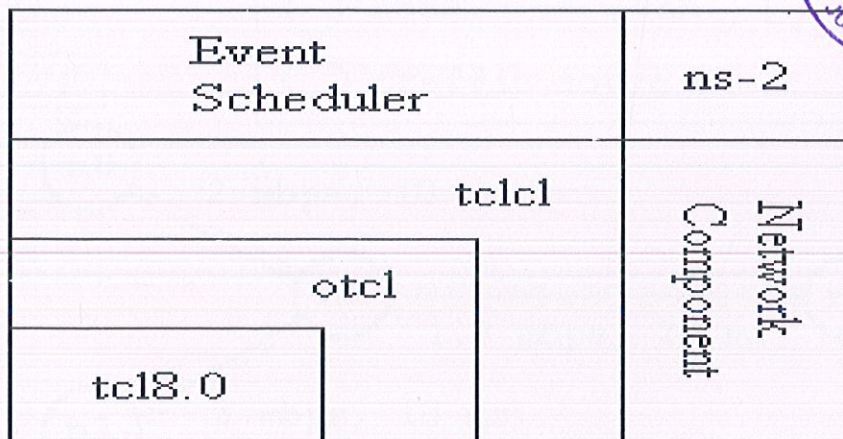


Figure 4.3: Architectural View of NS

Figure 4.3 shows the general architecture of NS. In this figure a general user (not an NS developer) can be thought of standing at the left bottom corner, designing and running simulations in Tcl using the simulator objects in the OTcl library. The event schedulers and most of the network components are implemented in C++ and available to OTcl through an OTcl linkage that is implemented using tclcl. The whole thing together makes NS, which is a OO extended Tcl interpreter with network simulator libraries.

This section briefly examined the general structure and architecture of NS. At this point, one might be wondering about how to obtain NS simulation results. As shown in Figure 4.1, when a simulation is finished, NS produces one or more text-based output files that contain detailed simulation data, if specified to do so in the input Tcl (or more specifically, OTcl) script. The data

can be used for simulation analysis (two simulation result analysis examples are presented in later sections) or as an input to a graphical simulation display tool called Network Animator (NAM) that is developed as a part of VINT project. NAM has a nice graphical user interface similar to that of a CD player (play, fast forward, rewind, pause and so on), and also has a display speed controller. Furthermore, it can graphically present information such as throughput and number of packet drops at each link, although the graphical information cannot be used for accurate simulation analysis.

4.2 Simple Simulation Example

This section shows a simple NS simulation script and explains what each line does. Example 4.1 is an OTcl script that creates the simple network configuration and runs the simulation scenario in Figure 4.4

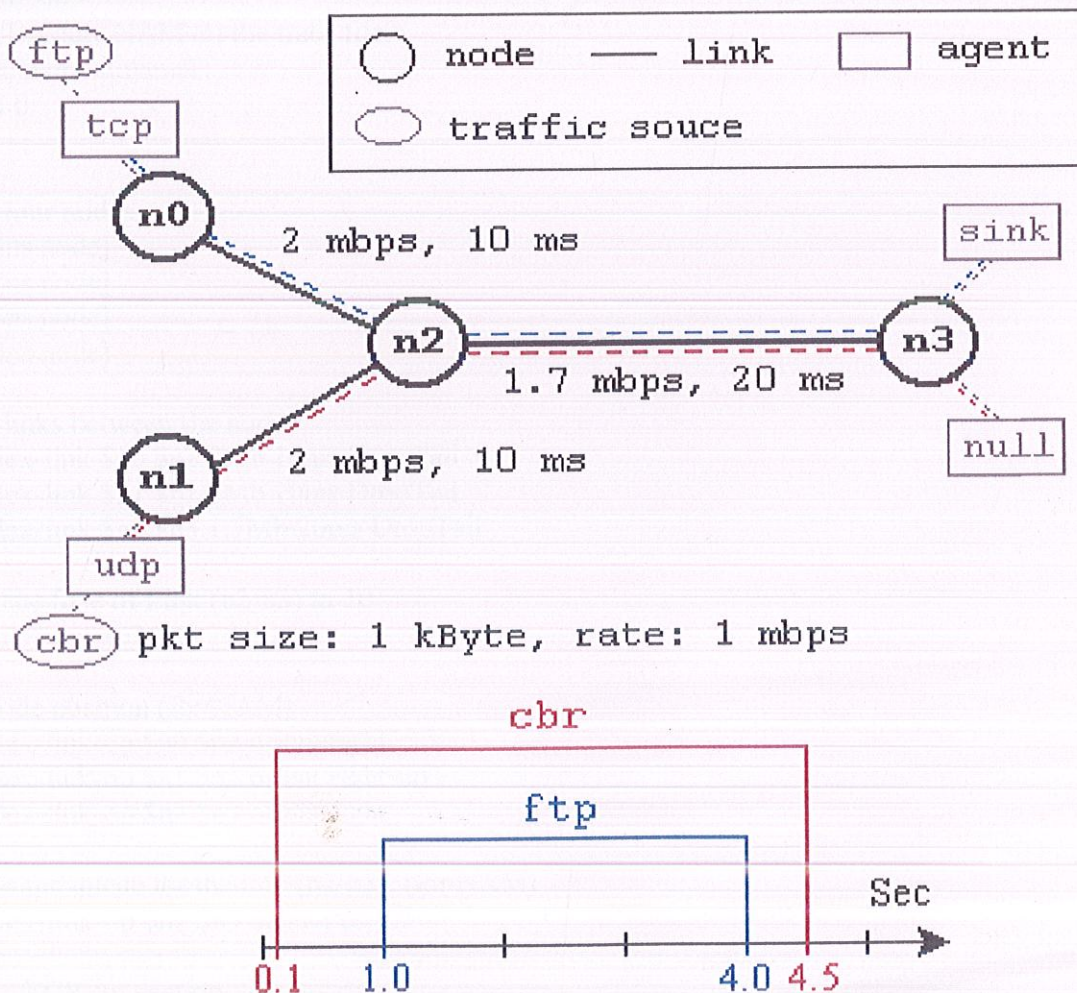


Figure 4.4: A Simple Network Topology and Simulation Scenario

Example 4.1: A Simple NS Simulation Script

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
    global ns nf
    $ns flush-trace
    #Close the NAM trace file
    close $nf
    #Execute NAM on the trace file
    exec namout.nam&
    exit 0
}

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of Link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for the link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5

# Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
```

```

$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

# Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

# Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false

#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run

```

This network consists of 4 nodes (n0, n1, n2, n3) as shown in above figure 4.4. The duplex links between n0 and n2, and n1 and n2 have 2 Mbps of bandwidth and 10 ms of delay. The duplex link between n2 and n3 has 1.7 Mbps of bandwidth and 20 ms of delay. Each node uses a DropTail queue, of which the maximum size is 10. A "tcp" agent is attached to n0, and a connection is

established to a tcp "sink" agent attached to n3. As default, the maximum size of a packet that a "tcp" agent can generate is 1Kbyte. A tcp "sink" agent generates and sends ACK packets to the sender (tcp agent) and frees the received packets. A "udp" agent that is attached to n1 is connected to a "null" agent attached to n3. A "null" agent just frees the packets received. A "ftp" and a "cbr" traffic generator are attached to "tcp" and "udp" agents respectively, and the "cbr" is configured to generate 1 Kbyte packets at the rate of 1 Mbps. The "cbr" is set to start at 0.1 sec and stop at 4.5 sec, and "ftp" is set to start at 1.0 sec and stop at 4.0 sec.

4.3 Post Simulation

The following is the explanation of the script above. In general, an NS script starts with making a Simulator object instance.

set ns [new Simulator]: generates an NS simulator object instance, and assigns it to variable *ns* (italics is used for variables and values in this section). What this line does is the following:

- Initialize the packet format
- Create a scheduler
- Select the default address format

The "Simulator" object has member functions that do the following

- Create compound objects such as nodes and links
- Connect network component objects created (ex. attach-agent)
- Set network component parameters (mostly for compound objects)

Create connections between agents (ex. make connection between a "tcp" and "sink")

Specify NAM display options

Most of member functions are for simulation setup (referred to as plumbing functions in the Overview section) and scheduling, however some of them are for the NAM display.

\$ns color fid color: is to set color of the packets for a flow specified by the flow id (fid). This member function of "Simulator" object is for the NAM display, and has no effect on the actual simulation.

\$ns namtrace-all file-descriptor: This member function tells the simulator to record simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command

\$ns flush-trace: Similarly, the member function trace-all is for recording the simulation trace in a general format.

proc finish {}: is called after this simulation is over by the command *\$ns at 5.0 "finish"*. In this function, post simulation process are specified.

set n0 [\$ns node]: The member function node creates a node. A node in NS is compound object made of address and port classifiers (described in a later section). Users can create a node by separately creating an address and a port classifier objects and connecting them together. However, this member function of Simulator object makes the job easier

\$ns duplex-link node1 node2 bandwidth delay queue-type: creates two simplex links of specified bandwidth and delay, and connects the two specified nodes. In NS, the output queue of a node is implemented as a part of a link, therefore users should specify the queue-type when creating links. In the above simulation script, DropTail queue is used. If the reader wants to use a RED queue, simply replace the word DropTail with RED. The NS implementation of a link is shown in a later section. Like a node, a link is a compound object, and users can create its sub-objects and connect them and the nodes.

\$ns queue-limit node1 node2 number: This line sets the queue limit of the two simplex links that connect node1 and node2 to the number specified. At this point, the authors do not know how many of these kinds of member functions of Simulator objects are available and what they are.

\$ns duplex-link-op node1 node2: The next couple of lines are used for the NAM display. To see the effects of these lines, users can comment these lines out and try the simulation.

Now that the basic network setup is done, the next thing to do is to setup traffic agents such as TCP and UDP, traffic sources such as FTP and CBR, and attach them to nodes and agents respectively.

set tcp [new Agent/TCP]: This line shows how to create a TCP agent. But in general, users can create any agent or traffic sources in this way. Agents and traffic sources are in fact basic

objects (not compound objects), mostly implemented in C++ and linked to OTcl. Therefore, there are no specific Simulator object member functions that create these object instances. To create agents or traffic sources, a user should know the class names these objects (Agent/TCP, Agent/TCPSink, Application/FTP and so on). This information can be found in the NS documentation or partly in this documentation.

\$ns attach-agent node agent: The **attach-agent** member function attaches an agent object created to a node object. Actually, what this function does is call the **attach** member function of specified node, which attaches the given agent to itself. Therefore, a user can do the same thing by, for example, **\$n0 attach \$tcp**. Similarly, each agent object has a member function **attach-agent** that attaches a traffic source object to itself.

\$ns connect agent1 agent2: After two agents that will communicate with each other are created, the next thing is to establish a logical network connection between them. This line establishes a network connection by setting the destination address to each others' network and port address pair.

Assuming that all the network configuration is done, the next thing to do is write a simulation scenario (i.e. simulation scheduling). The Simulator object has many scheduling member functions. However, the one that is mostly used is the following:

\$ns at time "string": This member function of a Simulator object makes the scheduler (scheduler_ is the variable that points the scheduler object created by [new Scheduler] command at the beginning of the script) to schedule the execution of the specified string at given simulation time. For example, **\$ns at 0.1 "\$cbr start"** will make the scheduler call a start member function of the CBR traffic source object, which starts the CBR to transmit data. In NS, usually a traffic source does not transmit actual data, but it notifies the underlying agent that it has some amount of data to transmit, and the agent, just knowing how much of the data to transfer, creates packets and sends them.

After all network configurations, scheduling and post-simulation procedure specifications are done, the only thing left is to run the simulation. This is done by **\$ns run**.

4.4 Trace Analysis Example

This section shows a trace analysis example

Example 4.2 Trace Enabled Simple NS Simulation Script

```
#Open the NAM trace file
```

```
set nf [open out.nam w]
```

```
$ns namtrace-all $nf
```

```
#Define a 'finish' procedure
```

```
proc finish {} {
```

```
    global ns nf
```

```
    $ns flush-trace
```

```
#Close the NAM trace file
```

```
close $nf
```

```
#Execute NAM on the trace file
```

```
exec namout.nam
```

```
exit 0
```

Running the above script generates a NAM trace file that is going to be used as an input to NAM and a trace file called "**out.tr**" that will be used for our simulation analysis. Figure 4.5 shows the trace format and example trace data from "**out.tr**".

event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id
-------	------	--------------	------------	-------------	-------------	-------	-----	-------------	-------------	------------	-----------

```

r : receive (at to_node)
+ : enqueue (at queue)          src_addr : node.port (3.0)
- : dequeue (at queue)         dst_addr : node.port (0.0)
d : drop      (at queue)

```

```

r 1.3556 3 2 ack 40 ----- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ----- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ----- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ----- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ----- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ----- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ----- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ----- 2 1.0 3.1 157 207

```

Figure 4.5: Trace Format Example

Each trace line starts with an event (+, -, d, r) descriptor followed by the simulation time (in seconds) of that event, and from and to node, which identify the link on which the event occurred. The next information in the line before flags (appeared as "-----" since no flag is set) is packet type and size (in Bytes). Currently, NS implements only the Explicit Congestion Notification (ECN) bit, and the remaining bits are not used. The next field is flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script. Even though fid field may not be used in a simulation, users can use this field for analysis purposes. The fid field is also used when specifying stream color for the NAM display. The next two fields are source and destination address in forms of "node.port". The next field shows the network layer protocol's packet sequence number. Note that even though UDP implementations do not use sequence number, NS keeps track of UDP packet sequence number for analysis purposes. The last field shows the unique id of the packet.

4.5 The AWK Command

The AWK utility is a data extraction and reporting tool that uses a data-driven scripting language consisting of a set of actions to be taken against textual data (either in files or data streams) for the

purpose of producing formatted reports. The language used by awk extensively uses the string data type, associative arrays (that is, arrays indexed by key strings), and regular expressions.

The basic syntax of AWK:

```
awk 'BEGIN {start_action} {action} END {stop_action}' filename
```

Here the actions in the begin block are performed before processing the file and the actions in the end block are performed after processing the file. The rest of the actions are performed while processing the file.

```
-rw-r--r-- 1 center center 0 Dec 8 21:39 p1
-rw-r--r-- 1 center center 17 Dec 8 21:15 t1
-rw-r--r-- 1 center center 26 Dec 8 21:38 t2
-rw-r--r-- 1 center center 25 Dec 8 21:38 t3
-rw-r--r-- 1 center center 43 Dec 8 21:39 t4
-rw-r--r-- 1 center center 48 Dec 8 21:39 t5
```

From the data, you can observe that this file has rows and columns. The rows are separated by a new line character and the columns are separated by a space characters. We will use this file as the input for the examples discussed here.

1. awk '{print \$1}' input_file

Here \$1 has a meaning. \$1, \$2, \$3... represents the first, second, third columns.. in a row respectively. This awk command will print the first column in each row as shown below.

```
-rw-r--r-- -r- -
-rw-r--r-- -r- -
-rw-r--r-- -r- -
-rw-r--r-- -r- -
-rw-r--r-- -r- -
-rw-r--r-- -r- -
```

To print the 4th and 6th columns in a file use awk '{print \$4,\$5}' input_file
Here the Begin and End blocks are not used in awk. So, the print command will be executed for each row it reads from the file.

2. `awk 'BEGIN {sum=0} {sum=sum+$5} END {print sum}' input_file`

This will print the sum of the value in the 5th column. In the Begin block the variable sum is assigned with value 0. In the next block the value of 5th column is added to the sum variable. This addition of the 5th column to the sum variable repeats for every row it processed. When all the rows are processed the sum variable will hold the sum of the values in the 5th column.

3. `awk '{ if($9 == "t4") print $0;}' input_file`

This awk command checks for the string "t4" in the 9th column and if it finds a match then it will print the entire line. The output of this awk command is

```
-rw-r- -r- - 1 pcenterpcenter 43 Dec 8 21:39 t4
```

4. `awk 'BEGIN { for(i=1;i<=5;i++) print "square of", i, "is",i*i; }'`

This will print the squares of first numbers from 1 to 5. The output of the command is

square of 1 is 1

square of 2 is 4

square of 3 is 9

square of 4 is 16

square of 5 is 25

CHAPTER 5

SIMULATION STRATEGY

For the simulation of the developed system, latest version 2.34 of NS-2 has been used in this project. NS-2 is a discrete event simulator targeted at networking research. It began as a part of the REAL network simulator and is evolving through an ongoing collaboration between the University of California at Berkeley and the VINT project.

5.1 Scenario

- Topology of 50*50 and 500*500 is taken for simulation.
- Nodes have being generated randomly at random position leading to high density and low density areas of nodes.
- No new node is entering into the topology.
- Nodes are moving at different random speed.

The simulation parameters are listed in Table 5. 1.

Simulation parameters	Parameter Value
Simulator	NS-2 (Version 2.34)
Channel type	Channel/Wireless channel
Radio-propagation model	Propagation/Two ray round wave
Network interface type	Phy/WirelessPhy
MAC Type	Mac /802.11
Interface queue Type	Queue/Drop Tail
Link Layer Type	LL
Antenna	Antenna/Omni Antenna
Maximum packet in ifq	50
Area (m*m)	50 * 50
Number of mobile node	20,50
Time of simulation end	50-150 seconds
Transmission Power (range in metres)	50

Table 5.1: Simulation Parameters

CHAPTER 6: SIMULATION GRAPHS

6.1SCENARIO: 1

Calculation of Packet Delivery Ratio, Average End to end Delay, Overhead

Area: 50*50 m²

No. of Nodes: 20, 50

Simulation Time: 50seconds- 150 seconds

Protocols: AODV, DSDV

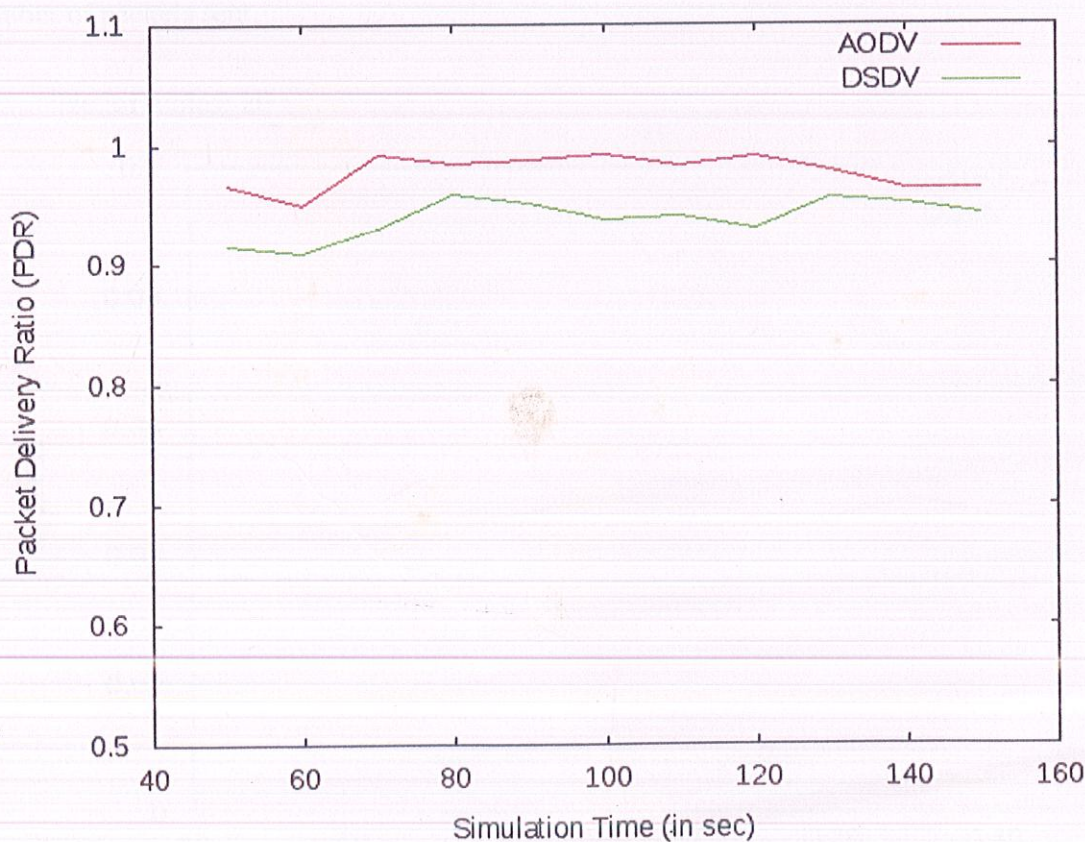
Without mobility of nodes, with mobility: 1m/s, 5m/s, 10m/s

6.1.1WITHOUT MOBILITY

Packet Delivery Ratio: Total number of delivered data packets divided by total number of data packets transmitted by all nodes. This performance metric will give us an idea of how well the protocol is performing in terms of packet delivery at different no. of nodes in the network using different traffic speeds.

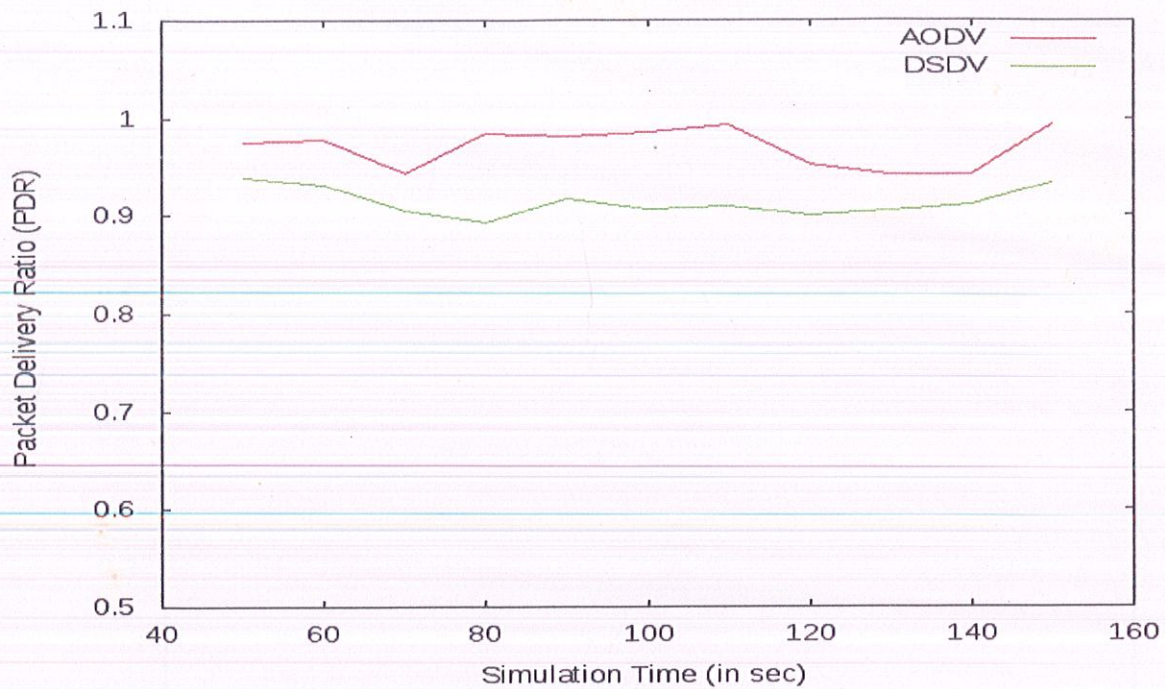
$$\text{Packet Delivery Ratio} = \frac{\sum_1^n \text{CBR received}}{\sum_1^n \text{CBR sent}}$$

No of nodes: 20



Graph 6.1.1.1: PDR_20

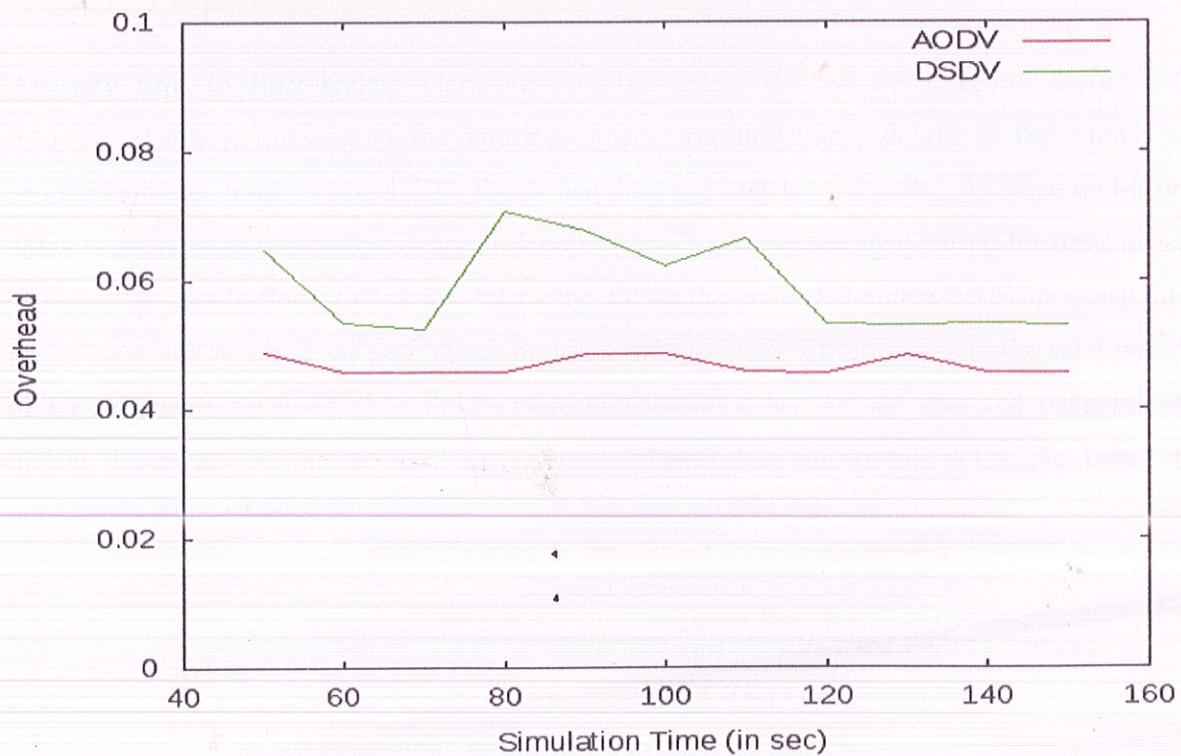
No of nodes: 50



Graph 6.1.1.2: PDR_50

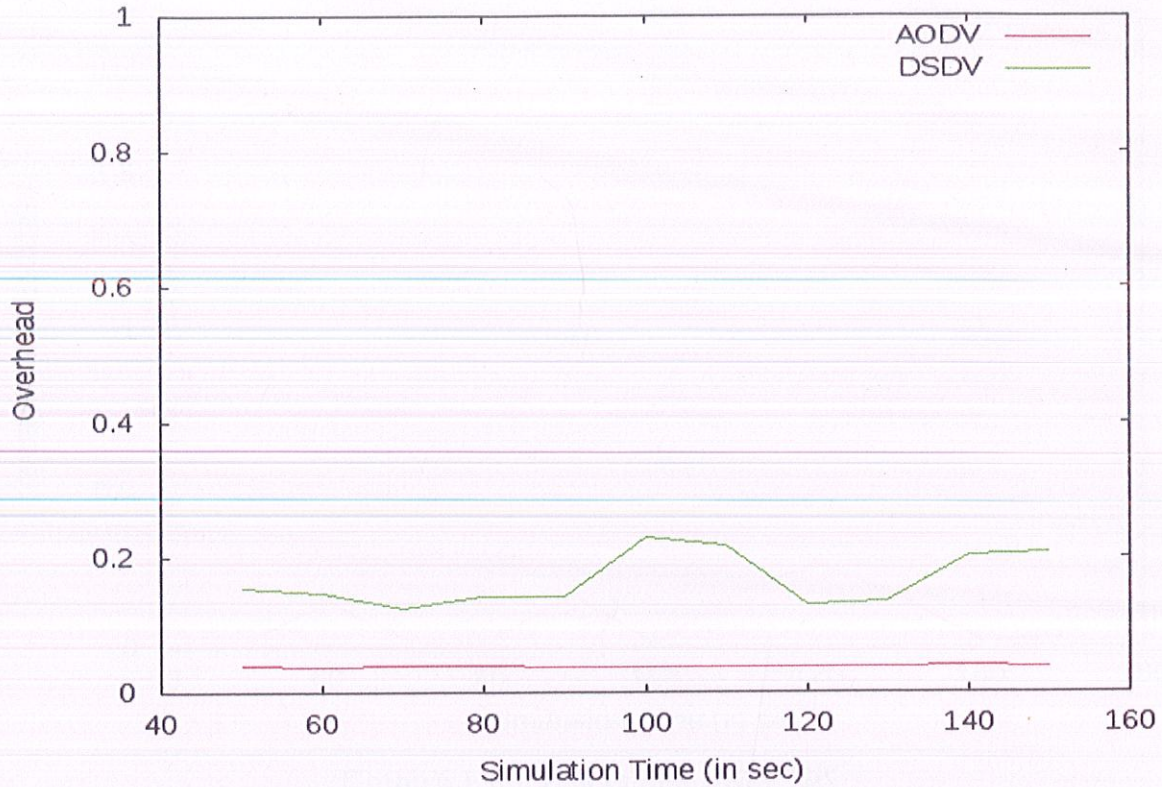
Overhead: Routing overhead, which measures the ratio of total routing packets sent and the total number of packets sent.

No. of nodes: 20



Graph 6.1.1.3: Overhead_20

No. of nodes: 50

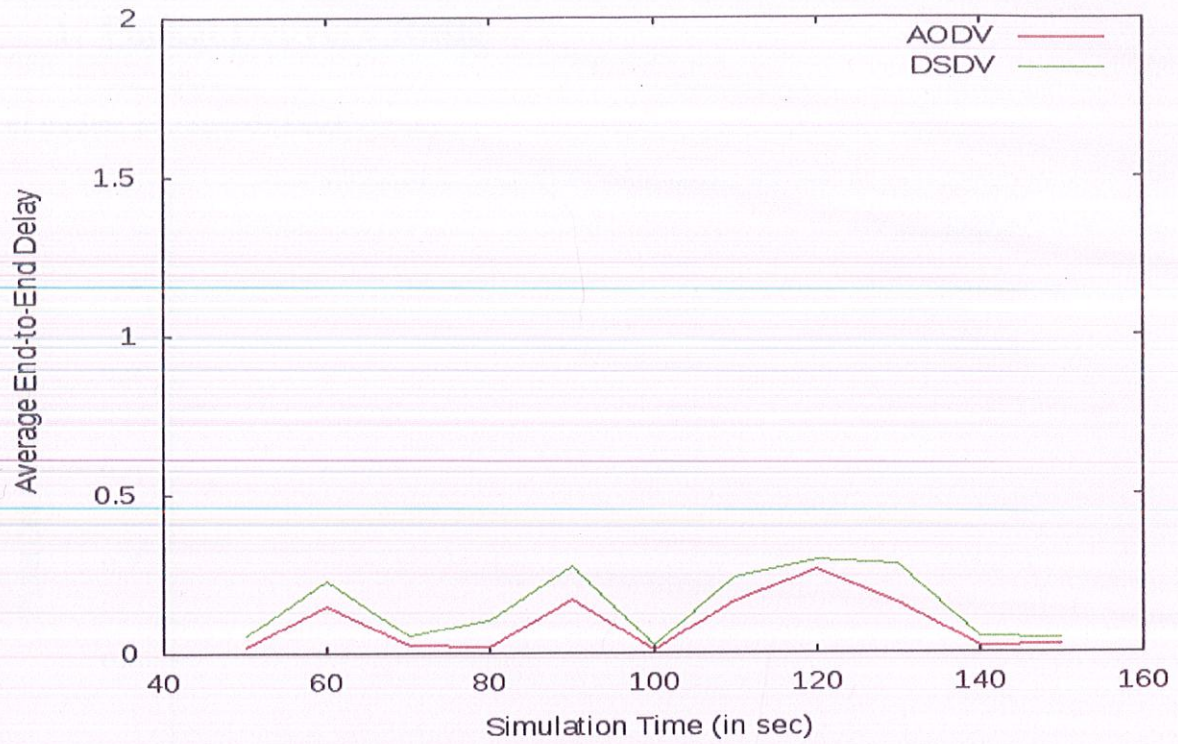


Graph 6.1.1.4: Overhead _50

Average End to End Delay: Here are possible delays caused by buffering during route discovery latency, queuing at the interface queue, retransmission delays at the MAC, and propagation and transfer times. The thesis use Average end-to-end delay. Average end-to-end delay is an average end-to-end delay of data packets. It also caused by queuing for transmission at the node and buffering data for detouring. Once the time difference between every CBR packet sent and received was recorded, dividing the total time difference over the total number of CBR packets received gave the average end-to-end delay for the received packets. This metric describes the packet delivery time: the lower the end-to-end delay the better the application performance.

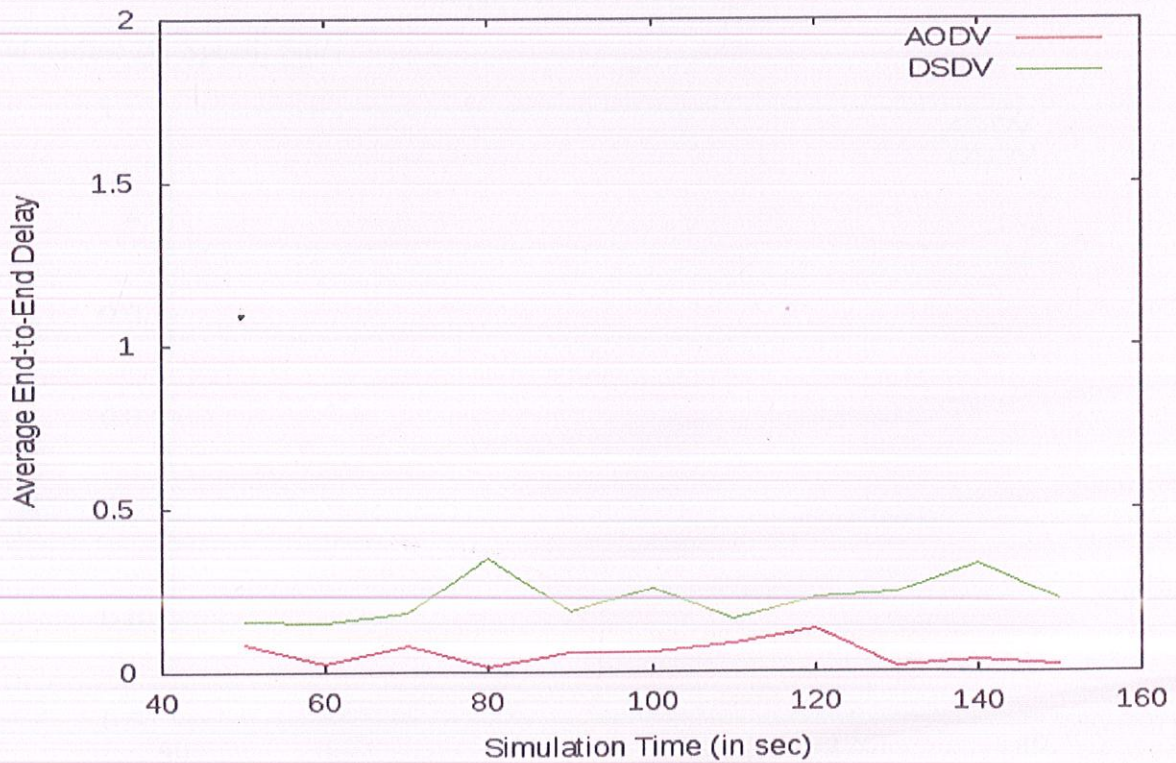
$$\text{Average End to End Delay} = \frac{\sum_1^n (\text{CBR sent time} - \text{CBR receive time})}{\sum_1^n \text{CBR receive packets}}$$

No of nodes: 20



Graph 6.1.1.5: End to End Delay _20

No. of nodes: 50

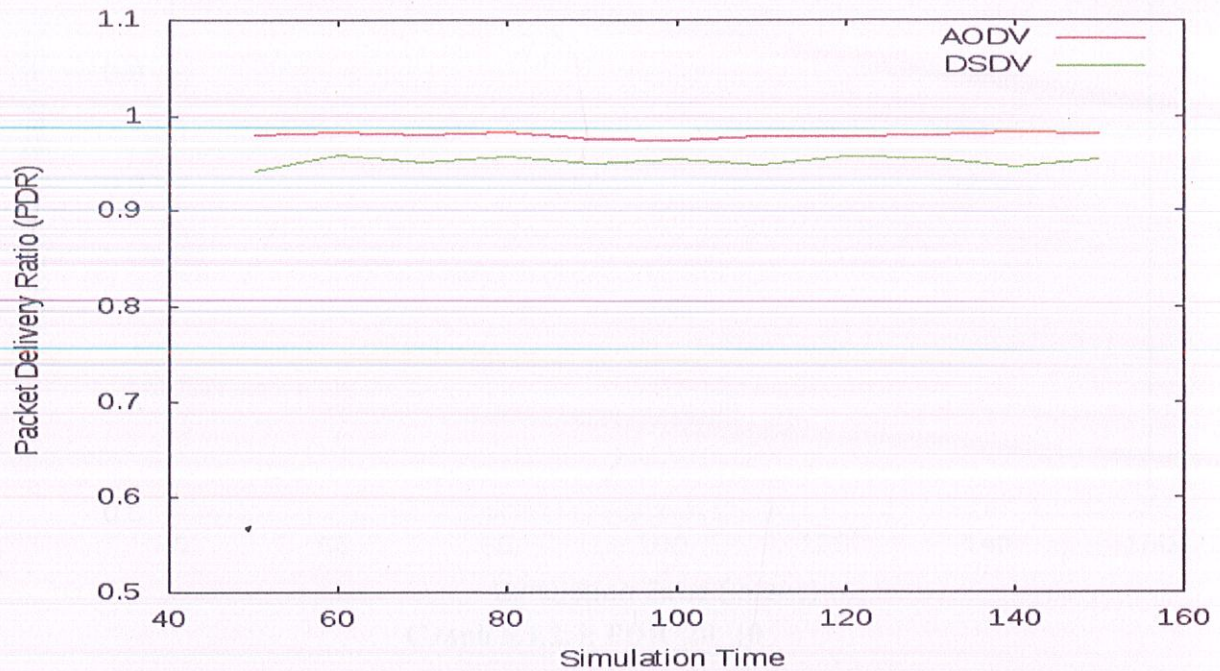


Graph 6.1.1.6: End to end Delay _50

6.1.2 WITH MOBILITY

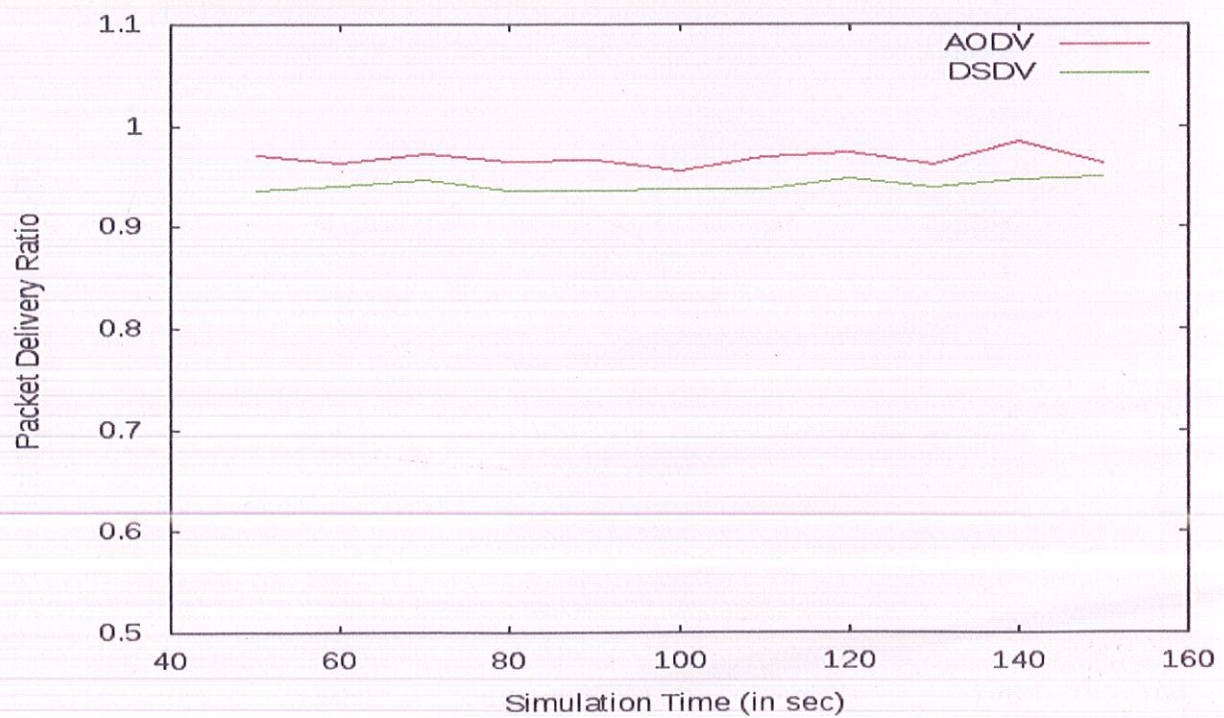
1. Packet Delivery Ratio

No of nodes: 20; Speed: 1m/s



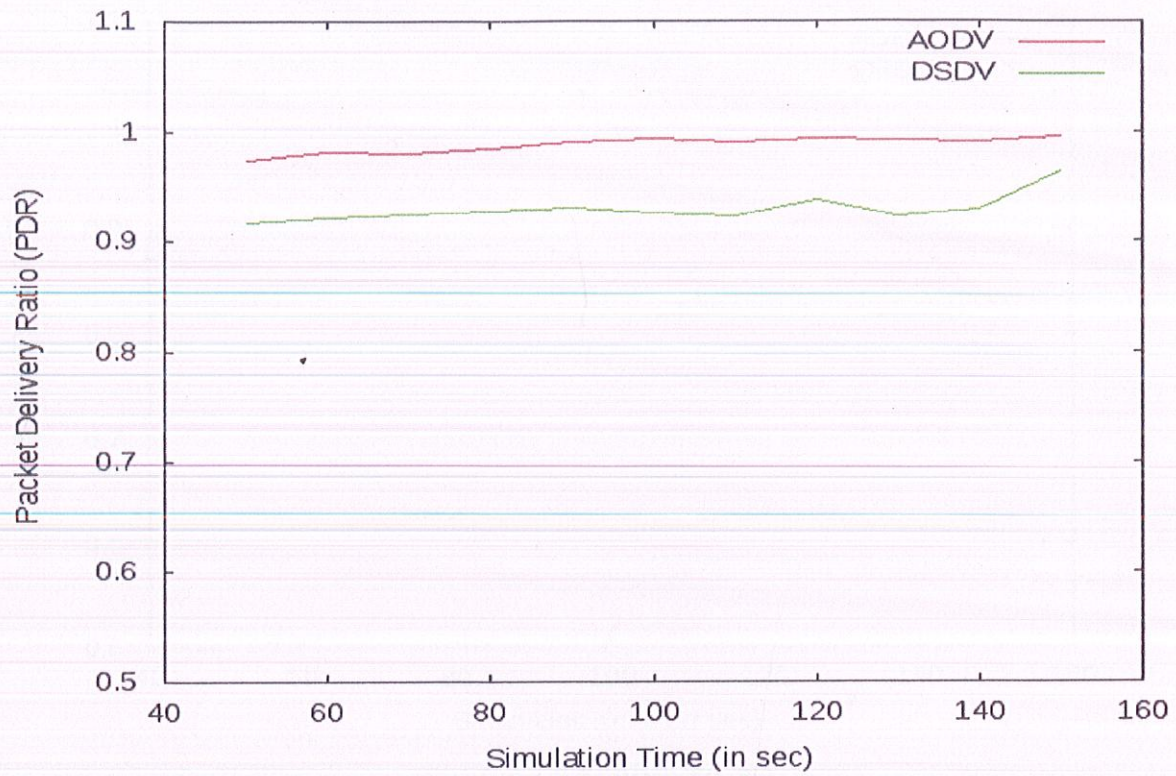
Graph 6.1.2.1: PDR_20_1

No of nodes: 20; Speed: 5m/s



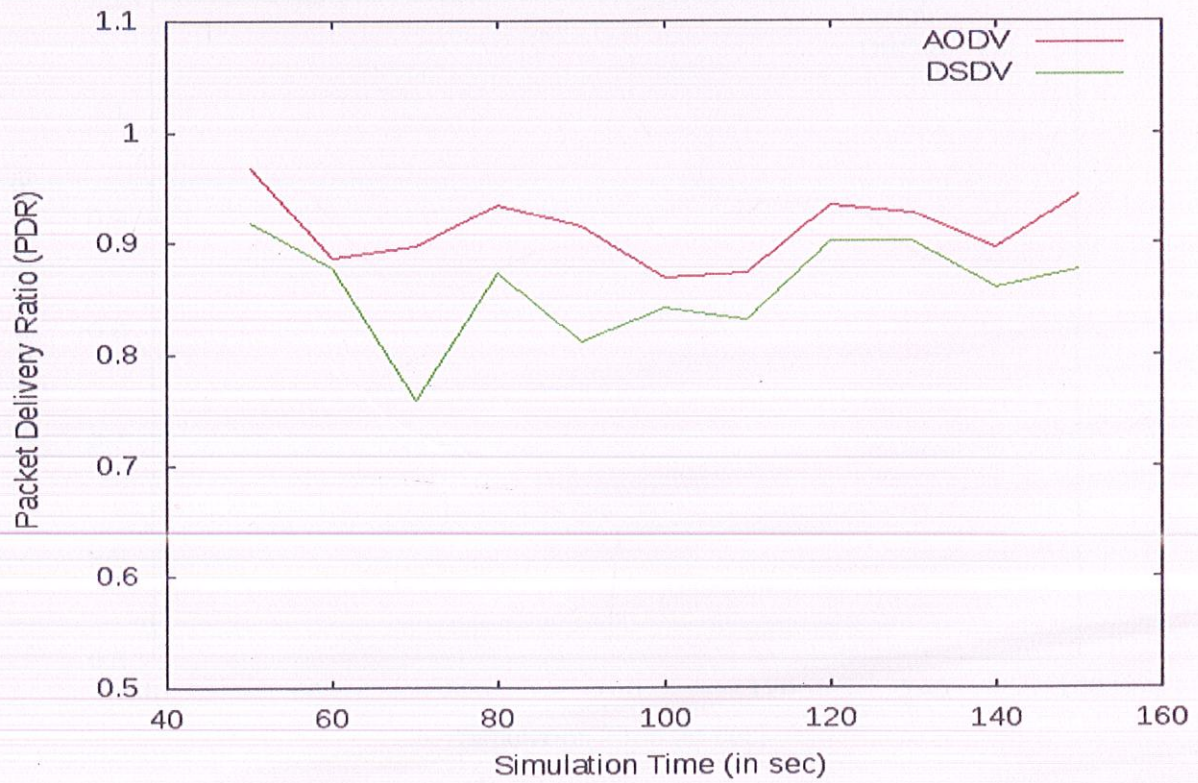
Graph 6.1.2.2: PDR_20_5

No of nodes : 20 ; Speed: 10m/s



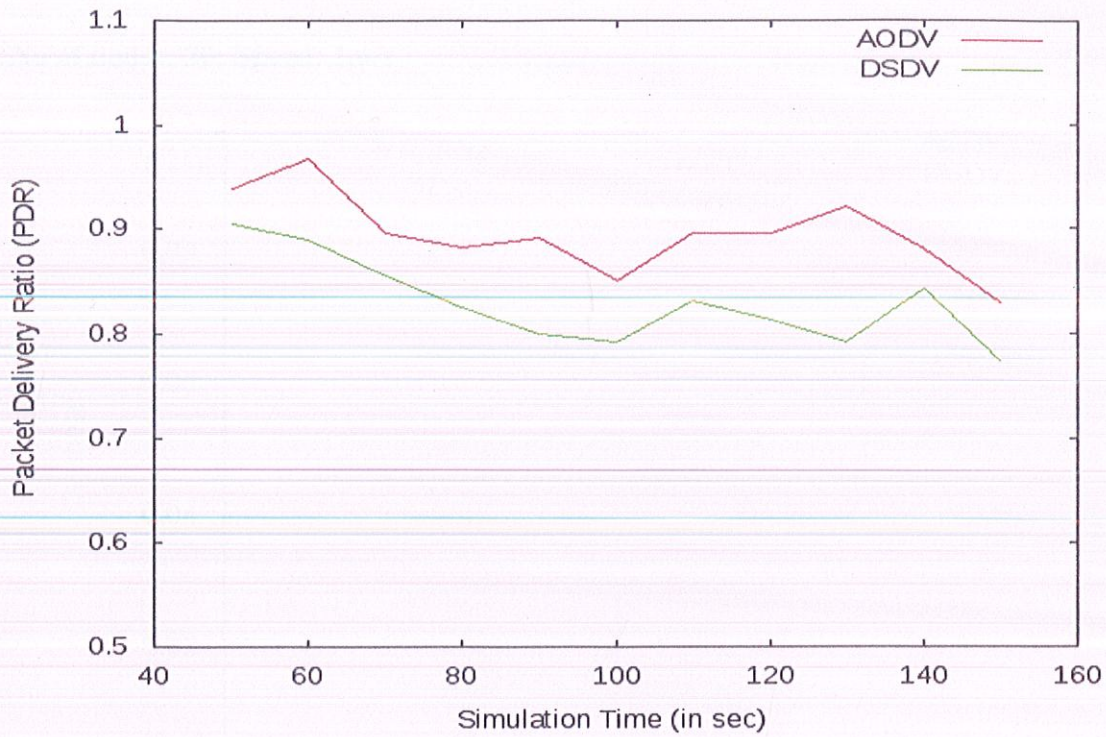
Graph 6.1.2.3: PDR_20_10

No. of nodes: 50; Speed: 1m/s



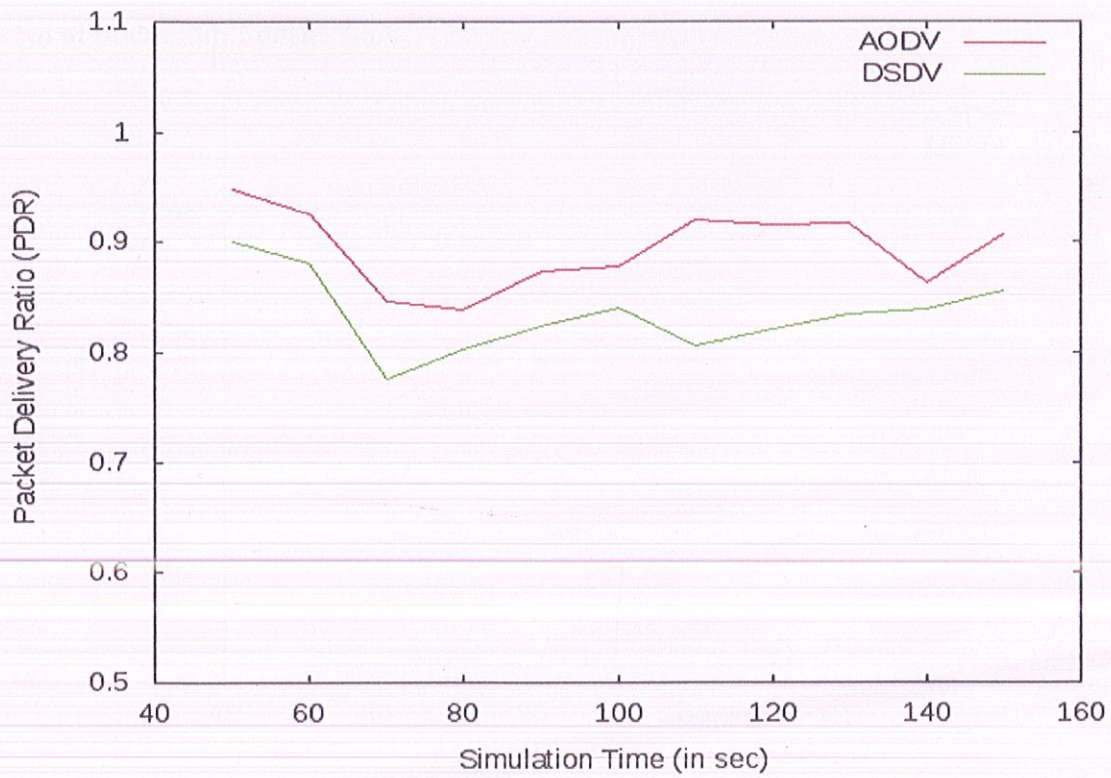
Graph: 6.1.2.4: PDR_50_1

No of node: 50; Speed: 5m/s



Graph 6.1.2.5: PDR_50_5

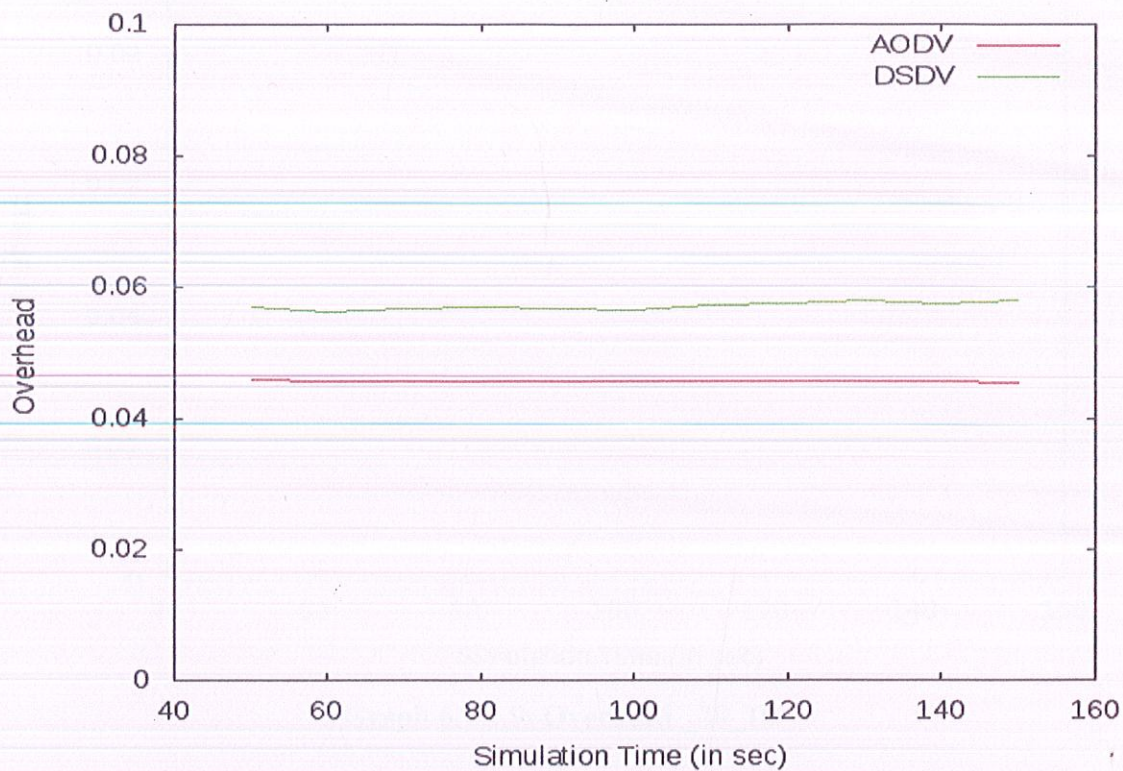
No of node: 50; Speed: 10m/s



Graph 6.1.2.6: PDR_50_10

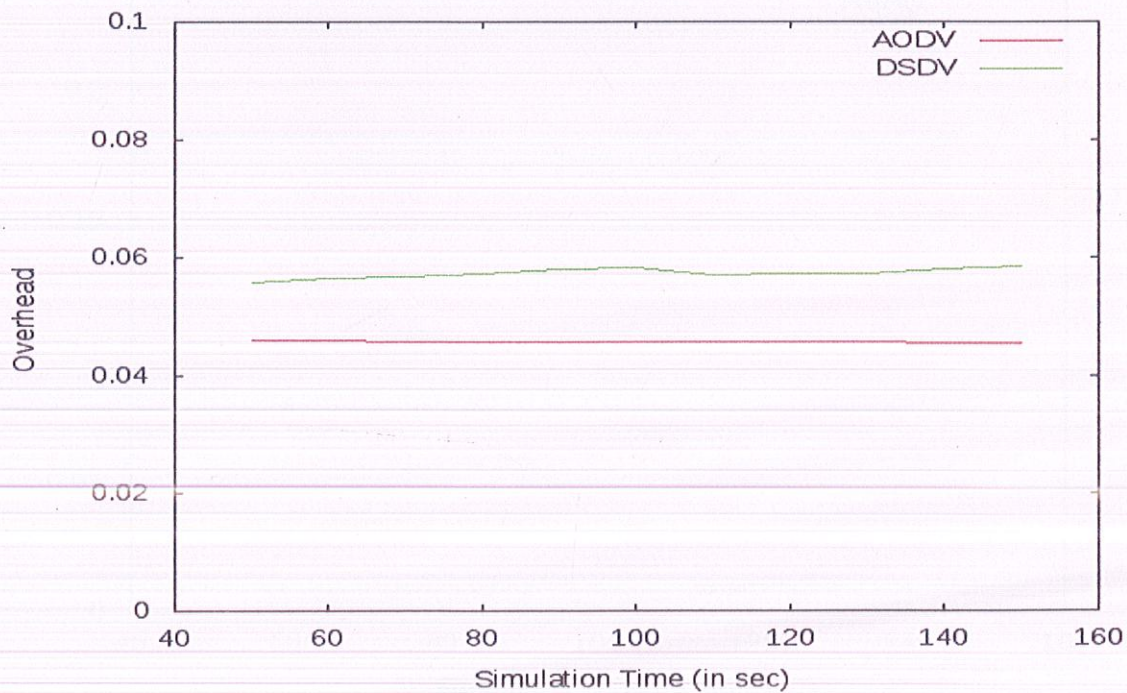
2. Overhead

No of nodes: 20; Speed: 1m/s



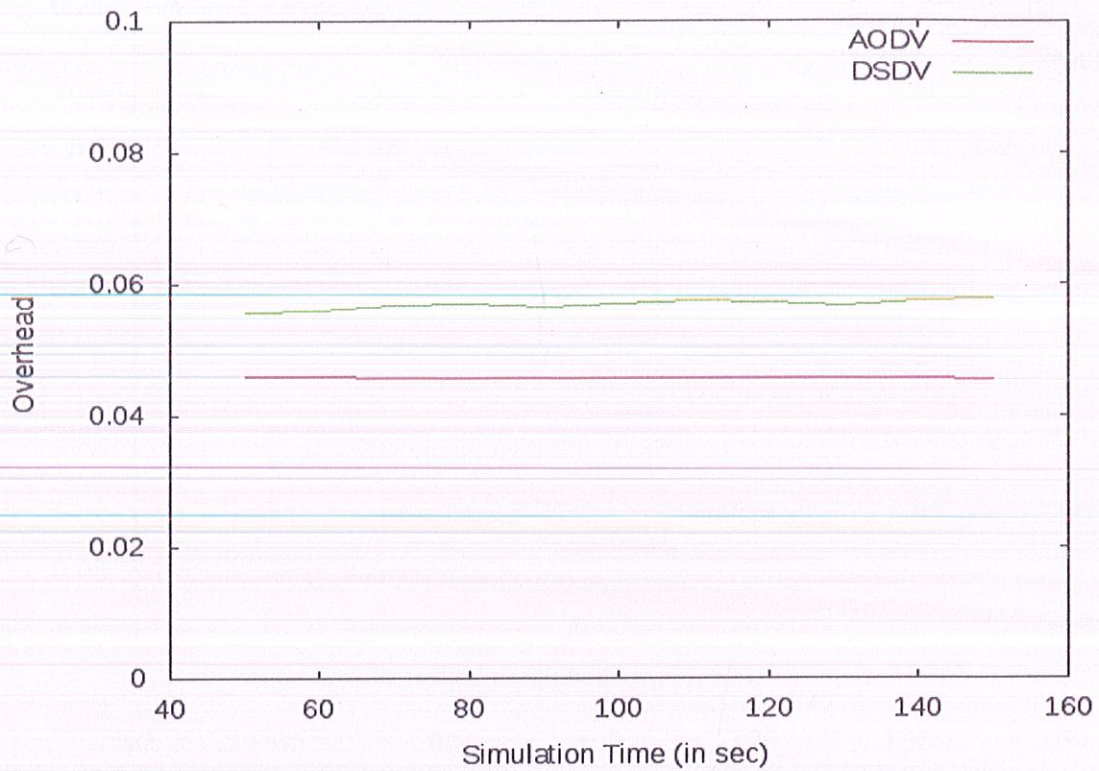
Graph 6.1.2.7: Overhead _20_1

No of nodes: 20; Speed: 5m/s



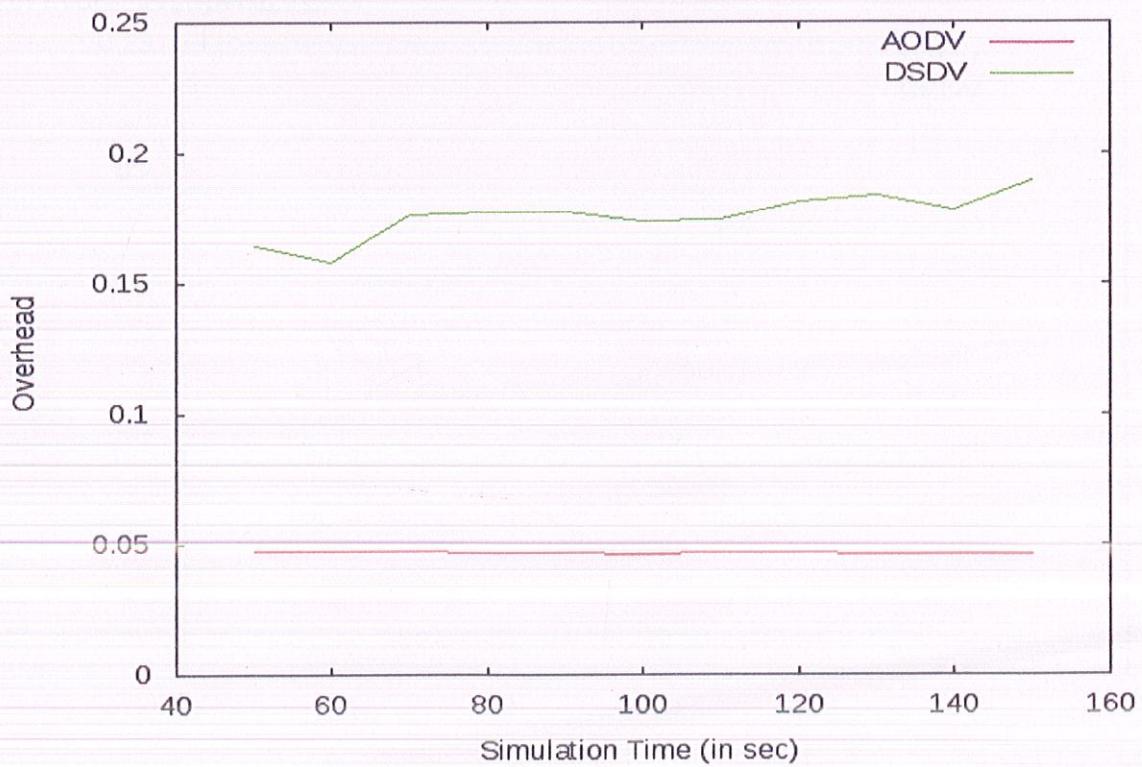
Graph 6.1.2.8: Overhead _20_5

No of nodes: 20; Speed: 10m/s



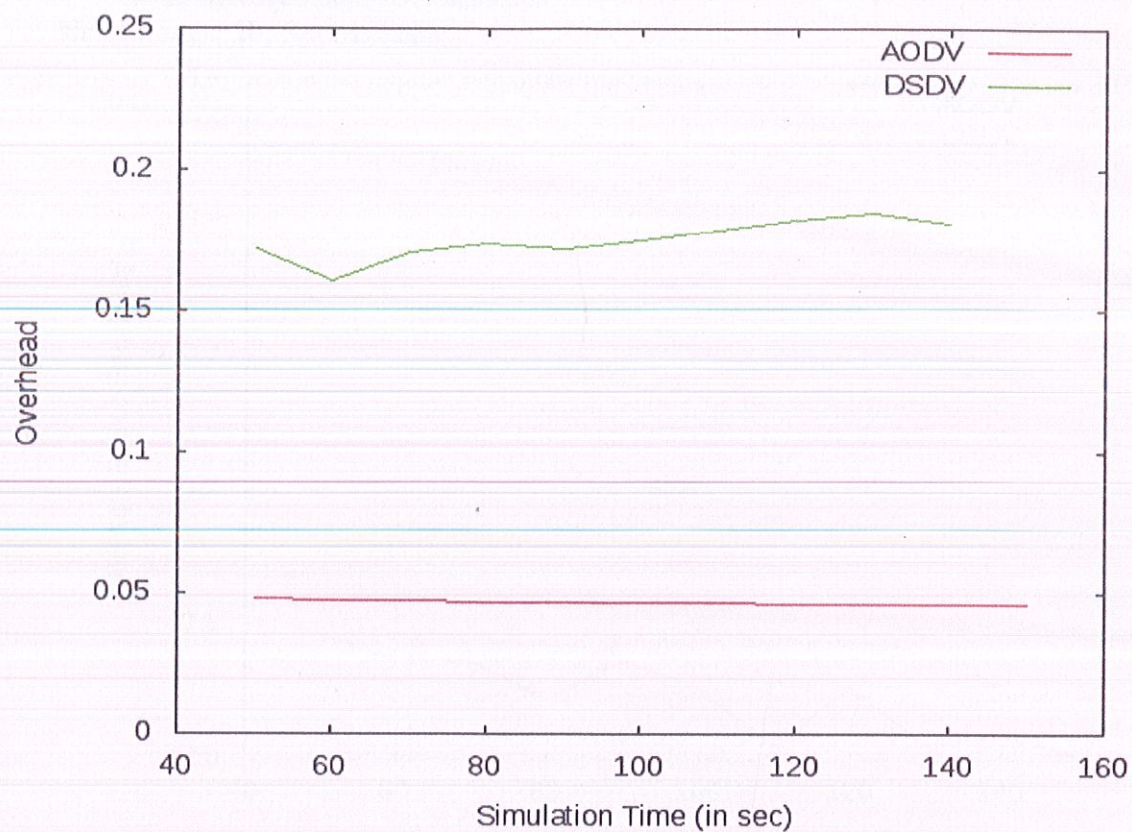
Graph 6.1.2.9: Overhead _20_10

No of nodes 50; Speed: 1m/s



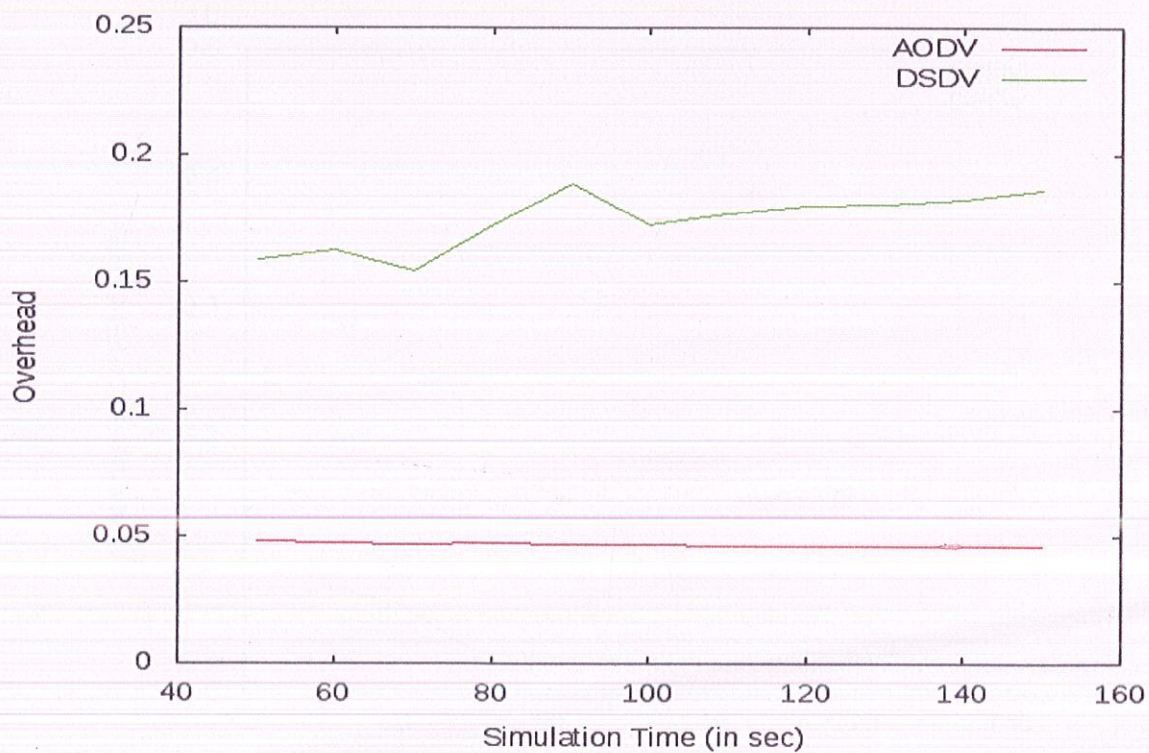
Graph 6.1.2.10: Overhead _50_1

No of nodes: 50; Speed: 5m/s



Graph 6.1.2.11: Overhead_50_5

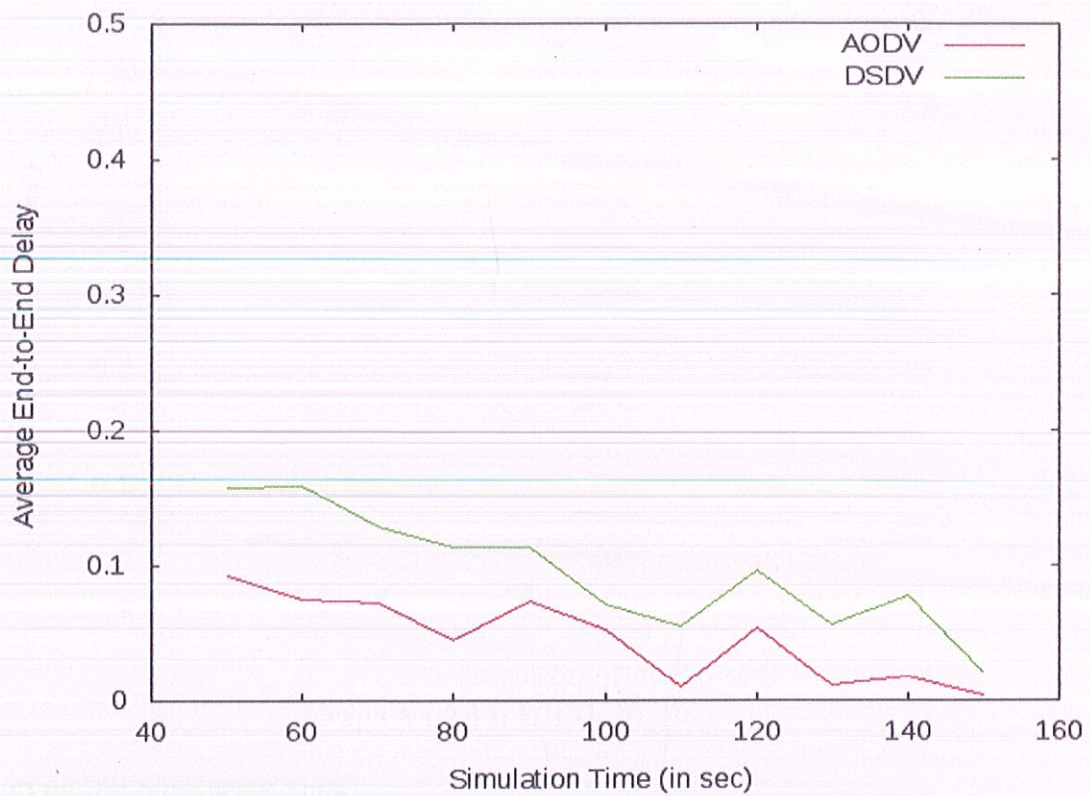
No of nodes: 50; Speed: 10m/s



Graph 6.1.2.12: Overhead_50_10

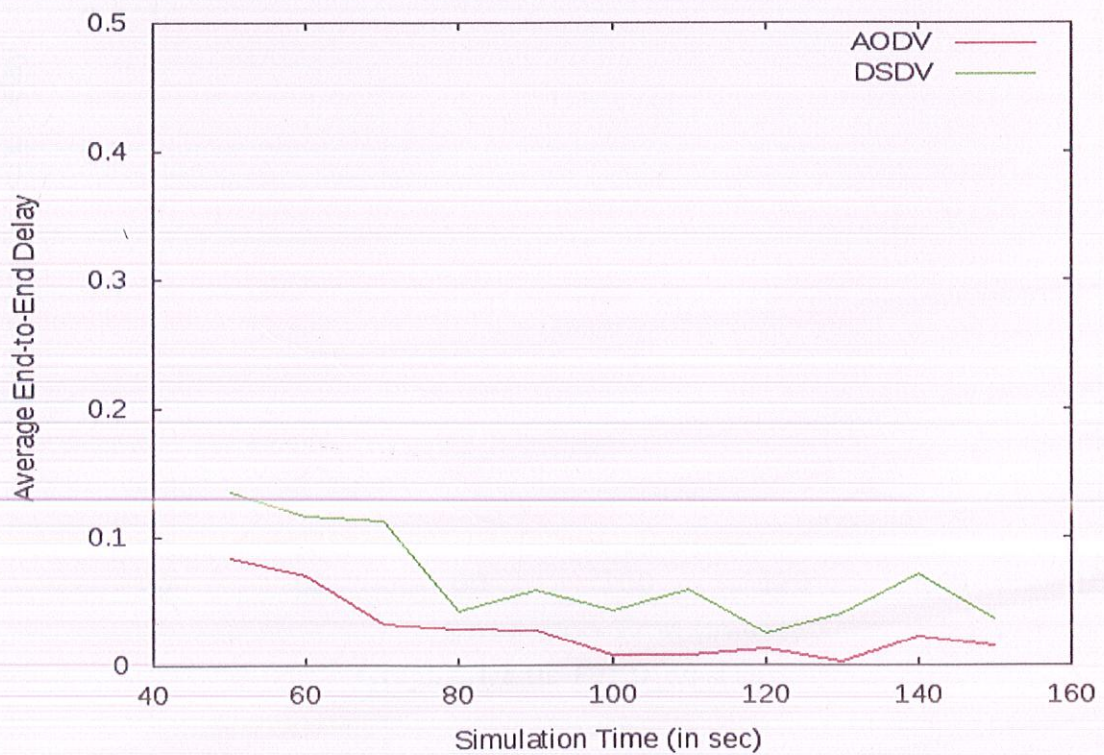
3. Average End to End Delay

No of nodes: 20; Speed: 1m/s



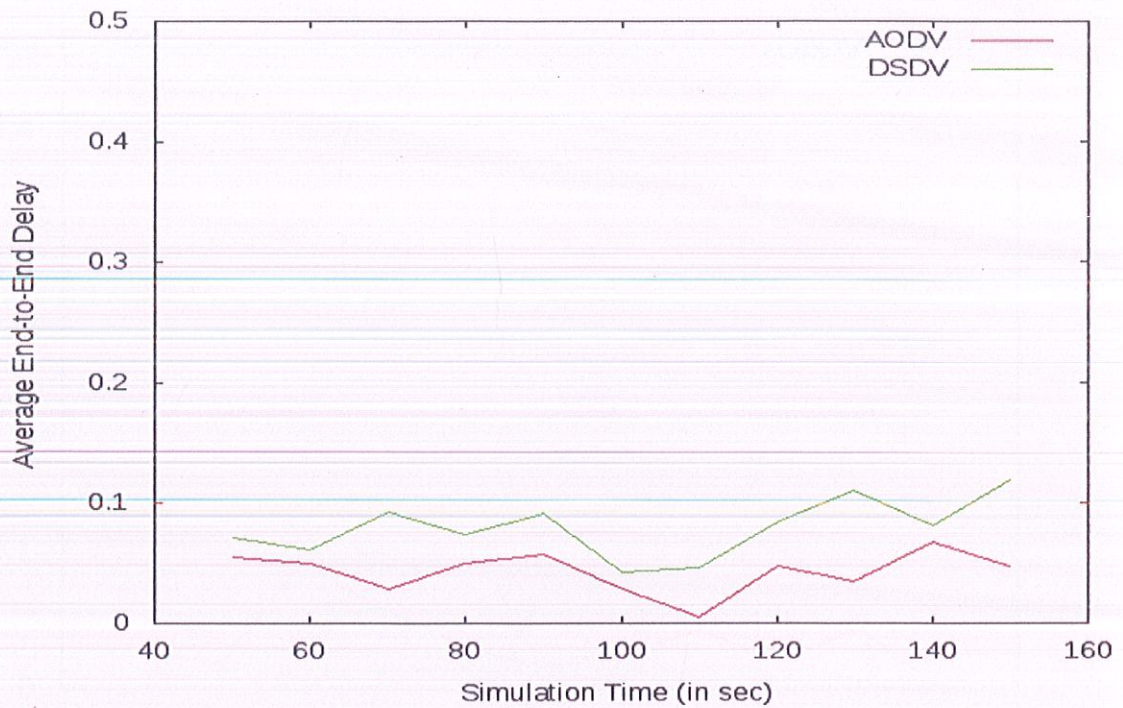
Graph 6.1.2.13: E2ED_20_1

No of nodes: 20; Speed: 5m/s



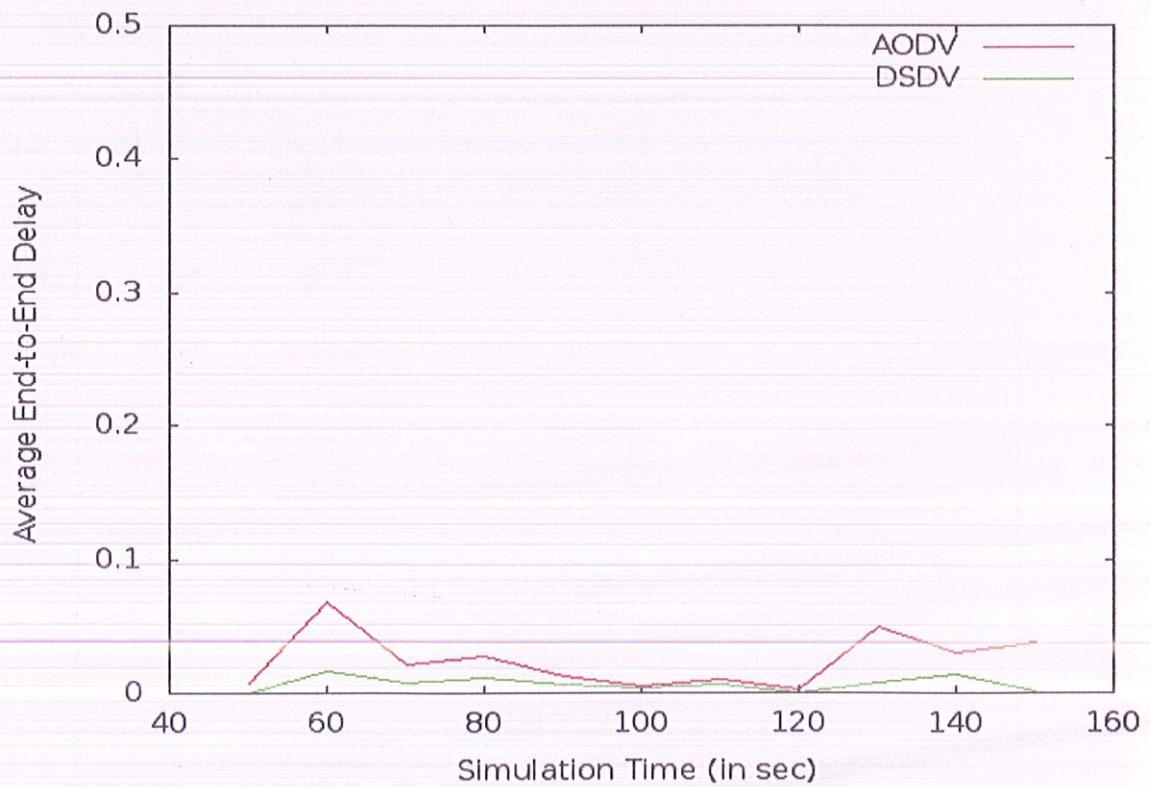
Graph 6.1.2.14: E2ED_20_5

No of nodes: 20; Speed: 10m/s



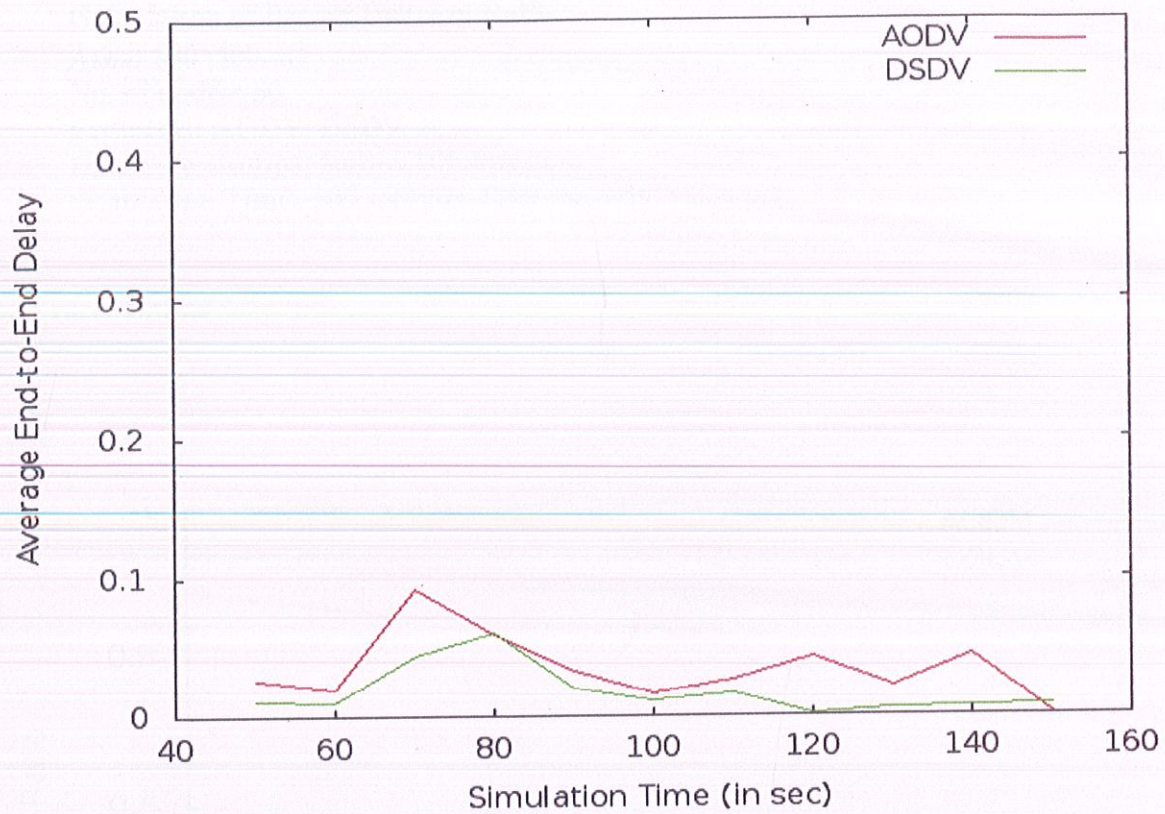
Graph 6.1.2.15: E2ED_20_10

No of nodes: 50; Speed: 1m/s



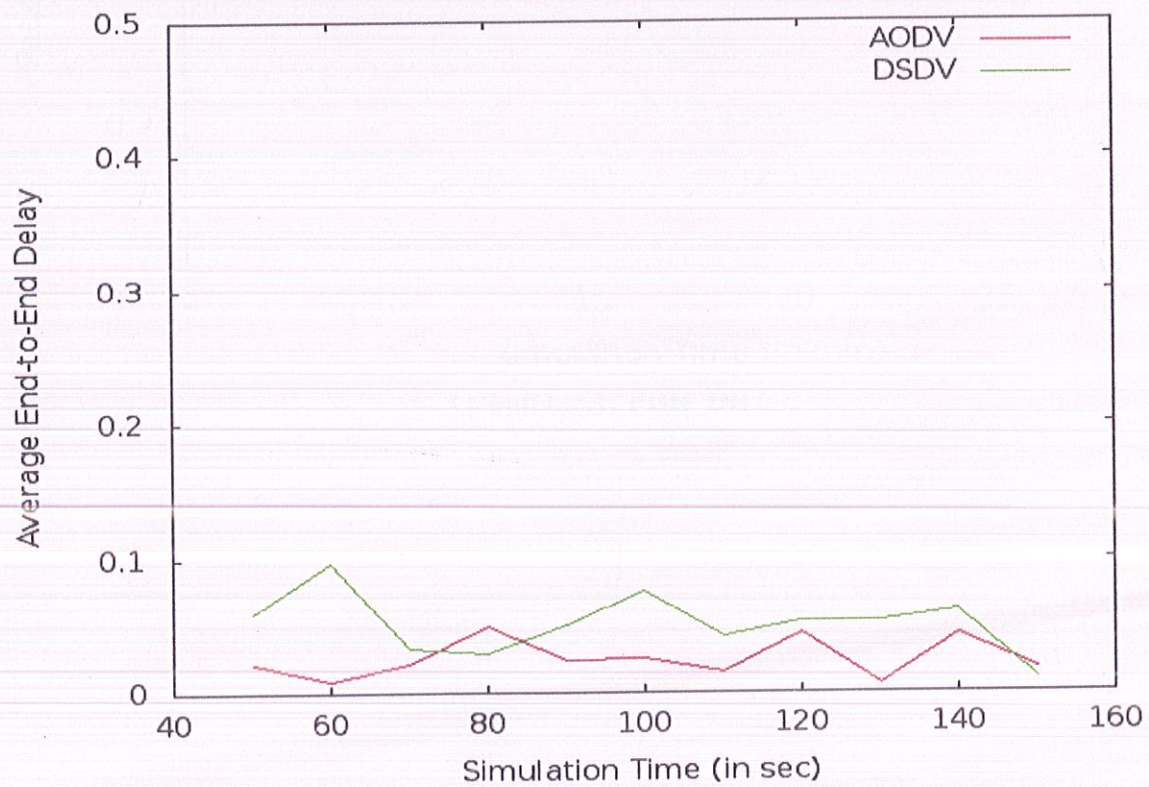
Graph 6.1.2.16: E2ED_50_1

No of nodes: 50; Speed: 5m/s



Graph 6.1.2.17: E2ED_50_5

No of nodes: 50; Speed: 10m/s



Graph 6.1.2.18: E2ED_50_10

6.2 SCENARIO: 2

Calculation of Packet Delivery Ratio

Area: $500 \times 500 \text{ m}^2$

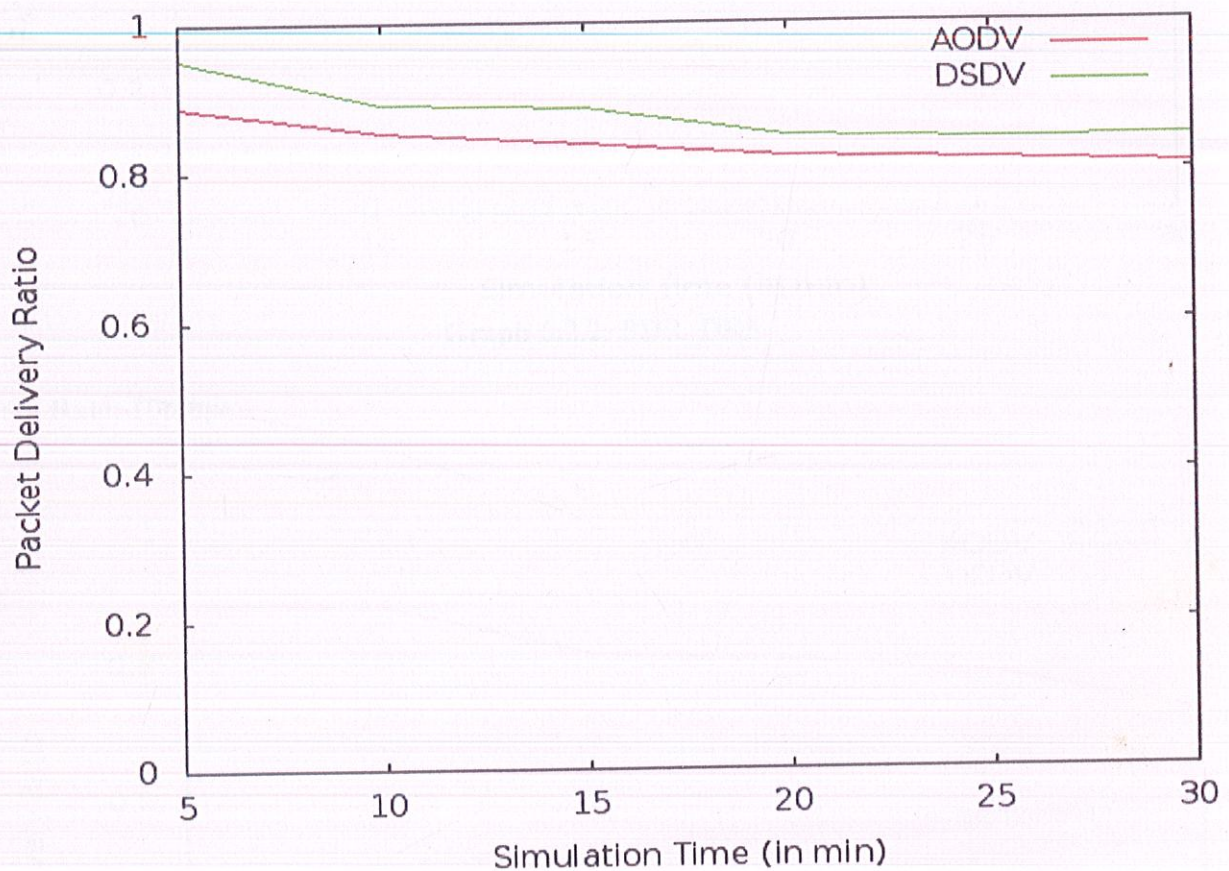
No. of nodes: 50

Protocols: AODV, DSDV

Data rate: 1mbps, 5mbps, 10mbps

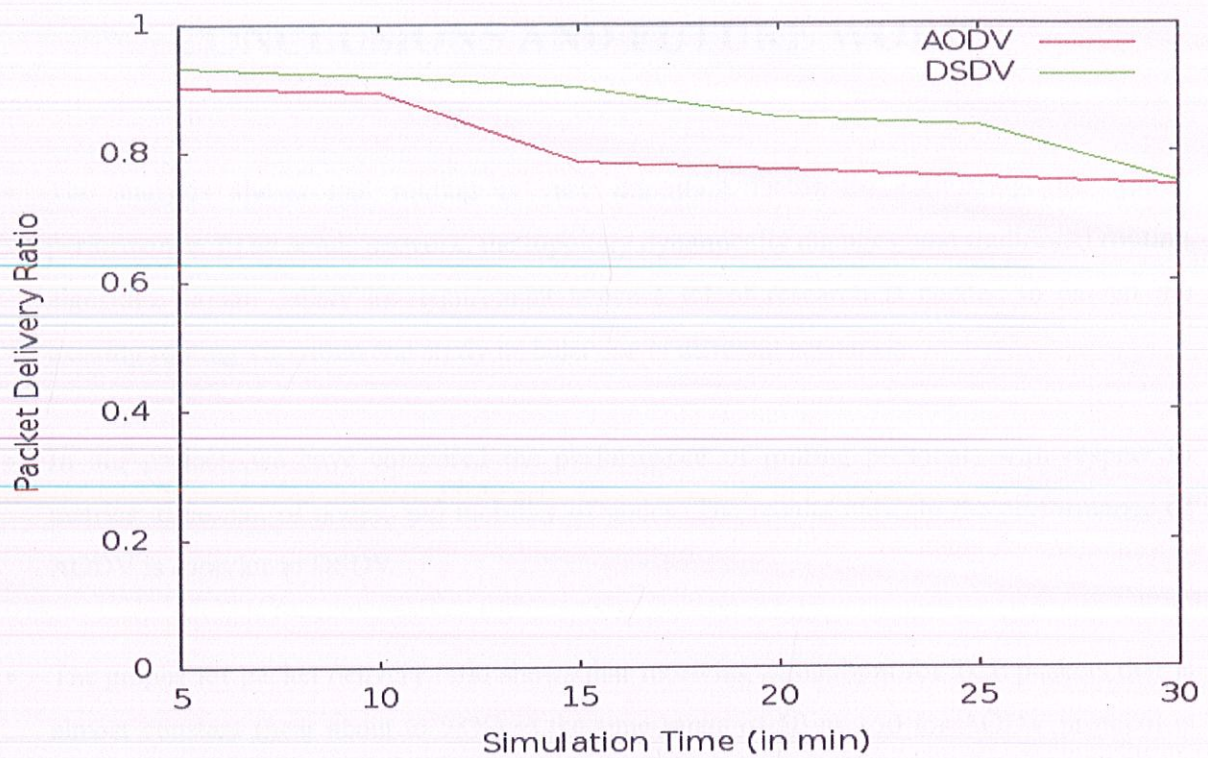
Simulation Time: 600 seconds-1800 seconds

Data rate: 1mbps



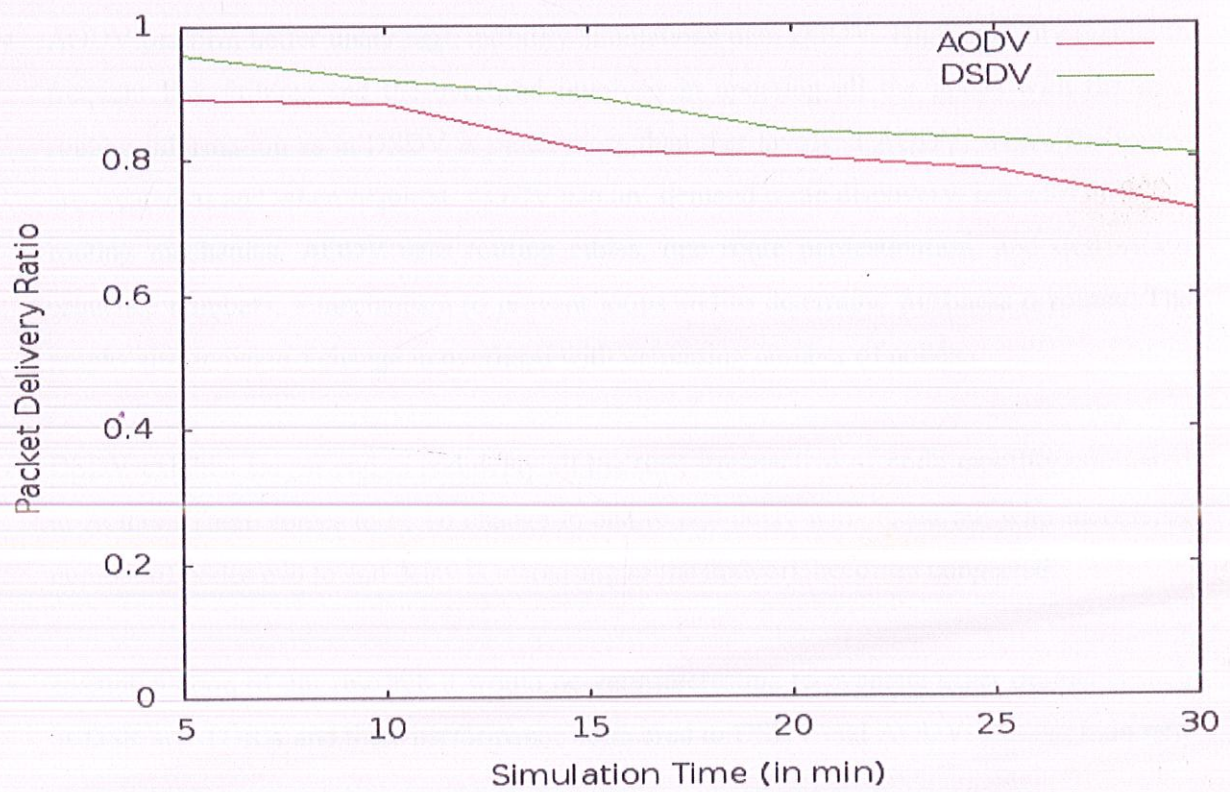
Graph 6.2.1: PDR_DR1

Data Rate: 5mbps



Graph 6.2.2: PDR_DR5

Data Rate: 10mbps



Graph 6.2.3: PDR_DR10

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

- The analysis shows that routing is very important factor for estimating the system performance. In an adhoc network, the topology dynamically changes, and traditional routing algorithm cannot satisfy its requirement hence a lot of research is needed to extend the existing routing algorithm and study its behavior in different scenarios.
- In our project, we have compared the performance of routing protocols with respect to metrics: time, no. of nodes, and mobility of nodes. The results indicate the performance of AODV is superior to DSDV.
- The graphs for packet delivery ratio shows that receiving throughput for TCP packets that is almost constant (near about to 98%) in the time range of 50 to 150 for AODV protocol is higher than DSDV. With increasing number of nodes PDR for both the protocols is decreasing. Speed of node has less impact on PDR. As the simulation is run for a longer time (5min-30min), the Packet Delivery Ratio of both the protocols starts to fall.
- AODV perform better under high mobility simulations than DSDV. High mobility results in frequent link failures and the overhead involved in updating all the nodes with the new routing information as in DSDV is much more than that involved AODV, where the routes are created as and when required. AODV use on -demand route discovery, but with different routing mechanics. AODV uses routing tables, one route perdestination, and destination sequence numbers, a mechanism to prevent loops and to determine freshness o routes. The graphs also indicate a change in overhead with increasing number of nodes.
- DSDV exhibits longer end to end delay all the time irrespective of node mobility compared to AODV. There comes to be no change in end to end delay with speed but with increasing number of nodes end to end delay is increasing as the network becomes congested.
- In continuation of our research it would be very interesting to evaluate other protocols such as DSR and TORA and there performance compared to DSDV and AODV. Research on new

simulation environment and calculating other performance parameters could also be undertaken.

APPENDIX

```
# Define options
setval(chan) Channel/WirelessChannel    ;# channel type
setval(prop) Propagation/TwoRayGround   ;# radio-propagation model
setval(netif) Phy/WirelessPhy           ;# network interface type
setval(mac) Mac/802_11                  ;# MAC type
setval(ifq) Queue/DropTail/PriQueue     ;# interface queue type
setval(ll) LL                           ;# link layer type
setval(ant) Antenna/OmniAntenna         ;# antenna model
setval(ifqlen) 50                       ;# max packet in ifq
setval(nn) 50                           ;# number of mobilenodes
setval(rp) AODV                          ;# routing protocol
setval(x) 500                           ;# X dimension of topography
setval(y) 500                           ;# Y dimension of topography
setval(stop) 1800                       ;# time of simulation end
setval(Pt) 50                           ;#Transmission Power/Range in meters
```

```
set ns [new Simulator]
settracefd [open simple.tr w]
setnamtrace [open simwrlds.nam w]
```

```
$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)
```

```
# set up topography object
settopo [new Topography]
$topoload_flatgrid $val(x) $val(y)
```

```
create-god $val(nn)
```

```
# configure the nodes
$ns node-config -adhocRouting $val(rp) \
    -llType $val(ll) \
    -macType $val(mac) \
    -ifqType $val(ifq) \
    -ifqLen $val(ifqlen) \
    -antType $val(ant) \
    -propType $val(prop) \
    -phyType $val(netif) \
    -channelType $val(chan) \
    -TransmissionRange $val(Pt) \
    -topoInstance $topo \
    -agentTrace ON \
    -routerTrace ON \
    -macTrace OFF \
    -movementTrace ON
```

```

for {set i 0} {$i < $val(nn) } { incr i } {
set xx [expr rand()*500]
set yy [expr rand()*400]
set n($i) [$ns node]
    $n($i) set X_ $xx
    $n($i) set Y_ $yy
    $n($i) set Z_ 0.0
    #Phy/WirelessPhy set Pt_ =7.214e-4; #100 meters
}

```

Set a TCP connection between n(2) and n(11)

```

settcp [new Agent/TCP]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $n(2) $tcp
$ns attach-agent $n(11) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.0 "$ftp start"

```

```

set udp0 [new Agent/UDP]
$ns attach-agent $n(2) $udp0

```

```

set cbr0 [new Application/Traffic/CBR]
#$cbr0 set packetSize_ 500
#$cbr0 set interval_ 0.0005
$cbr0 attach-agent $udp0

```

Set a TCP connection between n(10) and n(8)

```

settcp [new Agent/TCP]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $n(10) $tcp
$ns attach-agent $n(8) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.0 "$ftp start"

```

```

set udp1 [new Agent/UDP]
$ns attach-agent $n(10) $udp1

```

```

set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.0005
$cbr1 attach-agent $udp1

```

```
# Set a TCP connection between n(12) and n(17)
settcp [new Agent/TCP]
$tcp set class_ 3
set sink [new Agent/TCPSink]
$ns attach-agent $n(12) $tcp
$ns attach-agent $n(17) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.0 "$ftp start"
```

```
set udp2 [new Agent/UDP]
$ns attach-agent $n(12) $udp2
```

```
set cbr2 [new Application/Traffic/CBR]
#$cbr2 set packetSize_ 500
#$cbr2 set interval_ 0.0005
$cbr2 attach-agent $udp2
```

```
# Set a TCP connection between n(20) and n(40)
settcp [new Agent/TCP]
$tcp set class_ 4
set sink [new Agent/TCPSink]
$ns attach-agent $n(20) $tcp
$ns attach-agent $n(40) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.0 "$ftp start"
```

```
set udp3 [new Agent/UDP]
$ns attach-agent $n(20) $udp3
```

```
set cbr3 [new Application/Traffic/CBR]
#$cbr3 set packetSize_ 500
#$cbr3 set interval_ 0.0005
$cbr3 attach-agent $udp3
```

```
# Set a TCP connection between n(25) and n(1)
settcp [new Agent/TCP]
$tcp set class_ 5
set sink [new Agent/TCPSink]
$ns attach-agent $n(25) $tcp
$ns attach-agent $n(1) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.0 "$ftp start"
```

```
set udp4 [new Agent/UDP]
$ns attach-agent $n(25) $udp4
```

```
set cbr4 [new Application/Traffic/CBR]
#$cbr4 set packetSize_ 500
#$cbr4 set interval_ 0.0005
$cbr4 attach-agent $udp4
```

```
# Set a TCP connection between n(45) and n(39)
settcp [new Agent/TCP]
$tcp set class_ 6
set sink [new Agent/TCPSink]
$ns attach-agent $n(45) $tcp
$ns attach-agent $n(39) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.0 "$ftp start"
```

```
set udp5 [new Agent/UDP]
$ns attach-agent $n(45) $udp5
```

```
set cbr5 [new Application/Traffic/CBR]
#$cbr5 set packetSize_ 500
#$cbr5 set interval_ 0.0005
$cbr5 attach-agent $udp5
```

```
# Set a TCP connection between n(29) and n(30)
settcp [new Agent/TCP]
$tcp set class_ 7
set sink [new Agent/TCPSink]
$ns attach-agent $n(29) $tcp
$ns attach-agent $n(30) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.0 "$ftp start"
```

```
set udp6 [new Agent/UDP]
$ns attach-agent $n(29) $udp6
```

```
set cbr6 [new Application/Traffic/CBR]
#$cbr6 set packetSize_ 500
#$cbr6 set interval_ 0.0005
$cbr6 attach-agent $udp6
```

```
#$ns at 0.0 "$n(0) setdest 49 435 5"
#$ns at 0.0 "$n(1) setdest 100 250 5"
```

```

#Destination procedure..
$ns at 0.0 "destination"
proc destination {} {
    global ns val n
    set time 1.0
    set now [$ns now]
    for {set i 0} {$i < $val(nn)} {incr i} {
        set xx [expr rand()*490]
        setyy [expr rand()*490]
        #set s [expr rand()*10]
        $ns at $now "$n($i) setdest $xx $yy 1"
    }
    # $ns at [expr $now + $time] "destination"
}

```

```

# Define node initial position in nam
for {set i 0} {$i < $val(nn)} {incr i} {
    # 20 defines the node size for nam
    $ns initial_node_pos $n($i) 20
}

```

```

# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn)} {incr i} {
    $ns at $val(stop) "$n($i) reset";
}

```

```

# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 1800.1 "puts \"end simulation\" ; $ns halt"

```

```

#Graph procedure..
$ns at 1.0 "Graph"
set g [open simple.tr w]
proc Graph {} {
    global ns g
    set time 1.0
    set now [$ns now]
    puts $g "[expr rand()*8] [expr rand()*6]"

    $ns at [expr $now+$time] "Graph"
}

```

```

proc stop {} {

```

```
global ns tracefdnamtrace
  $ns flush-trace
close $tracefd
close $namtrace
execnamsimwrls.nam&
  exit 0
}

$ns run
```

REFERENCES

- [1] Alex Ali Hamidian "A study of Internet Connectivity for Mobile Ad-hoc Networks" Department of Communication Systems Lund Institute of Technology, Lund University Sweden, January 2003
- [2] Charles E. Perkins and Elizabeth M. Royer. "Ad hoc on-demand distance vector routing." In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 80–100. IEEE, February 1999.
- [3] David B. Johnson. "Routing in Ad Hoc Networks of Mobile Hosts". In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, pages 158–163. IEEE Computer Society, December 1994.
- [4] Vijayalakshmi M. et. al. QOS PARAMETER ANALYSIS ON AODV AND DSDV PROTOCOLS IN A WIRELESS NETWORK / Indian Journal of Computer Science and Engineering Vol. 1 No. 4 283-294
- [5] David A. Maltz, Josh Broch, and David B. Johnson. "Experiences Designing and Building a Multi-Hop Wireless Ad Hoc Network Testbed". Technical Report CMU-CS-99-116, School of Computer Science, Carnegie Mellon University.
- [6] David A. Maltz, Josh Broch, Jorjeta Jetcheva, and David B. Johnson. "The Effects of On-Demand Behavior in Routing Protocols for Multi-Hop Wireless Ad Hoc Networks" *IEEE Journal on Selected Areas of Communications*, 17(8):1439–1453, August 1999. Pittsburgh, Pennsylvania, March 1999.
- [7] M.S. Corson and A. Ephremides. "A Distributed Routing Algorithm for Mobile Wireless Networks." *ACM / Baltzer Wireless Networks Journal* 1(1):61-82, February 1995.
- [8] Gafni and Bertsekas. "Distributed Algorithms for Generating Loop-free Routes in Networks with Frequently Changing Topology" *IEEE Transactions on Communications* 29(1):11-15, January 1981.
- [9] Schiller J. *Mobile Communications*, 2000.
- [10] V. Park and M.S. Corson. "A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks." In *Proceedings of IEEE INFOCOM '97*, April 1997.

- [11] V.Ramesh et al. Performance Comparison and Analysis of DSDV and AODV for MANET / (IJCSE) International Journal on Computer Science and Engineering Vol. 02, No. 02, 183-188, 2010
- [12] Wakikawa R.; Malinen J.; Perkins C.; Nilsson A.; Tuominen A.J. *Global Connectivity for IPv6 Mobile Ad Hoc Networks*, IETF Internet Draft, November 2001.
- [13] <http://www.monarch.cs.cmu.edu/>
- [14] <http://www.isi.edu/nsnam/ns/>
- [15] http://nsnam.isi.edu/nsnam/index.php/Main_Page
- [16] [http://en.wikipedia.org/wiki/Ns_\(simulator\)](http://en.wikipedia.org/wiki/Ns_(simulator))
- [17] <http://nile.wpi.edu/NS/>
- [18] <http://en.wikipedia.org/wiki/AWK>
- [19] <http://www.folkstalk.com/2011/12/good-examples-of-awk-command-in-unix.html>