OPTIMIZING THE PERFORMANCE OF AD-HOC ROUTING PROTOCOLS USING HIERARCHICAL APPROACH

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

विद्या तत्व ज्योतिसम:

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

Ravinder Kumar 051003

Chinmay Dubey 051007

Shalender Rathor 051023

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,

WAKNAGHAT

MAY 2009

OPTIMIZING THE PERFORMANCE OF AD-HOC ROUTING PROTOCOLS USING HIERARCHICAL APPROACH

Submitted in partial fulfilment of the Degree of Bachelors of Technology

CERTIFICATE

This is to certify that the work entitled "Optimizing the performance of ad-hoc Routing Protocols using hierarchical approach" submitted by Ravinder Kumar, Chinmay Dubey and Shalender Rathor in fulfilment for the award of Degree of Bachelor of Technology in 2009 of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

[illegible]

Dr. D.S. Saini

Asst. Prof. ECE Department

ACKNOWLEDGMENT

We would like to express our gratitude to all those who gave us support and guidance to complete a part of our project.

We are deeply indebted to our Project mentor Asst. Prof. Dr. D.S. Saini, Department of Electronics and Communication, whose help, stimulating suggestions and encouragement helped us in all the stages of the project. His enthusiasm and his views for providing only "high quality work and not less", has made a deep impression on all of us.

[illegible]
Ravinder Kumar

[illegible]
Chinmay Dubey

[illegible]
Shalender Rathor

ABSTRACT

This project **"Optimizing the Performance of ad-hoc Routing Protocols using Hierarchical approach"** is a part of our B.Tech curriculum at Jaypee University of Information Technology, Solan.

In this project we have presented an approach to optimize the performance of existing ad-hoc routing protocols using the hierarchical approach for the network. For the purpose of designing and comparison we have used the simulator known as **NS2** (Network Simulator version 2).

The overhead in hierarchical approach is less because special nodes (cluster heads) are used to handle different group of users. This makes the routing table size small for each of the node but with the limitation of non optimum path possibility between two different nodes. This non optimum path gives rise to more bandwidth consumption as compared to other novel methods like AODV, DSR and DSDV. The existing ad-hoc routing protocols (DSDV, DSR and AODV etc.) can be the obvious choice networks but these algorithms suffers from large buffer requirement and large route discovery overhead. The design proposed in this paper use clustering technique along with DSDV or AODV to reduce routing overhead and improve packet delivery ratio for bigger networks.

CONTENTS

List of Figures

List of Abbreviations

AODV	Ad hoc On Demand Distance Vector
AGT	Agent Trace
CBR	Constant Bit Rate
DSDV	Destination-Sequenced Distance-Vector Routing
DSR	Dynamic Source Routing
HAODV	Hierarchical approach used with AODV
HDSDV	Hierarchical approach used with DSDV
MAC	Medium Access Control
MANET	Mobile Ad-Hoc Network
MH	Mobile Host (node)
NAM	Network Animator
NPDU	Network Protocol Data Units
NS	Network Simulator
OTcl	Object oriented Tool Control Language
PDU	Protocol Data Unit
RERR	Route Error
RREP	Route Reply
RREQ	Route Request
RTR	Router Trace
Tcl	Tool Command Protocol
TCP	Transmission control protocol
TTL	Time to Live
UDP	User Datagram protocol

Chapter-1
Project Outline

1.1 Ad-hoc networks

A wireless ad-hoc network is a decentralized wireless network. The network is ad-hoc because each node is willing to forward data for other nodes, and so the determination of which nodes forward data is made dynamically based on the network connectivity. This is in contrast to wired networks in which routers perform the task of routing. It is also in contrast to managed (infrastructure) wireless networks, in which a special node known as an access point manages communication among other nodes.

A variety of routing algorithms for ad hoc networks have been proposed. The most commonly used ad-hoc routing protocols include DSR, DSDV and AODV.

1.2 Aim of Project

In ad-hoc routing the main area of concern is the throughput and the overhead packet traffic in the network. Our aim of the project is to improve the values of packet delivery ratio for different pause times and to reduce the routing overhead.

1.3 Tool Used (NS2)

The tool used in our research work is NS2 (Network Simulator 2). It is a Linux based open source simulator, quiet popular for research work in the field of ad-hoc wireless networks.

1.4 Objective

Our main objective is to provide a qualitative performance comparison between the existing routing protocols and the design proposed to get better Packet delivery ratio and Routing overhead. (The comparison is done at different pause time and also with different network densities)

Chapter-2
Network simulator-2

2.1 Introduction

NS (version 2) is an object-oriented, discrete event driven network simulator developed at UC Berkeley written in C++ and OTcl (Tcl script language with Object-oriented extensions). It implements network protocols such as TCP and UPD; traffic source behaviour such as FTP, Telnet, Web, CBR; 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.

NS-2 includes a tool for viewing the simulation results, called NAM. NAM is a Tcl/TK based animation tool for viewing network simulation traces and real world packet trace data. The first step to use NAM is to produce the trace file. The trace file should contain topology information, e.g., nodes, links, as well as packet traces. Usually, the trace file is generated by NS. During an NS simulation, user can produce topology configurations, layout information, and packet traces using tracing events in ns. When the trace file is generated, it is ready to be animated by NAM. Upon start-up, NAM will read the trace file, create topology, pop up a window, do layout if necessary, and then pause at the time of the first packet in the trace file. Through its user interface, NAM provides control over many aspects of animation. The NS simulator covers a very large number of applications, of protocols of network types, of network elements and of traffic models.

2.1.1 OTcl Linkage

NS is an object oriented simulator, written in C++, with an OTcl interpreter as a frontend. The simulator supports a class hierarchy in C++ (also called the compiled hierarchy in this document), and a similar class hierarchy within the OTcl interpreter (also called the interpreted hierarchy in this document). The two hierarchies are closely related to each other; from the user's perspective, there is a one-to-one correspondence between a class in the interpreted hierarchy and one in the compiled hierarchy. The root of this hierarchy is the class TclObject.

Users create new simulator objects through the interpreter; these objects are instantiated within the interpreter, and are closely mirrored by a corresponding object in the compiled hierarchy. The interpreted class hierarchy is automatically established through methods defined in the class TclClass. User instantiated objects are mirrored through methods defined in the class TclObject. There are other hierarchies in the C++ code and OTcl scripts; these other hierarchies are not mirrored in the manner of TclObject.

2.1.2 Why two languages?

NS uses two languages because simulator has two different kinds of things it needs to do. On one hand, detailed simulations of protocols require a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important.

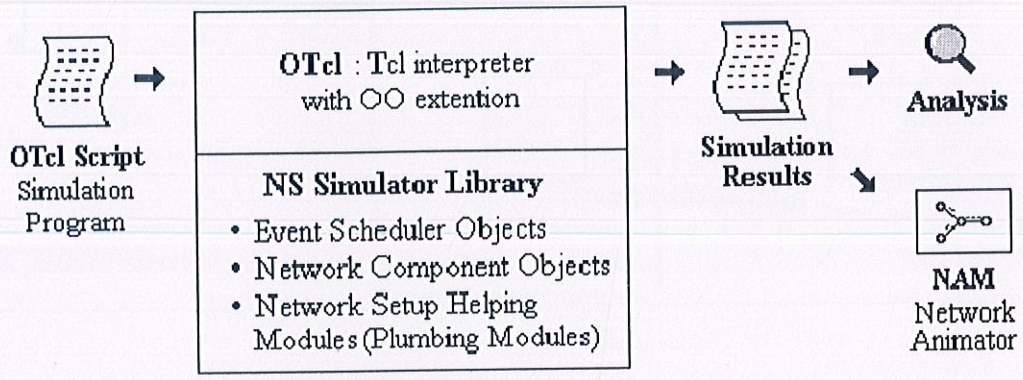
On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios. In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important.

NS meets both of these needs with two languages, C++ and OTcl. C++ is fast to run but slower to change, making it suitable for detailed protocol implementation. OTcl runs much slower but can be changed very quickly (and interactively), making it ideal for simulation configuration. NS (via tclcl) provides glue to make objects and variables appear on both languages. Tcl (Tool Command Language) was developed by Jhon ousterhout and have following characteristics:

- It allows a fast development
- It provide a graphical interface
- It is compatible with many platforms
- It is flexible for integration
- It is easy to use
- It is free

2.2 Modelling in NS-2

To model a Network simulating using NS-2 is necessary to write a Tcl script describing the topology (nodes, agents, applications, etc.), save it and invoke the NS2 interpreter. After that NS- 2 will simply store the results in form of trace files. We can visualize the result by using Network Animator (NAM) and xgraph (to see the result graphically), as shown in figure.



OTcl Script
Simulation Program
OTcl : Tcl interpreter with OO extention
NS Simulator Library
• Event Scheduler Objects
• Network Component Objects
• Network Setup Helping Modules (Plumbing Modules)
Simulation Results
Analysis
NAM
Network Animator

Fig 2.1 Modelling of NS-2

2.2.1 The NAM Visualisation Tool

Nam is a Tcl/TK based animation tool for viewing network simulation traces and real world packet trace data. The design theory behind NAM was to create an animator that is able to read large animation data sets and be extensible enough so that it could be used indifferent network visualization situations. Under this constraint NAM was designed to read simple animation event commands from a large trace file. In order to handle large animation data sets a minimum amount of information is kept in memory. Event commands are kept in the file and reread from the file whenever necessary.

The first step to use NAM is to produce the trace file. The trace file contains topology information, e.g., nodes, links, as well as packet traces. Usually, the trace file is generated by ns. During an NS simulation, user can produce topology configurations, layout information, and packet traces using tracing events in ns. However any application can generate a NAM trace file.

When the trace file is generated, it is ready to be animated by NAM. Upon start-up, NAM will read the trace file, create topology, pop up a window, do layout if necessary, and then pause at time 0. Through its user interface, NAM provides control over many aspects of animation.

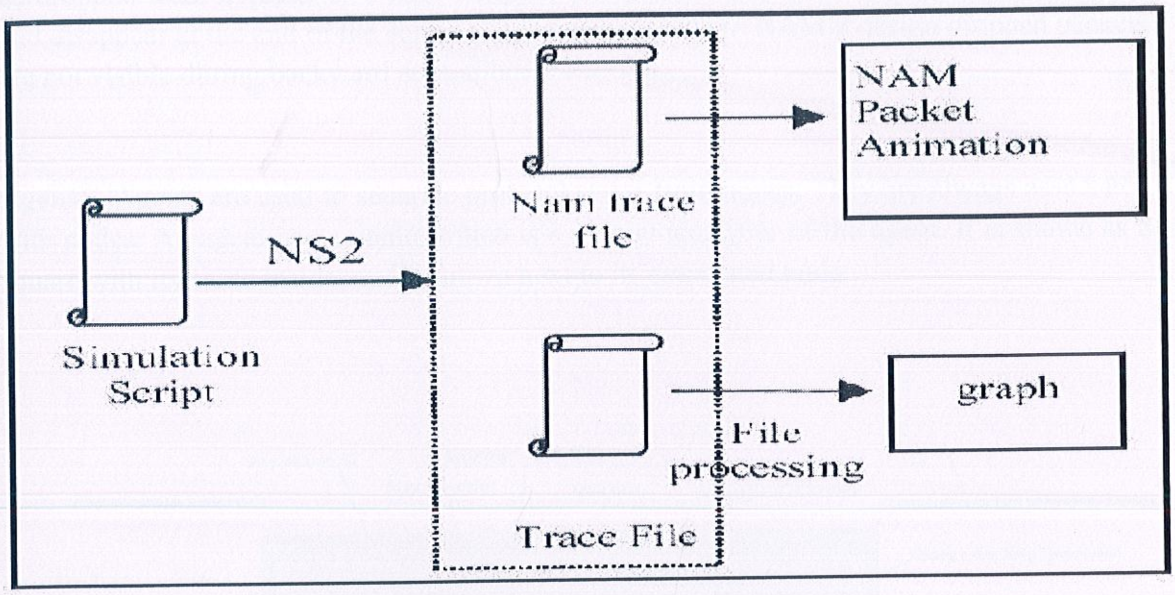


Fig 2.2 Figure showing the elements of NS-2

2.2.1a Animation Objects

Nam does animation using the following building blocks which are defined below:

Node: Nodes are created from 'n' trace event in trace file. It represents a source, host, or router. Nam will skip over any duplicate definitions for the same node. A node may have three shapes, (circle, square, and hexagon), but once created it cannot change its shape. Nodes can change its colour during animation. Nodes can be labelled.

Link: Links are created between nodes to form a network topology. Internally NAM links are consisting of 2 simplex links. The trace event 'l' creates two simplex links and does other necessary setup. Therefore, for a user's perspective all links are duplex links. Links can be labelled and also can change colour during the animation. Links cab is labelled as well.

Queue: Queues need to be constructed in NAM between two nodes. A NAM queue is associated to only one edge of a duplex link. Queues are visualized as stacked packets. Packets are stacked along a line, the angle between the line and the horizontal line can be specified in the queue trace event.

Packet: Packets are visualized as a block with an arrow. The direction of the arrow shows the flow direction of the packet. Queued packets are shown as little squares. A packet may be dropped from a queue or a link. Dropped packets are shown as falling rotating squares, and disappear at the end of the screen. Unfortunately, due to NAM's design dropped packets are not visible during backward animation.

Agent: Agents are used to separate protocol states from nodes. They are always associated with nodes. An agent has a name, which is a unique identifier of the agent. It is shown as a square with its name inside, and is drawn next to its associated node.

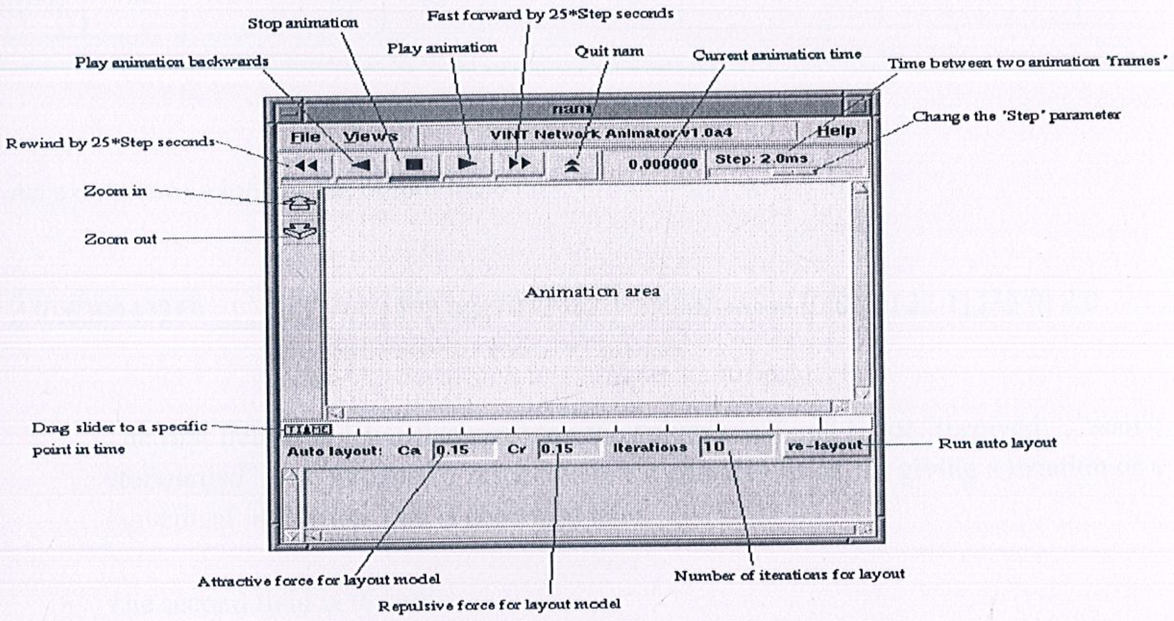

Stop animation
Fast forward by 25*Step seconds
Play animation
Quit nam
Current animation time
Play animation backwards
Time between two animation 'frames'
Change the 'Step' parameter
Rewind by 25*Step seconds
Zoom in
Zoom out
nam
File
Views
VINT Network Animator v1.0a4
Help
0.000000
Step: 2.0ms
Animation area
Drag slider to a specific point in time
Auto layout: Ca 0.15 Cr 0.15 Iterations 10 re-layout
Run auto layout
Attractive force for layout model
Repulsive force for layout model
Number of iterations for layout

Fig 2.3 A Typical NAM window

2.2.2 Tracing

NS simulation can produce both the visualisation trace (for NAM) as well as an ASCII file trace corresponding to the events registered at the network. There are a number of ways of collecting output or trace data on a simulation. Generally, trace data is either displayed directly during execution of the simulation, or (more commonly) stored in a file to be post-processed and analyzed. There are two primary but distinct types of monitoring capabilities currently supported by the simulator. The first, called *traces*, record each individual packet as it arrives, departs, or is dropped at a link or queue. Trace objects are configured into a simulation as nodes in the network topology, usually with a Tcl "Channel" object hooked to them, representing the destination of collected data.

The other types of objects, called monitors, record counts of various interesting quantities such as packet and byte arrivals, departures, etc. Monitors can monitor counts associated with all packets.

2.2.2.a Structure of the trace files

Event	Time	From node	To node	Pkt type	Pkt size	Flags	Fid	Src addr	Dst addr	Seq num	Pkt id

Fig 2.4 Structure of the trace file

An example of a lone in the output trace is:

r 40.639943289 _1_ AGT --- 1569 tcp 1032 [a2 1 2 800] ------- [0:0 1:0 32 1] [35 0] 2 0

- The first field is a letter that can have the values r, s, f and D for "received ", "sent", "forwarded" and "dropped", respectively . It can also be M for giving a location or a movement indication, this is described later.
- The second field is the time.
- The third field is the node number.
- The fourth field is MAC to indicate if the packet concerns a MAC layer; it is AGT to indicate the transport layer (e.g. tcp) packet, or RTR if it concerns the routed packet. it can also be IFQ to indicate events related to the interface priority queue (like drop of packets).
- After the dashes comes the global sequence number of the packet (this is not the tcp sequence number).
- At the next field comes more information on the type (e.g. tcp, ack or udp).
- Then comes the packet size in bytes.

- The 4 numbers in the first square brackets concern MAC layer information. The first hexadecimal number, a2 (which equals 162 in decimal) specifies the expected time in seconds to send this data packet over the wireless channel. The second number, 1, stands for the MAC-id of the sending node, and the third, 2, is that of the receiving node. The fourth number, 800, species that the MAC type is ETHERNTYPE_IP.

- The next numbers in the second square brackets concern the IP source and destination addresses, the TTL (Time To Live) of the packet (in our case 32).

- The third brackets concern the tcp information: its sequence number and the acknowledgement number.

2.2.2.b How to work with trace files

NS simulator can provide a lot of detailed data on events occur at the network. If one wishes to analyse the data we may need to extract relevant information form traces and to manipulate them. One can write programs in any programming language that can handle data files.

Yet several tools that seem particularly adapted for these purposes already exist and are freely available under various operating systems (Unix, Linux, windows, etc.). All they require is to write shorts scripts that are interpreted and executed without need for compilation.

Processing data files with awk

The awk utility allows us to do simple operations on data files such as averaging the values of a given column, summing or multiplying term by term between several columns, all data- reformatting tasks, etc.

Using grep

The grep command in Unix allows us to "filter "a file .we can create a new file which consists of only those lines from original files that contain a given charter sequence.

Processing data files with perl

PERL stands for “Practical Extraction and Report Language”. Perl allows easy filtering and processing of ASCII data files in Unix. Perl is an interpreted language who has many uses, but is mainly addressed to the search, extraction and report.

2.3 Xgraph

One part of the ns-allinone package is 'xgraph', a plotting program which can be used to create graphic representations of simulation results. As input, the xgraph command expects one or more ASCII files containing each x-y data point pair line. It allows creating postscript. It can be invoked within the tcl command which thus results in an immediate display after the end ‘of simulation

2.4 Generating node-movement and traffic connection files

To generate CMU's traffic and scenario generating scripts we have used following methods:

2.4.1 Creating random traffic-pattern for wireless scenarios.

Random traffic connections of TCP and CBR can be setup between mobile nodes using a traffic-scenario generator script. This traffic generator script is called cbrgen.tcl. It can be used to create CBR and TCP traffics connections between wireless mobile nodes. In order to create a traffic-connection file, we have defined the type of traffic connection (CBR or TCP), the number of nodes and maximum number of connections to be setup between them, a random seed and in case of CBR connections, a rate whose inverse value is used to compute the interval time between the CBR packets. So the command line looks like the following:

```
ns cbrgen.tcl [-type cbr|tcp] [-nn nodes] [-seed seed] [-mc connections][-rate rate]
```

The start times for the TCP/CBR connections are randomly generated with a maximum value set at 180.0s. Go through the tcl script cbrgen.tcl for the details of the traffic-generator implementation.

For example, let us try to create a CBR connection file between 10 nodes, having maximum of 8 connections, with a seed value of 1.0 and a rate of 4.0. So at the prompt type:

```
ns cbrgen.tcl -type cbr -nn 10 -seed 1.0 -mc 8 -rate 4.0 > cbr-10-test
```

From cbr-10-test file (into which the output of the generator is redirected) thus created, one of the cbr connections looks like the following:

```
#
# 2 connecting to 3 at time 82.557023746220864
#
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(2) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(3) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set pocketsize_ 512
$cbr_(0) set interval_ 0.25
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 82.557023746220864 "$cbr_(0) start"
```

Thus a UDP connection is setup between node 2 and 3. Total UDP sources (chosen between nodes 0-10) and total number of connections setup is indicated as 5 and 8 respectively, at the end of the file cbr-10-test.
Similarly TCP connection files can be created using "type" as tcp. An example would be:

```
ns cbrgen.tcl -type tcp -nn 25 -seed 0.0 -mc 8 > tcp-25-test
```

2.4.2 Creating node-movements for wireless scenarios.

The node-movement generator is consists of setdest{.cc,.h} and Makefile. CMU's version of setdest used system dependent /dev/random and made calls to library functions initstate() for generating random numbers. That was replaced with a more portable random number generator (class RNG) available in ns. In order to compile the revised setdest.cc do the following:

1. Go to ns directory and run "configure". This creates a makefile for setdest.

2. Go to indep-utils/cmu-scen-gen/setdest. Run "make", which first creates a stand-alone object file for ~ns/rng.cc and then creates the executable setdest.

3. Run setdest with arguments as shown below:

```
./setdest [-n num_of_nodes] [-p pausetime] [-s maxspeed] [-t simtime] [-x maxx]  [-y maxy]
> [outdir/movement-file]
```

For example, want to create a node-movement scenario consisting of 20 nodes moving with maximum speed of 10.0m/s with an average pause between movements being 2s. We want the simulation to stop after 200s and the topology boundary is defined as 500 X 500. So our command line will look like:

```
./setdest -n 20 -p 2.0 -s 10.0 -t 200 -x 500 -y 500 > scen-20-test
```

The output is written to stdout by default. We redirect the output to file scen-20-test. The file begins with the initial position of the nodes and goes on to define the node movements.

```
$ns_ at 2.000000000000 "$node_(0) setdest 90.441179033457 44.896095544010
1.373556960010"
```

This line from scen-20-test defines that node_(0) at time 2.0s starts to move toward destination (90.44, 44.89) at a speed of 1.37m/s. These command lines can be used to change direction and speed of movement of mobile nodes. Directives for GOD are present as well in node-movement file.

The General Operations Director (GOD) object is used to store global information about the state of the environment, network, or nodes that an omniscient observer would have, but that should not be made known to any participant in the simulation.
The god object does not calculate this on the fly during simulation runs, since it can be quite time consuming. The information is loaded into the god object from the movement pattern file where lines of the form

```
$ns_ at 899.642 "$god_ set-dist 23 46 2"
```

are used to load the god object with the knowledge that the shortest path between node 23 and node 46 changed to 2 hops at time 899.642.

The setdest program generates node-movement files using the random waypoint algorithm. These files already include the lines to load the god object with the appropriate information at the appropriate time.

Setdest calculate the shortest number of hops between nodes based on the nominal radio range, ignoring any effects that might be introduced by the propagation model in an actual simulation. The nominal range is either provided as an argument to the programs, or extracted from the header on the movement pattern file.

Chapter 3
Ad-hoc Routing

3.1 Mobile Ad hoc Networks

A mobile ad hoc network (MANET), sometimes called a mobile mesh network, is a self-configuring network of mobile devices connected by wireless links. Each device in a MANET is free to move independently in any direction, and will therefore change its links to other devices frequently. Each must forward traffic unrelated to its own use, and therefore be a router. The primary challenge in building a MANET is equipping each device to continuously maintain the information required to properly route traffic. Such networks may operate by themselves or may be connected to the larger Internet. MANETs are a kind of wireless ad hoc networks that usually has a routable networking environment on top of a Link Layer ad hoc network. They are also a type of mesh network, but many mesh networks are not mobile or not wireless.

What makes ad hoc networks different from wired networks is that all the usual rules about fixed topologies, fixed and known neighbours, fixed relationship between IP address and location, and more are suddenly tossed out the window. Routers can come and go or appear in new places at the drop of a bit. With a wired network, if a router has a valid path to some destination, that path continues to be valid indefinitely (barring a failure somewhere in the system). With an ad hoc network, the topology may be changing all the time, so desirability and even validity of paths can change spontaneously, without warning. Needless to say, these circumstances make routing in ad hoc networks quite different from routing in their fixed counterparts. A MANET environment is characterized by energy-limited nodes (Mobile Hosts), bandwidth-constrained, variable-capacity wireless links and dynamic topology, leading to frequent and unpredictable connectivity changes

3.2 Ad-hoc Routing

An ad hoc routing protocol is a convention, or standard, that controls how nodes decide which way to route packets between computing devices in a mobile ad-hoc network. In ad hoc networks, nodes do not start out familiar with the topology of their networks; instead, they have to discover it. The basic idea is that a new node may announce its presence and should listen for announcements broadcast by its neighbours. Each node learns about nodes nearby and how to reach them, and may announce that it, too, can reach them.

Different Routing Protocols included in our research are as follows:

3.2.1 Destination-Sequenced Distance-Vector Routing (DSDV)

Destination-Sequenced Distance-Vector Routing (DSDV) is a table-driven routing scheme for ad hoc mobile networks based on the Bellman-Ford algorithm. The main contribution of the algorithm was to solve the Routing Loop problem. Each entry in the routing table contains a sequence number, the sequence numbers are generally even if a link is present; else, an odd number is used. The number is generated by the destination, and the emitter needs to send out the next update with this number. Routing information is distributed between nodes by sending full dumps infrequently and smaller incremental updates more frequently.

The destination-sequenced distance-vector (DSDV) is a proactive hop-by-hop distance vector routing protocol, requiring each node to broadcast routing updates periodically. Here, every MH in the network maintains a routing table for all possible destinations within the network and the number of hops to each destination. Each entry is marked with a sequence number assigned by the destination MH. The sequence numbers enable the MHs to distinguish stale routes from new ones, thereby avoiding the formation of routing loops. Routing table updates are periodically transmitted throughout the network in order to maintain consistency in the tables. To alleviate potentially large network update traffic, two possible types of packets can be employed: full dumps or small increment packets. A full dump type of packet carries all available routing information and can require multiple network protocol data units (NPDUs). These packets are transmitted less frequently during periods of occasional movements. Smaller incremental packets are used to relay only the information that has changed since the last full dump. Each of these broadcasts should fit into a standard-size NPDU, thereby decreasing the amount of traffic generated. The MHs maintain an additional table where they store the data sent in the incremental routing information packets. New route broadcasts contain the address of the destination, the number of hops to reach the destination, the sequence number of the information received regarding the destination, as well as a new sequence number unique to the broadcast. The route labelled with the most recent sequence number is always used. In the event that two updates have the same sequence number, the route with the smaller metric is used in order to optimize (shorten) the path. MHs also keep track of settling time of the routes, or the weighted average time that routes to a destination could fluctuate before the route with the best metric is received. By delaying the broadcast of a routing update by the length of the settling time, MHs can reduce network traffic. Note that if each MH in the network advertises a monotonically increasing sequence number for itself, it may imply that the route just got broken.

3.2.2 Dynamic Source Routing (DSR)

Dynamic Source Routing (DSR) is a routing protocol for wireless mesh networks. It is similar to AODV in that it forms a route on-demand when a transmitting computer requests one. However, it uses source routing instead of relying on the routing table at each intermediate device. Many successive refinements have been made to DSR, including DSRFLOW.

Determining source routes requires accumulating the address of each device between the source and destination during route discovery. The accumulated path information is cached by nodes processing the route discovery packets. The learned paths are used to route packets. To accomplish source routing, the routed packets contain the address of each device the packet will traverse. This may result in high overhead for long paths or large addresses. To avoid using source routing, DSR optionally defines a flow id option that allows packets to be forwarded on a hop-by-hop basis.

This protocol is truly based on source routing whereby all the routing information is maintained (continually updated) at mobile nodes. It has only 2 major phases which are *Route Discovery* and *Route Maintenance*. Route Reply would only be generated if the message has reached the intended destination node (route record which is initially contained in Route Request would be inserted into the Route Reply).

To return the Route Reply, the destination node must have a route to the source node. If the route is in the Destination Node's route cache, the route would be used. Otherwise, the node will reverse the route based on the route record in the Route Reply message header (this requires that all links are symmetric). In the event of fatal transmission, the Route Maintenance Phase is initiated whereby the Route Error packets are generated at a node. The erroneous hop will be removed from the node's route cache; all routes containing the hop are truncated at that point. Again, the Route Discovery Phase is initiated to determine the most viable route.

Dynamic source routing protocol (DSR) is an on-demand protocol designed to restrict the bandwidth consumed by control packets in ad hoc wireless networks by eliminating the periodic table-update messages required in the table-driven approach. The major difference between this and the other on-demand routing protocols is that it is beacon-less and hence does not require periodic hello packet (beacon) transmissions, which are used by a node to inform its neighbours of its presence. The basic approach of this protocol (and all other on-demand routing protocols) during the route construction phase is to establish a route by flooding Route-Request packets in the network. The destination node, on receiving a Route-Request packet, responds by sending a Route-Reply packet back to the source, which carries the route traversed by the Route-Request packet received.

Consider a source node that does not have a route to the destination. When it has data packets to be sent to that destination, it initiates a Route-Request packet. This Route-Request is flooded throughout the network. Each node, upon receiving a Route-Request packet, rebroadcasts the packet to its neighbours if it has not forwarded it already, provided that the node is not the destination node and that the packet's time to live (TTL) counter has not been exceeded. Each Route-Request carries a sequence number generated by the source node and the path it has traversed. A node, upon receiving a Route-Request packet, checks the sequence number on the packet before forwarding it. The packet is forwarded only if it is not a duplicate Route-Request. The sequence number on the packet is used to prevent loop formations and to avoid multiple transmissions of the same Route-Request by an intermediate node that receives it through multiple paths. Thus, all nodes except the destination forward a Route-Request packet during the route construction phase. A destination node, after receiving the first Route-Request packet, replies to the source node through the reverse path the Route-Request packet had traversed. Nodes can also learn about the neighbouring routes traversed by data packets if operated in the promiscuous mode (the mode of operation in which a node can receive the packets that are neither broadcast nor addressed to itself). This route cache is also used during the route construction phase. If an intermediate node receiving a Route-Request has a route to the destination node in its route cache, then it replies to the source node by sending a Route-Reply with the entire route information from the source node to the destination node.

3.2.3 Ad hoc On Demand Distance Vector (AODV)

The Ad hoc On Demand Distance Vector (AODV) routing algorithm is a routing protocol designed for ad hoc mobile networks. AODV is capable of both unicast and multicast routing. It is an on demand algorithm, meaning that it builds routes between nodes only as desired by source nodes. It maintains these routes as long as they are needed by the sources. Additionally, AODV forms trees which connect multicast group members. The trees are composed of the group members and the nodes needed to connect the members. AODV uses sequence numbers to ensure the freshness of routes. It is loop-free, self-starting, and scales to large numbers of mobile nodes.

AODV builds routes using a route request / route reply query cycle. When a source node desires a route to a destination for which it does not already have a route, it broadcasts a route request (RREQ) packet across the network. Nodes receiving this packet update their information for the source node and set up backwards pointers to the source node in the route tables. In addition to the source node's IP address, current sequence number, and broadcast ID, the RREQ also contains the most recent sequence number for the destination of which the source node is aware. A node receiving the RREQ may send a route reply (RREP) if it is either the destination or if it has a route to the destination with corresponding sequence number greater than or equal to that contained in the RREQ. If this is the case, it unicasts a RREP back to the source. Otherwise, it rebroadcasts the RREQ. Nodes keep track of the RREQ's source IP address and broadcast ID. If they receive a RREQ which they have already processed, they discard the RREQ and do not forward it.

As the RREP propagates back to the source; nodes set up forward pointers to the destination. Once the source node receives the RREP, it may begin to forward data packets to the destination. If the source later receives a RREP containing a greater sequence number or contains the same sequence number with a smaller hop count, it may update its routing information for that destination and begin using the better route.

As long as the route remains active, it will continue to be maintained. A route is considered active as long as there are data packets periodically travelling from the source to the destination along that path. Once the source stops sending data packets, the links will time out and eventually be deleted from the intermediate node routing tables. If a link break occurs while the route is active, the node upstream of the break propagates a route error (RERR) message to the source node to inform it of the now unreachable destination(s). After receiving the RERR, if the source node still desires the route, it can reinitiate route discovery.

Chapter 4
Clustering

4.1 Clustering

In a clustering scheme the mobile nodes in a MANET are divided into different virtual groups, and they are allocated geographically adjacent into the same cluster according to some rules with different behaviours for nodes included in a cluster from those excluded from the cluster. A typical cluster structure is shown in **Fig. 4.1**.

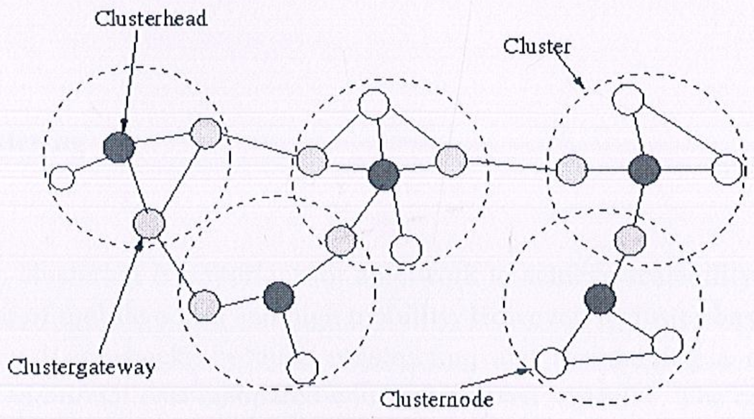


Fig 4.1 A typical Clustering Scheme

It can be seen that the nodes are divided into a number of virtual groups (with the dotted lines) based on certain rules. Under a cluster structure, mobile nodes may be assigned a different status or function, such as cluster-head, cluster-gateway, or cluster-member. A cluster-head normally serves as a local coordinator for its cluster, performing intra-cluster transmission arrangement, data forwarding, and so on. A cluster gateway is a non-cluster-head node with inter-cluster links, so it can access neighbouring clusters and forward information between clusters. A cluster-member is usually called an ordinary node, which is a non cluster-head node without any inter-cluster links.

4.2 Why do ad-hoc networks require Clustering?

It has been shown that cluster architecture guarantees basic performance achievement in a MANET with a large number of mobile terminals. A cluster structure, as an effective topology control means, provides at least three benefits. First, a cluster structure facilitates the spatial reuse of resources to increase the system capacity. With the non-overlapping multi-cluster structure, two clusters may deploy the same frequency or code set if they are not neighbouring clusters. Also, a cluster can better coordinate its transmission events with the help of a special mobile node, such as a cluster-head, residing in it. This can save much resources used for retransmission resulting from reduced transmission collision. The second benefit is in routing, because the set of cluster-heads and cluster-gateways can normally form a virtual backbone for inter-cluster routing, and thus the generation and spreading of routing information can be restricted in this set of nodes. Last, a cluster structure makes an ad hoc network appear smaller and more stable in the view of each mobile terminal. When a mobile node changes its attaching cluster, only mobile nodes residing in the corresponding clusters need to update the information. Thus, local changes need not be seen and updated by the entire network, and information processed and stored by each mobile node is greatly reduced.

4.3 Cost of Clustering

As we know, clustering is important for a network to achieve scalability in the presence of a large number of mobile nodes and high mobility. However, a cluster-based MANET has its side effects and drawbacks because constructing and maintaining a cluster structure usually requires additional cost compared with a flat-based MANET. The cost of clustering is a key issue to validate the effectiveness and scalability enhancement of a cluster structure. By analyzing the cost of a clustering scheme in different aspects qualitatively or quantitatively, its usefulness and drawbacks can be clearly specified. The clustering cost terms are described as follows:

- To maintain a cluster structure in a dynamically changing scenario often requires explicit message exchange between mobile node pairs. When the underlying network topology changes quickly and involves many mobile nodes, the clustering-related information exchange increases drastically. Frequent information exchange may consume considerable bandwidth and drain mobile nodes' energy quickly so that upper-layer applications cannot be implemented due to the inadequacy of available resources or the lack of support from related mobile nodes.

- Some clustering schemes may cause the cluster structure to be completely rebuilt over the whole network when some local events take place, e.g. the movement or "die" of a mobile node, resulting in some cluster-head re-election (re-clustering). This is called the ripple effect of re-clustering. In other words, the ripple effect of re-clustering indicates that the re-election of one cluster-head may affect the structure of many clusters and arouse the cluster-head re-election over the network. Thus, the ripple effect of re-clustering may greatly affect the performance of upper-layer protocols.

- In addition, most schemes separate the clustering into two phases, cluster formation and cluster maintenance, and assume that mobile nodes keep static when cluster formation is in progress. This is because for the initial cluster formation of these schemes, a mobile node can decide to become a cluster-head only after it exchanges some specific information with its neighbours and assures that it holds some specific attribute in its neighbourhood. With a frozen period of motion, each mobile node can obtain accurate information from neighbouring nodes, and the initial cluster structure can be formed with some specific characteristics. However, this assumption may not be applicable in an actual scenario, where mobile nodes may move randomly all the time.

- Another metric is the computation round, which indicates the number of rounds in which a cluster formation procedure can be completed. For clustering schemes relying on a frozen period of motion assumption, the computation round is an important metric since the more rounds that a clustering scheme requires for its cluster formation, the longer the frozen period that is required for mobile nodes. But in fact, the topology of a MANET changes frequently with the movement of mobile nodes. For most of the clustering schemes, their cluster formation procedure can be performed in parallel in the whole network, which should result in fast time convergence in cluster formation. But in these schemes, not all mobile nodes can decide their status at the same time (within one round), and they may require a non-constant number of rounds to finish the initial cluster construction. Thus, the time required for these algorithms cannot be bounded and may vary noticeably for different network topologies. Hence, the required explicit control message exchange, the ripple effect of re-clustering, and the stationary assumption for cluster formation are the main costs of a cluster-based MANET compared with a MANET with a flat structure.

Chapter 5 Simulation

5.1 Simulation Environment

The overall goal of our experiments was to measure the ability of the routing protocols to react to network topology change while continuing to successfully deliver data packets to their destinations. To measure this ability, our basic methodology was to apply to a simulated network a variety of workloads, in effect testing with each data packet originated by some sender whether the routing protocol can at that time route to the destination of that packet. We were not attempting to measure the protocols' performance on a particular workload taken from real life, but rather to measure the protocols' performance under a range of conditions.

For the comparison at different pause times the protocol evaluations are based on the simulation of 50 wireless nodes forming an ad hoc network, moving about over a rectangular (1500m x 300m) flat space for 900 seconds of simulated time. We chose a rectangular space in order to force the use of longer routes between nodes than would occur in a square space with equal node density. The physical radio characteristics of each mobile node's network interface, such as the antenna gain, transmit power, and receiver sensitivity, were chosen to approximate the Lucent WaveLAN direct sequence spread spectrum radio.

For the comparison with different network densities the protocol evaluations are based on the simulation of a rectangular (1000m x 500m) flat space for 200 seconds of simulated time, with different wireless nodes (18, 30, 48, 60, 78 and 90) forming an ad hoc network, moving about over. We ran our simulations for three pause times in this case (0, 100 and 200 seconds).

In order to enable direct, fair comparisons between the protocols, it was critical to challenge the protocols with identical loads and environmental conditions. Each run of the simulator accepts as input a scenario file that describes the exact motion of each node and the exact sequence of packets originated by each node, together with the exact time at which each change in motion or packet origination is to occur. We pre-generated 50 different scenario files with varying movement patterns and traffic loads, and then ran all four routing protocols against each of these scenario files. Since each protocol was challenged in an identical fashion, we can directly compare the performance results of the four protocols (DSDV, AODV and clustering using both).

5.2 Movement Model

Nodes in the simulation move according to a model that we call the "random waypoint" model. The movement scenario files we used for each simulation are characterized by a pause time and number of nodes. Each node begins the simulation by remaining stationary for pause time seconds. It then selects a random destination in the network space and moves to that destination at a speed distributed uniformly between 0 and some maximum speed. Upon reaching the destination, the node pauses again for pause time seconds, selects another destination, and proceeds there as previously described, repeating this behaviour for the duration of the simulation.

We ran our first experiment simulations with movement patterns generated for 7 different pause times: 0, 30, 60, 120, 300, 600, and 900 seconds. A pause time of 0 seconds corresponds to continuous motion, and a pause time of 900 (the length of the simulation) corresponds to no motion.

In our second experiment simulations, movement patterns are generated for 3 different pause times: 0, 100 and 200 seconds also 6 different node numbers: 18, 30, 48, 60, 78 and 90. A pause time of 0 seconds corresponds to continuous motion, and a pause time of 900 (the length of the simulation) corresponds to no motion.

Because the performance of the protocols is very sensitive to movement pattern, we generated scenario files with 14 different movement patterns, 2 for each value of pause time and with 18 different movement patterns, 6 for each value of pause time for second experiment. All four routing protocols were run on the same 70 movement patterns.

5.3 Communication Model

As the goal of our simulation was to compare the performance of each routing protocol, we chose our traffic sources to be constant bit rate (CBR) sources. When defining the parameters of the communication model, we experimented with sending rate of 4 packets per second, networks containing 8 CBR sources, and packet sizes of 64 and 1024 bytes.

Varying the number of CBR sources was approximately equivalent to varying the sending rate. Hence, for these simulations we chose to fix the sending rate at 4 packets per second. When using 1024-byte packets, we found that congestion, due to lack of spatial diversity, became a problem for all protocols and one or two nodes would drop most of the packets that they received for forwarding. As none of the studied protocols performs load balancing, and the goal of our analysis was to determine if the routing protocols could consistently track changes in topology, we attempted to factor out congestive effects by reducing the packet size to 64 bytes.

We did not use TCP sources because TCP offers a conforming load to the network, meaning that it changes the times at which it sends packets based on its perception of the network's ability to carry packets. As a result, both the time at which each data packet is originated by its sender and the position of the node when sending the packet would differ between the protocols, preventing a direct comparison between them.

5.4 Scenario Characteristics

To characterize the challenge our scenarios placed on the routing protocols, we measured the lengths of the routes over which the protocols had to deliver packets, and the total number of topology changes in each scenario.

When each data packet is originated, an internal mechanism of our simulator (separate from the routing protocols) calculates the shortest path between the packet's sender and its destination. The packet is labelled with this information, which is compared with the number of hops actually taken by the packet when received by the intended destination. The shortest path is calculated based on a nominal transmission range of 250m for each radio and does not account for congestion or interference that any particular packet might see.

Chapter 6
Results

6.1 First Scenario

In this scenario we considered a network of the size 1500m x 300m with 50 nodes in it. In this scenario we compared different protocols with their hierarchical approach at different pause times.

6.1.1 Packet Delivery Ratio

Following graphs show the Packet Delivery Ratio of ad-hoc routing protocols with their hierarchical approach at different pause times. In these graphs we see improved readings at the lower pause times (which refer to higher mobility in the network).

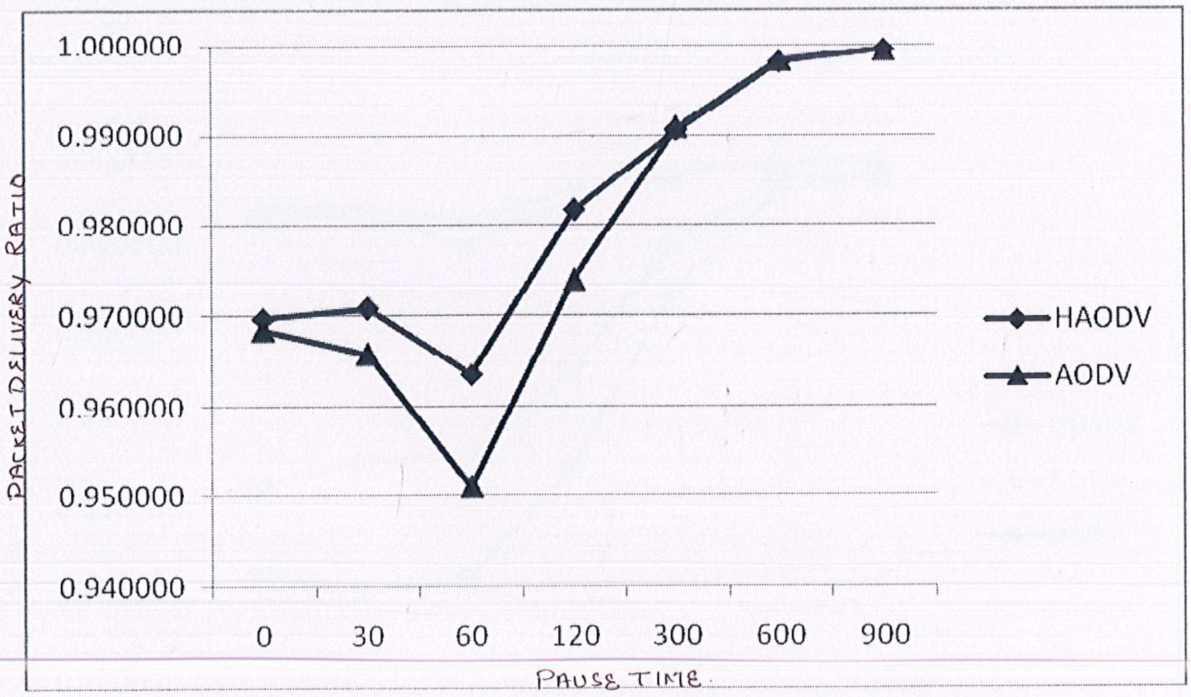


Fig 6.1 Packet Delivery Ratio of HDSDV and DSDV at different pause time

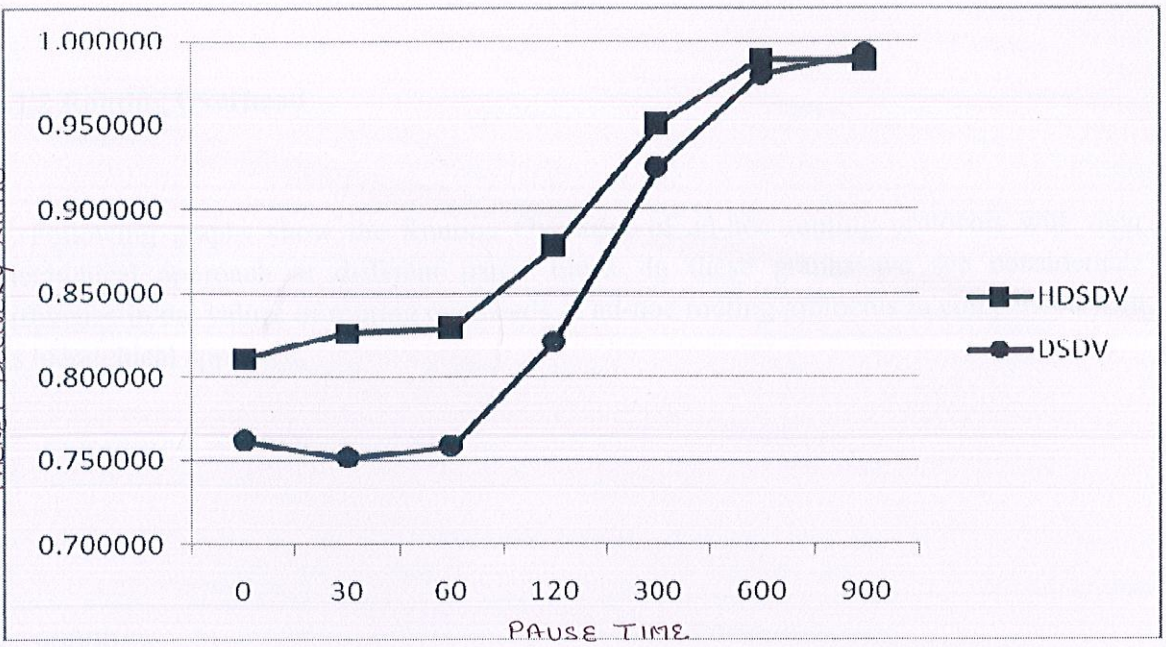


Fig 6.2 Packet Delivery Ratio of HDSDV and DSDV at different pause time

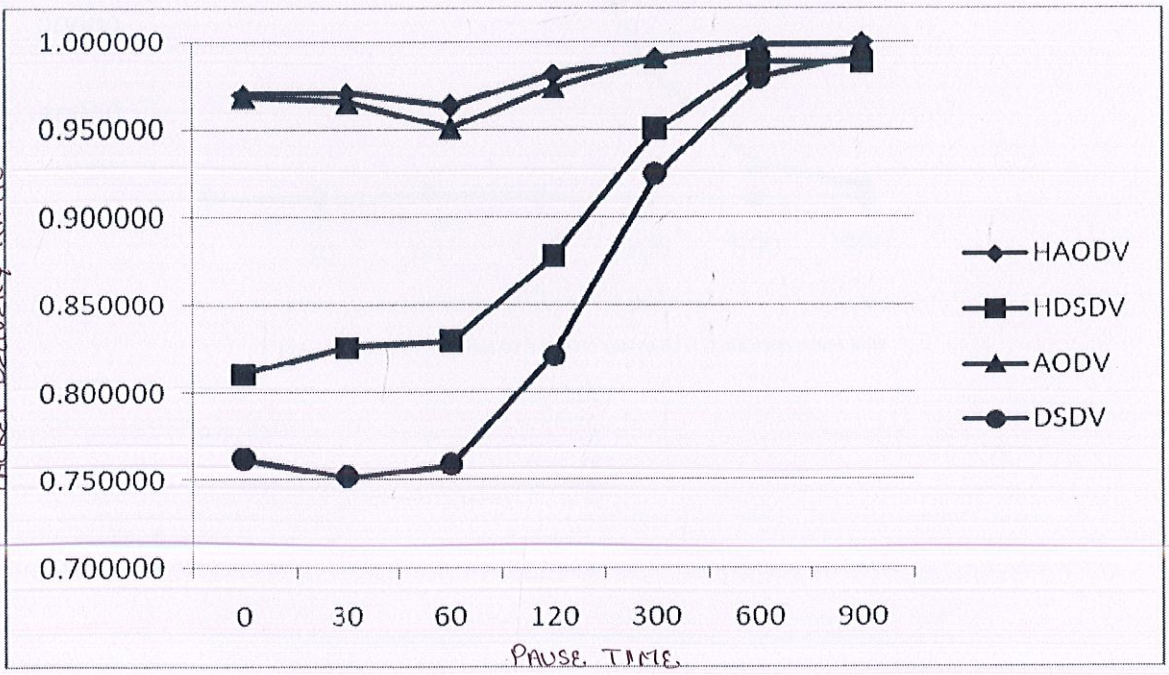


Fig 6.3 Packet Delivery Ratio of HAODV, HDSDV, DSDV and AODV at different pause time

6.1.2 Routing Overhead

Following graphs show the Routing Overhead of ad-hoc routing protocols with their hierarchical approach at different pause times. In these graphs we see considerable difference in the values of routing overheads of ad-hoc routing protocols in comparison with its hierarchical approach.

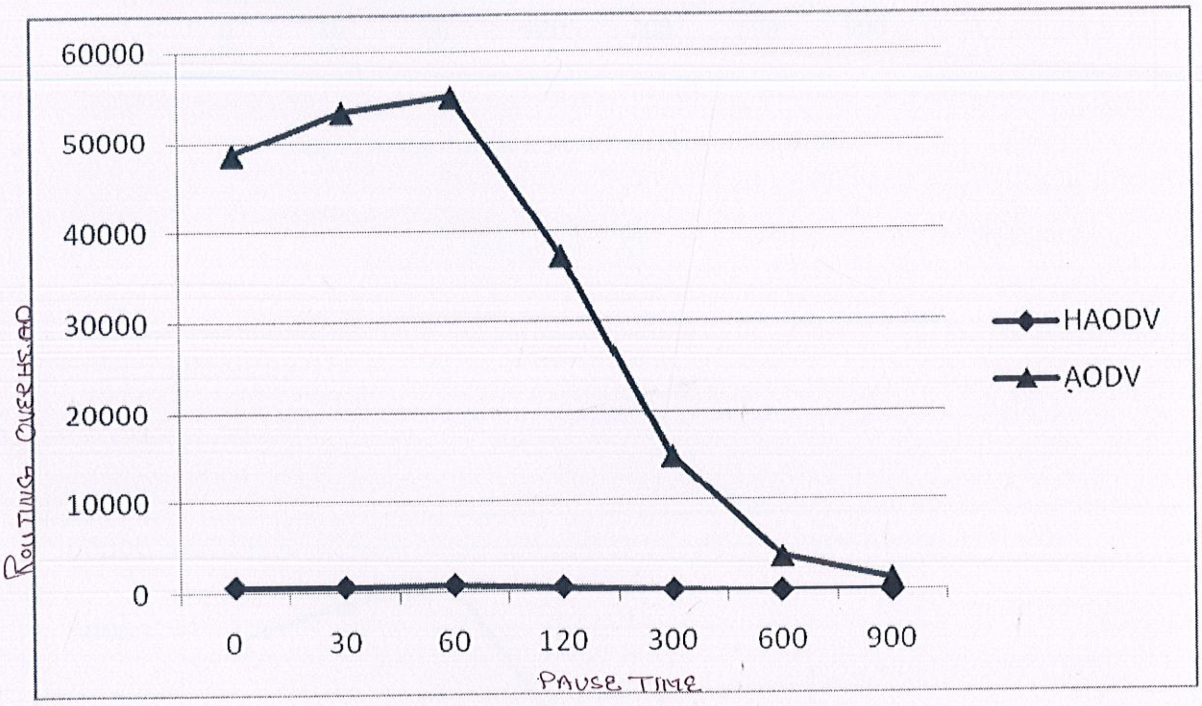


Fig 6.4 Routing Overhead of HAODV and AODV at different pause time

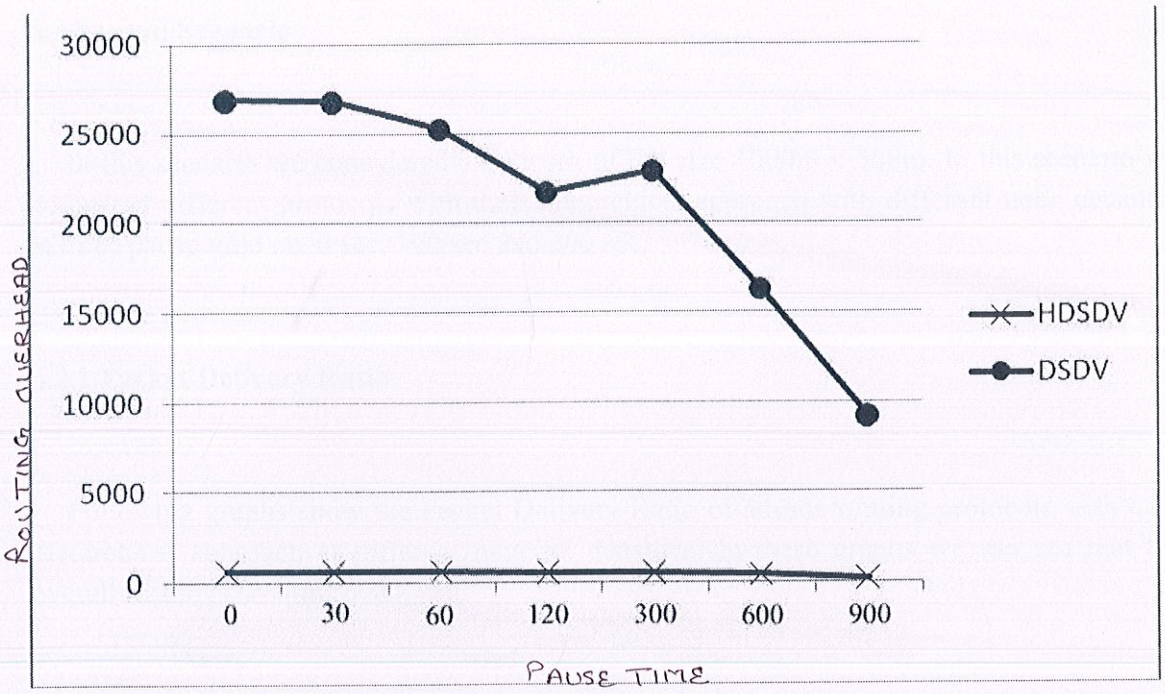


Fig 6.5 Routing Overhead of HDSDV and DSDV at different pause time

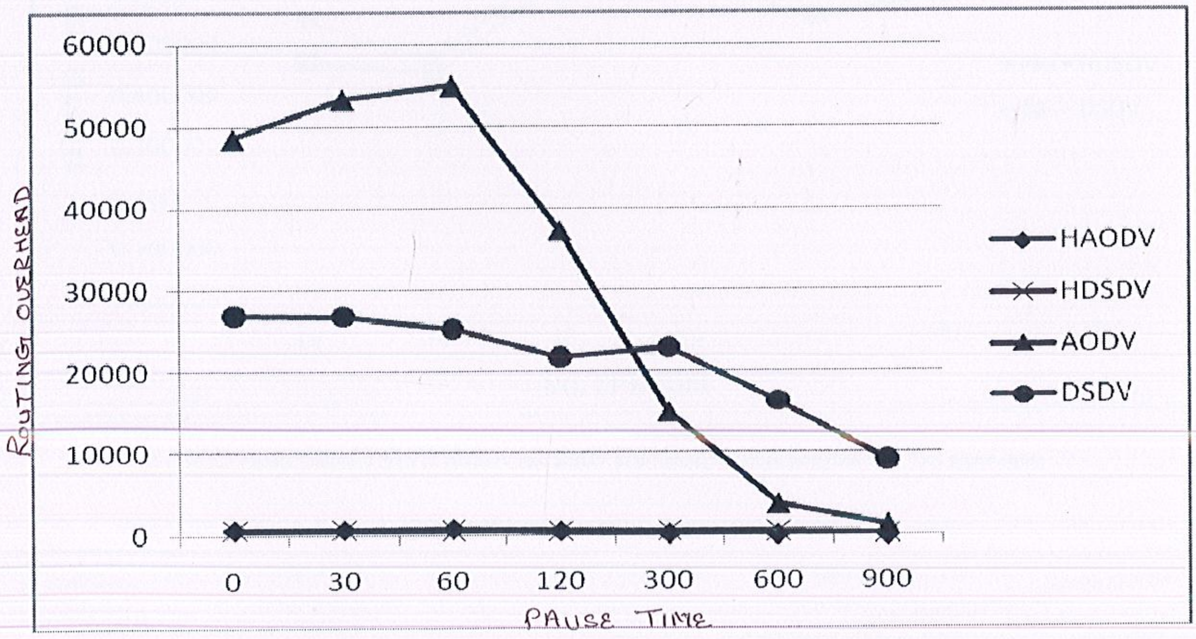


Fig 6.6 Routing Overhead of HAODV, HDSDV, AODV and DSDV at different pause time

6.2 Second Scenario

In this scenario we considered a network of the size 1000m x 500m. In this scenario we compared different protocols with their hierarchical approach with different node densities at three pause time i.e. 0 sec, 100 sec and 200 sec.

6.2.1 Packet Delivery Ratio

Following graphs show the Packet Delivery Ratio of ad-hoc routing protocols with their hierarchical approach at different network densities. In these graphs we can see that the overall readings are improved.

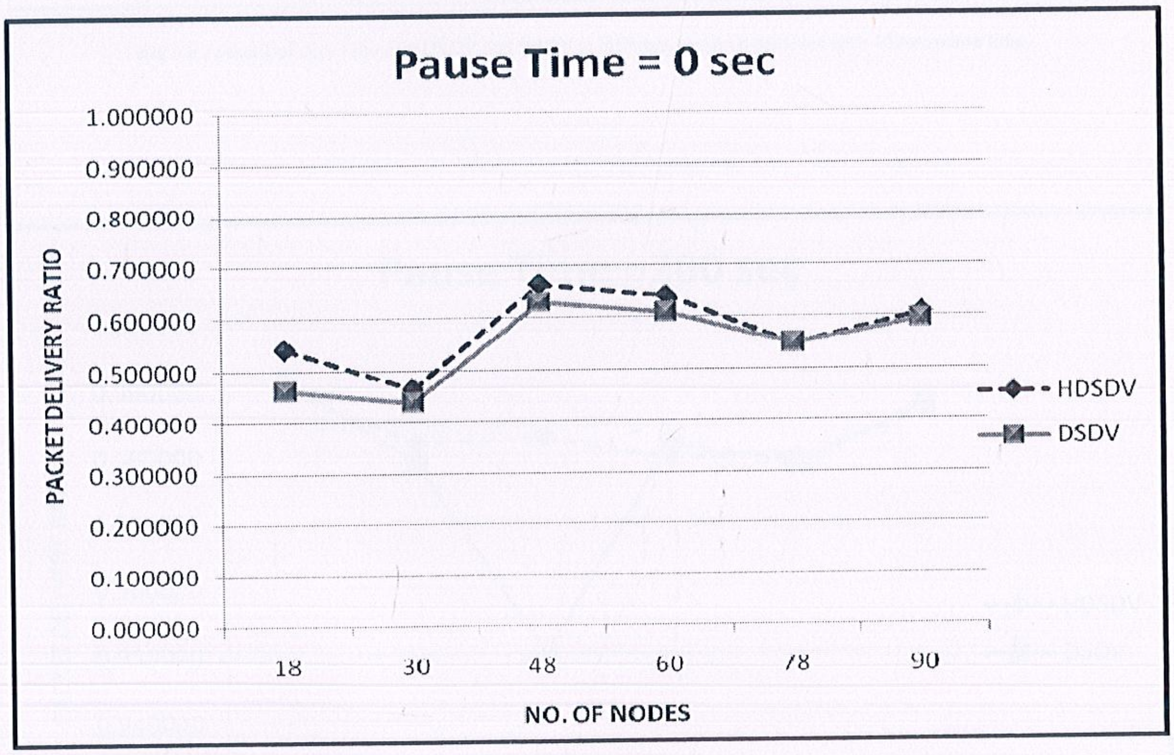


Fig 6.7 Packet Delivery ratio of HDSDV and DSDV at different Network densities with 0sec pause time

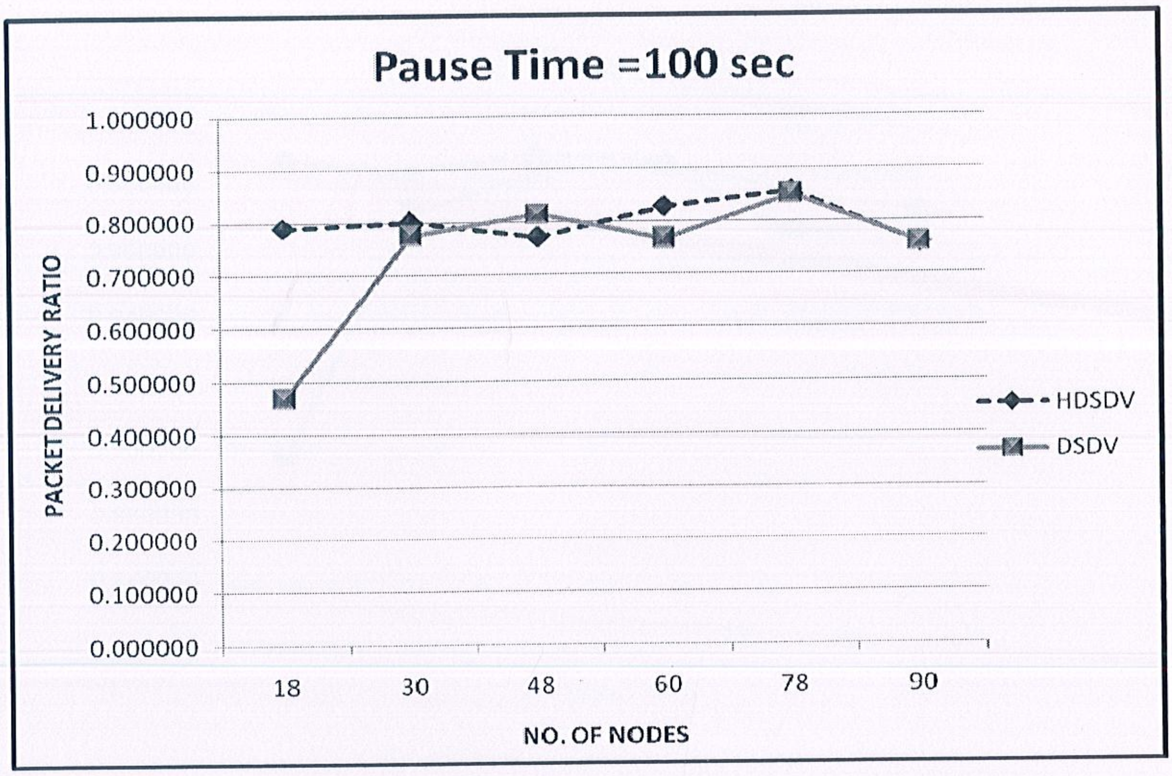


Fig 6.8 Packet Delivery ratio of HDSDV and DSDV at different Network densities with 100sec pause time

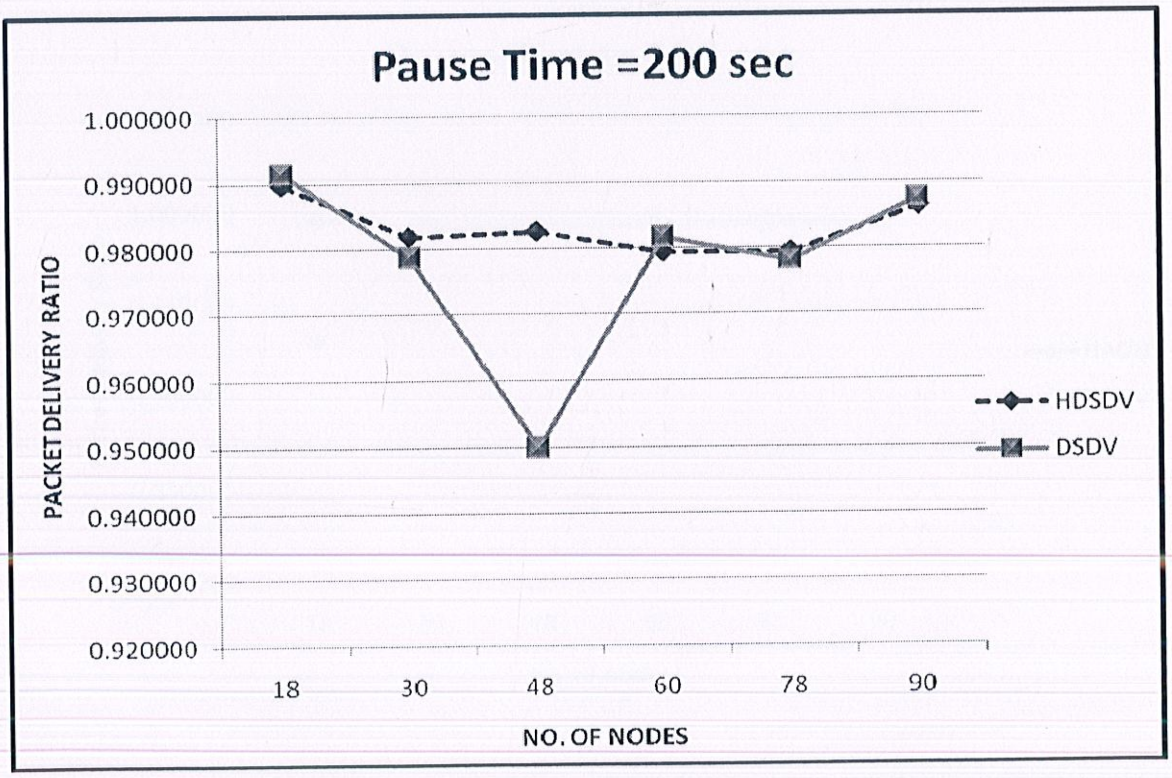


Fig 6.9 Packet Delivery ratio of HDSDV and DSDV at different Network densities with 200sec pause time

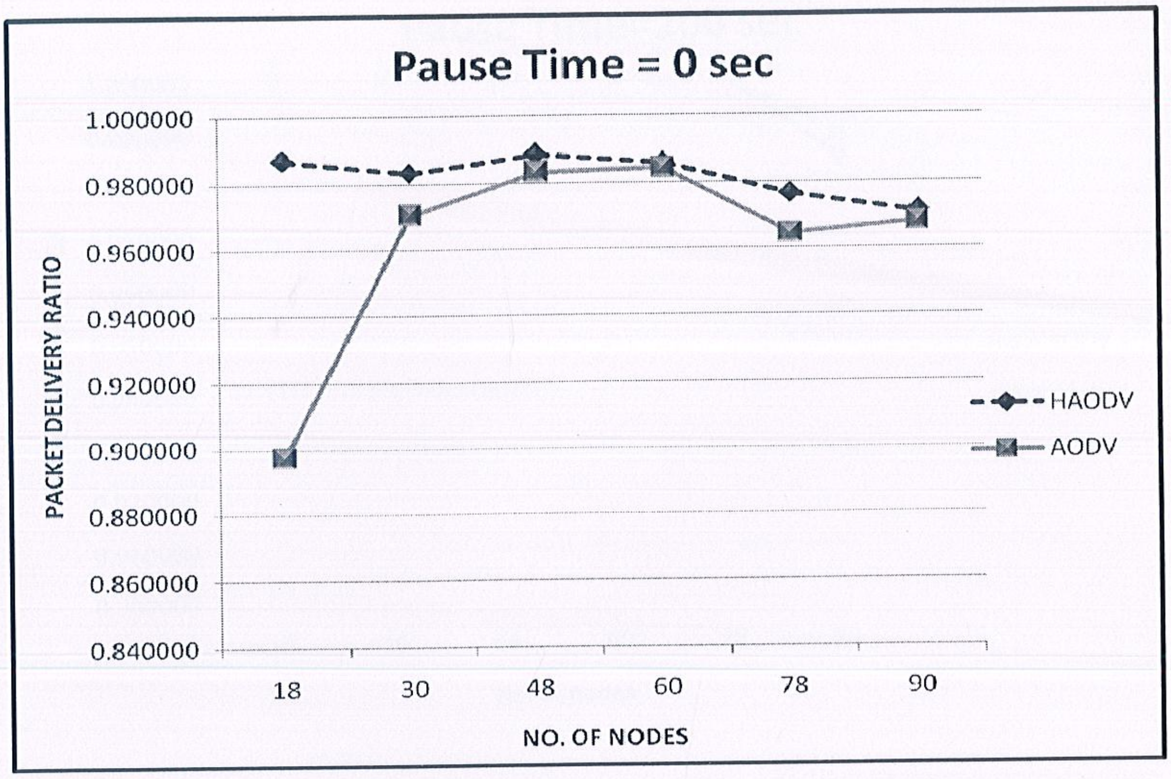


Fig 6.10 Packet Delivery ratio of HAODV and AODV at different Network densities with 0sec pause time

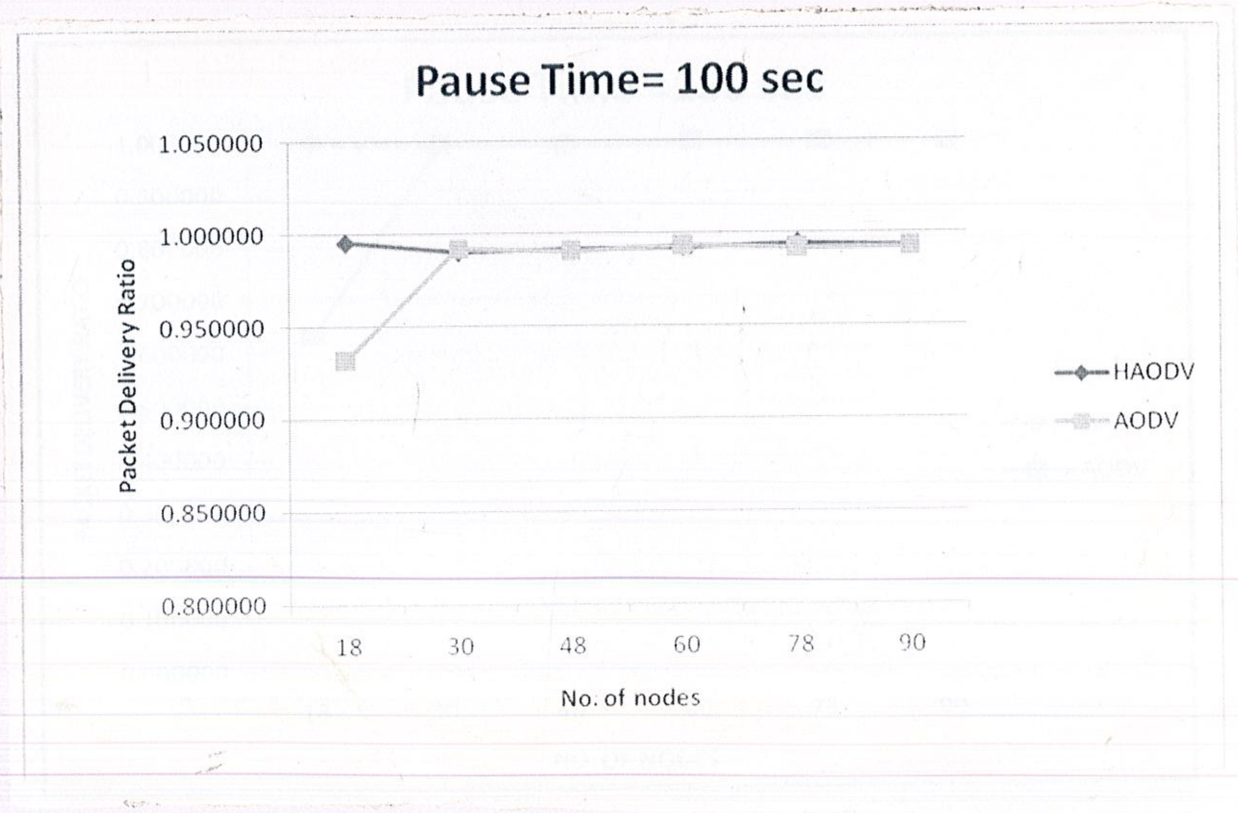


Fig 6.11 Packet Delivery ratio of HAODV and AODV at different Network densities with 100sec pause time

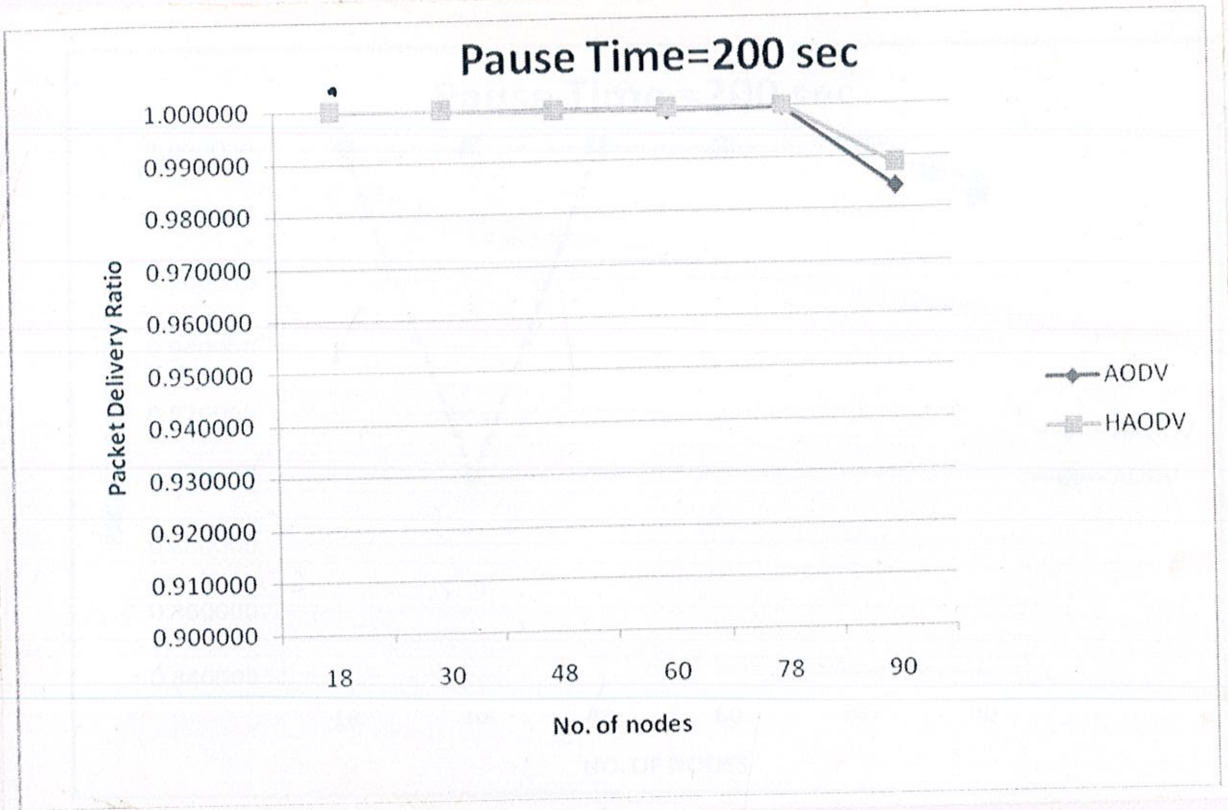


Fig 6.12 Packet Delivery ratio of HAODV and AODV at different Network densities with200sec pause time

6.2.2 Routing Overhead

Following graphs show the Routing Overhead of ad-hoc routing protocols with their hierarchical approach at different network densities, these readings are taken at three pause times i.e. 0 sec, 100 sec and 200 sec. In these graphs we see considerable improvements in the values of routing overheads of the hierarchical approach used in ad-hoc routing protocols as compared with the original form of ad-hoc routing protocols.

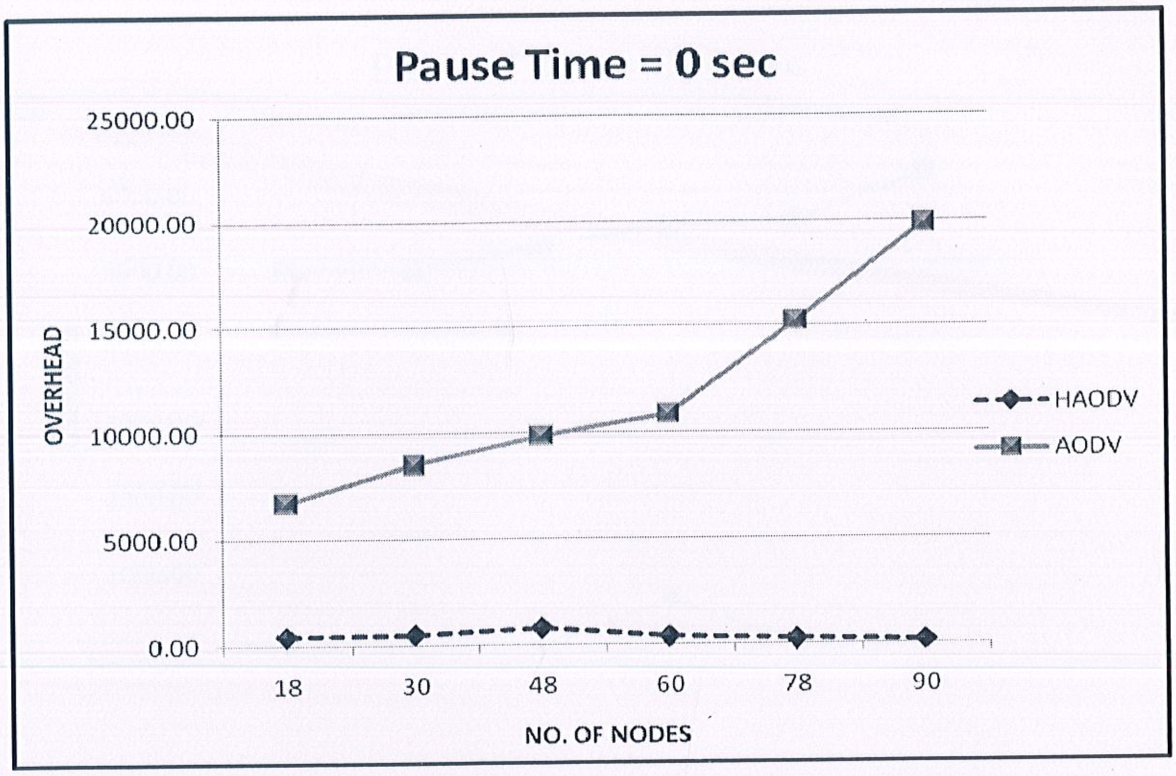


Fig 6.13 Routing overhead of HAODV and AODV at different Network densities with 0sec pause time

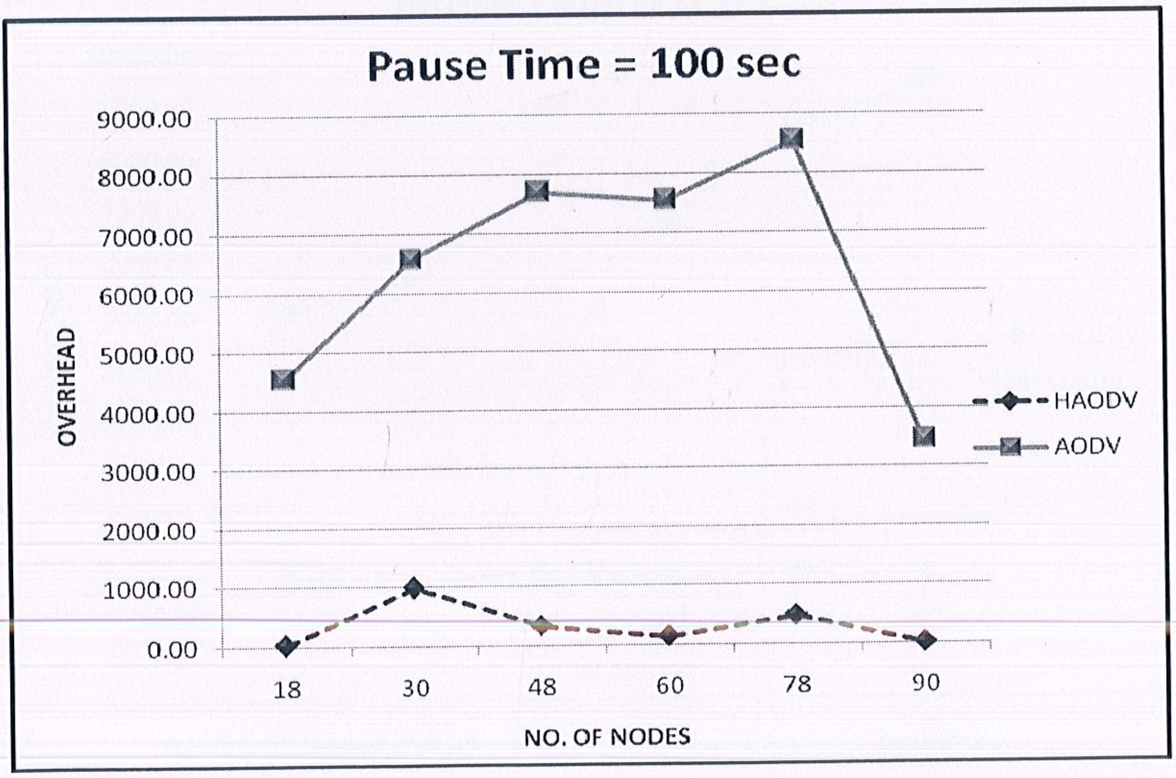


Fig 6.14 Routing overhead of HAODV and AODV at different Network densities with 100sec pause time

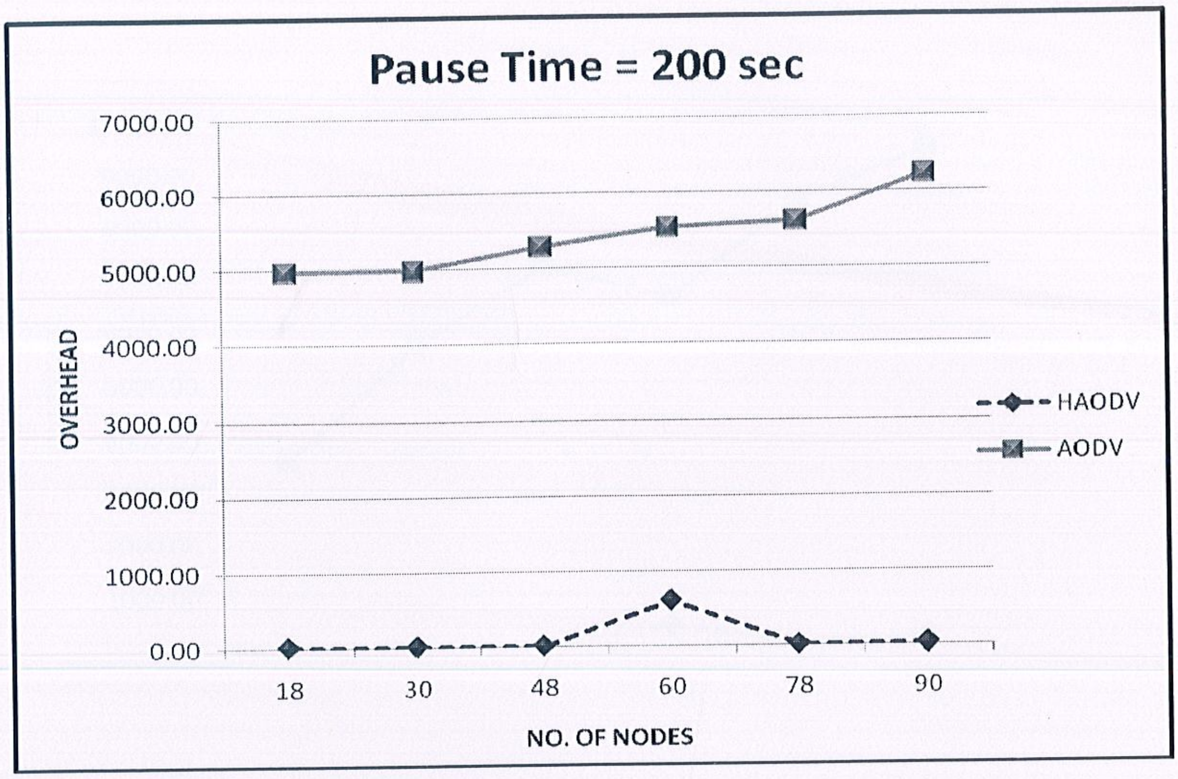


Fig 6.15 Routing overhead of HAODV and AODV at different Network densities with 200sec pause time

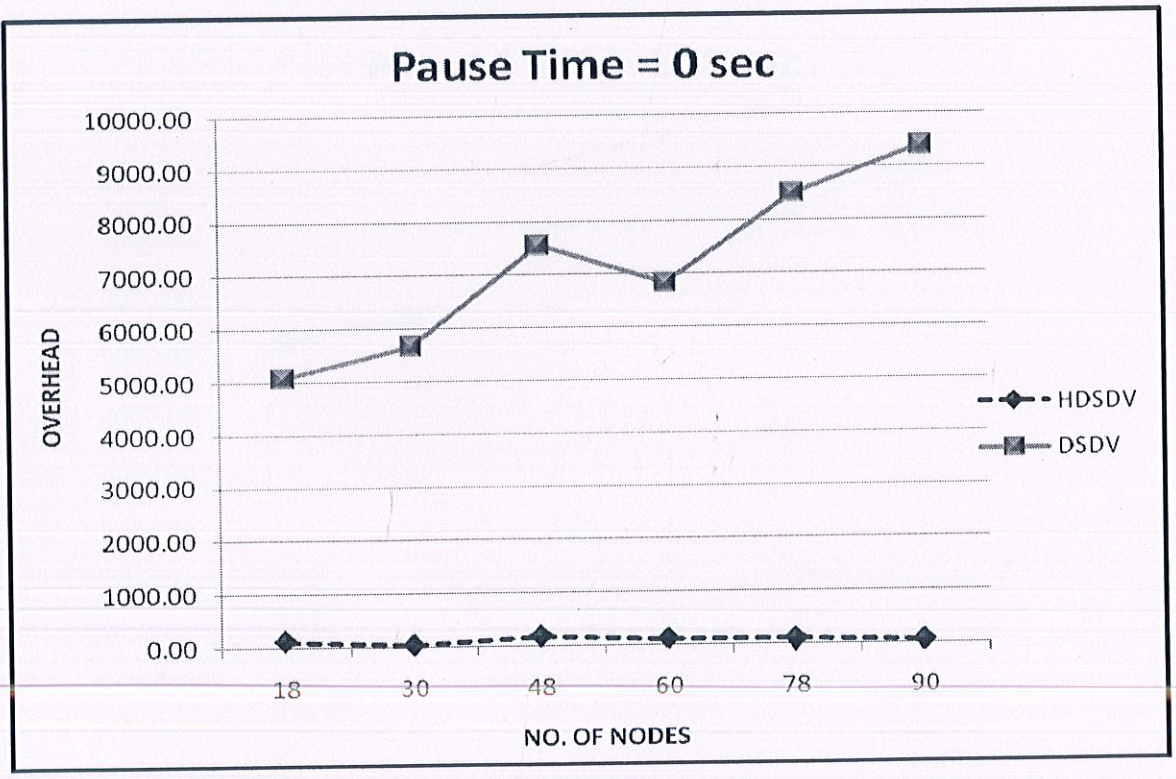


Fig 6.16 Routing overhead of HDSDV and DSDV at different Network densities with 0sec pause time

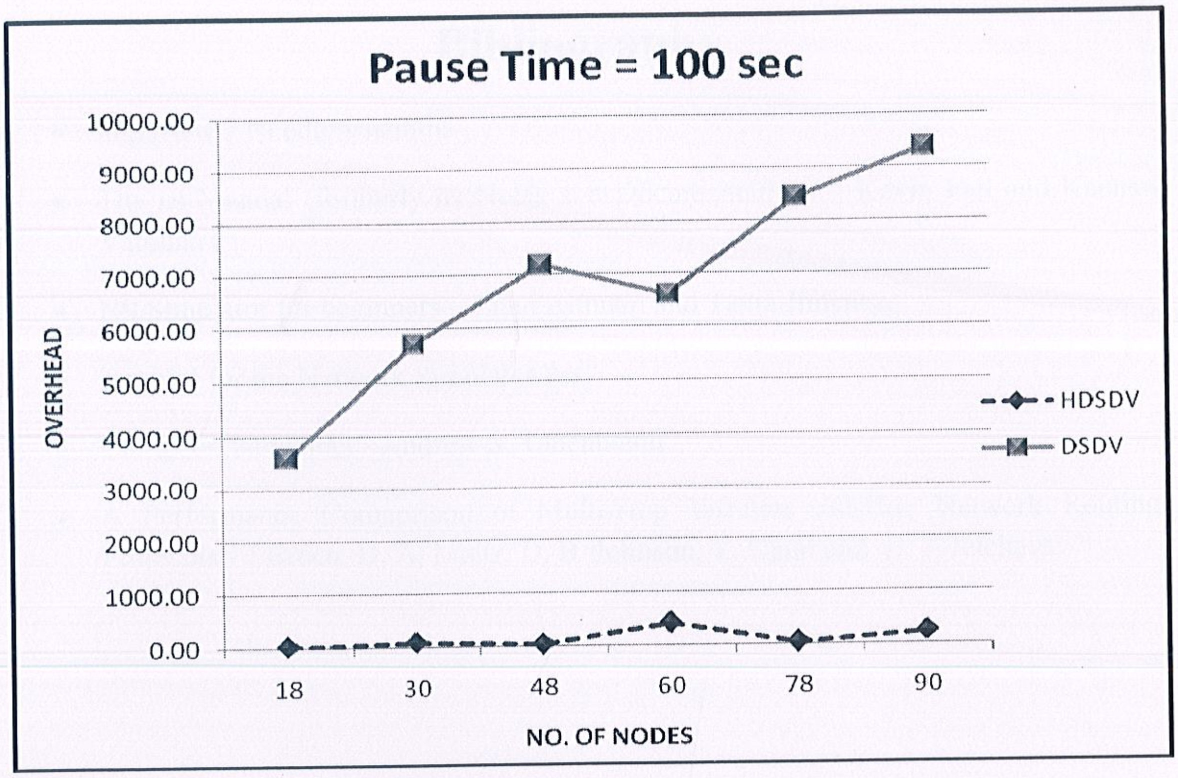


Fig 6.17 Routing overhead of HDSDV and DSDV at different Network densities with 100sec pause time

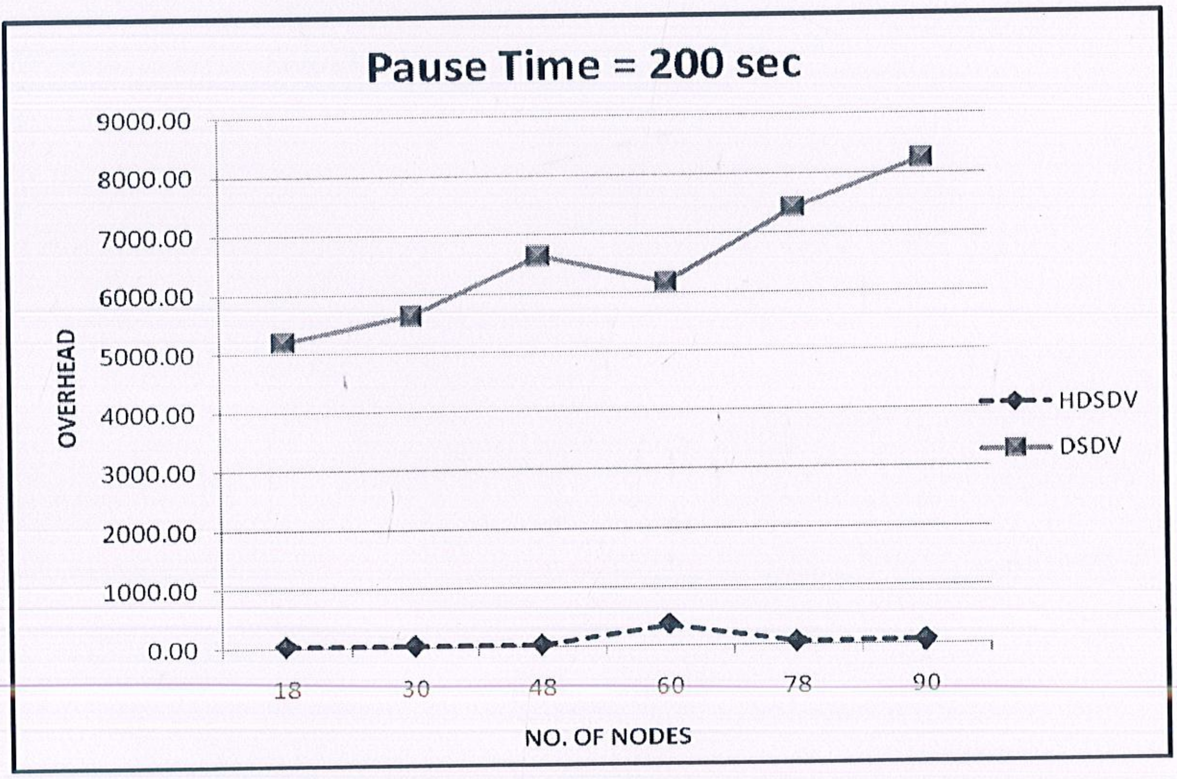


Fig 6.18 Routing overhead of HDSDV and DSDV at different Network densities with 200sec pause time

Bibliography

- http://www.isi.edu/nsnam/ns/
- The ns Manual (formerly ns Notes and Documentation) – Kevin Fall and Kannan Vardhan.
- NS Simulator for beginners – Eitan Altmam and Tania Jimenez.
- Tutorial for the Network Simulator "ns".
- Computer Networks - Andrew S. Tanenbaum.
- A Performance Comaprison of Multi-Hop Wireless Ad-Hoc Network Routing Protocols - J Broch, D. A. Maltz, D. B. Johnson, Y. Chun and H. J. Jetcheva.

Appendix

I. Main Program

i. First Scenario

A. Program for the ad-hoc routing protocols

```
=========================================================================
# Define options
=========================================================================

set val(chan)           Channel/WirelessChannel
set val(prop)           Propagation/TwoRayGround
set val(netif)          Phy/WirelessPhy
set val(mac)            Mac/802_11
set val(ifq)            Queue/DropTail/PriQueue ;#CMUPriQueue
set val(ll)             LL
set val(ant)            Antenna/OmniAntenna
set val(x)              1500    ;# X dimension of the topography
set val(y)              300     ;# Y dimension of the topography
set val(ifqlen)         50      ;# max packet in ifq
set val(seed)           1.0
set val(adhocRouting)   AODV
set val(nn)             50       ;# how many nodes are simulated
set val(cp)             "<Path of the file>"     ;# connection pattern file
set val(sc)             "<Path of the file>"    ;# node movement file.
set val(stop)            900.0               ;# simulation time

# ======================================================================#
Main Program
=========================================================================


#
# Initialize Global Variables
#

# create simulator instance

set ns_ [new Simulator]

# setup topography object

set topo [new Topography]

# create trace object for ns and nam

set tracefd       [open wireless11-out.tr w]
set namtrace    [open wireless11-out.nam w]

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

```
# define topology
$topo load_flatgrid $val(x) $val(y)

#
# Create God
#
set god_ [create-god $val(nn)]

#
# define how node should be created
#
# Create channel #1 and #2
set chan_1_ [new $val(chan)]
set chan_2_ [new $val(chan)]

#global node setting

$ns_ node-config -adhocRouting $val(adhocRouting) \
                 -llType $val(ll) \
                 -macType $val(mac) \
                 -ifqType $val(ifq) \
                 -ifqLen $val(ifqlen) \
                 -antType $val(ant) \
                 -propType $val(prop) \
                 -phyType $val(netif) \
                 -channeltype $val(chan) \
                 -topoInstance $topo \
                 -agentTrace ON \
                 -routerTrace ON \
                 -macTrace OFF
                 -movementTrace OFF \
                 -channel $chan_1_

#
#  Create the specified number of nodes [$val(nn)] and "attach" them  to the channel.

for {set i 0} {$i < $val(nn) } {incr i} {
        set node_($i) [$ns_ node]
        $node_($i) random-motion 0              ;# disable random motion
}


#
# Define node movement model
#
puts "Loading connection pattern..."
source $val(cp)

#
# Define traffic model
#
puts "Loading scenario file..."
source $val(sc)

# Define node initial position in nam
```

```
for {set i 0} {$i < $val(nn)} {incr i} {

# 20 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) 20
}


#
# Tell nodes when the simulation ends
#
for {set i 0} {$i < $val(nn) } {incr i} {
   $ns_ at $val(stop).0 "$node_($i) reset";
}

$ns_ at $val(stop).0001 "finish"
$ns_ at $val(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"

puts $tracefd "M 0.0 nn $val(nn) x $val(x) y $val(y) rp $val(adhocRouting)"
puts $tracefd "M 0.0 sc $val(sc) cp $val(cp) seed $val(seed)"
puts $tracefd "M 0.0 prop $val(prop) a nt $val(ant)"

proc finish {} {

   exec rm -f out1.xgr
   exec awk -f <path of awk file> <path of trace file> > <name of output file>


   puts "Finishing ns.."
   exit 0
}

puts "Starting Simulation..."
$ns_ run
```

B. Program for the Hierarchical Approach

```
# connected to a wireless domain through a base-station node.
# ==========================================================================
# Define options
# ==========================================================================
set opt(chan)           Channel/WirelessChannel    ;# channel type
set opt(prop)           Propagation/TwoRayGround   ;# radio-propagation model
set opt(netif)          Phy/WirelessPhy            ;# network interface type
set opt(mac)            Mac/802_11                 ;# MAC type
set opt(ifq)            Queue/DropTail/PriQueue    ;# interface queue type
#set opt(ifq)           CMUPriQueue

set opt(ll)             LL                         ;# link layer type
set opt(ant)            Antenna/OmniAntenna        ;# antenna model
set opt(ifqlen)         50                         ;# max packet in ifq
set opt(nn)             <no. Of nodes>             ;# number of mobilenodes
set opt(adhocRouting)   DSDV                       ;# routing protocol

set opt(cp)         "<Path of the file>"       ;# connection pattern file
set opt(sc)         "<Path of the file>" ;# node movement file.

set opt(x)              1500                       ;# x coordinate of topology
set opt(y)              300                        ;# y coordinate of topology
set opt(seed)           1.0                        ;# seed for random number gen.
set opt(stop)           900                        ;# time to stop simulation

set opt(ftp1-start)   160.0

set opt(ftp2-start)   170.0

set num_bs_nodes        6

#
==========================================================================
=====
# check for boundary parameters and random seed
if { $opt(x) == 0 || $opt(y) == 0 } {
        puts "No X-Y boundary values given for wireless topology\n"
}
if {$opt(seed) > 0} {
        puts "Seeding Random number generator with $opt(seed)\n"
        ns-random $opt(seed)
}

# create simulator instance
 set ns_   [new Simulator]

# set up for hierarchical routing
 $ns_ node-config -addressType hierarchical
AddrParams set domain_num_ 2            ;# number of domains
lappend cluster_num 3 3              ;# number of clusters in each domain
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel 9 10 9 10 9 9          ;# number of nodes in each cluster
```

```
AddrParams set nodes_num_ $eilastlevel ;# of each domain

set tracefd  [open wireless2mod-out.tr w]
set namtrace [open wireless2mod-out.nam w]
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)

# Create topography object
set topo   [new Topography]

# define topology
$topo load_flatgrid $opt(x) $opt(y)

# create God
set god_ [create-god $opt(nn)+$num_bs_nodes]
#create-god_ [expr $opt(nn) + $num_bs_nodes]

# Create channel #1 and #2
set chan_1_ [new $opt(chan)]
set chan_2_ [new $opt(chan)]


# configure for base-station node
$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 $topo \
                 -wiredRouting ON \
                 -agentTrace ON \
                 -routerTrace ON \
                 -macTrace OFF \
                 -channel $chan_1_



#create base-station node
set temp {0.0.0 0.1.0 0.2.0 1.0.0 1.1.0 1.2.0
          0.0.1 0.0.2 0.0.3 0.0.4 0.0.5 0.0.6 0.0.7 0.0.8
          0.1.1 0.1.2 0.1.3 0.1.4 0.1.5 0.1.6 0.1.7 0.1.8 0.1.9
          0.2.1 0.2.2 0.2.3 0.2.4 0.2.5 0.2.6 0.2.7 0.2.8
          1.0.1 1.0.2 1.0.3 1.0.4 1.0.5 1.0.6 1.0.7 1.0.8 1.0.9
          1.1.1 1.1.2 1.1.3 1.1.4 1.1.5 1.1.6 1.1.7 1.1.8
          1.2.1 1.2.2 1.2.3 1.2.4 1.2.5 1.2.6 1.2.7 1.2.8}


 ;# hier address to be used for wireless domain
set BS(0) [$ns_ node [lindex $temp 0]]
$BS(0) random-motion 0           ;# disable random motion
```

```
#provide some co-ord (fixed) to base station node
$BS(0) set X_ 175.0
$BS(0) set Y_ 150.0
$BS(0) set Z_ 0.0

#base station no 4 test imp

set BS(1) [$ns_ node [lindex $temp 1]]
$BS(1) random-motion 0              ;# disable random motion

#provide some co-ord (fixed) to base station node
$BS(1) set X_ 415.0
$BS(1) set Y_ 150.0
$BS(1) set Z_ 0.0

#base station no 4 test imp

set BS(2) [$ns_ node [lindex $temp 2]]
$BS(2) random-motion 0              ;# disable random motion

#provide some co-ord (fixed) to base station node
$BS(2) set X_ 655.0
$BS(2) set Y_ 150.0
$BS(2) set Z_ 0.0

#base station no 4 test imp

set BS(3) [$ns_ node [lindex $temp 3]]
$BS(3) random-motion 0              ;# disable random motion

#provide some co-ord (fixed) to base station node
$BS(3) set X_ 895.0
$BS(3) set Y_ 150.0
$BS(3) set Z_ 0.0


set BS(4) [$ns_ node [lindex $temp 4]]
$BS(1) random-motion 0              ;# disable random motion

#provide some co-ord (fixed) to base station node
$BS(4) set X_ 1135.0
$BS(4) set Y_ 150.0
$BS(4) set Z_ 0.0

set BS(5) [$ns_ node [lindex $temp 5]]
$BS(1) random-motion 0              ;# disable random motion

#provide some co-ord (fixed) to base station node
$BS(5) set X_ 1375.0
$BS(5) set Y_ 150.0
$BS(5) set Z_ 0.0


# create mobilenodes in the same domain as BS(0)
# note the position and movement of mobilenodes is as defined
# in $opt(sc)

$ns_ node-config -wiredRouting OFF

 for {set j 0} {$j < $opt(nn)} {incr j} {
   set node_($j) [ $ns_ node [lindex $temp \
              [expr $j+6]] ]
#    $node_($j) base-station [AddrParams addr2id \
#             [$BS(0) node-addr]]
}

#configure for mobilenodes

$ns_ node-config -wiredRouting OFF

            for {set i 0} {$i <= 7} {incr i} {
 #   set node_($j) [ $ns_ node [lindex $temp \
#             [expr $j+1]] ]

   $node_($i) base-station [AddrParams addr2id \
              [$BS(0) node-addr]]
}


#test imp
$ns_ node-config -wiredRouting OFF

 for {set n 8} {$n <= 16} {incr n} {
  # set node_($n+25) [ $ns_ node [lindex $temp \
         #    [expr $n+27]] ]

   $node_($n) base-station [AddrParams addr2id \
              [$BS(1) node-addr]]
}


#test imp
$ns_ node-config -wiredRouting OFF

 for {set n 17} {$n <= 24} {incr n} {
  # set node_($n+25) [ $ns_ node [lindex $temp \
         #    [expr $n+27]] ]

   $node_($n) base-station [AddrParams addr2id \
              [$BS(2) node-addr]]
}

$ns_ node-config -wiredRouting OFF

 for {set n 25} {$n <= 33} {incr n} {
  # set node_($n+25) [ $ns_ node [lindex $temp \
         #    [expr $n+27]] ]

   $node_($n) base-station [AddrParams addr2id \
                [$BS(3) node-addr]]
}

$ns_ node-config -wiredRouting OFF

 for {set n 34} {$n <= 41} {incr n} {
  # set node_($n+25) [ $ns_ node [lindex $temp \
           #    [expr $n+27]] ]

   $node_($n) base-station [AddrParams addr2id \
                [$BS(4) node-addr]]
}
$ns_ node-config -wiredRouting OFF

 for {set n 42} {$n <= 49} {incr n} {
  # set node_($n+25) [ $ns_ node [lindex $temp \
           #    [expr $n+27]] ]

   $node_($n) base-station [AddrParams addr2id \
                [$BS(5) node-addr]]
}

#test imp
$ns_ node-config -wiredRouting OFF

 for {set n 40} {$n <= 49} {incr n} {
  # set node_($n+25) [ $ns_ node [lindex $temp \
           #    [expr $n+27]] ]

   $node_($n) base-station [AddrParams addr2id \
                [$BS(3) node-addr]]
}



# source connection-pattern and node-movement scripts
if { $opt(cp) == "" } {
            puts "*** NOTE: no connection pattern specified."
       set opt(cp) "none"
} else {
            puts "Loading connection pattern..."
            source $opt(cp)
}
if { $opt(sc) == "" } {
            puts "*** NOTE: no scenario file specified."
       set opt(sc) "none"
} else {
            puts "Loading scenario file..."
            source $opt(sc)
            puts "Load complete..."
}

# Define initial node position in nam

for {set i 0} {$i < $opt(nn)} {incr i} {
    # 20 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) 20
}

# Tell all nodes when the simulation ends
for {set i } {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).0 "$node_($i) reset";
}
$ns_ at $opt(stop).0 "$BS(0) reset";

$ns_ at $opt(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"
$ns_ at $opt(stop).0001 "stop"
proc stop {} {

    exec rm -f <name of output file>
    exec awk -f    exec awk -f <path of awk file> <path of trace file> > <name of output file>
    global ns_ tracefd namtrace
#   $ns_ flush-trace
    close $tracefd
    close $namtrace

}

# informative headers for CMUTracefile
puts $tracefd "M 0.0 nn $opt(nn) x $opt(x) y $opt(y) rp \
        $opt(adhocRouting)"
puts $tracefd "M 0.0 sc $opt(sc) cp $opt(cp) seed $opt(seed)"
puts $tracefd "M 0.0 prop $opt(prop) ant $opt(ant)"



puts "Starting Simulation..."
$ns_ run
```

ii. Second Scenario

A. Program for the ad-hoc routing protocols

```
=======================================================================
# Define options
=======================================================================

set val(chan)           Channel/WirelessChannel
set val(prop)           Propagation/TwoRayGround
set val(netif)          Phy/WirelessPhy
set val(mac)            Mac/802_11
set val(ifq)            Queue/DropTail/PriQueue ;#CMUPriQueue
set val(ll)             LL
set val(ant)            Antenna/OmniAntenna
set val(x)              1000     ;# X dimension of the topography
set val(y)              500      ;# Y dimension of the topography
set val(ifqlen)         50       ;# max packet in ifq
set val(seed)           1.0
set val(adhocRouting)   <AODV or DSDV>
set val(nn)             50        ;# how many nodes are simulated
set val(cp)             "<Path of the file>"      ;# connection pattern file
set val(sc)             "<Path of the file>"  ;# node movement file.
set val(stop)            200.0              ;# simulation time

# ========================================================================#
Main Program
=======================================================================

#
# Initialize Global Variables
#

# create simulator instance

set ns_ [new Simulator]

# setup topography object

set topo [new Topography]

# create trace object for ns and nam

set tracefd       [open wireless11-out.tr w]
set namtrace    [open wireless11-out.nam w]

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

# define topology
$topo load_flatgrid $val(x) $val(y)

#
# Create God
```

```
#
set god_ [create-god $val(nn)]

#
# define how node should be created
#
# Create channel #1 and #2
set chan_1_ [new $val(chan)]
set chan_2_ [new $val(chan)]

#global node setting

$ns_ node-config -adhocRouting $val(adhocRouting) \
            -llType $val(ll) \
            -macType $val(mac) \
            -ifqType $val(ifq) \
            -ifqLen $val(ifqlen) \
            -antType $val(ant) \
            -propType $val(prop) \
            -phyType $val(netif) \
            -channeltype $val(chan) \
             -topoInstance $topo \
             -agentTrace ON \
            -routerTrace ON \
            -macTrace OFF
            -movementTrace OFF \
            -channel $chan_1_

#
#  Create the specified number of nodes [$val(nn)] and "attach" them  to the channel.

for {set i 0} {$i < $val(nn) } {incr i} {
         set node_($i) [$ns_ node]
         $node_($i) random-motion 0                ;# disable random motion
}


#
# Define node movement model
#
puts "Loading connection pattern..."
source $val(cp)

#
# Define traffic model
#
puts "Loading scenario file..."
source $val(sc)

# Define node initial position in nam

for {set i 0} {$i < $val(nn)} {incr i} {

 # 20 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) 20
}


#
# Tell nodes when the simulation ends
#
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop).0 "$node_($i) reset";
}

$ns_ at  $val(stop).0001 "finish"
$ns_ at  $val(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"

puts $tracefd "M 0.0 nn $val(nn) x $val(x) y $val(y) rp $val(adhocRouting)"
puts $tracefd "M 0.0 sc $val(sc) cp $val(cp) seed $val(seed)"
puts $tracefd "M 0.0 prop $val(prop) a nt $val(ant)"

proc finish {} {

    exec rm -f <name of output file>
    exec awk -f    exec awk -f <path of awk file> <path of trace file> > <name of output file>

    puts "Finishing ns.."
    exit 0
}

puts "Starting Simulation..."
$ns_ run
```

B. Program for the Hierarchical Approach

```
# connected to a wireless domain through a base-station node.
# ==========================================================================
# Define options
# ==========================================================================
set opt(chan)           Channel/WirelessChannel    ;# channel type
set opt(prop)           Propagation/TwoRayGround   ;# radio-propagation model
set opt(netif)          Phy/WirelessPhy            ;# network interface type
set opt(mac)            Mac/802_11                 ;# MAC type
set opt(ifq)            Queue/DropTail/PriQueue    ;# interface queue type
set opt(ll)             LL                         ;# link layer type
set opt(ant)            Antenna/OmniAntenna        ;# antenna model
set opt(ifqlen)         50                         ;# max packet in ifq
set opt(nn)             <no.of nodes>              ;# number of mobilenodes
set opt(adhocRouting)   DSDV                       ;# routing protocol

set opt(cp)        "<Path of the file>"     ;# connection pattern file
set opt(sc)        "<Path of the file>"     ;# node movement file.

set opt(x)              1000          ;# x coordinate of topology
set opt(y)              500           ;# y coordinate of topology
set opt(seed)           1.0           ;# seed for random number gen.
set opt(stop)           200           ;# time to stop simulation

set opt(ftp1-start)   160.0

set opt(ftp2-start)   170.0

set num_bs_nodes      6

#
==========================================================================
=====
# check for boundary parameters and random seed
if { $opt(x) == 0 || $opt(y) == 0 } {
        puts "No X-Y boundary values given for wireless topology\n"
}
if {$opt(seed) > 0} {
        puts "Seeding Random number generator with $opt(seed)\n"
        ns-random $opt(seed)
}

# create simulator instance
 set ns_   [new Simulator]

# set up for hierarchical routing
 $ns_ node-config -addressType hierarchical
AddrParams set domain_num_ 2          ;# number of domains
lappend cluster_num 3 3           ;# number of clusters in each domain
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel 9 10 9 10 9 9           ;# number of nodes in each cluster
AddrParams set nodes_num_ $eilastlevel ;# of each domain
```

```
set tracefd  [open wireless2mod-out.tr w]
set namtrace [open wireless2mod-out.nam w]
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)

# Create topography object
set topo   [new Topography]

# define topology
$topo load_flatgrid $opt(x) $opt(y)

# create God
set god_ [create-god $opt(nn)+$num_bs_nodes]
#create-god_ [expr $opt(nn) + $num_bs_nodes]

# Create channel #1 and #2
set chan_1_ [new $opt(chan)]
set chan_2_ [new $opt(chan)]


# configure for base-station node
$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 $topo \
            -wiredRouting ON \
            -agentTrace ON \
            -routerTrace ON \
            -macTrace OFF \
            -channel $chan_1_



#create base-station node
set temp {0.0.0 0.1.0 0.2.0 1.0.0 1.1.0 1.2.0
          0.0.1 0.0.2 0.0.3 0.0.4 0.0.5 0.0.6 0.0.7 0.0.8
          0.1.1 0.1.2 0.1.3 0.1.4 0.1.5 0.1.6 0.1.7 0.1.8 0.1.9
          0.2.1 0.2.2 0.2.3 0.2.4 0.2.5 0.2.6 0.2.7 0.2.8
          1.0.1 1.0.2 1.0.3 1.0.4 1.0.5 1.0.6 1.0.7 1.0.8 1.0.9
          1.1.1 1.1.2 1.1.3 1.1.4 1.1.5 1.1.6 1.1.7 1.1.8
          1.2.1 1.2.2 1.2.3 1.2.4 1.2.5 1.2.6 1.2.7 1.2.8}


 ;# hier address to be used for wireless domain
set BS(0) [$ns_ node [lindex $temp 0]]
$BS(0) random-motion 0              ;# disable random motion

#provide some co-ord (fixed) to base station node
```

```
$BS(0) set X_ 250.0
$BS(0) set Y_ 150.0
$BS(0) set Z_ 0.0

#base station no 4 test imp

set BS(1) [$ns_ node [lindex $temp 1]]
$BS(1) random-motion 0              ;# disable random motion

#provide some co-ord (fixed) to base station node
$BS(1) set X_ 250.0
$BS(1) set Y_ 350.0
$BS(1) set Z_ 0.0

#base station no 4 test imp

set BS(2) [$ns_ node [lindex $temp 2]]
$BS(2) random-motion 0              ;# disable random motion

#provide some co-ord (fixed) to base station node
$BS(2) set X_ 500.0
$BS(2) set Y_ 350.0
$BS(2) set Z_ 0.0

#base station no 4 test imp

set BS(3) [$ns_ node [lindex $temp 3]]
$BS(3) random-motion 0              ;# disable random motion

#provide some co-ord (fixed) to base station node
$BS(3) set X_ 500.0
$BS(3) set Y_ 150.0
$BS(3) set Z_ 0.0


set BS(4) [$ns_ node [lindex $temp 4]]
$BS(1) random-motion 0              ;# disable random motion

#provide some co-ord (fixed) to base station node
$BS(4) set X_ 750.0
$BS(4) set Y_ 150.0
$BS(4) set Z_ 0.0

set BS(5) [$ns_ node [lindex $temp 5]]
$BS(1) random-motion 0              ;# disable random motion

#provide some co-ord (fixed) to base station node
$BS(5) set X_ 750.0
$BS(5) set Y_ 350.0
$BS(5) set Z_ 0.0


# create mobilenodes in the same domain as BS(0)
# note the position and movement of mobilenodes is as defined
# in $opt(sc)
```

```
$ns_ node-config -wiredRouting OFF

 for {set j 0} {$j < $opt(nn)} {incr j} {
   set node_($j) [ $ns_ node [lindex $temp \
               [expr $j+6]] ]
#   $node_($j) base-station [AddrParams addr2id \
#              [$BS(0) node-addr]]
}

#configure for mobilenodes

$ns_ node-config -wiredRouting OFF

           for {set i 0} {$i <= 7} {incr i} {
 #   set node_($j) [ $ns_ node [lindex $temp \
#               [expr $j+1]] ]

    $node_($i) base-station [AddrParams addr2id \
               [$BS(0) node-addr]]
}


#test imp
$ns_ node-config -wiredRouting OFF

 for {set n 8} {$n <= 16} {incr n} {
   # set node_($n+25) [ $ns_ node [lindex $temp \
           #    [expr $n+27]] ]

    $node_($n) base-station [AddrParams addr2id \
               [$BS(1) node-addr]]
}


#test imp
$ns_ node-config -wiredRouting OFF

 for {set n 17} {$n <= 24} {incr n} {
   # set node_($n+25) [ $ns_ node [lindex $temp \
           #    [expr $n+27]] ]

    $node_($n) base-station [AddrParams addr2id \
               [$BS(2) node-addr]]
}

$ns_ node-config -wiredRouting OFF

 for {set n 25} {$n <= 33} {incr n} {
   # set node_($n+25) [ $ns_ node [lindex $temp \
           #    [expr $n+27]] ]

    $node_($n) base-station [AddrParams addr2id \
               [$BS(3) node-addr]]
}
```

```
$ns_ node-config -wiredRouting OFF

 for {set n 34} {$n <= 41} {incr n} {
  # set node_($n+25) [ $ns_ node [lindex $temp \
         #    [expr $n+27]] ]

   $node_($n) base-station [AddrParams addr2id \
              [$BS(4) node-addr]]
}
$ns_ node-config -wiredRouting OFF

 for {set n 42} {$n <= 49} {incr n} {
  # set node_($n+25) [ $ns_ node [lindex $temp \
         #    [expr $n+27]] ]

   $node_($n) base-station [AddrParams addr2id \
              [$BS(5) node-addr]]
}

#test imp
$ns_ node-config -wiredRouting OFF

 for {set n 40} {$n <= 49} {incr n} {
  # set node_($n+25) [ $ns_ node [lindex $temp \
         #    [expr $n+27]] ]

   $node_($n) base-station [AddrParams addr2id \
              [$BS(3) node-addr]]
}



# source connection-pattern and node-movement scripts
if { $opt(cp) == "" } {
            puts "*** NOTE: no connection pattern specified."
       set opt(cp) "none"
} else {
            puts "Loading connection pattern..."
            source $opt(cp)
}
if { $opt(sc) == "" } {
            puts "*** NOTE: no scenario file specified."
       set opt(sc) "none"
} else {
            puts "Loading scenario file..."
            source $opt(sc)
            puts "Load complete..."
}

# Define initial node position in nam

for {set i 0} {$i < $opt(nn)} {incr i} {

   # 20 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) 20
}

# Tell all nodes when the simulation ends
for {set i } {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).0 "$node_($i) reset";
}
$ns_ at $opt(stop).0 "$BS(0) reset";

$ns_ at $opt(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"
$ns_ at $opt(stop).0001 "stop"
proc stop {} {


    exec rm -f  <name of output file>
    exec awk -f    exec awk -f <path of awk file> <path of trace file> > <name of output file>
    global ns_ tracefd namtrace
#   $ns_ flush-trace
    close $tracefd
    close $namtrace

}

# informative headers for CMUTracefile
puts $tracefd "M 0.0 nn $opt(nn) x $opt(x) y $opt(y) rp \
            $opt(adhocRouting)"
puts $tracefd "M 0.0 sc $opt(sc) cp $opt(cp) seed $opt(seed)"
puts $tracefd "M 0.0 prop $opt(prop) ant $opt(ant)"



puts "Starting Simulation..."
$ns_ run
```

II. AWK files

i. AWK file to calculate Packet Delivery Ratio

```
BEGIN{
        sim_end = 900;
        sl=0;
        rl=0;
        }

{
        if ($1=="s" && $7=="cbr" && $4=="AGT" ) {
                sl++;

        };
}


{
        if ($1=="r" && $7=="cbr" && $4=="AGT" ) {
                rl++;

        };
}


END{print "throughput : " (rl/sl) }
```

ii. AWK file to calculate routing overheads

```
BEGIN{
        sim_end = 900;
        sl=0;
        rl=0;
}

{
        if ($1=="s" && $4=="RTR" ) {
                sl++;

        };
}


{
        if ($1=="f" && $4=="RTR" ) {
                rl++;

        };
}


END{print "Routing overheads : " (rl+sl) }
```