Volume 2, Issue 3, March 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Simulation Based Study of AODV, DSR, DSDV Routing Protocols in MANET Using NS-2

G. Jose Moses*
Research Scholar
Department of Computer Science
Adikavi Nannaya University
Rajahmundry, A.P, INDIA
josemoses@gmail.com

D. Sunil Kumar
Student
Department of MCA
Government College
Rajahmundry, A.P, INDIA

Prof.P.Suresh Varma
Professor
Department of Computer Science
Adikavi Nannaya University
Rajahmundry, A.P, INDIA

N.Supriya
Faculty
Department of CS
Adikavi Nannaya University
Rajahmundry, A.P,

Abstract— **Mobile Ad-hoc Networks are autonomously self-organized networks without infrastructure support. To facilitate communication within the network a routing protocol is used to discover routes between nodes. The main aim of the routing protocol is to have an efficient route establishment between a pair of nodes, so that messages can be delivered in a timely manner. Routing in the MANETs is a challenging task which has led to development of many different routing protocols for MANETs. In this paper, an attempt has been made to implement and compare performance of three well know protocols AODV, DSR and DSDV by using four performance metrics Throughput, Packet delivery ratio, End to End delay and Average Routing load, the performance analysis has been done by using simulation tool ns-2 which is the main simulator.**

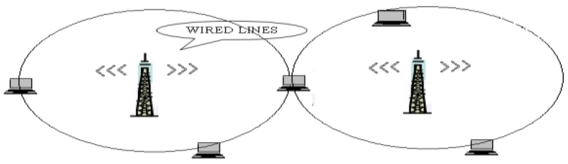
Keywords— **Ad-hoc, MANET, AODV, DSR, DSDV, NS-2.**

INTRODUCTION

Wireless networking is an emerging technology that allows users to access information and services electronically, regardless of their geographic position. Wireless networks can be classified in two types. [1]

Infrastructure Networks

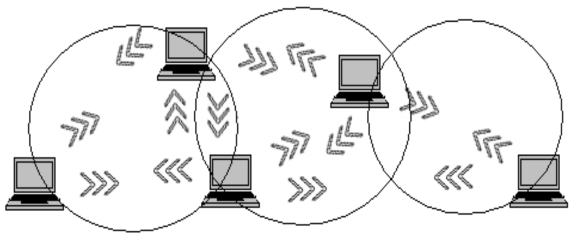
Infrastructure network consists of a network with fixed and wired gateways. A mobile host communicates with a bridge in the network (called base station) within its communication radius. The mobile unit can move geographically while it is communicating. When it goes out of range of one base station, it connects with new base station and starts communicating through it. This is called handoff. In this approach the base stations are fixed.



Infrastructure Network

Infrastructure Less (Ad hoc) Networks

In ad hoc networks all nodes are mobile and can be connected dynamically in an arbitrary manner. As the range of each host's wireless transmission is limited, so to communicate with hosts outside its transmission range, a host needs to enlist the aid of its nearby hosts in forwarding packets to the destination. So all nodes of these networks behave as routers and take part in discovery and maintenance of routes to other nodes in the network.



Mobile Ad hoc Network

Challenges in MANETs

- Power controlling at Physical layer
- Efficient routing at Network layer

- Quality of service at Transport layer

Routing Protocols

Design of the efficient routing protocol in the MANET environment is difficult because of its “short live” nature. And as the network topologies are dynamically changed. Routing protocols for MANETs can be broadly classified into two categories. Those are

- Proactive or table-driven routing protocols
- Reactive or on-demand routing protocols.

Proactive routing protocols

Proactive MANET protocols are table-driven and will actively determine the layout of the network. Through a regular exchange of network topology packets between the nodes of the network, a complete picture of the network is maintained at every single node. There is hence minimal delay in determining the route to be selected. Some Proactive MANET Protocols include: DSDV, DBF, GSR, WRP, ZRP, and FSR.

Reactive routing protocols

On-demand routing is a popular routing category for wireless ad hoc routing. It is a relatively new routing philosophy that provides a scalable solution to relatively large network topologies. The design follows the idea that each node tries to reduce routing overhead by only sending routing packets when communication is requested. Common for most on-demand routing protocols are the route discovery phase where packets are flooded into the network in search of an optimal path to the destination node in the network. Some Reactive MANET Protocols include: DSR, AODV and TORA.

The emphasis in this paper is concentrated on the comparison of various Proactive and Reactive protocols like DSDV [2], DSR [4] and AODV [5]

DSDV

This protocol is based on classical Bellman-Ford routing algorithm [3] designed for MANETS. Each node maintains a list of all destinations and number of hops to each destination. Each entry is marked with a sequence number. It uses full dump or incremental update to reduce network traffic generated by route updates. The broadcast of route updates is delayed by settling time. The only improvement made here is avoidance of routing loops in a mobile network of routers. With this improvement, routing information is always available, regardless whether the source node requires the information or not. With the addition of sequence numbers, routes for the same destination are selected based on the following rules: [2]

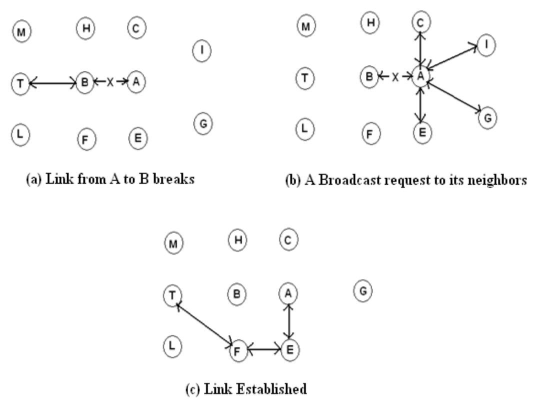
1. A route with a newer sequence number is preferred.
2. In the case that two routes have a same sequence number, the one with a better cost metric is preferred.

The list which is maintained is called routing table. The routing table contains the following:

Destination	Next Hop	No. of Hops	Sequence No	Install time

The sequence number is used to distinguish stale routes from new ones and thus it avoids the formation of loops. The stations periodically transmit their routing tables to their immediate neighbors. A station also transmits its routing table if a significant change has occurred in its table from the last update sent. So, the update is both time-driven and event-driven.
Each row of the update send is of the following form:

<Dest. IP address, Dest. sequence number, Hop count>
After receiving an update neighboring nodes utilizes it to compute the routing table entries.


(a) Link from A to B breaks
(b) A Broadcast request to its neighbors
(c) Link Established
Resolving route failure in DSDV

DSR

The Dynamic Source Routing protocol (DSR) [4] is a simple and efficient routing protocol designed specifically for use in multi-hop wireless ad hoc networks of mobile nodes. DSR allows the network to be completely self-organizing and self-configuring, without the need for any existing network infrastructure or administration. It uses source routing which means that the source must know the complete hop sequence to the destination. Each node maintains a route cache, where all the known routes are stored. The route discovery process is initiated only if the desired route cannot be found in the route cache.

To limit the number of route requests propagated, a node processes the route request message only if it has not already received the message and its address is not present in the route record of the message. The advantage is that intermediate nodes can learn routes from the source routes in the packets they receive.

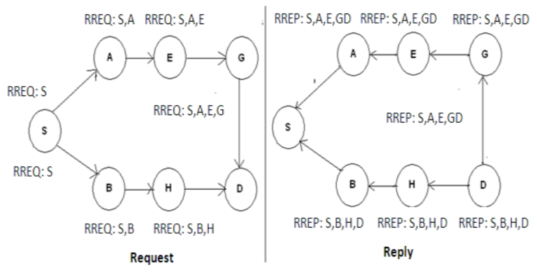
The protocol is composed of the two main mechanisms of "Route Discovery" and "Route Maintenance", which work together to allow nodes to discover and maintain routes to arbitrary destinations in the ad hoc network. The protocol allows multiple routes to any destination and allows each sender to select and control the routes used in routing its packets, for example, for use in load balancing or for increased robustness.

Route Discovery

Route Discovery is used whenever a source node desires a route to a destination node. First, the source node looks up its route cache to determine if it contains a route to the destination. If the source finds a valid route to the destination, it uses this route to send its data packets. If the node does not have a valid route to the destination, it initiates the route discovery process by broadcasting a route request message. The route request message contains the address of the source and the destination, and a unique identification number. An intermediate node that receives a route request message searches its route cache for a route to the destination. If no route is found, it appends its address to the route record of the message and forwards the message to its neighbors. The message propagates through the network until it reaches either the destination or an intermediate node with a route to the destination. Then a route reply message, containing the proper hop sequence for reaching the destination, is generated and unicast back to the source node.

Route maintenance

Route Maintenance is used to handle route breaks. When a node encounters a fatal transmission problem at its data link layer, it removes the route from its route cache and generates a route error message. The route error message is sent to each node that has sent a packet routed over the broken link. When a node receives a route error message, it removes the hop in error from its route cache. Acknowledgment messages are used to verify the correct operation of the route links.

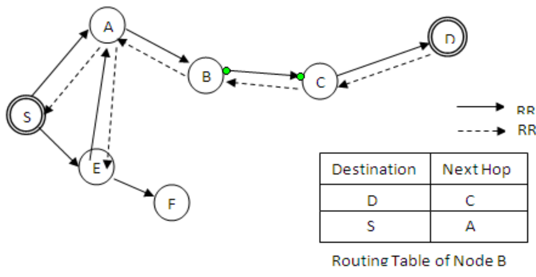


DSR request and Reply

AODV

The Ad hoc On Demand Vector routing algorithm [5] is a routing protocol designed for ad hoc mobile networks. AODV is capable of both unicast and multicast routing. It is an on demand routing algorithm, means that it builds routes between nodes only as desired by the source nodes. It maintains these routes as long as they are needed by the sources.

AODV uses sequence numbers to ensure the freshness of routes. It is a loop-free, self starting and scales to large number of mobile nodes. For example, node S intends to find a route to node D, the process is shown in the below figure



Destination	Next Hop
D	C
S	A

AODV Routing Protocol Model

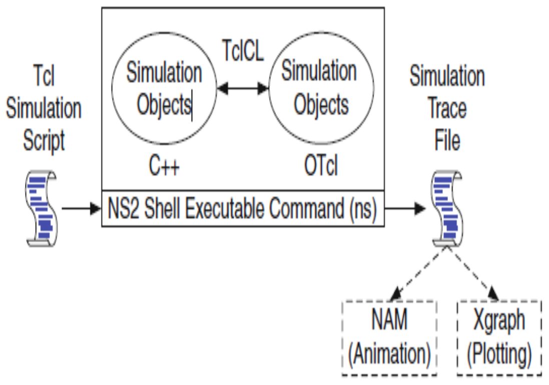
AODV builds routes using a route REQUEST and route REPLY query cycle. When a source node desires a route to a destination for which it does not have a route, it broadcasts a route request (RREQ) packet across the network. Nodes receiving the packet update their information for the source node and set 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 the 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.
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. 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 timeout 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.

ABOUT NS-2

Ns-2 is an object-oriented simulator developed as part of the VINT project at the University of California in Berkeley. Ns2 is extensively used by the networking research community. It provides substantial support for simulation of TCP, routing, multicast protocols over wired and wireless (local and satellite) networks, etc.

Basic Architecture of NS

NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events (i.e., a frontend).



Basic architecture of NS.

Main NS2 Simulation Steps

The followings show the three key steps guideline in defining a simulation scenario in a NS2: [6][7][8]

Step 1: Simulation Design: The first step in simulating a network is to design the simulation. In this step, the users should determine the simulation purposes, network configuration and assumptions, the performance measures, and the type of expected results.

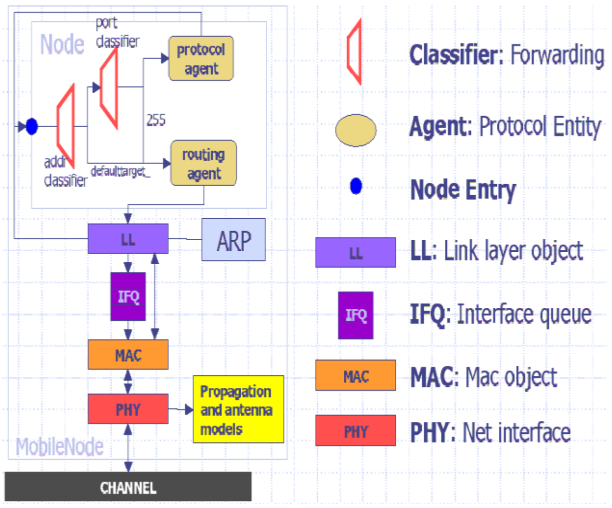
Step 2: Configuring and Running Simulation: This step implements the design in the first step. It consists of two phases:

Network configuration phase: In this phase network components (e.g., node, TCP and UDP) are created and configured according to the simulation design. Also, the events such as data transfer that are scheduled to start at a certain time.

Simulation Phase: This phase starts the simulation which was configured in the Network Configuration Phase. It maintains the simulation clock and executes events chronologically. This phase usually runs until the simulation clock reached a threshold value specified in the Network Configuration Phase.

Step 3: Post Simulation Processing: The main tasks in this step include verifying the integrity of the program and evaluating the performance of the simulated network. While the first task is referred to as *debugging*, the second one is achieved by properly collecting and compiling simulation results.

Wireless node model in Ns2 [7]



CORE IMPLEMENTATION

Here we discuss how the three protocols i.e. AODV, DSR and DSDV were simulated and implemented. We will present step by step procedure of how to do all the things done by one simulation in ns-2 with one Tcl scripts sequence:

Step 1: Create an instance of the simulator:

set ns_ [new Simulator]

Step.2: Setup trace support by opening file "sample_trace.tr" and call the procedure trace-all

set tracefd [open sampletrace.tr w]
$ns_ trace-all $tracefd

Step 3: Create a topology object that keeps track of all the nodes within boundary

set topo [new Topography]

Step 4: The topography is broken up into grids and the default value of grid resolution is 1. A different value can be passed as a third parameter to load_flatgrid {}.

$topo load_flatgrid $val(x) $val(y)

Step 5: Create the object God (General Operations Director) is the object that is used to store global information about the state of the environment, network or nodes. The procedure create-god is defined in $NS2_HOME/tcl/mobility/com.tcl, which allows only a single global instance of the God object to be created during a simulation. God object is called internally by MAC objects in nodes, so we must create god in every cases.

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

Step 6: Before we can create node, we first needs to configure them. Node configuration API may consist of defining the type of addressing (flat/hierarchical etc), for example, the type of adhoc routing protocol, Link Layer, MAC layer, IfQ etc.

```
$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) \
-channe
-channel [new $val(chan)] \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON \
-macTrace OFF \
-movementTrace OFF
```

Step 7: Create nodes and the random-motion for nodes is disabled here, as we are going to provide node position and movement (speed & direction) directives next

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

Step 8: Give nodes positions to start with, Provide initial (X,Y, for now Z=0) co-ordinates for node_(0) and node_(1). Node 0 has a starting position of (5,2) while Node 1 starts off at location (390,385).

```
$node_(0) set X_ 5.0
$node_(0) set Y_ 2.0
$node_(0) set Z_ 0.0
$node_(1) set X_ 390.0
$node_(1) set Y_ 385.0
$node_(1) set Z_ 0.0
```

Step 9: Setup node movement as the following example, at time 50.0s, node1 starts to move towards the destination (x=25, y=20) at a speed of 15m/s. This API is used to change direction and speed of movement of nodes.

```
$ns_ at 50.0 "$node_(1) setdest 25.0 20.0 15.0”
```

Step 10: Setup traffic flow between the two nodes as follows: TCP connections between node_(0) and node_(1)

```
set tcp [new Agent/TCP]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns_ attach-agent $node_(0) $tcp
$ns_ attach-agent $node_(1) $sink
$ns_ connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at 10.0 "$ftp start"
```

Step 11: Define stop time when the simulation ends and tell nodes to reset which actually resets their internal network components. In the following case, at time 150.0s, the simulation shall stop. The nodes are reset at that time and the "$ns_ halt" is called at 150.0002s, a little later after resetting the nodes. The procedure stop{} is called to flush out traces and close the trace file.

```
for {set i 0} {$i < $val(nn) } {incr i} {
$ns_ at 150.0 "$node_($i) reset";   }
                    $ns_ at 150.0001 "stop"
                $ns_ at 150.0002 "puts \"NS EXITING...\" ;
$ns_ halt"
                    proc stop {} {
                       global ns_ tracefd nf
                       $ns_ flush-trace
                       close $tracefd
                       close $nf
                    }
```

Step 12: Finally the command to start the simulation is

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

So, these 12 steps could finish one time simulation, and we can pack these 12 steps into one tcl file and do the simulation. However, there exist some problems on such kind of use on typical network performance test situations. Performance testing usually needs to be scalable in the number of nodes and network transmitting packets. Suppose in a network there are hundreds of nodes, we need to set all of the nodes positions and their movement, this needs a huge amount of workload, also, suppose if we setup all the possible sources and destinations and connections, it will be more complicated, Furthermore, even if we can set them, we cannot guarantee our input is randomly select, which is necessary for a fair comparison.
To overcome this we can use some third party tools included in ns-2.

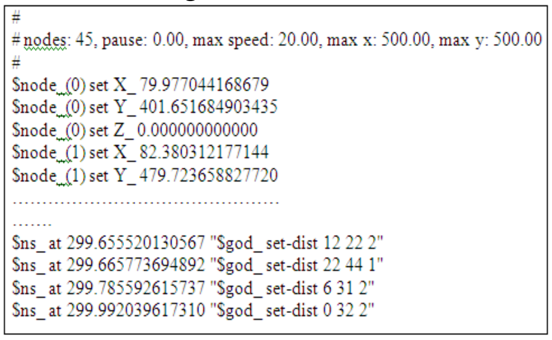
NETWORK SCENARIO GENERATING [8]

For nodes positions and their movement, we can generate a file with the statements which set nodes positions and nodes movement using CMU generator.

It is under $NS2_HOME/indep-utils/cmu-scen-gen/setdest. But, before we use it, we need to run “make” to create executable “setdest” program.

In fact, this is a third party tool which is CMU's version auxiliary scenario creation tool. It uses system dependent /dev/random and calls the library functions initstate() for generating random numbers.

The usage of this executable command is:
/setdest [-n num_of_nodes] [-p pausetime] [-s maxspeed][-t simtime] [-x maxx] [-y maxy] > [scenario_output_file]
For example, the command we use is like:
./setdest -n 40 -p 50.0 -s 10.0 -t 200 -x 500 -y 500 > scen-40-test
This means, the topology boundary is 500m X 500m, the scenario has 40 nodes with nodes' max moving speed of 10.0m/s and the pause between movements is 50s. Also, Simulation will stop in 200s, and finally, output generates tcl statements into a file scen-40-test. Some fragments of scen-40-test are shown in Figure below

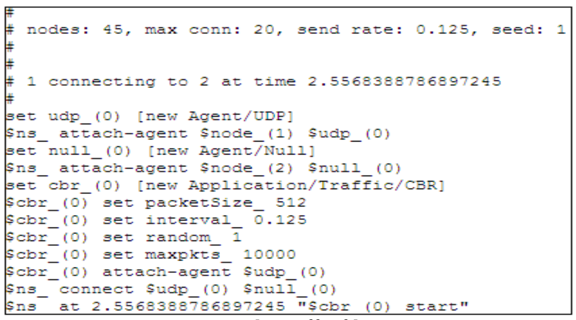


```
#
# nodes: 45, pause: 0.00, max speed: 20.00, max x: 500.00, max y: 500.00
#
$node_(0) set X_ 79.977044168679
$node_(0) set Y_ 401.651684903435
$node_(0) set Z_ 0.000000000000
$node_(1) set X_ 82.380312177144
$node_(1) set Y_ 479.723658827720
.........................................
.......
$ns_ at 299.655520130567 "$god_ set-dist 12 22 2"
$ns_ at 299.665773694892 "$god_ set-dist 22 44 1"
$ns_ at 299.785592615737 "$god_ set-dist 6 31 2"
$ns_ at 299.992039617310 "$god_ set-dist 0 32 2"
```

Sample Scenario file

NETWORK TRAFFIC GENERATING [8]

For network traffic generating, we must generate statements on sources, connections, and so on. This work could be done by a tcl file, which is in $NS2_HOME/indep-utils/cmu-scen gen/cbrgen.tcl.
For this network traffic generating tool, random traffic connections of TCP and CBR can be setup between nodes. It is used to create CBR and TCP traffic connections between wireless nodes. In order to create a traffic-connection file, we need to define 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 incase of CBR connections, a rate whose inverse value is used to compute the interval time between the CBR packets. So the command line is:
ns cbrgen.tcl [-type cbr/tcp] [-nn nodes] [-seed seed][-mc connections] [-rate rate]
Here, "-type cbr/tcp" means define the type of traffic connection, "-nn nodes" means the number of nodes could be used, "-mc connections" means maximum number of connections to be setup between those nodes, "-seed seed" means a random seed, if it not equal to 0, the traffic pattern will reappear if all the other parameters are the same. "-rate rate" means a rate whose inverse value is used to compute the interval time, which is called as packets sending rate.
For an example:
ns cbrgen.tcl -type cbr -nn 40 -seed 1.0 -mc 20 -rate 4.0> cbr-20-test
means create a CBR connection pattern between 40 nodes, having maximum of 20 connections, with a seed value of 1.0 and a rate of 4.0 pkts/second. Figure shows one fragment of traffic generating output



```
#
# nodes: 45, max conn: 20, send rate: 0.125, seed: 1
#
#
# 1 connecting to 2 at time 2.5568388786897245
#
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(1) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(2) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 0.125
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 2.5568388786897245 "$cbr_(0) start"
```

Sample Traffic file

To add them to main tcl file the following steps are done:

1. Add two variables in parameter options in the main tcl file
set val(sc) "/home/SUNIL/scenario/scen-40 test "
set val(cp) "/home/SUNIL/movement/cbr-40-test"
Here, "/home/SUNIL/scenario/" could be on directory where we store all your tcl files

2. In the main tcl file, just replace all the tcl statements which are related to node placement and movement, and also traffic creating with the following two statements
source $val(sc)
source $val(cp)

PERFORMANCE METRICS:

$$\text{Packet Delivery Ratio} = \frac{\text{Number of packets received successfully}}{\text{Number of packets sent}}$$

$$\text{Average Throughput} = \frac{\text{Total Received size}}{\text{Elapsed time between sent and receive}}$$

$$\text{Average Routing Load} = \frac{\text{Numbe r of Routing Control Packets}}{\text{Total Simulation Time}}$$

Average End to End Delay = "Sum (for each i equal to packet number, (packet i received time- packet i sent time))"

Trace format

<event> <time> <Destination id> <MAC/AGT.RTR/IFQ>---<seq no> <packet type><size><[exp time MAC sender ID Mac Receiver ID MAC type]>----<[source IP Destination IP TTL]><[tcp seq no tcp ck no]>

AWK SCRIPT TO ANALYZE TRACE FILES [9]

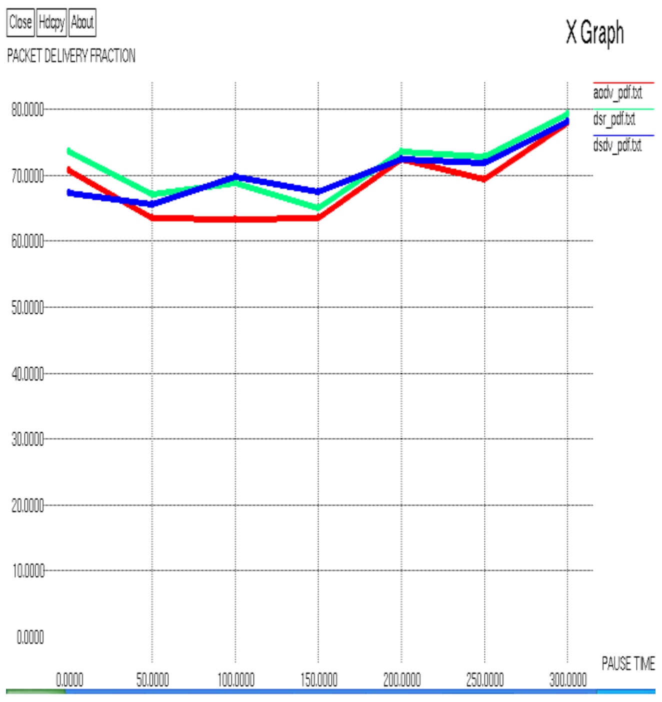
```
BEGIN {
    seqno = -1,  droppedPackets = 0;
    receivedPackets = 0, count = 0;
    sentpackets = 0, ctrlpac = 0,  pdf = 0;
}
{
if($4 == "AGT"&&$1=="s"&&seqno<$6) {
        seqno = $6;
```

```
         sentpackets++;
 }
else if(($4 == "AGT") && ($1 == "r")) {
            receivedPackets++;
    }
else if ($1=="d"&&($7=="tcp"||$7=="cbr") && $8 > 512){
           droppedPackets++;
}
else  if($4=="RTR"&&($1=="s"  ||  $1  ==  "f")  &&   ($7  ==
"DSR" || $7 == "AODV" || $7 == "message")) {
            ctrlpac++;
    }
   #end-to-end delay calculation
if($4 == "AGT" && $1 == "s") {
         start_time[$6] = $2;
    }
else if(($7 == "tcp"|| $7== "cbr")&& ($1 == "r")) {
         end_time[$6] = $2;
    }
else if($1 == "d" && ($7 == "tcp" || $7 == "cbr")) {
         end_time[$6] = -1;
    }
}
END {
for(i=0; i<=sentpackets; i++) {
 if(end_time[i] > 0) {
  delay[i] = end_time[i] - start_time[i];
  count++;
         }
  else{
                delay[i] = -1;
           }
 }
  for(i=0; i<=seqno; i++) {
   if(delay[i] > 0) {
    n_to_n_delay = n_to_n_delay + delay[i];
        }         }
   n_to_n_delay = n_to_n_delay/count;
print "GeneratedPackets           = " seqno+1;
print  "SentPackets                    =  "  sentpackets;       print
"ReceivedPackets = " receivedPackets;
pdf = receivedPackets/(sentpackets)*100
print "Packet Delivery Ratio     = " pdf " %";
print"TotalDroppedPackets = " droppedPackets;
print  "Average  End-to-End  Delay      =  "  n_to_n_delay*1000
"ms" ;
print "Control Packets = "ctrlpac;
print "Average Routing Load = "ctrlpac/300;
}
```

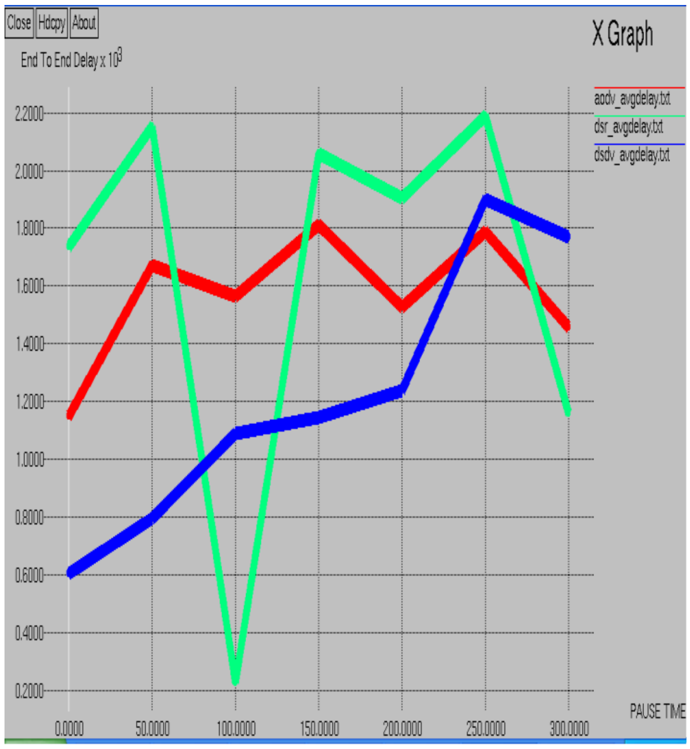
SIMULATION PARAMETERS

Parameter	Value
Transmission range	250 m
Simulation Time	300 s
Topology size	800m X 800m
No. of Mobile nodes	45
No. of Sources	20
Traffic Type	CBR (Constant Bit Rate)
Packet Rate	8 packets/sec
Packet size	512 bytes

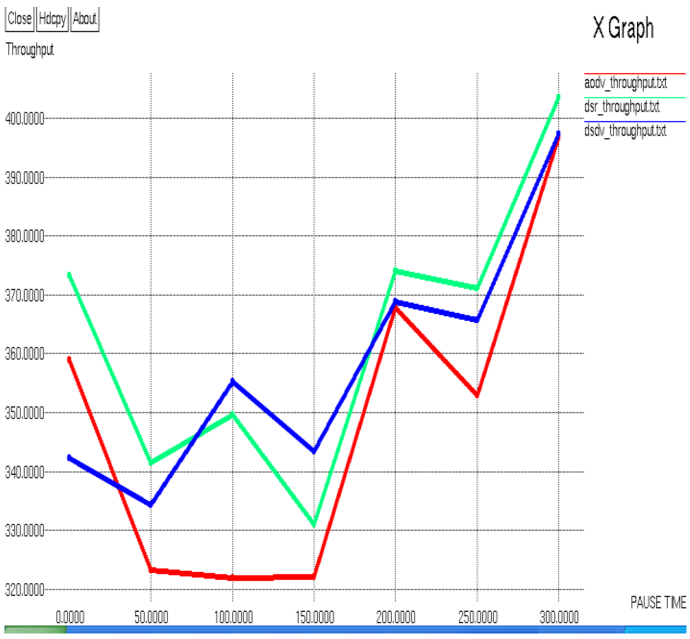
RESULTS



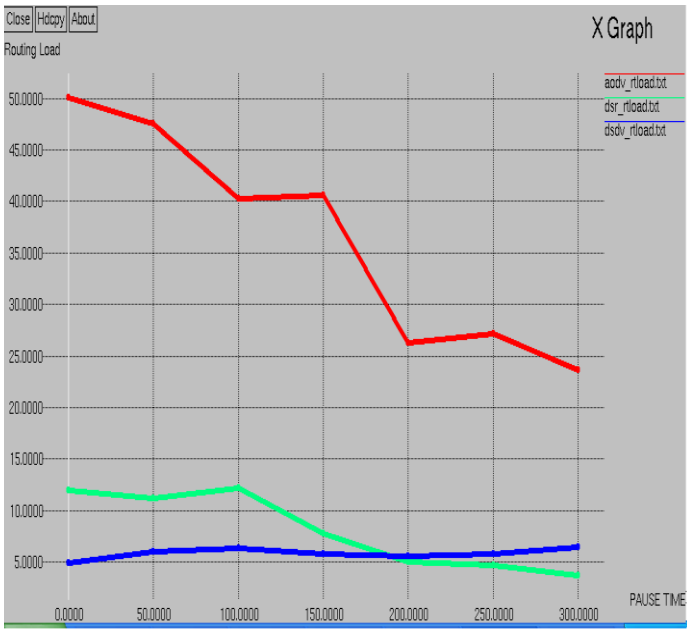
Graph 1 Packet Delivery Ratio



Graph 2 Average END to END Delay



Graph 3Throughput



Graph 4 Routing Load

Graph 1 shows packet delivery ratio with pause time varying from 0 to 300 for DSR, DSDV and AODV routing protocols. The red line shows graph for DSR, the green line shows the graph for DSDV and the blue line shows the graph for AODV protocol. The delivery ratio for all the protocols is always greater than 65 percent. The basic difference between these protocols is very less. But generally the graph for the DSR protocol lies above that of DSDV and AODV for most cases. However in certain cases the DSDV protocols is also better.
Graph 2 shows the average end-to-end delay is less for the DSDV approach than for the DSR approach, mean while AODV maintains consistency in this performance metric. The reason is that the periodic gateway information sent by the gateways allows the mobile nodes to update their route entries for the gateways more often, resulting in fresher and shorter routes in DSDV. With the DSR (reactive approach) a mobile node continues to use a route to a gateway until it is broken.

Graph 3 shows the result shows that the average throughput for DSDV and DSR are better with high mobility nodes. At the end of simulation times the AODV goes closer to the other in Average throughput.

Graph 4 shows the routing overhead for AODV always in a high peak rate when compared to the other protocols this is because AODV periodically sends RREQ, RREP packets. The routing overload for DSDV is less in all situations with any number of nodes. The routing overhead for DSR is somewhat closer to DSDV and which are far from AODV.

CONCLUSION

This simulation based study was conducted to evaluate the performance of the MANET protocols DSDV, DSR, and

AODV based on CBR traffic. The successful test on the comparison shows that our performance evaluation mechanism developed by this project is effective for scalable performance test in NS-2. These routing protocols were compared in terms of Packet delivery ratio, Average end-to-end delay, Throughput and Routing load. Our simulations have shown that performance of a routing protocol varies widely across different performance differentials. It is observed both AODV and DSR perform better in simulations than DSDV. So we can conclude that DSR and AODV outperforms DSDV for the CBR based traffic in Ad-hoc networking environments, so DSR and AODV could be used as a base protocol when we talk of developing a new protocol for Ad-hoc networks and the future research must be focused on improving and implementing the new protocol in Ad-hoc networks.

REFERENCES

[1] D.P. Aggarwal and Qing-An Zeng. "Introduction to wireless and Mobile Systems". Brooks/Cole, 2005.

[2] C.E. Perkins & P. Bhagwat, "Highly Dynamic Destination Sequence-Vector Routing (DSDV) for Mobile Computers", Computer Communication Review, vol. 24, no.4, 1994, pp. 234-244.

[3] Cheng C, Riley R, Kumar SPR, Garcia-Luna-Aceves JJ (1989) A Loop-Free Extended Bellman-Ford Routing Protocol Without Bouncing Effect. ACM SIGCOMM Computer Communications Review, Volume 19, Issue 4:224–236.

[4] Johnson, D.B. and D.A. Maltz, "Dynamic source routing in ad hoc wireless networks", Mobile Computing, 1996, pp: 153-181.

[5] C.E. Perkins and E.M. Royer, "Ad-Hoc on-Demand Distance Vector Routing," Proc. Workshop Mobile Computing Systems and Applications (WMCSA '99), Feb. 1999 pp. 90-100.

[6] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/

[7] K. Fall and K. Varadhan (Eds.), ns notes and documentation, 1999. http://www.mash.cs.berkeley.edu/ns/

[8] NS by Example, http://nile.wpi.edu/NS/

[9] Awk - A Tutorial and Introduction - by Bruce Barnett.

Author's profile:

Mr. G. Jose Moses is a Research Scholar in the Department of Computer Science, Adikavi Nannaya University, Rajahmundry, A.P., India. He obtained his B.Tech from JNTU, Kakinada, M.Tech from Acharya Nagarjuna University. His research interests lies in Computer Networks, Cloud Computing.

Mr.D.Sunil Kumar is student in Department of MCA. Government College, Rajahmundry, A.P., India. His area of interest include wireless communications and networking.

Dr. P. Suresh Varma received the Master's degree M.Tech in Computer Science & Technology from Andhra University. He received Ph.D. degree in Computer Science & Engineering from Acharya Nagarjuna University. He is currently working as Professor in Department of Computer Science in Adikavi Nannaya University, Rajahmundry, A.P., India. He published several papers in National and International Journals. He is active member of various professional bodies. His current research is focused on Computer Networks, Cloud Computing and Data Mining.

Mrs. N.Supriya Working as Academic Consultant, in the department of Computer Science, Adikavi Nannaya university, Rajahmundry. She is at present pursuing Ph.D at Acharya Nagarjuna University. Her research interests lies in Software Engineering, Data Mining and Computer Networks.