Performance Evaluation of Routing Protocols in Ad Hoc Network

Satish Goswami, Neha Sharma, Amit Kumar, Kamlesh Puri, Rahul Desai, Nilima Walde

***Abstract*— NS2 is discrete network simulator which is used for simulating various types of networks such as wired, wireless and ad hoc network. Different traffic patterns are generated and different mobility models are used to evaluate various performance metrics such as packet delivery ratio and delay etc. This paper describes the use of NS2 simulation tools starting from installation, compilation, various necessary tools such as XGraph, BonnMotion, cbrgen and setdest tools used along with NS2 to support and evaluate the network.**

***Index Terms*— AODV, DSR, DSDV, AOMDV, Cbrgen, BonnMotion, XGraph etc**

I. INTRODUCTION

System modeling refers to an act of representing an actual system in a simply way. System modeling is extremely important in system design and development, since it gives an idea of how the system would perform if actually implemented. With modeling, the parameters of the system can be changed, tested, and analyzed. More importantly, modeling, if properly handled, can save costs in system development. To model a system, some simplifying assumptions are often required. It is important to note that too many assumptions would simplify the modeling but may lead to an inaccurate representation of the system. Traditionally, there are two modeling approaches: analytical approach and simulation approach.

Simulation is widely-used in system modeling for applications ranging from engineering research, business analysis, manufacturing planning, and biological science experimentation, just to name a few. Compared to analytical modeling, simulation usually requires less abstraction in the model (i.e., fewer simplifying assumptions) since almost every possible detail of the specifications of the system can be put into the simulation model to best describe the actual system. When the system is rather large and complex, a straightforward mathematical formulation may not be feasible. In this case, the simulation approach is usually preferred to the analytical approach. According to Shannon, simulation is "the process of designing a model of a real system and conducting experiments with this model for the purpose of understanding the behavior of the system and/or evaluating various strategies for the operation of the system."

The structural components of a simulation consist of the following:

A. Entities: Entities are objects which interact with one another in a simulation program to cause some changes to the state of the system.

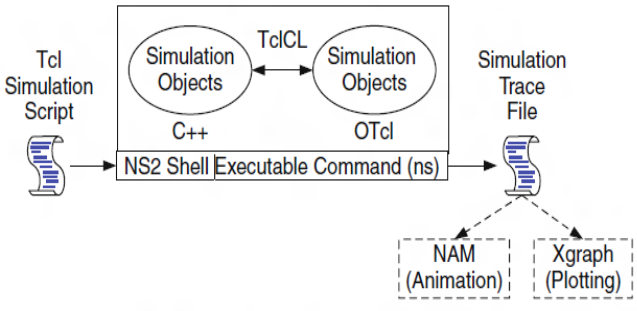
B. Resources: Resources are a part of complex systems. In general, a limited supply of resources has to be shared among a certain set of entities.

C. Activities and Events: From time to time, entities engage in some activities. This engaging creates events and triggers changes in the system states. Common examples of activities include delay and queuing.

D. Scheduler: A scheduler maintains the list of events and their execution time. During a simulation, it runs a simulation clock creates events, and executes them.

E.Global variable: In simulation, a global variable is accessible by any function or entity in the system, and basically keeps track of some common values of the simulation.

Network Simulator (Version 2) is simply an event driven simulation tool that has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2. In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors. Due to its flexibility and modular nature, NS2 has gained constant popularity in the networking research community since its birth in 1989. University of California and Cornell University who developed the REAL network simulator which is the foundation of NS. Since 1995, the Defense Advanced Research Projects Agency (DARPA) starts development of NS through the virtual Internetwork Test bed (VINT) project.



Manuscript received April 13, 2014.

Satish Goswami, Neha Sharma, Amit Kumar, Kamlesh Puri, Student in Department of Information Technology, Army Institute of Technology, Pune, India

Rahul Desai, Asst Professor, Department of Information Technology, Army Institute of Technology, Pune, India

Nilima Walde, Asst Professor, Department of Information Technology, Army Institute of Technology, Pune, India

Fig 1: Network Simulator Architecture

NS2 consist of 200K lines of C++ code, 80K lines of OTcl (Object Oriented Tool Command language) and 50K+ lines of test suite, examples and docs etc. It works on all Windows Platform using Cygwin and Unix/Linux platforms. NS2 provides users with executable command ns which take on input argument, the name of a Tcl simulation scripting file. Users are feeding the name of a Tcl simulation script as an input argument of an NS2 executable command ns. In most cases, a simulation trace file is created, and is used to plot graph and/or to create animation.

NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines backend of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events. The C++ and the OTcl are linked together using TCL. Mapped to a C++ object, variables in the OTcl domains are sometimes referred to as handles. Conceptually, a handle is just a string in the OTcl domain, and does not contain any functionality. Instead, the functionality (e.g., receiving a packet) is defined in the mapped C++ object (e.g., of class Connector). In the OTcl domain, a handle acts as a frontend which interacts with users and other OTcl objects. It may define its own procedures and variables to facilitate the interaction.

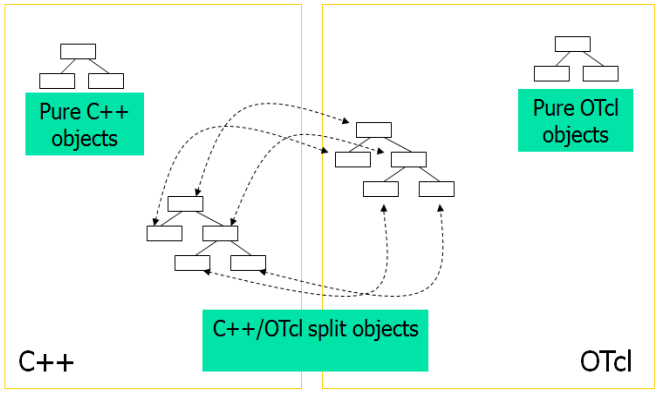


Fig 2: NS2 Language Support

To run a simulation, a user needs to define a network scenario in a Tcl Simulation script, and feeds this script as an input to executable file ns. During the simulation, the packet flow information can be collected through text-based tracing or NAM tracing. After the simulation, an AWK program or a Perl program can be used to analyze a text-based trace file. The NAM program, on the other hand, utilizes a NAM trace file to replay the network simulation using animation. After simulation, NS2 outputs either text-based or animation-based simulation results. To interpret these results graphically and interactively, tools such as NAM (Network Animator) and XGraph are used. To analyze a particular behavior of the network, users can extract a relevant subset of text-based data and transform it to a more conceivable presentation.

Simulation using NS2 consists of three main steps. First, the simulation design is probably the most important step. Here, we need to clearly specify the objectives and assumptions of the simulation. Secondly, configuring and running simulation implements the concept designed in the first step. This step also includes configuring the simulation scenario and running simulation. The final step in a simulation is to collect the simulation result and trace the simulation if necessary.

Written mainly in C++, NS2 employs a make utility to compile the source code, to link the created object files, and create executable file ns. It follows the instruction specified in the default descriptor file Makefile. The make utility provides a simple way to incorporate newly developed modules into NS2. After developing a C++ source code, we simply add an object file name into the dependency, and re-run make.

II. INSTALLATION AND COMPILATION

Installation Procedure on LINUX operating System is as follows:

1) Copy ns-allinone-2.34.tar_1.gz into /usr/local folder.
2) Unzip ns-allinone-2.34.tar_1.gz, you will get ns-allinone-2.34.tar_1.
3) Extract ns-allinone-2.34.tar_1, you will get ns-allinone-2.34 folder.
4) Go to ns-allinone-2.34 folder and say ./install.

It is necessary to configure NS2 before proceeding with sample programs.

1) Open terminal and edit .bashrc file.
2) Add the TCL library, LD library and NS library path in .bashrc file. Save the changes.

To run NS2 on Windows-based operating systems, a bit of tweaking is required. Basically, the idea is to make Windows-based machines emulate the functionality of the Unix-like environment. A popular program that performs this job is Cygwin. After getting Cygwin to work, the same procedure as that of Unix-based installation can be followed.

Go to /usr/local/ns-allinone-2.34/ns-2.34/ directory and do

```
#  ./configure
#  make clean
#  make
#  make install
```

After the installation and/or recompilation, an executable file ns is created in the NS2 home directory. NS2 can be invoked by executing the following statement from the shell environment:

 # ns [<file>] [<args>]

Where <file> and <args> are optional input argument. If no argument is given, the command will bring up an NS2 environment, where NS2 waits to interpret commands from the standard input (i.e., keyboard) line-by-line. If the first input argument <file> is given, NS2 will interpreted the input scripting <file> (i.e., a so-called Tcl simulation script) according to the Tcl syntax. Finally, the input arguments <args>, each separated by a white space, are fed to the Tcl file <file>. From within the file <file>, the input argument is stored in the built-in variable argv.

III. NS2 FOR WIREESS NETWORK

There are two approaches for wireless communication between two hosts. The first is the centralized cellular network in which each mobile is connected to one or more fixed base stations, so that a communication between two mobile stations require to involve one or more based stations. A second decentralized approach consists of an ad-hoc network between users that wish to communicate between each other. Due to more limited range of mobile nodes, nodes not only act as a senders and receivers but also forward the packets between other mobile nodes. Cellular stations have a much larger range than ad-hoc networks. However, ad-hoc networks have the advantage of being quickly deployable as they do not require an existing infrastructure. In cellular networks, the wireless part is restricted only to the access to a network, and within the network classical routing protocols can be used. Ad-hoc network in contrast rely on special routing protocols that have to be adapted to frequent topology changes. The current routing protocols implemented by ns are DSDV, DSR, AODV and OLSR routing protocols.

DSDV [1,2] is the destination sequenced distance vector routing protocols where routing messages are exchanged between neighboring mobile nodes (i.e. mobile nodes that are within range of one another). Routing updates may be triggered or routine. Updates are triggered in case routing information from one of the neighbors forces a change in the routing table. A packet for which the route to its destination is not known is cached while routing queries are sent out.

Optimized List State Routing (OLSR) [3,4] is an optimized version of traditional link state protocol such as OSPF. It uses the concept of Multipoint relays (MPRs) to efficiently disseminate link state updates across the network. Only the nodes selected as MPRs by some node are allowed to generate link state updates. Moreover, link updates contain only the links between MPR nodes and their MPR- selectors in order to keep the update size small. Thus, only partial topology information is made available at each node.

The Dynamic Source Routing (DSR) [5-7] protocol is characterized by the use of source routing. That is, the sender knows the complete hop-by-hop route to the destination. These routes are stored in a route cache. The data packets carry the source route in the packet header. When a node in the ad hoc network attempts to send a data packet to a destination for which it does not already know the route, it uses route discovery process to dynamically determine such a route.

Ad Hoc on Demand Distance Vector Routing (AODV) [8-9] is pure on-demand routing protocol. AODV uses traditional routing tables, one entry per destination. This is in contrast to DSR, which can maintain multiple route cache entries for each destination. Without source routing, AODV relies on routing table entries to propagate a RREP back to the source and, subsequently, to route data packets to the destinations. AODV uses destination sequence numbers as in DSDV to prevent routing loops and to determine freshness of routing information.

IV. TRAFFIC PATTERN GENERATION

In order to evaluate above routing protocols, it is necessary to generate traffic (TCP or UDP) among nodes. Random traffic connections of TCP and CBR can be setup between mobile nodes using a traffic-scenario generator script. This traffic generator script is available under ~ns/indep-utils/cmu-scen-gen and 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 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 looks like the following:

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

The start times for the TCP/CBR connections are randomly generated with a maximum value set at 180.0s, thus the simulation time is at least 180 seconds. And the number of nodes has no relationship to the maximum number of connections (mc), we can have 10 nodes, also 10 connections as one node could have multiple simultaneous connections to other nodes. The parameter "rate" means how many packets per second, thus, for CBR traffic, the packet interval is the reversal of this parameter. And for TCP traffic, we don't have to specify rate, ftp connections are going to be used. the default packet size is 512 bytes.

AWK is a general-purpose programming language designed for processing of text files. AWK refers to each line in a file as a record. Each record consists of fields, each of which is separated by one or more spaces or tabs. Generally, AWK reads data from a file consisting of fields of records, processes those fields with certain arithmetic or string operations, and outputs the results to a file as a formatted report. To process an input file, AWK follows an instruction specified in an AWK script. An AWK script can be specified at the command prompt or in a file.

AWK can be invoked from a command prompt as following:
awk [-F<ch>] {<pgm>} [<vars>] [<data_file>]

The bracket <> contains a variable which should be replaced with actual values at the invocation. These variables include ch Field separator, pgm An AWK script, pgm_file a file containing an AWK script, vars Variables used in an AWK file and data_file an input text file.

V. MOBILITY PATTERN GENERATION

The scenario for a particular experiment is defined using the tool BonnMotion, Java software which creates and analyses mobility scenarios. It is developed within the Communication Systems group at the Institute of Computer Science IV of the University of Bonn, Germany, where it serves as a tool for the investigation of mobile ad hoc network characteristics. The scenarios can also be exported for the network simulator ns-2 and GlomoSim/QualNet. Several mobility models are supported, namely the Random Waypoint model, the

Gauss-Markov model, the Manhattan Grid model and the Reference Point Group Mobility model.

Since mobility patterns may play a significant role in determining the protocol performance, it is desirable for mobility models to emulate the movement pattern of targeted real life applications in a reasonable way. Otherwise, the observations made and the conclusions drawn from the simulation studies may be misleading. Thus, when evaluating routing protocols, it is necessary to choose the proper underlying mobility model. For example, the nodes in Random Waypoint model behave quite differently as compared to nodes moving in groups. It is not appropriate to evaluate the applications where nodes tend to move together using Random Waypoint model. Therefore, there is a real need for developing a deeper understanding of mobility models and their impact on protocol performance. One intuitive method to create realistic mobility patterns would be to construct trace-based mobility models, in which accurate information about the mobility traces of users could be provided. However, since MANETs have not been implemented and deployed on a wide scale, obtaining real mobility traces becomes a major challenge. Therefore, various researchers proposed different kinds of mobility models, attempting to capture various characteristics of mobility and represent mobility in a somewhat 'realistic' fashion. Much of the current research has focused on the so-called synthetic mobility models that are not trace-driven. [10-12]

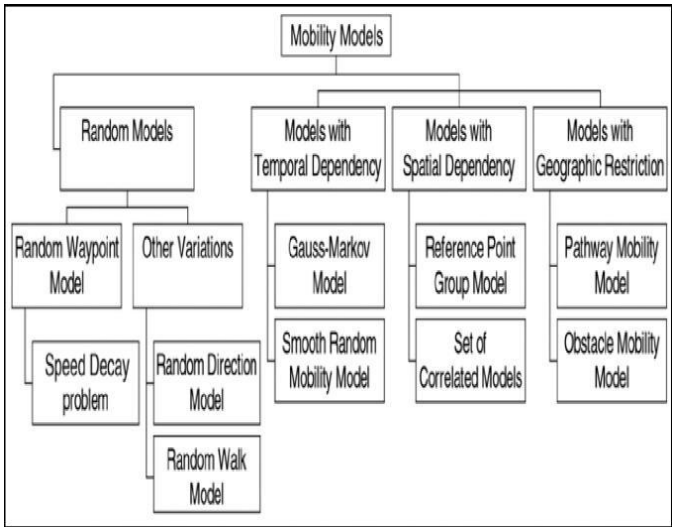


Fig 3: Different Mobility Models

Random Direction Mobility Model (RDMM) is designed to avoid concentration of mobile nodes (MNs) at centre of the simulation area, as seen in the Random Waypoint model. In this model, MNs choose a random direction in which to travel similar to the Random Walk Mobility Model. An MN then travels to the border of the simulation area in that direction. Once the simulation boundary is reached, the MN pauses for a specified time, chooses another angular direction (between 0 to 180 degrees) and continues the process.

In Random Walk Mobility Model (RWkMM), a mobile node moves from its current location to a new location by randomly choosing a direction and speed in which to travel [14]. The new speed and direction are both chosen from predefined ranges, respectively [min-speed, max-speed] and [0, 2*pi] respectively based on uniform distribution. Each movement in the RWkMM occurs in either a constant time interval 't' or a constant travelled distance 'd', at the end of which a new direction and speed are calculated.

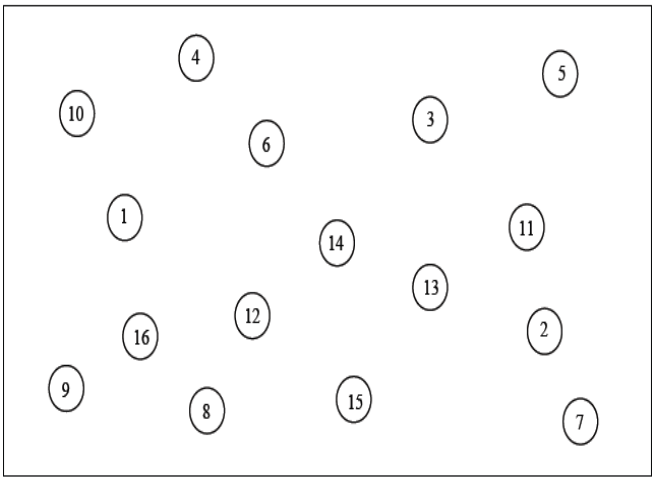


Fig 4: Topography showing Random Waypoint Mobility Models

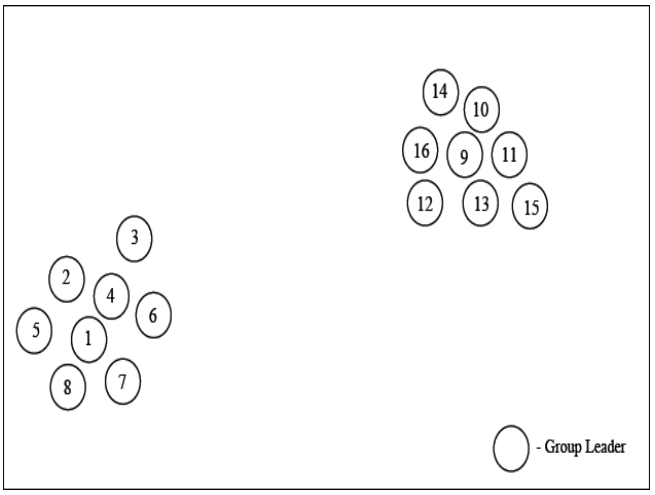


Fig 5: Topography showing Random Group Mobility Model

Gauss-Markov Mobility Model (GMMM) was designed to adapt to different levels of randomness via tuning parameters. Initially each mobile node is assigned a current speed and direction. At each fixed intervals of time n a movement occurs by updating the speed and direction of each mobile node. Specifically, the value of speed and direction at the nth instance is calculated based on the basis of the value of speed and direction at the (n-1)st instance and a random variable using the following equations:

VI. ANALYSIS OF AD HOC NETWORK

We have generated the movement scenario files using the *setdest* program which comes with the NS-2 distribution. These scenario files are characterized by pause time. The total duration of each simulation run is 200 seconds. Different movement patterns for five different pause times are created: 0, 50, 100, 150 and 200 seconds. These varying pause times affect the relative speed of the mobile nodes. A pause time of 200 seconds corresponds to the motionless state of the nodes in the simulation environment as the total duration of the

simulation run time is 200 seconds. On the contrary when we choose the pause time of 0 second, it indicates continuous motion of the nodes.

#./setdest –v 2 –n 10 –s 1 –m 1 –M 10 –t 300 –P 1 –p 0 –x 800 –y 800 > motion/scen-10-0

This command will generate motion file having 10 nodes movement in 800 by 800 area, -m is the minimum speed, -M is the maximum speed, thus speed varying from 0 to 10 m/s, -p is the pause time which is changing from 0 to 300 in steps of 50s. –s and –P is the speed type (uniform or normal) and pause time type (uniform and normal) respectively.

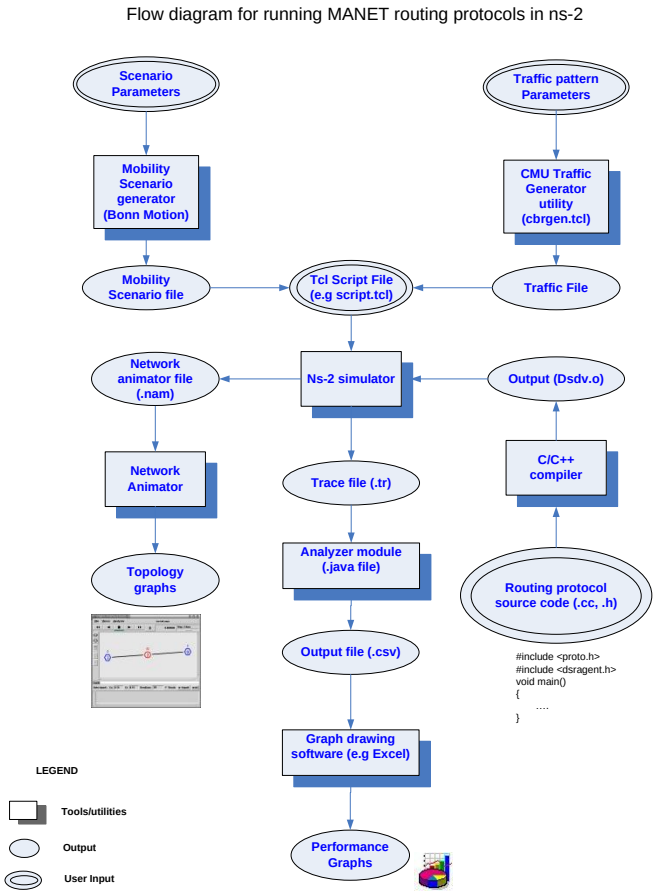


Fig 6: Flow Diagram for running routing protocols in NS2

Thus for simulation network parameters listed below in table 1 are used.

TABLE I
SIMULATION PARAMETERS

Parameter	Value
Number of nodes	10 to 100 nodes
Mobility model	Random Waypoint Mobility Model, Random Group Mobility Model
Simulation time	400 s
Topology Size	800 m × 800 m
Routing protocols analysed	DSDV, DSR, AODV,
Packet size	512 bytes
Mobility rate	10m/s to 50 m/s
Pause time	0, 50, 100, 150 and 200 s

Packet delivery ratio (PDR) and end to end delay are used to evaluate the performance of above mentioned routing protocols.

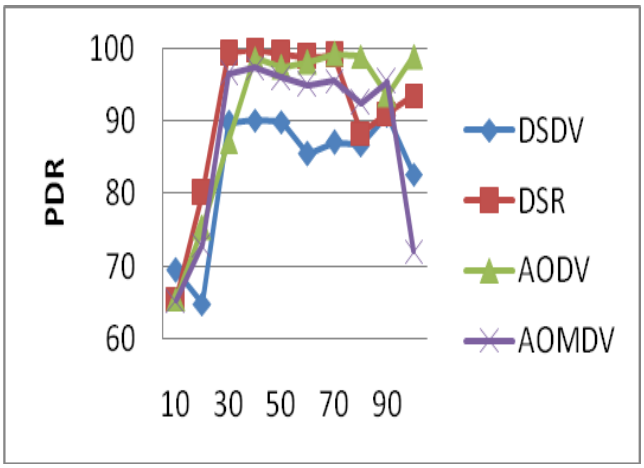


Fig 7: Packet Delivery Ratio vs. Number of Nodes

Here the provided graphical result shows the packet delivery ratio over increasing number of nodes. From the above given results we can say, that DSDV returns poor result as we start increasing the number of nodes. AODV and DSR protocols returns best result and thus achieves packet delivery ratio in range of 95% to 99%. But as we start increasing number of nodes results falls down below 95%.

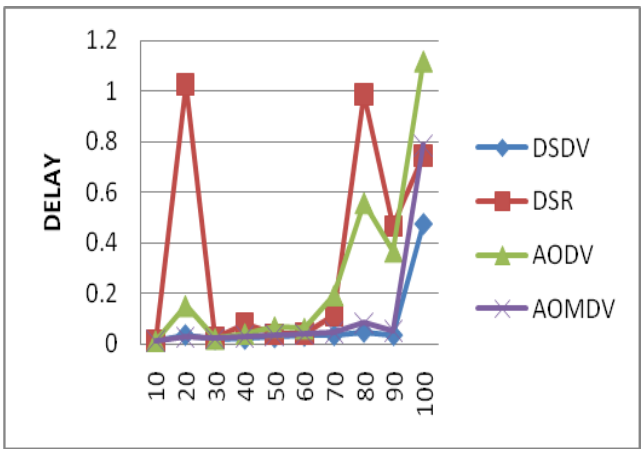


Fig 8: Delay vs. Number of Nodes

It is observed that as we start increasing the number of nodes, delay will starts increasing. DSDV with the minimum delay as it is proactive protocol.

REFERENCES

[1] E. Kulla, M. Hiyama, M. Ikeda, L. Barolli, V. Kolici, R. Miho, "MANET performance for source and destination moving scenarios considering OLSR and AODV protocols", Mobile Information Systems, vol. 6, no. 4, 2010, pp. 325–339.

[2] C. E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance Vector Routing (DSDV) for Mobile Computers" Proc. ACM SIGCOMM'94, London, U.K., Sep. 1994, PP. 234-44.

[3] T. Clauses et al, Optimized Link State Routing Protocol, Http://www.ietf.org.internet-drafts/draft-ietf-manet-olsr-11.txt, July 2003, IETF Internet Draft.

[4] E. Spaho, L. Barolli, G. Mino, F. Xhafa, V. Kolici, R. Miho, "Implementation of CAVENET and its usage for performance evaluation of AODV, OLSR and DYMO protocols in vehicular networks", Mobile Information Systems, vol. 6, no. 3, 2010, pp. 213–237.

[5] C E Perkins, E M Royer, S R Das and M K Marina. Performance Comparison of two On Demand Routing Protocols for ad hoc networks. IEEE Personal Communications, 8(1):16-28, 2001

[6] Y-C Hu and D. Johnson. Caching strategies in On Demand Routing Protocols for wireless Ad Hoc Networks. In Proceedings of IEEEE/ACM MobiCom, pages 231-242, 2000

[7] Y-C Hu and D. Johnson. Implicit Source Routes for on Demand Ad Hoc Network Routing. In Proceedings of ACM MOBIHOC, pages 1-12, 2001

[8] Z. Haas, J.Halpern, and L. Li. Gosssip-based Ad Hoc Routing. In Proceedings of IEEE infoCom, pages 1707-1716, 2002

[9] E. Kulla, M. Hiyama, M. Ikeda, L. Barolli, V. Kolici, R. Miho, "MANET performance for source and destination moving scenarios considering OLSR and AODV protocols", *Mobile Information Systems*, vol. 6, no. 4, **2010**, pp. 325–339.

[10] Megat Zuhairi, Haseeb Zafar, David Harle, The impact of mobility models on the performance of mobile Ad Hoc network routing protocol. IETE Journal of Research, Vol.29, Issue 5, pages 414-420,2012

[11] A. Boukerche, S.K. Das and A. Fabbri, "Analysis of randomized congestion control with DSDV routing in ad hoc wireless networks", Journal of Parallel and Distributed Computing (JPDC) 61 (2001) 967–995.

[12] Fan Bai, Ahmed Helmy "A Framework to systematically analyze the Impact of Mobility on Performance of Routing Protocols for Adhoc Networks", IEEE INFOCOM 2004