PERFORMANCE EVALUATION OF NS2 AND OMNET++ SIMULATORS FOR AODV PROTOCOL IN MANET

Jekishan K. Parmar[1], Mrudang Mehta[2]

[1] M.Tech Student, [2]Associate Professor,Department of Computer Engineering, Dharmsinh Desai University Nadiad, Gujarat, India

Abstract

In order to observe the behaviour of a protocol in various scenarios, network simulators are used. There are few network simulators available for usage but which one will provide optimum performance and suitability of network simulator for a particular scenario is always an important decision. In this paper we analyse behaviour of two different network simulators for a same MANET routing protocol. The MANET routing protocol that we have selected is popular Ad-hoc on demand Distance Vector (AODV). In this paper, we analyse the performance of routing protocol by using NS2 simulator and then the same on INET, which is a simulation framework from OMNeT++. The main purpose behind this work is to understand what role simulator plays when we try to simulate a particular protocol on desired Simulator. This study of difference in performance of simulator shows how the underlying architecture of a simulator affects the performance of the simulator. We considered these simulators for comparison purpose due to the amount of their usage in the industry as well as in education and research area. NS2 is one of the oldest and most preferred simulators by researchers and industry people while usage of OMNeT++ is also increasing with the time.

Keywords: *MANET, AODV, NS2, OMNeT++, Performance Evaluation.*

----------***----------

1.INTRODUCTION

Often, the simulation of a new network protocol is preferred over its evaluation in testbed experiments. The reasons are manifold, e.g., the increased speed of getting evaluation results, the reduced hardware demands and thus the reduced cost, or the flexibility in the scenario definition. As a result, many network simulators have been developed over the last decades. Today, ns-2 is most widely used network simulation tool. Kurkowski et al. [1] found that 44% of the simulations in their MobiHoc survey used ns-2 as network simulator. Its development began in 1989 as collaboration between a number of different researchers and institutions. Meanwhile, a vast number of models for all kinds of network protocols have been written for ns-2. At the time of writing this paper, a popular ns-2 web site [2] lists 59 models for media access, routing, and transport protocols, as well as various topology and traffic generators. OMNeT++ is another simulation tool that is free for academic use [3]. OMNeT++ features a simple, object oriented design, which leads to good scalability. It is found that OMNeT++ is particularly well suited for performance evaluations of large networks. Still, outside a small community of OMNeT++ enthusiasts few people seem to know this tool.

If a protocol implementation for NS2 is available in public domain, it is difficult to write same protocol in OMNeT++ Simulator, since architecture of both simulators differs. This restricts extension of research work in a different simulator. So, choosing network simulation tool is an important decision. Hence, in this paper we try to observe various issues that arise when a same protocol is implemented in two different simulators. After getting the results from both the simulators we compare them and try to understand what might be the cause of this difference in performance.

2.RELATED WORK

There are several works in literature, which describes differences in the performance of simulators, but it is really very difficult to analyse a simulator from all perspectives. Hence, the approach of every work done so far is based on the perspective, from which the author has analysed the differences in the performance of simulator. In [4], authors have described and analysed wireless network simulators like Qualnet, NS-2, J-Sim, OMNeT++ and Opnet. The analysis done of these different simulators are on the basis of their features like language supported, platform supported, GUI support, licensed or not, animation support is there or not. After comparison of various simulators on the basis of their features given above, authors have suggested that NS-2 and OMNeT++ should be the best choice when open source network simulators are considered for research work. The authors also have suggested that Qualnet satisfies most features when all commercial network simulation tools are considered. In [5], importance of selecting a proper simulator for carrying out research in MANET is described by simulating a MANET routing protocol in open-source network simulators like NS-2, NS-3, OMNeT++ and GloMoSiM. The authors in this work, analyses these simulators by considering the performance metrics like computational time taken, CPU utilization, memory usage and scalability. Hence, in this work, the evaluation is done from the perspective of hardware requirements of various simulators. In [6] also various simulators like NS-2, NS-3, OMNeT++ are analysed along with couple of recently

developed simulators like SimPy and JiST/SWANS. Here, the authors have analysed these simulators on the basis of the factors like, impact of simulation runtime on the network size and probability of dropping packets. They have also considered the memory usage as a metric in order to analyse the memory requirements of various simulators. Large variations in runtime performance as well as in memory usage were found when the simulation results were analysed.

After analysing some of the existing works where the performance comparison of various wireless network simulators has been done, we found that it would be good if two simulators which are more predominantly used, are analysed and compared, especially for their performance in MANETs. Hence, here we chose to simulate and analyse the performance of AODV protocol which is a MANET routing protocol into NS-2 and OMNeT++. The simulation results and analysis that we have discussed in this study will be helpful for choosing simulator whenever a research work is to be carried out for MANET routing.

3.AODV PROTOCOL

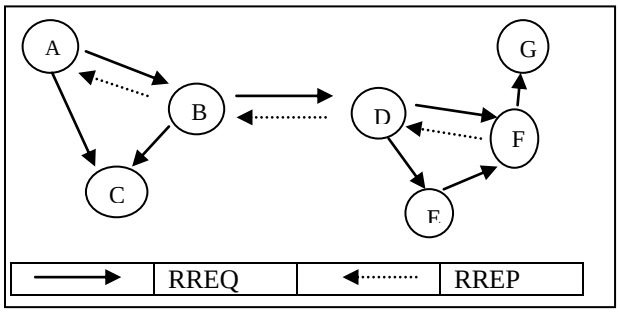

Fig-1:.AODV Protocol Working

AODV stands for Ad-hoc On demand Distance Vector (AODV). In AODV there are four different types of messages used for route establishment and route maintenance. Route Requests (RREQs) and Route Replies (RREPs) are the two message types used in AODV for establishment of route. When a route to a new destination is needed, the node uses a *broadcast* RREQ to find a route to the destination as shown in figure 1. Here node A is a source node, wanting to transfer the data to node F which is a destination node. A route can be determined when the request reaches either the destination itself or an intermediate node with a fresh route to the destination. Thus in order to determine the route to destination RREQ is sent out to every intermediate node also. This can be seen in figure 1 where RREQ sent by node A, which is the source node. Every node receives this RREQ sent by source node A. The route is made available by unicasting a RREP back to the source of the RREQ. Since each node receiving the request keeps track of a route back to the source of the request, the RREP Reply can be unicast back from the destination to the source, or from any intermediate node that is able to satisfy the request back to the source. This can be well understood from figure 1 above, when the destination F receives the RREQ it replies with RREP and all the intermediate nodes receiving this RREP, forwards it to the source which had sent the RREQ. In figure 1, replies by unicasting an RREP to D, which unicasts it to B, which in turn sends it to A which is the source node and hence the route is established. It is only possible as all intermediate are keeping track of incoming RREQ.

There are two other types of messages in AODV for route maintenance. They are: HELLO and RERR. These two messages are for maintenance of the network. A HELLO message is a local advertisement for the continued presence of the node. Neighbours that are using routes through the broadcasting node will continue to mark the routes as valid. If hello messages from a particular node stop coming, the neighbour can assume that the node has moved away. When that happens, the neighbour will mark the link to the node as broken and may trigger a notification to some of its neighbours telling that the link is broken. On receipt of this notification, neighbour node broadcasts a RERR message. RERR message is also broadcast when destination is not reachable as well as when there are no more active routes for the nodes to which the packets are destined. After the receipt of RERR message by each and every node, the routing table is updated with a broken link on a route i.e. that route is deleted from the table.

In AODV, each node maintains route table entries with the destination IP address, destination sequence number, hop count, next hop ID and lifetime. This information must be kept even for ephemeral routes, such as those created to temporarily keep track of reverse paths towards nodes originating the RREQs.

4.NETWORK SIMULATOR 2

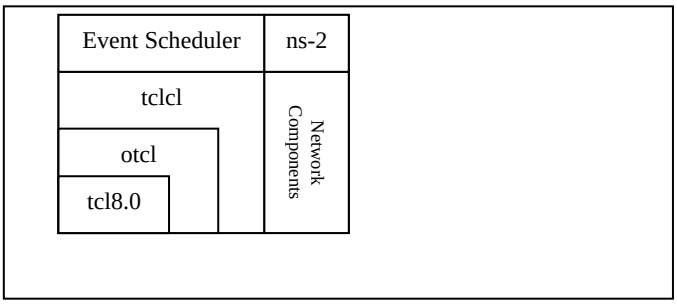

Fig-2: Architecture of NS-2

The above figure 2 shows the architectural view of NS2 simulator. As shown in figure 2, NS2 is composed of TCL, OTCL, TCLCL, Event Scheduler and Network Component [7]. TCL stands for Tool Command Language which is used for creating various simulation scenarios in NS2. OTCL is Object TCL programming language. In NS2, programs are written in OTCL as it provides Object-oriented support. In order to link the simulation scenario script written in TCL and programs written in C++ there is TCLCL which stands for TCL Cross Linking. Above all of this there is NS2 simulator which co-ordinates with models of various network components and event scheduler implemented in C++. In order to create a simulation, OTCL is used to link this C++ files to the simulation script written in TCL and simulation program which is generated with OTCL.

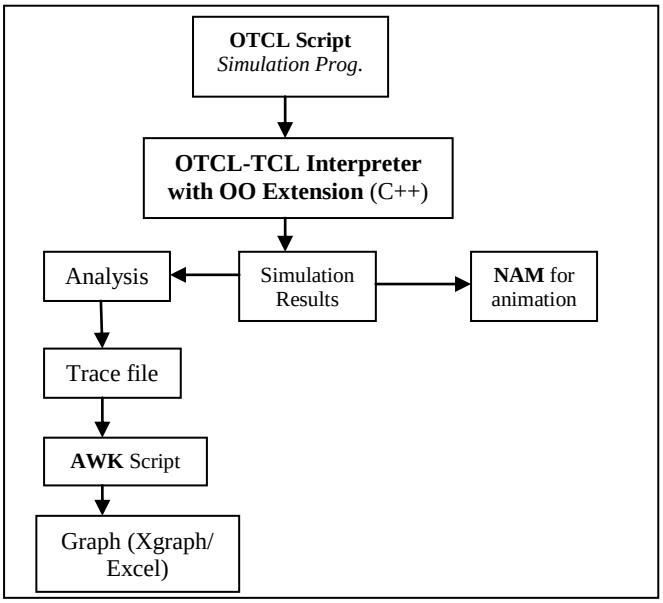

Fig-3:NS2 Simulation Execution

Figure 3 shows the procedure of executing the simulation in NS2. First of all we create our simulation script which contains our simulation scenario and then parameters which we want to apply. This simulation script is nothing but the TCL file in which we have mentioned our simulation parameters like protocol to be used, energy model to be used, physical layer details, etc. These parameters are modelled in NS2 using object-oriented extension of C++ that is linked to current simulation script using OTCL linkage.

The simulation script also mentions the protocols that we are going to use. In our case we will be giving AODV as a parameter to routing protocol in the simulation script. The protocol i.e. AODV in our case will be present as C++ file in the NS-2 directory. This C++ source file of AODV will be linked with our simulation script by OTCL linkage.

After the completion of simulation, a trace file is generated and then to fetch the output from the trace file , an AWK script may be coded that extracts the required output from that trace file. Also from this trace file, direct generation of charts and graphs is possible by XGraph, which is also tool supported by NS2.

NS2 also provides support for visualization of the network with the help of NAM, which is a network animator tool. NAM uses the trace file generated by the simulation carried out in NS2 and generates an animation based on it.

5.OBJECTIVE MODULAR NETWORK TESTBED IN C++ (OMNET++)

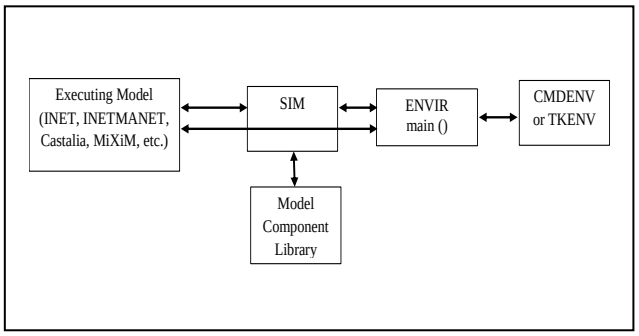

Fig-4:OMNeT++ Simulation Architecture

OMNeT++ simulation programs possess a modular structure. The logical architecture is shown on Figure 4. The Model Component Library consists of the code of compiled simple and compound modules. Modules are instantiated and the concrete simulation model is built by the simulation kernel and class library (*Sim*) at the beginning of the simulation execution.

The simulation executes in an environment provided by the user interface libraries (*Envir*, *Cmdenv* and *Tkenv*) – this environment defines where input data comes from, where simulation results go to, what happens to debugging output arriving from the simulation model, controls the simulation execution, determines how the simulation model is visualized and (possibly) animated, etc.

In our case of AODV, we'll be having AODV implementation residing in model component library and our NED file will create a module/sub module where we actually create the network where we'll be running the simulation and INI file will be used to configure this network i.e. to set parameters, simulation times, etc. The detailed simulation execution process is shown in figure 5.

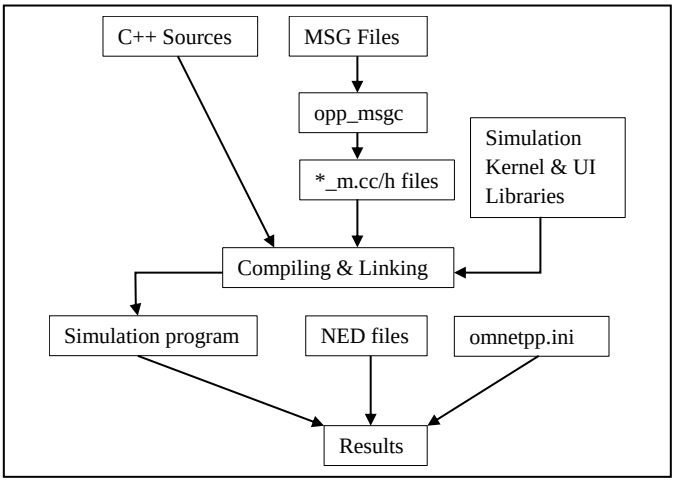

Fig-5:OMNeT++ Simulation Execution

In the above figure 5, a complete process that is followed whenever any simulation is carried out in OMNeT++, is shown. There are MSG files available in OMNeT++ which

contains messages required for any special processing required for any source files as well as messages about inclusion of any other header files or source files if required. Moreover, due to compilation and linking process done in every simulation, we do not have any manual compilation process even if we have done modification in existing code

After compiling and linking process is over, the simulation scenario is generated as per NED file. NED stands for Network Description. The parameters are assigned as described in INI file. After this, simulation runs and outputs the results in scalar and vector format

.

6.PERFORMANCE EVALUATION

Here we evaluate the performance of AODV protocol in both OMNeT++ and NS2 by keeping identical simulation scenario

.

6.1 Simulation Scenario

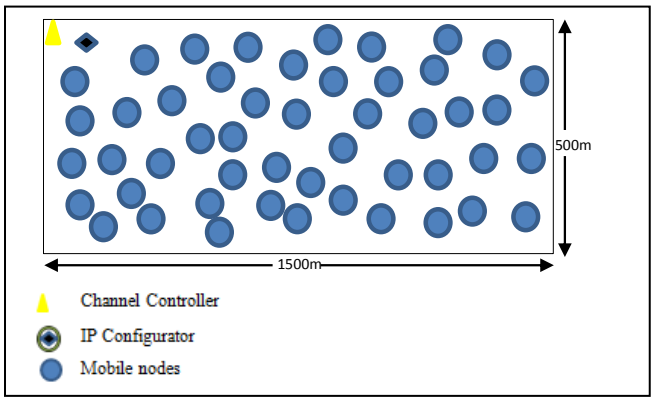

Fig-6: Visualization of simulation scenario

In figure 6, we see two objects i.e. channel controller and IP Configurator, their work is to manage the connections with hosts in the network. These two objects resides on every mobile node that exists in the network. We have displayed them as one single entity just for the sake of clarity of diagram. Basically the aim behind displaying these two entities in our simulation is to show the components which handle termination and establishment of connections when the nodes move out of the range. This task of channel allotment and channel access is done by channel controller. In OMNeT++, there is a Channel Control module described in its mobility framework [8] in INET, which takes care of this while in NS2 there is also a link layer module which does this task but on the base of node positions as described in [2]. The task for allotting and managing IP addresses in OMNeT++ is done by an IPv4 Configurator module which rests in mobility framework but in NS2 there is *netIF*, network interface which manages all network related tasks. As given in [2], the allocation and management of IP addresses in NS2 not handled by any specific module.

The simulation runs using movement patterns generated for 7 different pause times: 0, 20, 40, 80,120, 160, 200 seconds and constant speeds of 20s. A **pause time** of 0 seconds corresponds to continuous motion, and a pause time of 200 (the length of the simulation) corresponds to no motion. Constant Bit Rate (CBR) traffic generators used as sources to run the simulation. The simulation parameters are summarized in the table below.

Table 1: Simulation Settings

Name of Parameter	Value
Dimensions	1500 m X 500 m
Number of Nodes	50
Pause time	0, 20, 40, 80,120, 160, 200 seconds
Mobility Speed	20 m/s
Simulation time	200 s
Type of Traffic	CBR

6.2 Performance Metrics

While considering the evaluation of AODV protocol on NS2 and OMNeT++ we chose the following metrics.

Table 2: Performance Metrics

Name	Definition
Packet Delivery Ratio	According to [9], packet delivery ratio is calculated by dividing the number of packets received by the destination through the number of packets originated by the application layer of the source. It specifies the packet loss rate, which limits the maximum throughput of the network. The better the delivery ratio, the more complete and correct is the routing protocol.
Throughput	The throughput (messages/second) is the total number of delivered data packets divided by the total duration of simulation time [10]. In this case, the throughput of each of the routing protocol in terms of number of messages delivered per one second is evaluated.
Average End-to-End Delay	Average End-to-End delay (seconds) is the average time it takes a data packet to reach the destination. This metric is calculated by subtracting "time at which first packet was transmitted by source" from "time at which first data packet arrived to destination". This includes all possible delays caused by buffering during route discovery latency, queuing at the interface queue, retransmission delays at the MAC, propagation and transfer times. This metric is significant in understanding the delay introduced by path discovery.

6.3 Simulation Results

We simulate AODV protocol using NS2 with the parameters described in Table 1 for our simulation. After configuring we extract the results from it using AWK script. Following are the results of our simulation on NS2.

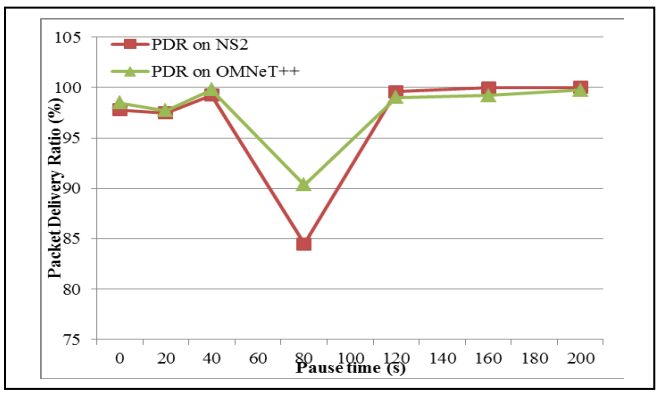

Fig-7: Packet delivery ratio for AODV

In figure 7, packet-delivery ratio for AODV on both the simulators is plotted. It can be noticed that packet delivery ratio remains nearly similar at all points for both NS2 and OMNeT++ but still when we consider the value of packet-delivery ratio through various pause time values, it is seen that the lowest value attained by OMNeT++ for packet-delivery ratio is also much more then NS-2 and so we can say that even in worst case, OMNeT++ can still outperform NS2.

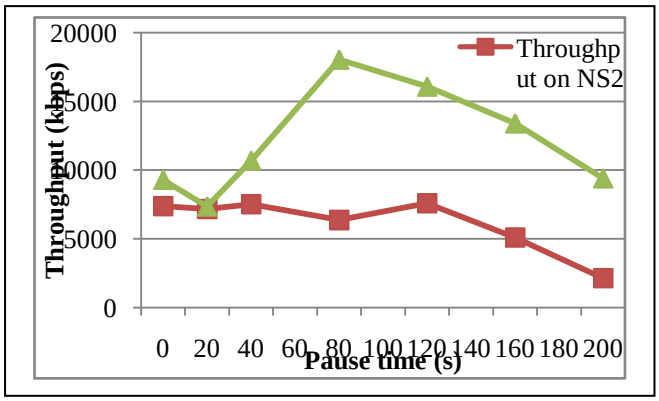

Fig- 8: Throughput for AODV

In figure 8, results for throughput on both simulators are plotted and it can be noticed here that OMNeT++ throughput results are far better than NS2 when we consider the highest throughput received throughout the simulation. NS2 gives constant throughput but it is lower than OMNeT++. Hence, we can say that NS2 is very much stable when throughput is considered against varying pause times while OMNeT++ gives much higher throughput compared to NS2 but it is affected more due to variation in pause times.

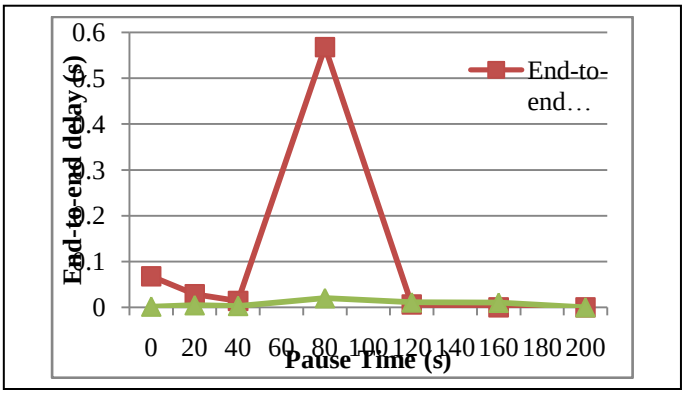

Fig- 9: End-to-end delay for AODV

In figure 9, end-to-end delay for AODV on both the simulators is graphed. We can clearly see that OMNeT++ gives constant end-to-end delay throughout all the pause times and also are lower than NS2 at most at all the variations of pause times.

Considering all the results we see that OMNeT++ provides better outputs for all the performance metrics that we have considered. In every performance metric we find that OMNeT++ provides highest output for e.g. we can note in throughput, the highest value achieved by OMNeT++ is around 18000 kbps while with NS2 we only have the highest value around 7500 kbps. Similarly in figure 3, the lowest packet delivery ratio of OMNeT++ is around 90% which is much greater than 84% of NS2. Same can be noticed with end-to-end delay also.

6.4 Observations

While carrying out the simulations and analysing the results, we also viewed the internal structure of OMNeT++ and NS2. In our study of the internal structure, we carefully analysed and understood the functioning of their architecture. We also found out some of the parameters which were processed in a little different manner in both of these simulators. The differences that we found out in these simulators during our study are listed below:

a) Some parameters are not available in the configuration file but only in the source code. For example, in ns-2, the maximal contention window `CWMax` is set in the configuration file, with a default value of 1023. In OMNeT++ this parameter is defined as a constant of 255 in one of the source files. This size of contention window does have effect on traffic parameters like throughput in the network [11].

b) Sometimes there is no corresponding parameter in the other simulator at all. For example, in ns-2, `longRetryLimit=4` is configured as the retry limit for data packets and `shortRetryLimit=7` is configured as the retry limit for a control packet. In OMNeT, there is only one `RetryLimit=7` for all the packets. Hence, in OMNeT++ we don't get any option of setting the different retry limits for control packets and data packets.

c) These simulators use a different modeling approach for the parameters. For example, ns-2 and OMNeT++ have the following different modeling approach:

Propagation delay — In ns-2, the delay is a constant defined in configuration file. In OMNeT++, the delay is a function of the nodes' distance. Due to this, the same parameter will produce different results even if the simulation scenario is exactly the same in both the simulators.

d) In NS2, memory consumed when we the number of nodes are increased is much more than in OMNeT++. This happens in NS2 due to OTcl as it does not carry out the process of garbage collection [12].

e) The overall documentation of NS2 also seems a bit fragmented as we don't see any central access site where there are all simulators or frameworks that are based on NS2 are located along with their documentation while when we consider OMNeT++ we do have the documentation of it very much intact and up-to-date.

Because of these implementation differences, it is impossible to make identical simulations without modifying the source code of either module. The latter would be time consuming and error-prone; and it would further limit the comparability of the simulation results unless all relevant publications would use the same modifications.

7.CONCLUSIONS

To this end, we have compared simulations that involved the MANET Routing module in NS2 and OMNeT++ with INET framework. Analysing the simulators including their source code, we have found that differences in the implementation of the simulators and frameworks do not allow reproducing the simulation scenarios from another simulator. Furthermore, we have shown that even for scenarios where two simulators allow the choice of identical parameters, the different simulators lead to vastly different results. From our observations we can say that OMNeT++ is good to work with parameters in which their value is the point of interest and not the form (linear, exponential, etc.) that those values follow when they are plotted. We can also notice from the results that NS-2 is good when consistency is taken into consideration while OMNeT++ we have considerable amount of variation in results.

When the source code of AODV protocol was analysed in particular, we found that both the simulators had same version of AODV protocol implemented in it. Hence, we can say that the difference in the performance of these simulators is not due to different source codes. It is just due to the manner in which these simulators carry out the simulation.

As a consequence, we conclude that protocol evaluations from different simulators are not comparable even when the authors use the very same simulation scenario and source code. Based on the experience of study, it can be proposed that modules, i. e. the implementation of a particular model or protocol, should be the level of abstraction on which different model and protocol implementations should be compared. Also in this study the analysis and evaluation are done by concentrating on the mobility of network as we have considered pause time in our simulation, hence, in future the same study can also be done by varying network simulation time, bandwidth or by varying mobility pattern.

REFERENCES

[1]S. Kurkowski, T. Camp, and M. Colagrosso. MANET Simulation Studies: The Incredibles. *Mobile Computing and Communications Review, 9(4):50–61, 2005*.

[2]S. McCanne and S. Floyd. *ns Network Simulator. <http://www.isi.edu/nsnam/ns>*.

[3]Varga. The OMNeT++ discrete event simulation system. *In Proceedings of the European Simulation Multiconference (ESM'2001). June 6-9, 2001. Prague, Czech Republic, 2001.*

[4]V. Mishra, S. Jangale, Analysis and Comparison of different wireless network simulators. *VESIT, International Technological Conference-2014 (I-TechCON), Jan. 03 – 04, 2014.*

[5]A. R. Khan, S. M. Bilal, M. Othman. A Performance Comparison of Network Simulators for Wireless Networks. *ICCSCE 2012*.

[6]E. Weingartner, H. vom Lehn, K. Wehrle. A performance comparison of recent network simulators. *IEEE International Conference on Communications, 2009.*

[7]M. Karl. A Comparison of the architecture of network simulators NS-2 and TOSSIM, *Seminar, Universität Stuttgart, 2005.*

[8]W. Driytkiewicz, S. Sroka, V. Handziski, A Kopke and H. Karl A Mobility Framework for OMNeT++, *Telecommunication Networks Group, Technische Universität Berlin, 2003*

[9]Jörg, David Oliver. (2003). Performance Comparison of MANET Routing Protocols in Different Network Sizes. Retrieved February 5, 2008, from *<http://www.iam.unibe.ch/~rvs/research/publications/projekt_david_joerg.pdf>*

[10]Al-Maashri, A. and Ould-Khaoua, M. (2006). Performance analysis of MANET routing protocols in the presence of self-similar traffic. *Proceedings of the 31st IEEE Conference on Local Computer Networks, 2006, 14-16 November 2006, pages pp. 801-807, Tampa, Florida, USA. Retrieved February 3, 2008, from <http://eprints.gla.ac.uk/3545/01/almaashri3545.pdf>*

[11]A. Khalaj, N. Yazdani, M. Rahgozar, Effect of the contention window size on performance and fairness of the IEEE 802.11 standard. *Wireless Pers Communications, Springer, 2007.*

[12]NS- Debugging website
< http://www.isi.edu/nsnam/ns/ns-debugging.html>

BIOGRAPHIES

Jekishan K. Parmar is a student of M.Tech in Computer Engineering at Faculty of Technology, Dharmsinh Desai University, Nadiad, Gujarat, India. He received his B.E. in Information Technology from L. D. College Of Engineering, Ahmedabad in 2012. His research interests include Mobile Computing, Ad hoc Networks and Wireless Sensor Network.

Mrudang T. Mehta is an Associate Professor at Department of Computer Engineering, Faculty of Technology, Dharmsinh Desai University, Nadiad, Gujarat, India. He received his M.Tech. in Information Technology from Indian Institute of Technology, Roorkee in 2008 and B.E. in Computer Engineering from DDIT, Nadiad in 2001. His research interests include Wireless Sensor Network, Embedded System and Mobile Computing.