ISSN 2072-0149

The AUST

Journal of Science and Technology

Volume-2 Issue-2 July 2010

Ahsanullah University
of Science and Technology

Editorial Board

Editor

Prof. Dr. Kazi Shariful Alam
Treasurer
Ahsanullah University of Science and Technology

THE AUST
Journal of Science and Technology
Volume-2, Issue-2, July 2010

Development of a network simulator modifying the NS2 network simulator for user-friendly QoS monitoring of TCP/IP based data networks.

Muhammad Jakaria Rahimi[1], Selina Parveen[2]

Abstract: *Network Simulator 2 (NS-2) is one of the most popular tools in academia for evaluation of network, protocols and topologies. It has become virtually the standard of network simulation. It represents a discrete, event-based simulator that has an ability to be easily extendible and modifiable due to its open source nature. However, a major shortcoming of ns-2 is its limited scalability in terms of memory usage and simulation run-time which get worsen due to further post processing of huge, multi-format output files. In this research work, the existing NS2 simulator is modified to include the new modules and corresponding scripts and codes so that it would be capable of taking various inputs and providing appropriate outputs representing the QoS for the wired, wireless and hybrid data networks without requiring any post-processing. Two new modules have been devised with their inherent capability of collecting data, calculating different QoS parameters during simulation, and present them graphically immediately after completion of the simulation without any further post-processing. To facilitate the work of the newly developed agents, the existing packet format is also modified. The newly devised agents added to NS2, not only facilitate the representation of output data in user-friendly format, but also reduce the total simulation time and memory consumptions. Moreover, with a slight post-processing of the output produced by the two QoS monitoring agents, users would be able to get more versatile outputs.*

Key words*: Agents, IP networks, Network Simulator 2, QoS monitors,QoS parameters.*

Introduction

NS or network simulator is a very popular and versatile discrete event network simulator. It is widely called *NS2*, network simulator 2, in reference to its current generation. *NS* began as a variant of the REAL network simulator in 1989 and has evolved substantially over the past few years. In 1995 *NS* development was supported by DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI.

Currently *NS* development is supported through DARPA with SAMAN and through NSF with CONSER, both in collaboration with other researchers including ACIRI. *NS* has always included substantial contributions from other researchers, including wireless code from the UCB, Daedalus and CMU Monarch projects and Sun Microsystems. Due to its open source model and its substantial support for simulation of TCP, routing, and multicast protocols over

[1] *Assistant Professor, Department of EEE, Ahsanullah University of Science and Technology.*
[2] *Lecturer, Department of CSE, Darul Ihsan University.*

wired and wireless (local and satellite) networks it is very popular among the network researchers and *NS2* has established itself as virtually the standard network simulation tool [1],[2].

With the advent of multimedia networking, sincere attempts in guaranteeing quality of service (QoS) through appropriate devices and protocols are essential in designing the next generation advanced networks. Although test-beds and real world experiments can produce much more realistic results than simulations, implementing a large and complex experimental scenario is rather challenging. Therefore, many researchers employ *NS2* simulations to test their ideas in the early stage of protocol design. Typically, each researcher writes his/her own *NS2* scripts, and much time is spent in processing the raw data produced by the simulator, during extraction of the valuable information.

On the other hand, the community still lacks a set of ready-made module to measure QoS parameters and to demonstrate how the QoS parameters are affected by variations of other network parameters. Several researchers have proposed some post-processing techniques [3],[4]. However, they involve a huge amount of time and consume a huge memory. Moreover, the existing techniques are not so versatile to be deployed for wired, wireless and hybrid networks. In this paper, to alleviate the short comings of ns2, we have developed two smart 'quality of service' (QoS) monitors for NS2. We have also introduced a changed packet format to facilitate the data processing capabilities of the simulator. They altogether produce a very small, well formatted output file for wired, wireless or hybrid networks, especially to evaluate the quality of service of any IP based network reducing the memory usage. In addition, the QoS monitors are capable of generating the user friendly graphical representation of throughput, jitter, loss probability and delay without any post processing which greatly reduces the post processing time also.

A Brief Overview of NS2

NS2 is an object oriented simulator, written in C++, with an OTcl interpreter as a front end. The simulator supports a compiled class hierarchy in C++, and a similar interpreted class hierarchy within the OTcl interpreter. 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.

NS2 uses two languages because simulator has two different kinds of tasks it needs to perform. On one hand, detailed simulations of protocols require a system 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. To setup and run a simulation network, a user should write an OTcl script that initiates an event scheduler, sets up the network topology using the network objects and the plumbing functions in the library, and tells traffic sources when to start and stop transmitting packets through the event scheduler.

The term "plumbing" is used for a network setup, because setting up a network is plumbing possible data paths among network objects by setting the "neighbor" pointer of an object to the address of an appropriate object. When a user wants to make a new network object, he or she can easily make an object either by writing a new object or by making a compound object from the object library, and plumb the data path through the object.. The power of *NS* comes from this plumbing.with out a doubt, *NS2* is one of the most powerful network simulators now-a-days; however it has the following limitations [5-8].

1 Large multi format output files, which require post processing.
2 Huge memory space consumption due to a very large output file.
3 Relatively slower.
4 Lack of built-in QoS monitoring modules.
5 Lacks the capability of visual output representation.

The main output file is just a block of ASCII data in a file and quite cumbersome to access using some form of the post-processing techniques. The format of data representation is different for wired, wireless and hybrid network and also varies widely from protocol to protocol. In addition all the information are stored in this single file for every packets on the network. Therefore the output file becomes too bulky consuming huge memory spaces and also incurring additional time to record every pros and cons.

Modification of the *NS2*

In developing the modified structure of the NS2 simulation that would facilitate the QoS based observations, the basic objectives were

1. to embed the data collecting, calculating and storing in C++ domain for faster simulation,
2. to eliminate the post processing of the huge trace file while evaluating QoS,
3. to introduce QoS monitoring object independent of traffic type, to ease the visualization of result without any post processing.

To enhance the capabilities of present *NS2* following structure shown in Figure 1 has been proposed.

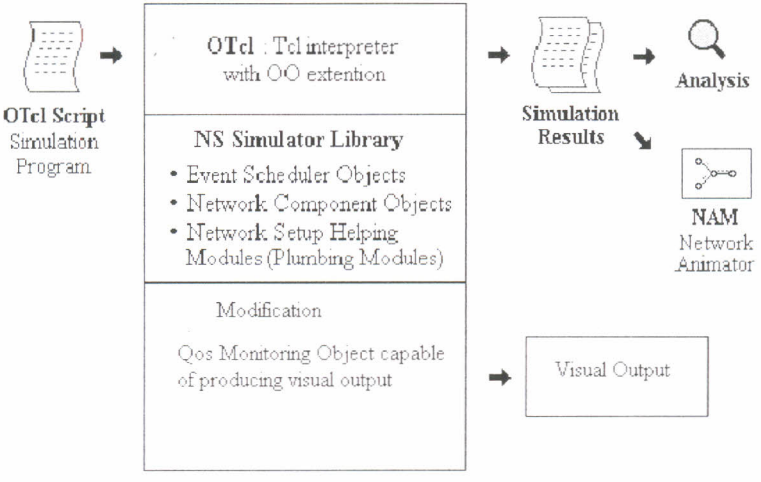


Figure 1 The structure of the proposed simulator (Modified *NS2*).

QoS Monitoring Agents

To fulfill the objective stated above some QoS monitoring agents is need to be created using C++ programming language. The basic requirements for those agents are as follows:

1. It must be capable of calculating the four QoS parameters throughput, jitter, number of packet loss and end to end packet delay and store the result in the corresponding files.
2. In order to calculate those parameters it must be capable of gathering information on
   i. the source address and the destination address of a packet,
   ii. the unique ID of that packet,
   iii. the send time of the packet,
   iv. the receive time, and
   v. the number of lost packet.

3. To find out the individual QoS parameters of each network host in a complex network environment, consisting of many different hosts handling different type of traffics. It must also be
   a) independent of type of traffic it is handling,
   b) independent of network topology and equally deployable in all IP based wired, wireless and hybrid networks.

Considering all the above mentioned criteria, two new agents are proposed as follows.

1. QosMonitor for UDP based applications and
2. TCPQosMonitor for TCP based applications.

These two agents are defined in C++ as a subclass of the *NS2* 'AGENT' class. In order to make its dual in OTcL class hierarchy, a link has to be established in between C++ and OTcL. A constructor is defined in each QoS monitoring agent in order to invoke the variables from both C++ and TCL by binding them together.bBoth the agents work similarly being attached to a traffic sink. The basic difference between them is the way they collect necessary information from the received packets.

The QosMonitor

The QosMonitor is a QoS measuring agent for UDP based applications, which is inherited from the class AGENT of *NS2*. It works as a UDP packet sink. On receiving a packet it stores the packet receiving time and updates the packet count.

It extracts the following information from the received packet header:

1.the flow ID. From the IP header
2.the send time and unique packet ID from a common header (modified), and
3.the *sequence number* from the RTP header.

The following figure shows the information extraction of QosMonitor.

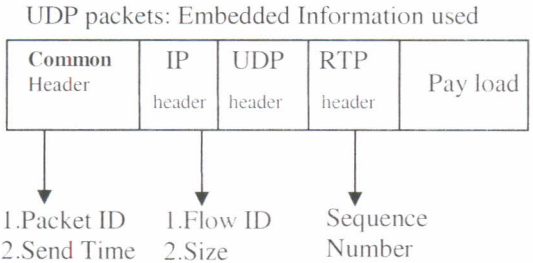


Figure 2: Extracting information for UDP based packets.

Instead of using the flow ID, we can also use the source address and destination address to identify a specific data flow. After collecting these information, it also calculates the Instant throughput, jitter, number of loss packets and end to end packet delay for different packet flows. Instant throughput is calculated as the ratio of the additional bytes received to the time interval elapsed. The jitter is the variation of end to end packet delay, and it is the ratio of delay variation to the packet ID variation. The loss of packet is identified by missing *sequence number* and thus updating a counter. The end to end delay is just the difference between the receive time and send time [9].

In addition, the other statistics kept by the agent are the following:

- bytes_ : are the total number of bytes sent to the application layer.
- nlost_ : is the number of packets lost. This is calculated from the *sequence number* in the header of TCP packets. If the received packet has a *sequence number* higher than the expected value it means that some packets have been lost.
- npkts_ : is the total number of received packets

After the calculation is performed, the result representing the QoS parameters are stored in different files further differentiated by the flow ID , those can be readily plotted with any graph plotter.

The TCPQosMonitor

The TCPQosMonitor is a QoS measuring agent for the TCP based applications, which is inherited from the class AGENT of *NS2*. It works as a TCP packet sink. On receiving a packet it stores the packet receiving time and updates the packet count. It extracts following information from the received packet header:

1. the flow ID. From the IP header,
2. from TCP and the Common header, it also extracts
   - the send time,
   - unique packet sequence number, and
   - the size of the packet data.

After gathering information, it calculates and stores all the four QoS parameters in specific files.

Besides all the information related to QoS, like the UDP qosMonitor it also stores following parameters

- recvbytes_: are the total number of bytes sent to the application layer.
- nlost_ : is the number of packets lost. This is calculated from the *sequence number* in the header of TCP packets. If the received packet has a

sequence number higher than the expected, it means that some packet have been lost.

- ndup_ : is the number of duplicated packets. Duplicated packets may arrive if some acknowledgements get lost. This is calculated from the *sequence number* in the TCP header of the packet. If the received packet has a *sequence number* lower than expected it means that source has re-sent some packets due to *ack* loss or timeout.
- expected_ : is the *sequence number* of the next expected packet.
- npkts_ : is the total number of received packets (including duplicate).

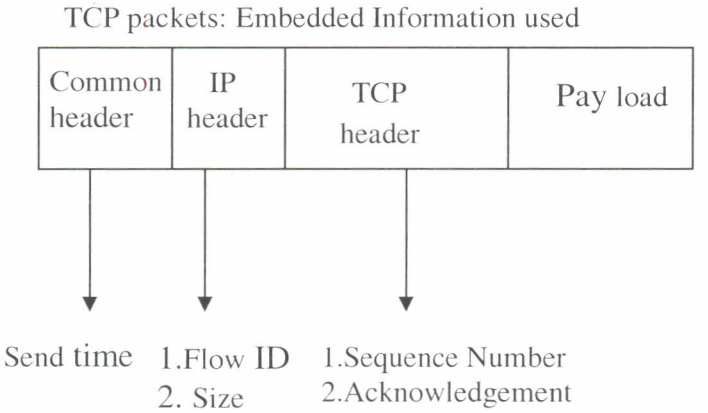


Figure 3: Extracting information from a TCP packet.

The original packet structure of *NS2* is slightly modified to ease the data collecting capabilities of these two powerful QoS monitoring Agents. The common header of every packet in the simulator has a "common" header which is defined in *~NS*/packet.h. Two different fields are introduced for identifying each packet individually and to record the send time of the packet.

How to use the agents

Like all other agent of *NS2*, we have to incorporate the newly introduced agents through proper making procedures. After successful rebuild, the qosMonitor and the TCPQosMonitor agents can be used in OTcl scripts to work as the UDP sink and a TCP sink with the capabilities of measuring QoS for UDP and TCP based application respectively. Two parameters should be carefully initialized. One is the IP level flow ID and the other is the maximum number of flow that a QoS

Monitor is going to handle. Use of flow ID help to distinguish each flow having separate common source and destination address pair, and to store their values in separate and specific files in an easy to plot format. These two QoS monitors are versatile enough to handle any application based on TCP and UDP, like the simple CBR, FTP, VOIP, VIDEO and even the HTTP traffic with a minor modification in the existing HTTP structure, in any network topology. These two agents by default produce 4 types of output file those can be easily plot able by Xgraph (usually available with all *NS2* installation packages) and also with any other plotting software, without any further processing on the data. The output files are

a) throughput *vs.* time
b) jitter *vs.* time
c) delay *vs.* time
d) packet loss *vs.* time

There is also another well formatted output file containing all the layer 3 relevant information for a complete insight of error checking. Further more, with slight post-processing, we can have any type of graphical representation of the following variables to show their effect of variation on those 4 QoS parameters any TCP/IP compatible network.

1) the effect of varying sending rate of the traffic,
2) the effect of varying the packet size,
3) the effect of varying congestion window of the TCP,
4) the effect of increasing payload and
5) the effect of handoff. etc

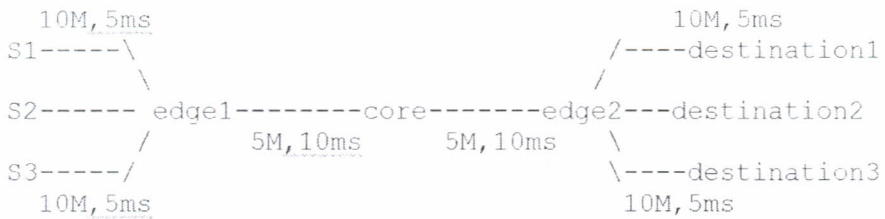
We can also observe the inter-relation of any of the 4 QoS parameters. These agents can also be used with *differentiated* and *integrated* service enabled network and learn their behavior.

Output Obtained by Modified *NS2*

A *diffserv* enabled network will be examined, where the QoS is guaranteed for a single sender and two other best effort services competing with each other. The network topology is as follows where source S1 gets ensured service and other to gets best effort service.

The sender S1 and S2 starts from the very beginning and the S1 sender is intentionally stopped for some times to show its effect on network. The sender S3 starts sending after 5 seconds; however, being a best effort service it remains quiet until enough network resource is free as the S1 refrains from transmission from 10 seconds to 15 seconds. For every particular Flow ID there will be 4 files as follows.

1. Throughput vs. Time (udp_tp_fidi.xg)
2. End-to-End packet delay vs. Time (udp_delay_fidi.xg)
3. Instant Jitter vs. Time (udp_jitter_fidi.xg)



4. No of Packet Loss vs. Time (udp_loss_fidi.xg)

Figure 4: A *Diffserv* Enabled network handling UDP data.

For example for UDP flow having a Flow ID 2 the outputs are shown in the Figure 5. In order to get a better insight of the whole simulation process, two information file is generated; one associated with TCP agent other with UDP agent. It holds information of the all the TCP and UDP flows together.

Time	Throughput
0.036400	0.219780
0.041200	1.666667
0.046000	1.666667
0.052400	2.500000
0.057200	1.666667

Fig 5.1: '*udp_tp_fid2*' file

Time	Delay
0.000000	0.036400
0.002667	0.038533
0.005333	0.040667
0.008000	0.044400
0.010667	0.046533

Fig 5.2: u*dp_delay_fid2.xg*

Time	Jitter
0.000000	0.000000
0.002667	0.002133
0.005333	0.002133
0.008000	0.003733
0.010667	0.002133

Fig 5.3: u*dp_jitter_fid2.xg*

Time	Loss
0.261333	0
0.264000	0
0.269333	1
0.272000	1
0.280000	3

Fig 5.4: *dp_Loss_fid2.xg*

Figure 5: Example of output files.(Partial).

For the above simulation the layer three information file for the UDP agents looks as in Figure 6.It contains byte sent, flow id, packet id, delay, received time, sent time as follows.

```
1000 2    0    0.100000    0.138706    0.038706
2000 2    1    0.108000    0.146706    0.038706
3000 2    2    0.116000    0.154706    0.038706
4000 2    3    0.124000    0.162706    0.038706
5000 2    4    0.132000    0.170706    0.038706
6000 2    5    0.140000    0.178706    0.038706
7000 2    6    0.148000    0.186706    0.038706
8000 2    7    0.156000    0.194706    0.038706
.....
```

Figure 6: Layer three file 'udp_qos_tr' for UDP based flows (partially).

Corresponding visual representation without post processing of the all the data files are given below.

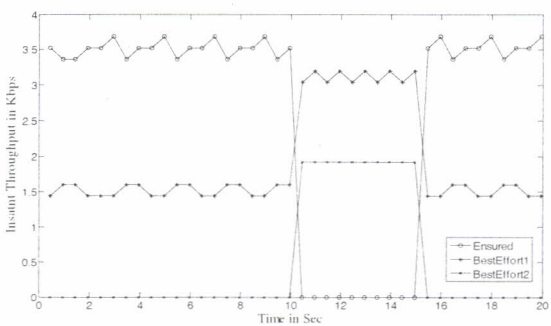


Figure: 7: Throughput in kbps *vs.* Time in second in *Diffserv* network

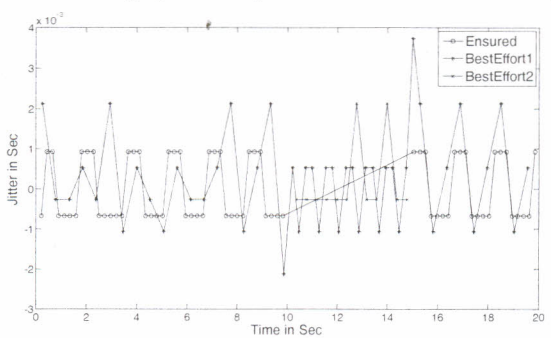


Figure: 8: Packet delay in sec *vs.* Time in second in *Diffserv* network.

The modified NS2 thus producing a similar output as like as the original NS2 in much less time. In addition to these files we can have the outputs for delay and packet loss rate readily without post processing.

The improvement in Performance

In comparing the performance of the existing *NS2*, with modified *NS2* enriched with qos-monitors we picked a hybrid network starting with five wired node along with five wireless nodes. Two way communications between two pairs of nodes for moderate data rate is observed. The size of the network was varied and a Pentium Celeron pc with 512 MB RAM is used for simulation.

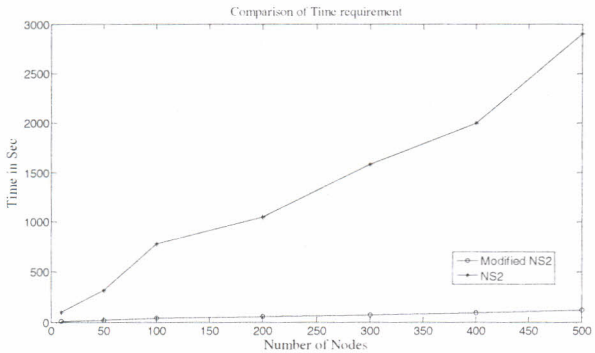


Figure 11: Comparison of Total processing times of the *NS2* and modified *NS2*.

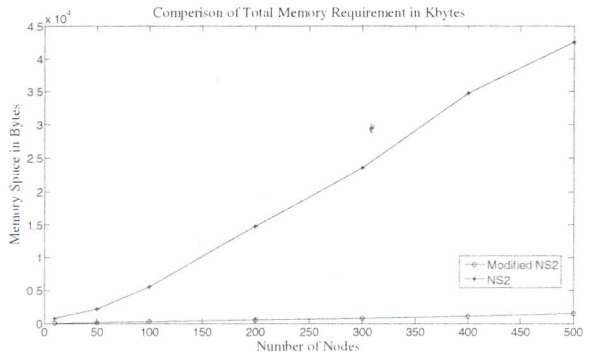


Figure 12: Comparison of memory req. by the *NS2* and Modified *NS2* simulators.

The improvement in the modified simulator is that it works comparatively faster consuming much less memory space and provides formatted output without any

post-processing. Looking at the memory requirements of both the simulator, as found in Figure 10, it is clear that the memory requirement of the modified *NS2* is also very low. Therefore it is evident that the modified version of *NS2* is outperforming the performance of the existing version of *NS2*.

Conclusion

The modified structure of *NS2* provides advantages especially for computing of the QoS related calculations in the network, and it is especially designed for TCP/IP based networks. Moreover, it is concerned with the layer three packets only. So it is at its first stage of evolution and it is highly centralized on QoS related issues of TCP/IP networks. There is also a modified version of ns2 known as Fast ns-2, developed by Laboratories of software technologies at ETH Zurich [10].The Fast ns-2 actually works faster for large networks beyond 1000 nodes but the memory requirement of Fast ns-2 is higher than present version of NS2.So we can conclude that the newly devised modified NS2 is performing better than Fast ns-2 in terms of memory requirement. However the total Installation process is still a bit cumbersome and lengthy.

References

[1] T. Issariyakul and E. Hossain, "Introduction to Network Simulator *NS2*", 1st edition. Springer, 2008.

[2] *NS*-2 The Network Simulator *ns*-2, http://www.isi.edu/nsnam/*ns*/. Oct. 30, 2006.

[3] A. Umair Salleh, Z. Ishak , N. Md. Din, Md Z. Jamaludin ,"A trace Analyzer for *NS2*", IEEE 4th Student Conference on Research and Development (SCOReD 2006), 27-28 June, 2006.

[4] J. Malek, K. Nowak, " Trace Graph – Data Presentation System for *NS*-2 " ISAT 2003.

[5] P. Novák. "Simulation of network structures". Master's thesis, Department of software Engineering, Charles University in Prague, August 2006.

[6] Elias Weingärtner et. al "A performance comparison of recent network simulators", Proceedings of the IEEE International Conference on Communications 2009 (ICC 2009), Dresden, Germany, IEEE.

[7] Yi Lu," Report for CS641 Project", Department of Computer Sciences, Purdue University, December 9, 2002 .

[8] B. Schilling. "Qualitative comparison of network simulation tools". Technical report, Institute of Parallel and Distributed Systems (IPVS), University of Stuttgart, January 2005.

[9] Ray Hunt, "IP Quality of Service Architectures", *IEEE 2001*, pp. 338-343, June 2001.

[10] http://www.lst.inf.ethz.ch/research/ad-hoc/fast-ns2/ accessed at January, 2012.