

UTILITIES FOR PORTING TCP EVALUATION SUITE TO NS3

Project Report Submitted by

B. Pratheek
(4NM12CS041)

H.J. Bhargav
(4NM12CS058)

Sourabh S. Shenoy
(4NM12CS161)

Gopika S. Pai
(4NM12IS020)

UNDER THE GUIDANCE OF

Mrs. Sharada U. Shenoy
Associate Professor, Dept. of CSE

In partial fulfillment of the requirements for the award of the Degree of

Bachelor of Engineering (Computer Science)

from

Visvesvaraya Technological University, Belgaum

NITTE



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

N.M.A.M. INSTITUTE OF TECHNOLOGY

(An Autonomous Institution under VTU, Belgaum)

(AICTE approved, NBA Accredited, ISO 9001:2008 Certified)

NITTE -574 110, Udupi District, KARNATAKA

March 2016



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the project work entitled

"Utilities for porting TCP Evaluation Suite to NS 3"

is a bonafide work carried out by

Mr. B Pratheek - 4NM12CS041

Mr. H J Bhargav - 4NM12CS058

Mr. Sourabh S Shenoy - 4NM12CS161

Mrs. Gopika S Pai - 4NM12IS020

in partial fulfilment of the requirements for the award of

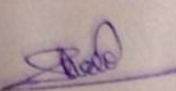
Bachelor of Engineering Degree in Computer Science and Engineering

Prescribed by Visvesvaraya Technological University, Belgaum

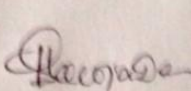
during the year 2015-2016.

It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library.

The project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the Bachelor of Engineering Degree.


Signature of Guide

Prof. Sharada U. Shenoy


Signature of HOD

Signature of Principal

Semester End Viva Voce Examination

Name of the Examiners

Signature with Date

1. _____

2. _____

Abstract

Researchers often use NS-3 to evaluate the performance of their congestion control algorithms against the existing variants. However, there exists no agreed upon framework that specifies the usage of similar topologies, traffic or the metrics used to evaluate the performance in NS-3. The TCP Evaluation Suite is once such framework that aims to solve this problem.

The main aim of this project is to develop the utilities required to port the TCP Evaluation Suite to NS-3. The TCP Evaluation Suite facilitates the comparison of various TCP implementations, by varying factors such as Bandwidth, RTT and FTP. The output consists of metrics like link utilization, packet drop rate, congestion window, percentile queue length, queue length, RTT, Sequence Number and throughput. The outputs are presented in statistical and graphical format.

Tmix is a tool for generating realistic traffic on a given network in NS-3. The module takes connection vectors as input, which are extracted from a real network, and uses that to simulate the traffic. DelayBox adds delays at the intermediate nodes. These modules, present in NS-3.21, have to be ported to NS-3.24 to aid in traffic generation for the TCP evaluation suite.

Acknowledgements

There have been several individuals who have played a key role in the development and completion of this project. Our sincere gratitude goes to our guide Mrs. Sharada U. Shenoy, Associate Professor, Department of Computer Science and Engineering Mrs. Manasa S., Assistant Professor, Department of Information Science and Engineering and for their constant support.

We express our heartfelt gratitude to Dr. Sarojadevi, Professor and HOD, Department of Computer Science and Engineering and Dr. Balasubramani R, Professor and HOD, Department of Information Science and Engineering who encouraged us throughout the development of the project.

We would also like to thank our Principal Dr. Niranjana N Chiplunkar for being an inspirational leader and for his constant encouragement to all students.

The project would not have been successful if not for the guidance of Dr. Mohit P. Tahiliani, Assistant Professor, NITK Surathkal and Mr. Dharmendra and Mr. Pranav, M.Tech, NITK Surathkal. We would like to thank them for the continuing guidance and constant support in helping us throughout our project.

Dr. Uday Kumar Shenoy, Professor, Department of Information Science and Engineering, has also played a pivotal role in helping shape the project. We wish to thank him for the suggestions and feedback given, which helped us immensely.

We would like to thank our Project Coordinators Mr. Raju K., Associate Professor, Department of Computer Science and Engineering, Mr. Ranjan Kumar H S, Assistant Professor, Department of Computer Science and Engineering, Mrs. Asmita Poojary, Assistant Professor, Department of Computer Science and Engineering and Mr. Devidas Bhat, Associate Professor, Department of Information Science and Engineering for

their valuable inputs which made the project a better one.

Lastly we would like to thank all our friends who have been directly or indirectly responsible for the completion of our project.

B. Pratheek

H.J. Bhargav

Sourabh S Shenoy

Gopika S Pai

Contents

Abstract	iii
Acknowledgement	iv
List of Figures	viii
1 INTRODUCTION	1
1.1 ORGANISATION OF CHAPTERS	2
2 LITERATURE REVIEW	3
2.1 INTRODUCTION	3
2.2 TMIX	4
2.2.1 Introduction	4
2.2.2 Characterizing TCP Connections	5
2.2.3 TMIX Workload Generation Tool	6
2.2.4 Implementation of TMIX in ns	8
2.3 TCP EVALUATION SUITE	9
2.3.1 Network Topology Model	9
2.3.2 Traffic Model	11
2.3.3 Performance Metrics	11
2.3.4 Simulation Results	13
3 COMPONENTS	14
3.1 NS-3	14
3.1.1 NS-2 vs NS-3	15

3.1.2	Salient Features of NS-3	17
3.1.3	Software Organization of NS-3	17
3.2	TMIX AND DELAYBOX	19
3.2.1	Connection Vectors	19
3.2.2	Traffic Generation	20
3.3	NETANIM	21
3.3.1	Features of NetAnim	22
3.3.2	Using NetAnim	22
4	METHODOLOGY	23
4.1	PORTING OF TMIX AND DELAYBOX MODULE	23
4.2	TRAFFIC GENERATION	25
4.3	STATS COLLECTION	26
5	RESULTS	30
5.1	TMIX AND DELAYBOX	30
5.2	RESULTANT GRAPHS	33
6	CONCLUSION	37
6.1	PROBLEMS FACED	38
7	SCOPE FOR FUTURE WORK	39
	References	40

List of Figures

2.1	The pattern of ADU exchange in an HTTP 1.0 Connection [1]	5
2.2	A pattern of Concurrent ADU exchanges in a Bit Torrent Connection [1]	6
2.3	Architecture of TCP Evaluation tool [8]	9
2.4	Single-Bottleneck Dumb-bell Topology [8]	10
2.5	Multiple Bottleneck Parking-Lot Topology [8]	10
2.6	Simple Network Topology [8]	10
3.1	Programming Language	15
3.2	Output of Simulation	16
3.3	Performance	16
3.4	Community Support	16
3.5	Software Organization of NS-3 [9]	18
3.6	Connection Vectors for Tmix	19
3.7	Sample animation of Dumb-bell Network in NetAnim	21
3.8	Sample PacketStats for WiFi Application in NetAnim	21
5.1	Trace file generated after porting Tmix and DelayBox to NS-3.24	30
5.2	Netanim output for tmix traffic generated on a dumbbell topology . . .	31
5.3	Tmix Test Suite Output 1	32
5.4	Tmix Test Suite Output 2	32
5.5	Graph of Congestion Window vs Time for TCP Tahoe and Reno, for Bandwidths 5, 10 and 20	33
5.6	Graph of Average Queue Length vs Time for TCP Tahoe and Reno, for Bandwidths 3, 5, 10 and 20	34

5.7	Graph of RTT vs Time for TCP Tahoe and Reno, for Bandwidths 5, 10 and 20	34
5.8	Graph of Sequence Number vs Time for TCP Tahoe and Reno, for Bandwidths 3, 5, 10 and 20	35
5.9	Graph of Link Utilization vs Time for TCP Tahoe and Reno, for Bandwidths 5, 10 and 20	35
5.10	Graph of 95th Percentile Queue Length vs Time for TCP Tahoe and Reno, for Bandwidths 3, 5, 10 and 20	36
5.11	Graph of Throughput vs Time for TCP Tahoe and Reno, for Bandwidths 5, 10 and 20	36

INTRODUCTION

A TCP Evaluation Suite is a simulation tool for determining interactions among different congestion control schemes. The logic behind the tool is to generate identical network experiments for all the different congestion control algorithms to run, for accurate results. Across these identical experiments, one can experiment new congestion control algorithms and performance of different features of the protocols on identical environments.

There are several utilities required for porting the TCP evaluation suite to NS-3.

In order to perform realistic network simulations, one needs a traffic generator that is capable of generating realistic synthetic traffic that looks like traffic found on an actual network. Tmix takes as input a packet header trace taken from a network link of interest. TCP connections as a pattern of ADU (Application Data Unit) transmissions provides a unified view of connections that does not depend on the specific applications driving each TCP connection. Tmix DelayBox simply adds the ability to specify delays and losses for each connection using a connection vector file instead of setting up rules for each source-destination pair.

This tool was not supported by the recent version of NS-3, in which the TCP evaluation suite was planned to be ported, and this work remained as the incomplete GSoC 2015 project. The Tmix and DelayBox are running efficiently on NS-3.24 as well as the recently released NS-3.25.

There are several other utilities required for porting TCP evaluation suite to NS-3. Several post processing tools for plotting the graph and checking the flow of the packets in the network. Gnuplot is a portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms. The Flow Monitor module's goal is to provide a flexible system to measure the performance of network protocols. The module uses probes installed in network nodes, to track the packets exchanged by the nodes, and it will measure a number of parameters. Packets are divided according to the flow they belong to, where each flow is defined according to the probes characteristics.

1.1 ORGANISATION OF CHAPTERS

The project report has been organized under seven chapters, which are as follows:

Chapter 1 introduces the main objective of our project.

Chapter 2 deals with the literature survey conducted on the topics related to the project.

Chapter 3 gives description of the software components used in this project.

Chapter 4 deals with the methodology used in the project.

Chapter 5 discusses the results obtained.

Chapter 6 deals with conclusions and interpretations derived from the relevant studies.

Chapter 7 deals with the further scope for future work and recommendations.

LITERATURE REVIEW

2.1 INTRODUCTION

For realistic network simulations, a traffic generator that is capable of spawning realistic synthetic traffic in a closed loop fashion which looks like traffic on actual network is needed. The system takes a packet header trace taken from any network link of interest as input. The collected trace is then reverse compiled into a source-level characterization of each TCP connection present in the trace. The characterization, called a connection vector, is then used as input to an ns module called Tmix that emulates the socket-level behaviour of the source application that created the corresponding connection in the trace. Another ns module DelayBox is designed to add delay of certain amount at the intermediate nodes. The resulting traffic generated represents the traffic measured on the real link. Such an approach to generate the traffic can reproduce full range of TCP connections found on an arbitrary link. These tools aids for traffic generation in TCP Evaluation Suite.

The TCP Evaluation Suite is to help researchers in evaluating their proposed modifications to TCP. It defines a small number of evaluation scenarios, including traffic and delay distributions, network topologies, and evaluation parameters and metrics like Queue Length, Link Utilization, Packet Drop Rate, Congestion Window, Percentile Queue Length, RTT, Sequence Number and Throughput by varying the FTP Flows, RTT and Bandwidth.

2.2 TMIX

2.2.1 Introduction

TCP congestion control is an end-to-end closed-loop mechanism. In the case of TCP-based applications, TCPs end-to-end congestion control shapes the low-level packet-by-packet traffic processes. For simulating networks used by TCP applications, the generation of network traffic must be accomplished by using models of applications layered over TCP/IP protocol stacks. This is in contrast to an open-loop approach in which packets are injected into the simulation according to some model of packet arrival processes (e.g., a Pareto process). The open-loop approach is now largely deprecated as it ignores the essential role of congestion control in shaping packet-level traffic arrival processes. Therefore a critical problem in doing network simulations is generating application-dependent, network-independent workloads that correspond to contemporary models of application or user behaviour.

The goal is to create an automated method for characterizing the full range of TCP-based applications using a network of interest without any knowledge of which applications are actually present in the network. This characterization should be sufficiently detailed to allow one to statistically reproduce the applications workload in an ns simulation.

The general paradigm we follow is an empirically based method. One first takes one or more packet header traces on a network link and uses the trace(s) to construct a source-level characterization of applications uses of the network. This source-level workload model is constructed by reverse compiling TCP/IP headers into a higher-level, abstract representation that captures the dynamics of both end-user interactions and application-level protocols above the socket layer. Each TCP connection is represented as a connection vector. A connection vector, in essence, models how applications use TCP connections as a series of data-unit exchanges between the TCP connection initiator and the connection acceptor. The data units modelled are not packets or TCP segments but instead correspond to the objects (e.g., files or email messages) or protocol elements

(e.g., HTTP GET requests or SMTP HELO messages) as defined by the application and the application protocol. The data units exchanged may be separated by time intervals that represent application processing times or user think times. A sequence of such exchanges constitutes the connections vector.

Collectively, the set of connection vectors derived from a network trace is a representation of the aggregate behaviour of all the applications found on the measured network. These connection vectors are input to a trace-driven workload generating program called TMIX that replays the source-level (socket-level) operations of the original applications. In this manner, TMIX creates inputs to TCP that are statistically similar to the TCP inputs from the applications that created the original packet trace. TMIX can generate realistic synthetic TCP traffic in either a network testbed or in an ns simulation.

2.2.2 Characterizing TCP Connections

Application level protocols are based on simple patterns of data exchanges within a local connection between the end point processes. These endpoint processes exchange the data in units expressed by their specific application level protocol. The sizes of these application-data units (ADUs) depend only on the application protocol and the data objects used in the application and, therefore, are (largely) independent of the sizes of the network-dependent data units employed at the transport layer and below. For example, HTTP requests and responses depend on the sizes of headers defined by the HTTP protocol and the sizes of files referenced but not on the sizes of TCP segments used at the transport layer.

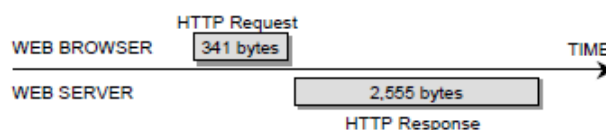


Figure 2.1: The pattern of Adu exchange in an HTTP 1.0 Connection [1]

Application protocols that exchange multiple ADUs between endpoints in a single

TCP connection include HTTP/1.1, SMTP, FTP-CONTROL, and NNTP.

To represent concurrent ADU exchanges, the actions of each endpoint are considered to operate independently of each other so each endpoint is a separate source generating ADUs that appear as a sequence of epochs following a unidirectional flow pattern.

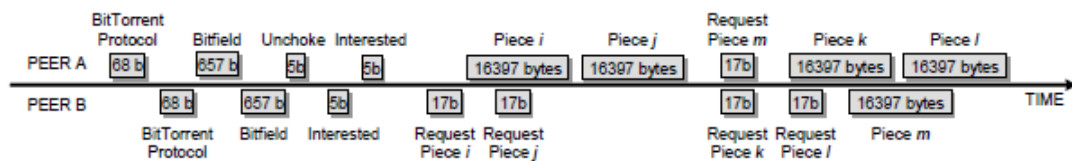


Figure 2.2: A pattern of Concurrent ADU exchanges in a Bit Torrent Connection [1]

From Packet Traces to Connection Vectors

ADU transmissions do not depend on the specific applications driving each TCP connection and provides a unified view of connections. In modelling, after acquiring a trace of TCP/IP headers, the trace is processed to produce a set of connection vectors; one vector for each TCP connection in the trace.

Connection Vectors for sequential connections are computed from unidirectional traces by examining the sequence numbers and acknowledgements numbers in TCP Segments. Changes in sequence numbers in the TCP Segments is used to compute ADU sizes flowing in the traced direction and ACK Values is used to compute ADU size in the opposite direction.

Classification of concurrent connection vector requires bi-directional header trace.

2.2.3 TMIX Workload Generation Tool

TMIX takes a set of connection vectors and replays them to generate synthetic TCP traffic.

The following is the representation of the connection vector used by TMIX:

```
SEQ 102345 3 // Sequential connection
w 16384 16384 // Window size
r 13450 // Minimum RTT
> 329 // Epoch 1
< 402
t 12000
> 403 // Epoch 2
< 25821
t 312000
> 356 // Epoch 3
< 1198
```

The input specifies that a sequential connection should be started at time 102,345 milliseconds and that the connection will consist of 3 epochs. Each endpoint has a maximum TCP receiver window of 16,384 bytes. The base (minimum) RTT of the connection is 13.450 milliseconds. The window sizes and RTTs are optional.

Similarly, a connection vector describing a concurrent connection with 2 epochs in each direction:

```
CONC 3737620 2 2 // Concurrent connection
w 65535 64240 // Window size
r 166883 // Minimum RTT
c> 91
t> 2514493
c< 111
t< 2538395
```


2.2.4 Implementation of TMIX in ns

By default, TMIX models a single direction with initiators on one side of the network and acceptors on the other. Users who wish to generate two-way traffic should use two separate TMIX modules to create two-way traffic.

tmix-end

This module controls all the activities of a set of initiators and acceptors. It reads the set of connection vectors from a file and creates two lists one to hold each connection's ID and start time, and the other which is indexed by the connection ID holds the remaining parameters for each connection type of connection, window size of initiator and acceptor and etc.

Once the connection vector file is processed, new connections are started according to the connection start time list. The TMIX-end module sets the appropriate window sizes for both the initiator and acceptor. The remaining operation of TMIX-end depends upon whether the connection vector describes a sequential or concurrent connection. In sequential connections, the initiator sends a-type ADUs ("requests") that the acceptor "responds to" with a b-type ADU. No pipelining of requests is used. In concurrent connections, the initiator uses pipelining to send multiple a-type ADUs without waiting for a response from the acceptor.

tmix-net

The tmix-net module allows a user to create per-flow delays and losses. The delay and loss rates for each connection are contained in the connection vector. A tmix-net node should be placed in the network between nodes used by tmix-end. Upon start-up, the tmix-net module reads each connections ID, source, destination, RTT, and loss rate into a table. When a packet is received, tmix-net looks up the connection ID, source, and destination in the table to find the appropriate delay and loss values. tmix-net uses the DelayBox component to implement per-flow delays.

The above mentioned modules TMIX and DelayBox were implemented on NS-2 which

were later re-written for NS-3 to aid the similar functionality in NS-3.

2.3 TCP EVALUATION SUITE

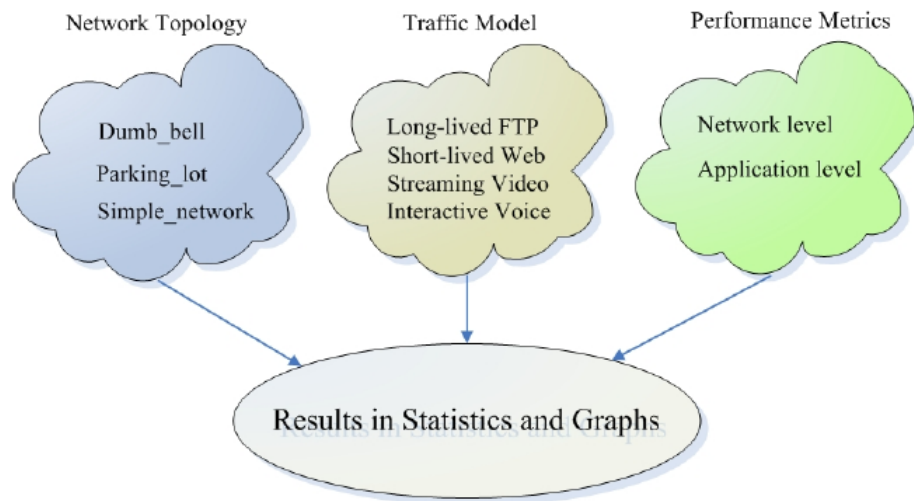


Figure 2.3: Architecture of TCP Evaluation tool [8]

The tool primarily consists of three components: Topology Model, Traffic Model and Performance Metrics. The result statistics and graphs are generated on completion of the simulation.

2.3.1 Network Topology Model

The tool includes three topologies for the performance evaluations. They are as follows:

Dumb-bell Topology

In this, two routers are connected by a single link called the bottleneck link and the routers can have number of nodes of connected to them just like a dumbbell.

Parking-Lot Topology

It is similar to dumbbell topology except that it introduces cross traffic at the intermediate routers.

Simple Network Topology

In this kind of topology, the core routers represent the backbone of the network with access routers responsible for sender or receiver nodes to connect to the network.

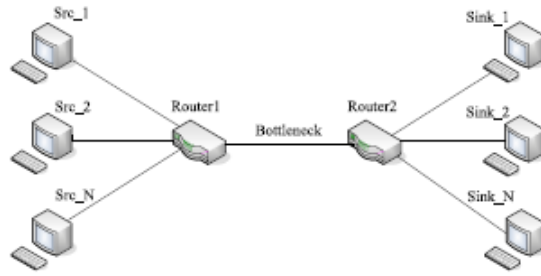


Figure 2.4: Single-Bottleneck Dumb-bell Topology [8]

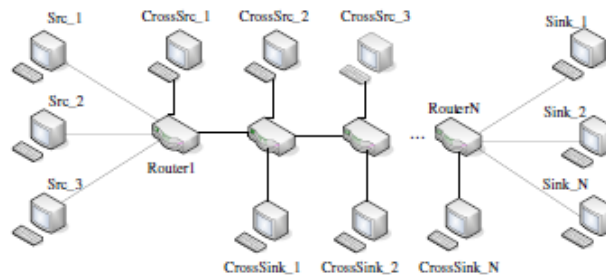


Figure 2.5: Multiple Bottleneck Parking-Lot Topology [8]

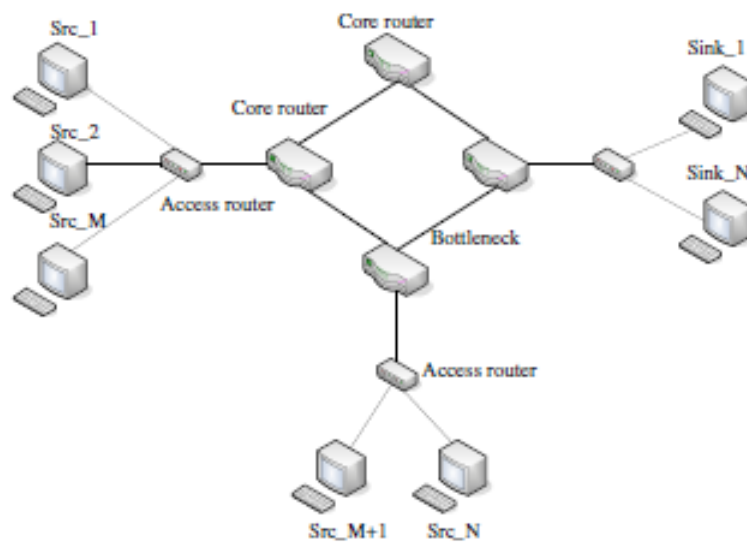


Figure 2.6: Simple Network Topology [8]

2.3.2 Traffic Model

The tool includes four common types of traffic. They are as follows:

1. **Long-lived FTP traffic**

FTP traffic uses infinite, non-stop traffic transmission, starting at random time and runs on the top of TCP.

2. **Short-lived Web traffic**

The Short-lived Web traffic is generated by another ns module called PackMime.

3. **Streaming Video traffic**

The streaming video traffic is modelled using Constant Bit Rate (CBR) Traffic over UDP with a freedom of setting packet size and sending rate.

4. **Interactive Voice traffic**

There are two methods to generate Voice traffic in this tool. One is similar to CBR-based streaming traffic. Other method uses ON/OFF model. The mean ON period is 1.0 sec, and the mean OFF duration is 1.35 sec. These values are set in accordance with ITU-T recommendations, but are changeable if needed. The voice packet size is 200 bytes, including the 160 bytes data packet (codec G.711, 64 kbps rate and 20 ms duration), 20 byte IP header, 8 byte UDP header, and 12 byte RTP header. These parameters can be changed by using other voice/audio codecs.

2.3.3 Performance Metrics

Throughput, Delay, Jitter and Loss rate

1. **Throughput**

Throughput is taken as the bottleneck link utilization for the network metrics.

It is the total traffic transmitted in bits per second. Goodput is the measurement of only useful transmission of traffic whereas throughput includes even retransmission.

For long-lived FTP traffic it computes transmitted traffic during some intervals in bps.

For short-lived Web traffic it collects the goodput and response time for measuring web traffic performance.

For Voice and Video traffic the goodput is measured in transmitted packets excluding lost packets and delayed packets.

2. Delay

Bottleneck queue size is used as an indication for queue delay in bottlenecks. Mean and max/min queue size, percentile queue length to determine the queue length.

For FTP traffic delay isn't much affected by delay.

For Web traffic delay can be considered as the time duration between a request sent and a response received.

For streaming and voice traffic packet delay is the duration between sending and receiving packets at the end nodes.

3. Jitter

Jitter is crucial for traffic such as voice and video. Large jitter means significant packet loss and need for more buffer size at the receiver.

4. Loss rate

FTP and Web traffic aren't much affected by this metric.

For interactive and streaming traffic higher loss rate results in failure of receiver to decode packets.

5. Response Times and Oscillations

Response time to sudden changes in network is a key concern in the design of congestion control mechanisms. On one hand it should rapidly respond to changes in the network and on the other hand it should make sure that changes are not too critical to maintain network stability. The tool is capable of observing response time and changes with the help of figures it generates, in case of variable bandwidth, RTT, and other parameters.

6. Fairness and Convergence

Using Jain's Fairness index the fair bandwidth share of end-to-end FTP flows that traverse the same route is measured.

Convergence time is measured as the time elapsed between multiple flows from an unfair share of link bandwidth to a fair state.

2.3.4 Simulation Results

The tool includes the RPI graphing package to automatically generate the above-discussed performance metrics. At the end of a simulation, it also automatically generates a series of user defined statistics (e.g. bottleneck average utilization, bottleneck 90-percentile queue length, average per-flow goodput, etc.) and graphs (like bottleneck utilization and queue length variation over time, per-flow throughput over time, etc.). It can create latex and html files in order to present the simulation results in a paper or webpage form. All the simulation-generated data is stored in a temporary directory for later use.

COMPONENTS

In this project we use the following components :

1. NS-3.24 a discrete-event network simulator for Internet systems.
2. Tmix - an Internet traffic generator for NS-3 simulator, And Delaybox, a module to introduce delays in traffic flow at specific nodes.
3. NetAnim - an offline animator for NS-3 based on QT toolkit.

3.1 NS-3

NS-3 is a discrete event network simulator, used mainly for research and educational use. NS-3 is a free and open source software, and is licensed under GNU General Public License v2.

NS-3 is built using C++ and Python. It is split into different modules, each containing models for real world network components. A general simulation in NS-3 is composed of several steps, usually starting with defining the topology of the network, followed by adding the different components to the network elements.

Then the nodes and links in the defined topology are configured (using NS-3 attribute system). The simulation is run, and the data is logged when the simulation runs. The user can use the data to study the consequences of the design and variables on the network. Multiple tools are used for post processing, notably, NetAnim for displaying animations and Gnuplot for plotting graphs.

3.1.1 NS-2 vs NS-3

NS-3 was preceded by NS-2. NS-2 was one of the most widely used network simulators and still continues to be used in education and sometimes in research. NS-2 used C++ as the backend and had a TCL frontend. It was designed with low memory devices in mind, thus used TCL scripting language as frontend. Unlike NS-3, a person working with NS-2 would need knowledge of both C++ and TCL for effective debugging. Plus the error messages were not obvious and could cause confusion to novice users. While NS-2 has lots of diverse modules due its longer existence, NS-3 is more active in development and features are being added at a fast rate. NS-3 supports Direct Code Execution, by which the entire Linux stack can be encapsulated in a NS-3 node. Also many of the bugs and limitations of NS-2 have been overcome in NS-3.

NS2	NS3
<ul style="list-style-type: none"> NS2 is implemented using a combination of OTcl (for scripts describing the network topology) and C++ (The core of the simulator) 	<ul style="list-style-type: none"> NS3 is implemented using C++
<ul style="list-style-type: none"> With modern hardware capabilities, compilation time is an issue for NS2 	<ul style="list-style-type: none"> Compilation time is not an issue ,NS3 can be developed with C++ entirely.
<ul style="list-style-type: none"> Simulation script should be written on OTcl 	<ul style="list-style-type: none"> A simulation script can be written as a C++ program
<ul style="list-style-type: none"> OTcl is the sole language you can depend on , for NS2. 	<ul style="list-style-type: none"> There is a limited support for Python in scripting.

Figure 3.1: Programming Language

NS2	NS3
Trace Files	.pcap and .tr files
NAM for animator	NetAnim for the animation

Figure 3.2: Output of Simulation

NS2	NS3
<ul style="list-style-type: none"> The total computation time required to run a simulation scales more in NS2. 	<ul style="list-style-type: none"> NS3 performs better than NS2 in terms of memory management. (automatic de-allocation of objects)
<ul style="list-style-type: none"> This is due to the removal of the overhead associated with interfacing OTcl with C++, and the overhead associated with the OTcl interpreter. 	<ul style="list-style-type: none"> The aggregation system prevents unneeded parameters from being stored, and packets don't contain unused reserved header space.

Figure 3.3: Performance

NS2	NS3
<ul style="list-style-type: none"> Its has got more diverse set of contributed modules than NS3 	<ul style="list-style-type: none"> Improving in NS3
<ul style="list-style-type: none"> NS2 doesn't have an emulation mode 	<ul style="list-style-type: none"> NS3 has an emulation mode, which allows for the integration with real networks.

Figure 3.4: Community Support

3.1.2 Salient Features of NS-3

1. NS-3 allows users to save packets to pcap files which are compatible with tools like wireshark. This can help in easier post-processing.
2. NS-3 is a discrete event simulator. In NS-3 time jumps from event to event, instead of flowing like in the real world.
3. Network Simulation Cradle : allows users to run the Linux kernel TCP stack inside simulation.
4. POSIX Emulation : Allows to run POSIX programs inside NS-3. This allows NS-3 to talk to the outside world and communicate over real networks
5. Tracing in NS-3 is quite easy and available, and almost everything can be traced using the trace framework.
6. Allows packets with virtual zero bytes for applications where we dont care about the data in the packet. This can save memory.
7. Optional features on nodes can easily enabled or disabled. Eg, mobility model is optional feature for nodes.
8. Built for easy extension and feature addition.

3.1.3 Software Organization of NS-3

The NS-3 simulator is written mainly in C++, with Python bindings. The simulation core and models are implemented in C++. The whole system is built as a system of libraries which are linked to the main program only if the functionality provided by the library is needed. NS-3 API is also ported to Python, thus allowing users to write Python code for scripting and running simulations.

The source code of NS-3 lives in the /src directory. Components and models are implemented in several modules as subfolders of src directory. The simulation core is implemented in src/core and the network objects like packets in src/network. Events

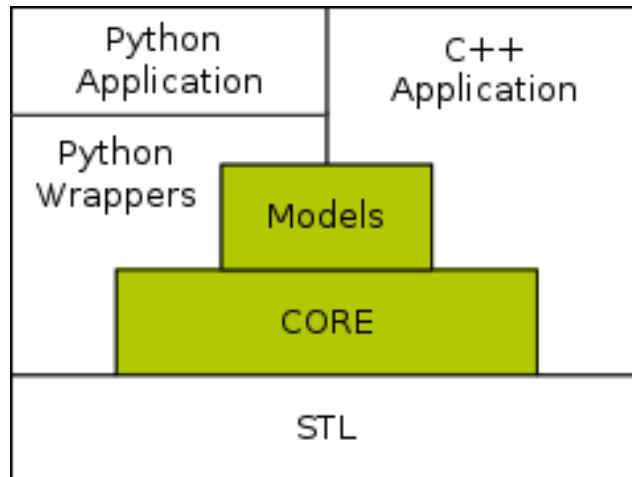


Figure 3.5: Software Organization of NS-3 [9]

are scheduled to be run at a particular time in the simulation code, and the event simulator handles the execution of the events. The simulator and the event mechanism are part of the core design of NS-3.

The different network elements like models for physical devices, links, various protocols, applications etc are written in various folders of the src folder.

Each module (ie., folder in src) has around six folders : bindings, doc, examples, helper, model and test. The source code is written in the model folder . The helper has API which enable users to write simulations more easily, but with lesser fine grain control. The doc, test and examples folders have the documentation, test cases and suites and example programs respectively. There is a wscript file which is used by the waf build tool to build the source.

3.2 TMIX AND DELAYBOX

Tmix is a synthetic traffic generating tool used to generate traffic that looks like traffic generated from a real network. Tmix takes as input a trace file of packet headers, reverse compiled into a source level characterization of all TCP connections in the link. This trace is given as input to the Tmix module that replicates the behavior of the source applications. This actually emulates the socket level reads and writes the application had actually done in the real network.

3.2.1 Connection Vectors

```
#
# tcvec format version 2
# Number of Cvecs: 26983
S 3412 1 21217 555381
W 64800 6432
R 1118156
L 0.000000 0.000000
I 0 0 253
A 0 123693 510
A 6308497 0 0
S 3724 34 15715 439847
W 8760 6656
R 3146992
L 0.000000 0.157900
I 0 0 250
A 0 48877 1024
A 1082014 0 1536
A 5420122 0 1536
A 3880570 0 1536
A 3824097 0 2048
A 4089707 0 2048
A 3750349 0 1024
A 1259480 0 1024
A 2957493 0 1536
A 691951 0 1024
A 2823464 0 1024
A 517197 0 512
A 542478 0 1024
A 4927824 0 3072
A 4129545 0 2560
```

Figure 3.6: Connection Vectors for Tmix

The application level protocols exchange data in terms of application level packets. The size of the application data units (ADUs) are dependent on the application layer headers and not on the transport layer TCP segment sizes. A set of ADU exchanges can be represented as a connection vector, each with set of epochs.

$$C(i) = \langle E(i), E(i+1), \dots, E(k) \rangle$$

And each epoch

$$E(i) = (a(i), b(i), t(i))$$

Where $a(i)$ is the size of the i th ADU from initiator to acceptor, $b(i)$ is the size of the i th ADU from acceptor to initiator, and $t(i)$ is the think time between the i th and $(i+1)$ th ADU.

The analysis is done by examining sequence numbers and acknowledgment numbers and computing the ADU sizes based on these numbers.

3.2.2 Traffic Generation

The tmix tool takes a set of connection vectors as input. The tmix tool will faithfully reproduce the traffic observed in the actual link. The tool will initiate TCP connections for each connection vector given to it based on the time $t(i)$ and then generate packets based on the sizes given in the connection vectors, ie., $a(i)$ and $b(i)$.

For instance, consider the connection vector

$$C(i) = \langle (100, 150, 0.1), (200, 25000, 3), (300, 1500, 0) \rangle$$

Once the initiator establishes a new connection with the acceptor, the following steps occur:

1. Initiator writes 100 bytes and reads 150 bytes through the socket.
2. Acceptor reads 150 bytes and reads 100 bytes through its socket
3. The initiator then sleeps for 100 milliseconds and writes 200 bytes and reads 25000 bytes.
4. Acceptor reads 25000 bytes and writes 200 bytes.
5. Finally after sleeping for 3000 milliseconds, the receiver and sender terminate the exchange.

DelayBox is a module that works in tandem with Tmix in order to introduce random delays so as to ensure realistic traffic scenarios.

3.3 NETANIM

NetAnim stands for Network Animator, which is used to simulate the network for which a scenario is created. It is an offline animator based on Qt toolkit. It animates the simulation using a XML file that is created during simulation.

The header file to be included to use NetAnim is netanim-module.h

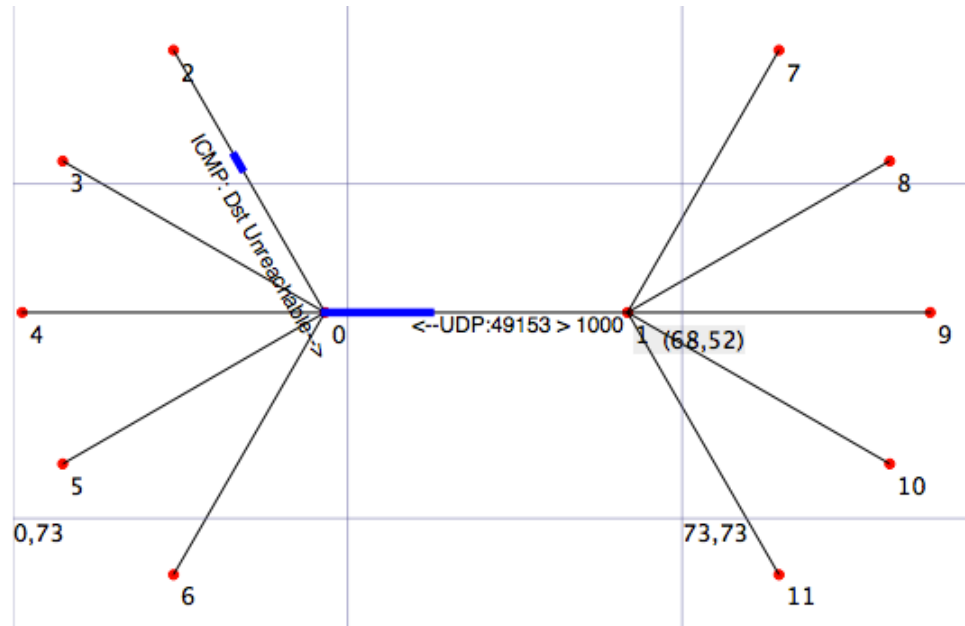


Figure 3.7: Sample animation of Dumb-bell Network in NetAnim

	Tx Time	From Node Id	To Node Id	
1	2.5e-05	0	5	Wifi MGT_BEACON FromDS: 0 toDS: 0 DA: ff:ff:ff:ff:ff:ff
2	2.5e-05	0	6	Wifi MGT_BEACON FromDS: 0 toDS: 0 DA: ff:ff:ff:ff:ff:ff
3	2.5e-05	0	7	Wifi MGT_BEACON FromDS: 0 toDS: 0 DA: ff:ff:ff:ff:ff:ff
4	0.000167033	5	6	Wifi MGT_ASSOCIATION_REQUEST FromDS: 0 toDS: 0
5	0.000167033	5	7	Wifi MGT_ASSOCIATION_REQUEST FromDS: 0 toDS: 0
6	0.000167033	5	0	Wifi MGT_ASSOCIATION_REQUEST FromDS: 0 toDS: 0
7	0.000279066	0	5	Wifi CTL_ACK RA:00:00:00:00:00:07
8	0.000279066	0	6	Wifi CTL_ACK RA:00:00:00:00:00:07
9	0.000279066	0	7	Wifi CTL_ACK RA:00:00:00:00:00:07
10	0.000402103	6	5	Wifi MGT_ASSOCIATION_REQUEST FromDS: 0 toDS: 0
11	0.000402103	6	0	Wifi MGT_ASSOCIATION_REQUEST FromDS: 0 toDS: 0
12	0.00051414	0	5	Wifi CTL_ACK RA:00:00:00:00:00:08
13	0.00051414	0	6	Wifi CTL_ACK RA:00:00:00:00:00:08
14	0.00051414	0	7	Wifi CTL_ACK RA:00:00:00:00:00:08

Figure 3.8: Sample PacketStats for WiFi Application in NetAnim

3.3.1 Features of NetAnim

1. Animate packets over wired-links and wireless-links (Limited support for LTE traces. No support for Ipv6).
2. Packet timeline with regex filter on packet meta-data.
3. Node position statistics with node trajectory plotting (path of a mobile node).
4. Print brief packet-meta data on packets.
5. Use custom icons for nodes.
6. Parse flow-monitor XML files and display statistics for each flow.
7. Show IP and MAC information, including peer IP and MAC for point-to-point links.
8. Display double or uint32 valued counters vs time for multiple nodes in a chart or a table.
9. Step through a simulation one event at a time and pause the simulation at a given time.
10. Print the routing table at nodes at various points in time.

3.3.2 Using NetAnim

Using NetAnim involves two steps, first step the generation of XML file using ns3::AnimationInterface during the simulation. Next is to load the XML trace file generated into NetAnim.

METHODOLOGY

4.1 PORTING OF TMIX AND DELAYBOX MODULE

The procedure involves the creation of skeleton modules for both Tmix and DelayBox. For doing the above operation, we used a python script `create-module.py` already present in the `ns` package. We then replicated the older Tmix and DelayBox modules to the newer modules that we created. The next step was to replace the deprecated API function calls present to make it functional with the latest Network Simulator.

According to the changelog in the `nsnam` website, from NS-3.14 and above the usage of `RandomVariable` class (present in `NSDIR/src/core`) was deprecated, from NS-3.21 and above it has been eliminated, and was replaced by `RandomVariableStream` class (present in `NSDIR/src/core`) and this class was made as an abstract class.

Tmix and DelayBox used few methods of `RandomVariable` class, which now had to be replaced by their equivalents in the `RandomVariableStream` class. The `RandomVariableStream` class has several sub classes, each representing a type of random variable. Eg: `ConstantRandomVariable`, `ExponentialRandomVariable`, `GammaRandomVariable`, `NormalRandomVariable`, `ParetoRandomVariable`, `SequentialRandomVariable`, `UniformRandomVariable`, `ZetaRandomVariable` etc. Tmix and DelayBox earlier used the `RandomVariable` sub class equivalent of `UniformRandomVariable`.

Therefore, we had to replace all the instances of `UniformVariable` and related methods with instances and methods of `UniformRandomVariable`. In addition, the earlier

code was using simple instances, but it is better to use Smart Pointers to enable safe and easy passing of instances. So we made changes to use Smart Pointers (Ptr <ClassName>InstanceName) instead of simple instances. There were a few changes to the logic also, due to the usage of Smart Pointers. With this, we managed to get Tmix and DelayBox running on NS-3.24 producing output similar to the ones the older version was producing on NS-3.20 and earlier.

Next, we wrote the test suite, examples and documentation for Tmix so that it could be integrated to NS-3.

The test suite has three tests:

1. TmixTopologyTest:

This test creates a Tmix topology (similar to a dumbbell topology). Next, we create some number of nodes on the left and right sides. Then we check if the nodes have been created or not. If the nodes have been created, the test is passed, else failed.

2. TmixCvecParseTest:

This test reads connections vectors from a specified file. We fixed the file and the path is hard coded. Once the first connection vector has been read and stored in ConnectionVector object, the values are compared with expected hard coded values to ensure that the parsing has been done properly. If any of the values do not match, the test fails.

3. TmixTrafficTest:

In this, the traffic is generated based on the connection vectors from a specified file. The connection vectors are read and passed onto Tmix pair, which is a construct of TmixTopology used to store pairs of receivers and senders.

The traffic is generated based on the connection vectors, and stored into a temporary file. Later the temporary file is compared to a file with expected values. If

the files match, the test is passed.

Next, we will look at the example program for Tmix.

The example program that we wrote, reads connection vectors from files. It has the option to use two files, i.e; two-way traffic, to use only a certain portion of the file using random variables, and to create animation (NetAnim).

We assume two files inbound.ns and outbound.ns to be available in the scratch folder. These files contain the connection vectors. These files are parsed and the connection vectors are added to the Tmix constructs.

Then we install a trace helper Ns2StyleTraceHelper, which will log all the packets being generated and output on some ostream object (console or a file). If animation is enabled, then we allocate position for each object in the simulation. The simulation is run for 10 seconds.

Then we documented the working and usage of Tmix in a text (.rst) file according to the format readable for the automatic documentation utilities.

4.2 TRAFFIC GENERATION

The next step was to find out details of the different types of traffic required for the TCP Evaluation Suite and provide simple implementation generating these kinds of traffic. The different types of traffic are:

1. **Tmix Traffic:**

Traffic generated by Tmix from a connection vector file. Achieved using Tmix module that was ported to NS-3.24.

2. **Streaming Traffic:**

Traffic simulating a streaming connection between two points. Achieved by using OnOff Application with an off time of 0 seconds. This ensures there is continuous

flow of traffic from one node to another. The data rate is settable according to the requirement.

3. **Voice Traffic:**

Simulating a voice call between two parties. This is achieved by using OnOff traffic with On time of 1s and Off time of 1.35 s. In addition, we set the data rate to 80 kbps and packet size to 172 bytes. This is in accordance with the standards set by ITU for stimulation of voice traffic.

4. **FTP traffic:**

FTP traffic uses BulkSend Application to copy real life FTP file transfers. We set the MaxBytes attribute of the application to 0, thereby making the application to send traffic infinitely.

We managed to get these kinds of traffic running on a dumbbell like topology.

4.3 STATS COLLECTION

To collect the different statistics on the router nodes of a topology we had to hook the nodes to different callback functions. Whenever an event happens in the simulation, it can be configured to call a callback method. There are different types of events for which callbacks can be configured, some of them being Enqueue event on a queue, CongestionWindowChange event for TCP sockets etc. These events are called trace sources in NS-3.

To collect the different statistics, we had to write callbacks for different trace sources. To enable the hooking of the callback methods to the trace sources of the correct nodes, we created multiple Stats classes as described below. An Install method was provided with each class, which when called on a node, hooked up the associated trace sources with the required callbacks.

The different statistics we provided collection facilities are as follows. The first four are collected at the router nodes.

1. Average Queue Length:

To get the average queue length we needed to sample the queue length at some intervals. To do this, we hook the Enqueue trace source of the queue of the given router to a callback method. This callback method is called when a packet gets enqueued, and it gets the queue length and adds it a static variable. Later at every one-second interval, the total size is divided by the number of times the method had been called, to get the average queue length over the interval.

2. Percentile Queue Length:

The percentage queue length is calculated similar to Average Queue Length. Instead of adding the queue lengths, the queue length values are stored in a vector. Later at the end of every second, the values are sorted and the 90th percentile value is found.

3. Link Utilization:

To find the link utilization, we connect the Tx trace source of the Ipv4L3Protocol object on the node to a callback method. Whenever a transmission event happens over the node, through the Ipv4L3Protocol, the callback is fired. In this method, the size of the packet that has been sent over the medium is found and added to a static variable. At the end of every second, the link utilization is calculated according to the formula for link utilization based on the bandwidth of the link.

4. Drop Rate:

The drop rate is calculated from the queue again. The queue in NS-3 is designed to have variables to store certain statistics by default. The queue maintains details of the number of packets it has dropped and received, and these can be recovered using Getter methods. The total number of dropped packets and received packets are gathered every second and their ratio is taken as the drop rate.

The next set of statistics are collected at the sending nodes.

1. Congestion Window:

These set of statistics are collected at the sending nodes. The Trace source for Congestion Window Change is available from the socket only. The socket is unfortunately not made public by the applications. There is a Getter method, to get the socket from the applications. However, the applications do not create the socket unless the app actually starts. That is the socket is created during runtime. So we are unable to access the socket during compile time to hook up trace source with the callback method.

To get the access to the callback, we schedule a call to a function which takes the node as a parameter and extracts the socket from the application inside the node. The call is scheduled just after the app is started. In this function, the application is extracted from the node, and then the socket is got from the application. Then the trace source is connected to the required callback.

The CongestionWindow trace source is hooked to a method, which will direct output the time of the event (time of Congestion Window Change) and the value (Congestion Window Size) to a dat file.

2. RTT:

To get the RTT value, we have a trace source called RTT associated with the socket. Whenever the value of the RTT changes, the trace source calls a method, which logs the time and RTT value into a dat file.

3. Sequence Number:

Similar to the above two statistics, a trace source exists which trigger when the sequence number of the next packet to be sent changes. A similar callback logs the time and sequence number to a dat file.

The final group of statistics is collected at the destination side.

1. **Throughput:**

Throughput is measured at the sink application. The Sink application at the receiving side is extracted from the node. The PacketSink application keeps track of the data that is sent into it. A Getter method exists for the PacketSink application, which gives the total amount of data that it has received. Every second, a method is scheduled to be called, which gets this information and then calculates the throughput based on the current and previous values of the total transmitted data.

RESULTS

Following is the output after porting Tmix and DelayBox to NS-3.24. The trace file generated is about 51mb in size and is similar to the one obtained by the same modules in NS-3.24. The output is the same even when run with NS-3.25.

5.1 TMIX AND DELAYBOX

```
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2649841 22199 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2651289 22200 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2652737 22201 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2654185 22202 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2655633 22203 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2657081 22204 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2658529 22205 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2659977 22206 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2661425 22207 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2662873 22208 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2664321 22209 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2665769 22210 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2667217 22211 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2668665 22212 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2670113 22213 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2671561 22214 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2673009 22215 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2674457 22216 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2675905 22217 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2677353 22218 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2678801 22219 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2680249 22220 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2681697 22221 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2683145 22222 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2684593 22223 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2686041 22224 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2687489 22225 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2688937 22226 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2690385 22227 1 0x10 40 0
+ 1.841369 0 4 ack 1500 ----- 0 0.180324 1.132195 2691833 22228 1 0x10 40 0
```

Figure 5.1: Trace file generated after porting Tmix and DelayBox to NS-3.24

Next, we have the output of the example script written for Tmix that accepts an

inbound and outbound file and simulates the traffic on a dumbbell topology as shown. The simulation is graphically shown using netanim.

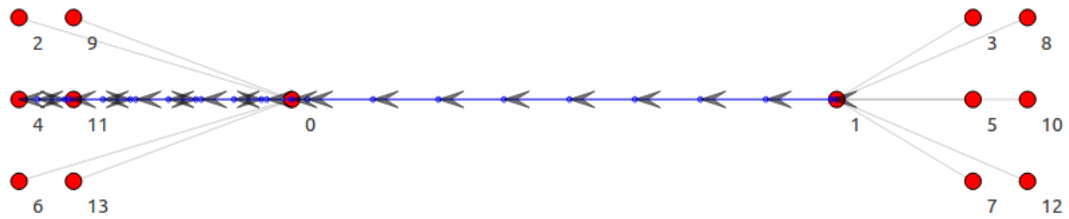


Figure 5.2: Netanim output for tmix traffic generated on a dumbbell topology

Further, we wrote a test suite with 3 test cases to test the working of the API calls made by the tmix module. The test suite makes use of Assert function calls provided by NS3 to ensure the proper creation of objects of various classes, test the values of variables to be appropriate, etc.


```
pratheek@kasu:~/ns3_tcp_eval/ns-allinone-3.24.rc2/ns-3.24.rc2$ ./test.py --suite=tmix
Waf: Entering directory `/home/pratheek/ns3_tcp_eval/ns-allinone-3.24.rc2/ns-3.24.rc2/build'
Waf: Leaving directory `/home/pratheek/ns3_tcp_eval/ns-allinone-3.24.rc2/ns-3.24.rc2/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1.850s)

Modules built:
antenna          aodv              applications
bridge           buildings         config-store
core            csma              csma-layout
delaybox (no Python) dsdv              dsr
energy          fd-net-device     flow-monitor
internet        lr-wpan           lte
mesh            mobility          mpi
netanim (no Python) network          nix-vector-routing
olsr            point-to-point    point-to-point-layout
propagation      sixlowpan         spectrum
stats           tap-bridge        tcp-eval (no Python)
test (no Python) tmix (no Python)  topology-read
uan             virtual-net-device wave
wifi            winax

Modules not built (see ns-3 tutorial for explanation):
brite           click             openflow
visualizer

PASS: TestSuite tmix
1 of 1 tests passed (1 passed, 0 skipped, 0 failed, 0 crashed, 0 valgrind errors)
pratheek@kasu:~/ns3_tcp_eval/ns-allinone-3.24.rc2/ns-3.24.rc2$
```

Figure 5.3: Tmix Test Suite Output 1

```
PASS: Test Suite "tmix" (0.090)
PASS: Test Suite "Creates nodes and checks if they have been created" (0.020)
PASS: Test Suite "Tries to parse a Cvec file" (0.000)
PASS: Test Suite "Creates traffic and checks if they have been created" (0.070)
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
NORMAL >- testResults.txt          text <= 41 words < 25% :    1:   1
"testResults.txt" 4L, 244C
```

Figure 5.4: Tmix Test Suite Output 2

5.2 RESULTANT GRAPHS

The evalStats module collect the statistics and the create graph module parses the .dat file to plot the graphs shown below. Statistics such as congestion window size, RTT, Sequence Number are collected at the sender node. Statistics such as Average Queue Length, Queue Length, Throughput etc. are collected at the bottleneck router and Average Packet drop rate is collected at the receiver node.

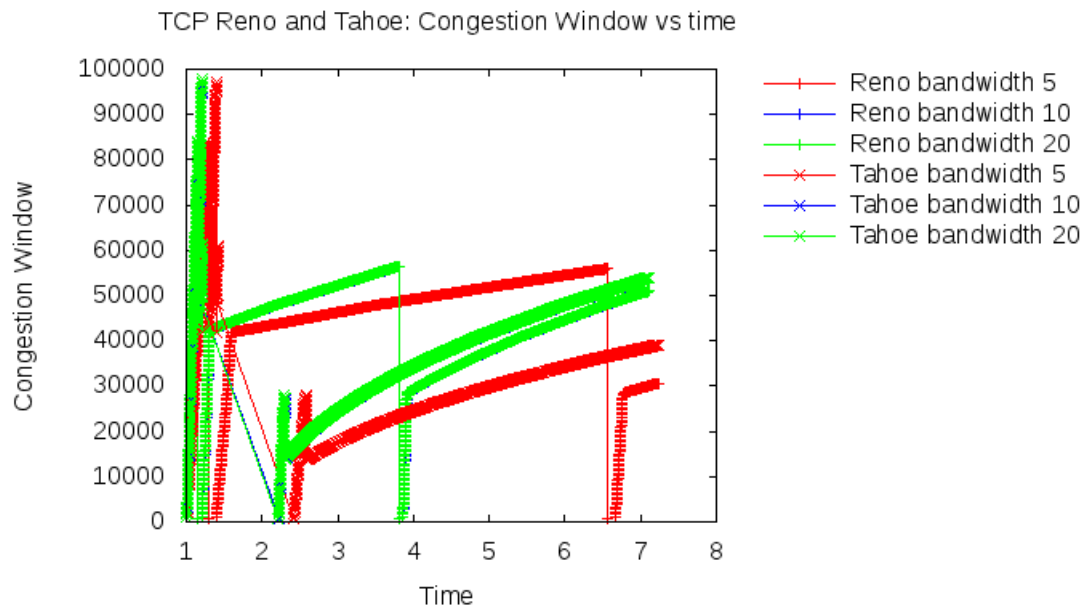


Figure 5.5: Graph of Congestion Window vs Time for TCP Tahoe and Reno, for Bandwidths 5, 10 and 20

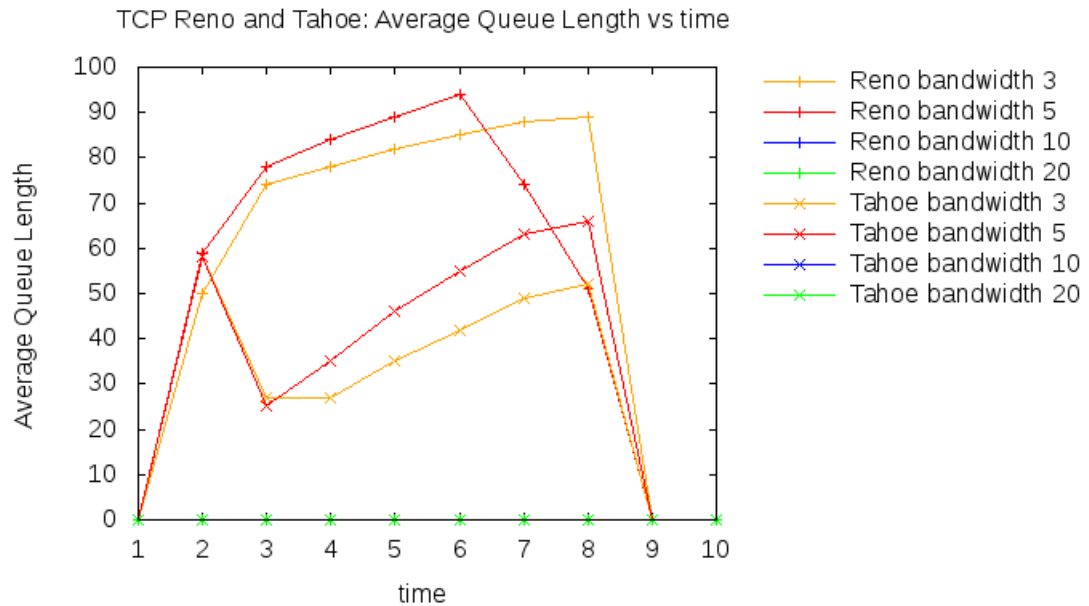


Figure 5.6: Graph of Average Queue Length vs Time for TCP Tahoe and Reno, for Bandwidths 3, 5, 10 and 20

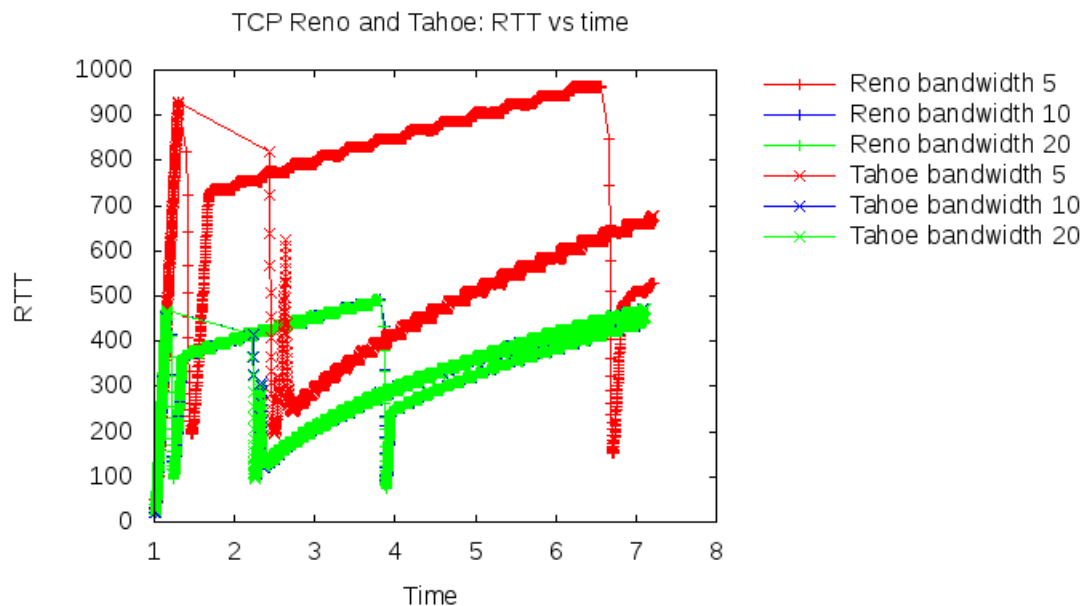


Figure 5.7: Graph of RTT vs Time for TCP Tahoe and Reno, for Bandwidths 5, 10 and 20

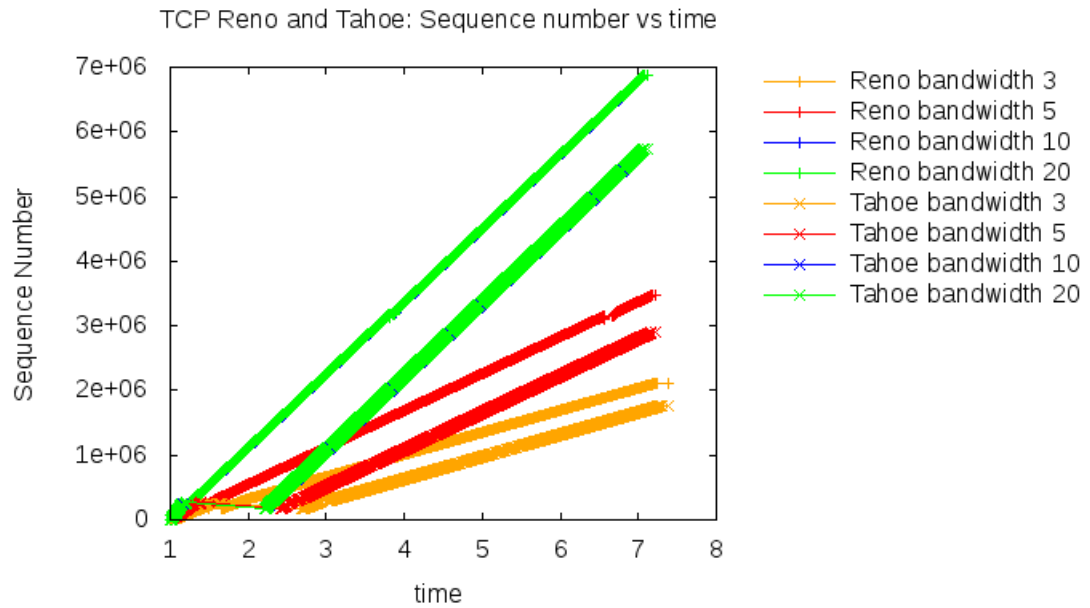


Figure 5.8: Graph of Sequence Number vs Time for TCP Tahoe and Reno, for Bandwidths 3, 5, 10 and 20

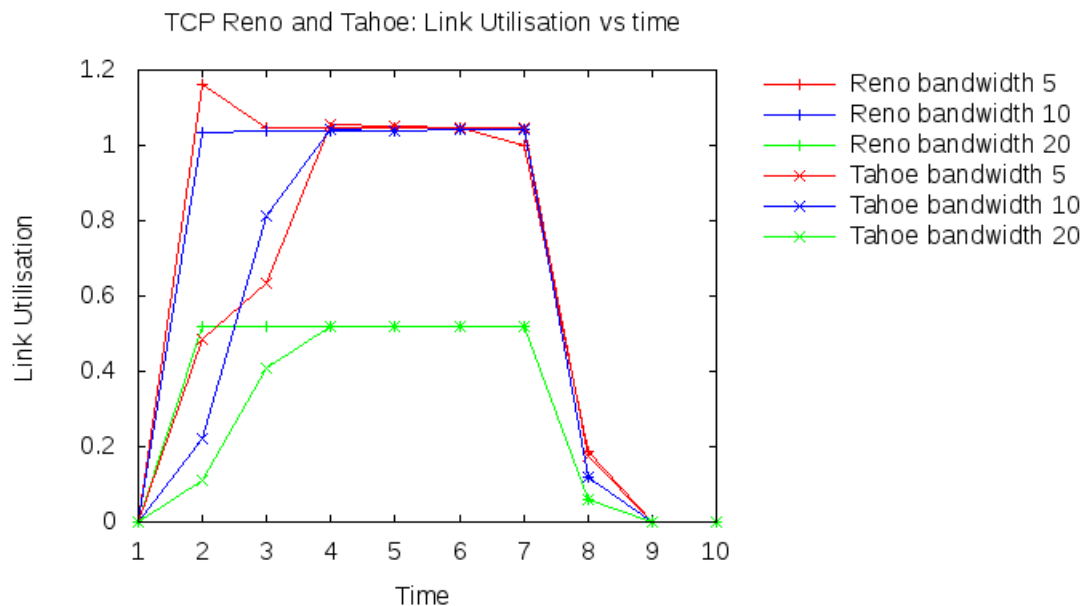


Figure 5.9: Graph of Link Utilization vs Time for TCP Tahoe and Reno, for Bandwidths 5, 10 and 20

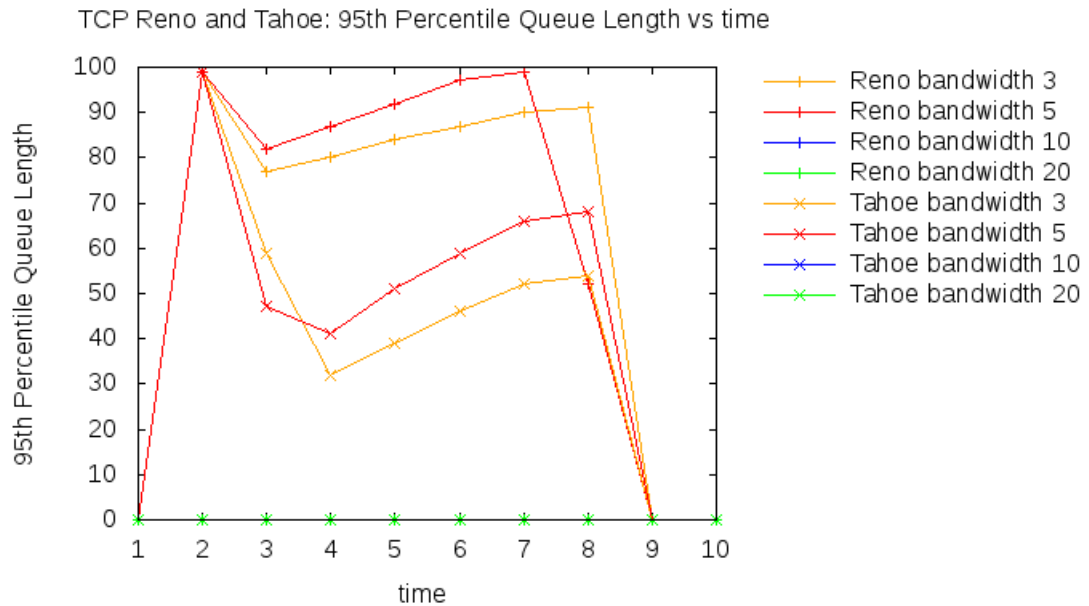


Figure 5.10: Graph of 95th Percentile Queue Length vs Time for TCP Tahoe and Reno, for Bandwidths 3, 5, 10 and 20

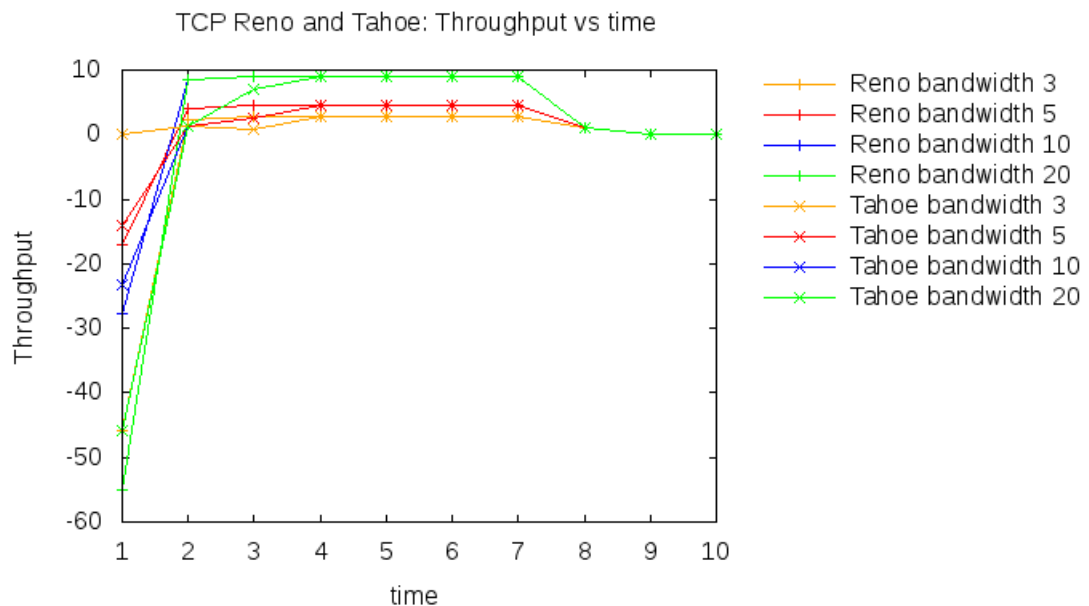


Figure 5.11: Graph of Throughput vs Time for TCP Tahoe and Reno, for Bandwidths 5, 10 and 20

CONCLUSION

In this project, we successfully completed the utilities required to port the TCP Evaluation Suite to NS3. Tmix and DelayBox were ported to NS 3.24 by making necessary changes. The ported modules also work seamlessly with the latest version, NS 3.25. In addition, a test suite consisting of three test cases and example scenarios demonstrating the usage of Tmix and DelayBox were written. Also, create graph and evalStats modules helped in collecting various stats from the bottleneck routers and some nodes. These stats were later used to plot graphs using gnuplot.

In Conclusion,

1. Tmix and DelayBox output were successfully tested against the output of previous versions and found to give the correct result.
2. Example Scenario showcasing the usage of Tmix and DelayBox to produce traffic and a test suite to verify the results were coded.
3. A code to generate different types of traffic, including Tmix traffic on a dumbbell topology was written.
4. Modules named evalStats and create graph to collect various statistics and run the simulation by varying various parameters and selecting TCP variants and plotting graphs for the same.

6.1 PROBLEMS FACED

1. Tmix and DelayBox used methods that belonged to a class that was depreciated in the latest version of NS3. As a result, they had to be patched with the latest version by using the newer APIs provided in the latest version.
2. In evalStats module, the stats for the congestion window have to be obtained from the socket of the source node. However, it is not possible to obtain a reference to the socket at compile time, since the socket is generated only after the application starts. To circumvent this, when the application starts, we use a callback to hook the socket to the required trace source at the run time.
3. HTTP Application module is not present in NS3. Instead, we have to create a subclass of the Application class and implement it.

SCOPE FOR FUTURE WORK

Now that the utilities required to port TCP Evaluation Suite to NS3 are completed, the next step is to enhance it by modifying it to work with other topologies of interest, for instance parking lot topology. Doing so would give a better picture of how the algorithms would behave in a real life scenario.

The work could also be extended to include generation of some more kinds of traffic, such as HTTP. NS-3 currently does not have an application module for HTTP. In addition, the evalStats module could be extended to collect more statistics, such as Sending rate, Goodput, Cumulative loss, queuing delay, SRTT, fairness of flows etc.

Ultimately, we hope that the project is extended to include more widely accepted and well-defined benchmark configurations that encompass all possible real-life scenarios.

References

- [1] Michele C. Weigle, Prashanth Adurthi, Flix Hernandez-Campos, Kevin Jeffay, F. Donelson Smith. Tmix: A Tool for Generating Realistic TCP Application Workloads in ns-2. ACM SIGCOMM Computer Communication Review, Volume 36, Number 3, July 2006.
- [2] Common TCP Evaluation Suite RFC, July 6, 2008 from <https://tools.ietf.org/html/draft-irtf-tmrg-tests-00>
- [3] TCP Evaluation Suite, UCLA, <http://nrlweb.cs.ucla.edu/tcpsuite/>
- [4] Tmix: Internet Traffic Generation, <http://www.isi.edu/nsnam/ns/doc/node563.html>
- [5] Tmix code, Github, <https://github.com/weiglemc/tmix-ns2>
- [6] A Short Tutorial for Using Tmix Traffic Generator in ns-2, Rebecca Lovewell, <http://cs.unc.edu/lovewell/research/tutorial/tmix/index.html>
- [7] NS3 Tutorial, nsnam, <https://www.nsnam.org/docs/release/3.25/tutorial/ns-3-tutorial.pdf>
- [8] An NS2 TCP Evaluation Tool: Installation Guide and Tutorial; Gang Wang, Yong Xia, David Harrison and TMRG RFC Common TCP Evaluation Suite; L. Andrew, S. Floyd
- [9] <http://research-jai.blogspot.in/2012/06/what-is-ns3.html>