Common TCP Evaluation Suite for ns-3: Design, Implementation and Open Issues

Kriti Nagori, Meenakshy Balachandran, Ankit Deepak, Mohit P. Tahiliani, B. R. Chandavarkar
Wireless Information Networking Group (WiNG)
National Institute of Technology Karnataka, Surathkal
Mangalore, India, 575025
kriti123.nagori@gmail.com, bmeenakshy@gmail.com, adadeepak8@gmail.com, tahiliani@nitk.ac.in

ABSTRACT

This paper presents the design and implementation of a Common TCP Evaluation Suite for ns-3. The proposed evaluation suite uses Tmix to generate realistic synthetic TCP traffic, and is designed in line with the recommendations from Internet Congestion Control Research Group (ICCRG). We discuss the Tmix integration in ns-3, shuffling connection vectors of the real traces, architecture and validation of the proposed evaluation suite. The correctness of the evaluation suite is verified by comparing the results obtained from it to those from an existing implementation of such a suite in *ns*-2. Several open issues discovered with Tmix are also discussed in the paper.

CCS CONCEPTS

•Networks →Transport protocols; Network simulations; •Computing methodologies →*Simulation environments;*

KEYWORDS

ns-3, TCP Evaluation, Tmix

ACM Reference format:
Kriti Nagori, Meenakshy Balachandran, Ankit Deepak, Mohit P. Tahiliani, B. R. Chandavarkar. 2017. Common TCP Evaluation Suite for ns-3: Design, Implementation and Open Issues. In *Proceedings of the 2017 Workshop on ns-3, Porto, Portugal, June 2017 (WNS3 2017),* 8 pages.
DOI: http://dx.doi.org/10.1145/3067665.3067676

1 INTRODUCTION

Over the past few years, extensive work has been done to enhance the TCP performance, and as a result, many different extensions of it have been developed to cater to the changing needs of the Internet applications. Evaluating these extensions and comparing them with others is troublesome, one of the reasons being the absence of a standard set of performance metrics to evaluate them. Also, some TCP extensions cater to the needs of certain scenarios only, like TCP Westwood [2], which is aimed to improve TCP performance in wireless networks. Hence, it is crucial that researchers are able to emphasise on the exact scenarios in which their TCP works better than the others. To standardise the scenarios and enable researchers to gain quick insights into the working of TCP extensions, ICCRG has proposed a common test suite [6]. The aim of this suite is not to exhaustively evaluate a new TCP extension, but to help the researchers to easily and quickly derive initial results of their work.

In this paper, we propose an implementation of the test suite proposed by ICCRG for ns-3. The paper describes the architecture of the proposed suite with a major focus on the integration of Tmix traffic generator [11] with latest ns-3 release, shuffling the real Internet traces as recommended in the ICCRG draft, automating the simulation setup, execution and results collection. We have selected four experiments from ICCRG draft and implemented them in our proposed suite, namely *access link*, *dial up*, *transoceanic* and the experiment related to *delay throughput tradeoff*. The original implementation of Tmix for ns-3 [4] takes care of the topology setup inherently. However, only one type of topology design is currently possible with Tmix, namely point-to-point dumbbell. Setting up other topologies such as wireless dumbbell or parking lot is not supported in Tmix, and extending it is beyond the scope of this paper. This is the primary reason behind selecting only four experiments for implementation in the proposed test suite.

The paper is organised as follows: Section 2 reviews the prior work done towards the implementation of a TCP evaluation suite in ns-2 and ns-3. Section 3 describes the architecture of the test suite we built for ns-3, along with its integration with Tmix. Section 4 presents the results of the various experiments run using the proposed test suite, followed by Section 5 which contains a preliminary validation of our test suite by comparing its results to those obtained from ns-2 evaluation suite. The open issues identified during the process of building the evaluation suite for ns-3 are discussed in Section 6. Lastly, Section 7 concludes the paper and provides directions for extending the proposed evaluation suite.

2 RELATED WORK

There has been a lot of work done towards designing a suite for evaluating the TCP extensions, but mainly for ns-2, like the proposals by Shimonishi et al. [10] and Li et al. [8]. However, the Internet draft by ICCRG [6] contains a set of experiments suitable for a wide range of TCP variants, unlike [10] and [8] which focus only on tests to evaluate high speed TCP extensions. The suite developed in this paper aims to follow the ns-2 TCP evaluation suite proposed in [5], which has been developed by the same authors who prepared the ICCRG draft. Our aim is to create an evaluation suite in ns-3 which equals the one developed for ns-2, thereby providing the researchers with all the useful features of the draft in addition to the benefits of using recently developed models in ns-3, such as the traffic control layer.

WNS3 2017, June 2017, Porto, Portugal
© 2017 ACM. 978-1-4503-5219-2/17/06…$15.00
DOI: http://dx.doi.org/10.1145/3067665.3067676

Dharmendra et al. [9] implemented a TCP evaluation suite in ns-3 which mainly provides features described in [1] and some of those listed in [6]. However, this suite uses ns-3 traffic for evaluation, unlike the realistic traffic generated by Tmix which is recommended in ICCRG draft. An alternative method to evaluate TCP extensions is to run the simulations manually by using the default TCP implementations and traffic generators provided in ns-3. However, this process is tedious and the proposed evaluation suite simplifies it. It currently works only with the realistic traffic generated by Tmix and does not support ns-3 traffic due to some limitations, more details of which are explained in Section 5.

3 DESIGN AND IMPLEMENTATION

This section describes the integration of Tmix with ns-3, the additional features which have been implemented in Tmix to meet the requirements of the proposed suite, and finally, the architecture of the proposed suite in ns-3.

3.1 Tmix

To accurately analyse the nature of congestion control protocols in actual networks, the Internet draft mentions that it is essential to have a fairly realistic traffic generated for simulation. Tmix is one such traffic generator. It uses connection vectors to represent each TCP connection. Every connection vector is composed of a sequence of triples (request-size, response-size, think-time) representing bidirectional traffic. These connection vectors are retrieved from Internet traffic traces such as the ones provided in [6]. To generate steady flows, the process is as follows: trace files must be shuffled first, then the simulation topology must be created, and lastly, each connection vector from the trace file must be parsed to generate the traffic. The following sections describe how these functionalities have been implemented in our suite.

3.1.1 Shuffling of Traces: Ideally, it is preferred that the experiments are run until some sort of equilibrium results can be obtained. In order to achieve this, the trace files must be shuffled to remove non-stationary load. This also ensures that the estimate of traffic remains constant throughout the execution of the experiment.

In the proposed suite, a class named `TmixShuffle` has been implemented for shuffling the Internet trace files provided in [6]. Initially, the Tmix traces are divided into bins of different sizes based on the scenario and the load, and then each individual bin is shuffled by using Fisher Yates Shuffling algorithm [13]. Every scenario specifies parameters like the target load, test time, warm up time, prefill time, and scale which are required to generate the shuffled trace files. Following are the main methods implemented in `TmixShuffle`:

- **ShuffleTraces:** Creates the bins and shuffles the connection vectors based on the parameters listed in each scenario.
- **FisherYatesShuffle:** This method implements the Fisher Yates algorithm for shuffling the connection vectors within a bin.
- **AddBurstStats:** Calculates the burst duration of the data.
- **ProcessBurst:** Calculates the burst statistics of the connection vectors.

3.1.2 Creating the Topologies: Every scenario specifies the parameters to configure a dumbbell topology for the simulation. These parameters are set using the `TmixTopologyParameters` class in our suite. The topology parameters that are set include:

- Edge link delay
- Bottleneck delay
- Edge bandwidth
- Bottleneck bandwidth
- End device queue limit
- Router queue limit

Once the configuration parameters are obtained, a dumbbell topology is created by a class named `TmixTopology` which is implemented in the Tmix module. Although this class belongs to the Tmix module, we have largely modified its implementation and shifted it to our suite. The methods that set up the topology are:

- **NodeTypeByAddress:** this method finds a node from its IP address.
- **AssignNodes:** this method creates the left and the right side nodes of the dumbbell topology.
- **NewPair:** it uses the nodes created in `AssignNodes` and creates a new sender/receiver pair, with initiator and acceptor applications installed on them.
- **ConnectNodeToRouter:** this method is invoked from `NewPair`. It creates a channel between each `NetDevice` and the router, setting its properties to the ones obtained from `TmixTopologyParameters` class. Furthermore, queues for these channels are installed on the respective edge nodes and the routers in this method.
- **ConnectRouter:** this method installs the queues on the routers which are constantly monitored for the purpose of evaluation. The trace sources installed on these queues are used to extract the evaluation metrics of interest.

After the topology is created, Tmix initiator and acceptor applications are installed on every sender/receiver pair using the `TmixHelper` class, and the connection vectors are added to these applications to send the data.

3.1.3 Traffic Generation: Traffic generation requires converting the felix connection vectors [7] obtained from the shuffled trace files to a ns-3 parsable connection vector format. The converted vectors are then parsed and traffic is generated in ns-3 from it. This functionality is provided by the Tmix module in the `Tmix` class.

3.2 Architecture of the Module

Like every other module in ns-3, our suite is placed in a new directory called `common-tcp-eval-suite` inside `src` which contains `models`, `helpers`, and `examples`. The simulation output is stored in `tcp-eval-output` which is a directory at the same level as `src`. Figure 1 describes the structure of the `common-tcp-eval-suite` module for ns-3.

3.2.1 Model: This directory contains the code which is common to all scenarios like shuffling of traces, topology creation, calculating performance metrics, and collecting results.

3.2.2 Helper: This consists of the helper classes that assist in the creation of the topology and generation of the Tmix traffic.

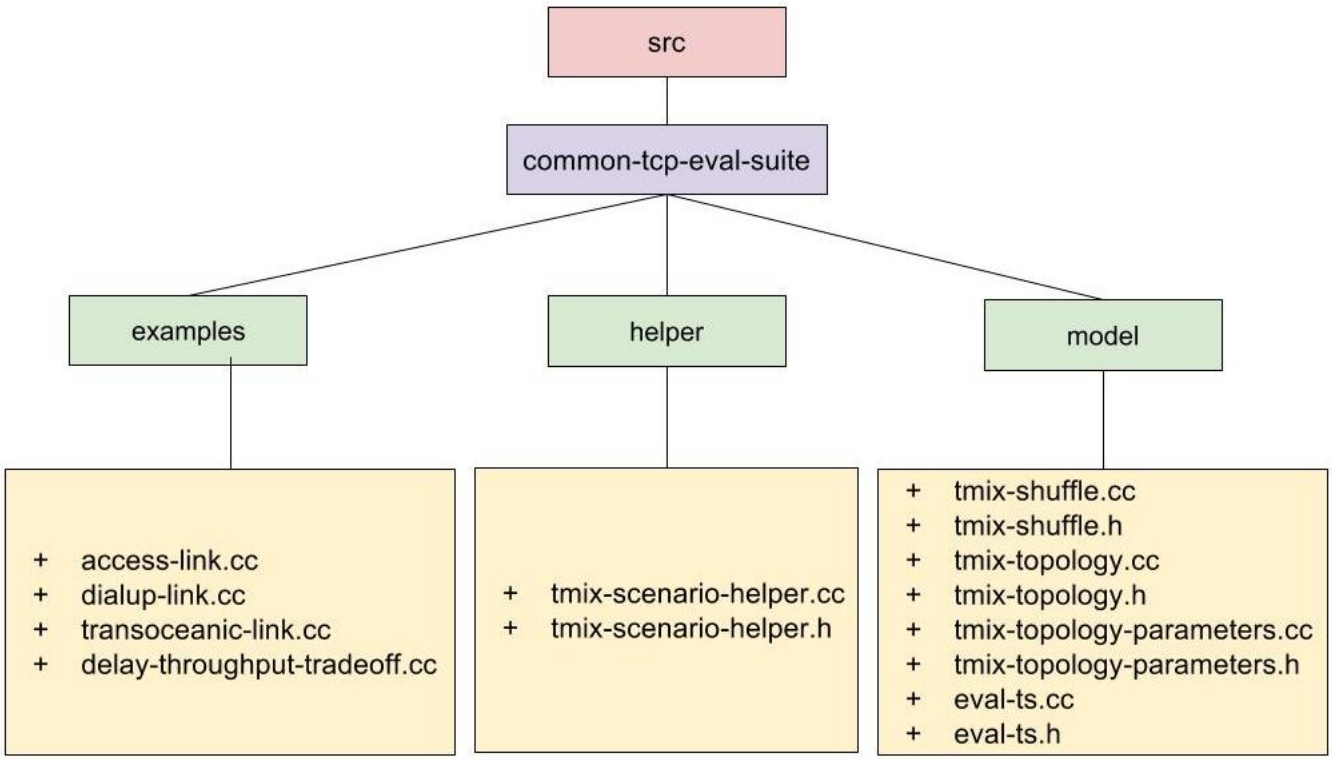


Figure 1: Architecture of the Common TCP Evaluation Suite for ns-3

The `tmix-scenario-helper {.h, .cc}` contain functions that set up the topology parameters and the traffic parameters which are passed to the `TmixTopology` class. They also facilitate running the scenario once the parameters are set. The services provided by each method are listed below:

(1) **SetExptParameters:** this method sets the parameters for the topology such as bandwidth, link delay, and number of node pairs.
(2) **SetTmixParameters:** the parameters required for shuffling the trace files are set here, such as prefill time, maximum segment size, load tolerance, scale and warm up time etc. It also calculates the total simulation time from these values.
(3) **AddCvecsToPairs:** this converts the shuffled traces to a ns-3 parsable connection vector format, so that they can be parsed by the nodes. The flows are then defined and the connection vectors extracted from the files are installed into the nodes.
(4) **RunScenario:** this method is invoked to run the scenario after setting the required parameters. It first calls the method `ShuffleTraces()` which takes the Tmix parameters as arguments and creates the shuffled traces. This scenario is then run for the TCP extensions specified by the user. Subsequently, the `addCvecsToPairs()` method is called to install the traces on the nodes. Lastly, the simulator is run for the specified simulation time computed in the `SetExptParameters()` method.
(5) **DestroyTrace:** it invokes `destroyConnection()` which cleans the trace files installed on the bottleneck routers to collect the metrics.

Based on the descriptions of different classes provided above, Figure 2 presents the class diagram of the proposed suite.

3.2.3 Examples. All the experiments implemented in this suite are placed here. The parameters specific to each experiment are set and passed to the functions of the helper class as they are invoked. The results for these examples are collected in the `tcp-eval-output` folder in the same level as `src` directory. The output is collected in separate folders for every experiment. As an example, the interaction of the user with our module for simulating the AccessLink experiment is shown in Figure 3.

3.3 Evaluation of Metrics

As recommended in the draft, we have considered three performance metrics for evaluation: average throughput, average queueing delay, and the average packet drop rate. These metrics are calculated on the bottleneck link in both directions.

To extract these values, trace sources are installed on `QueueDisc` and `NetDevice` of the routers. Each time a packet is enqueued, dequeued or received by the device, the method which calculates the

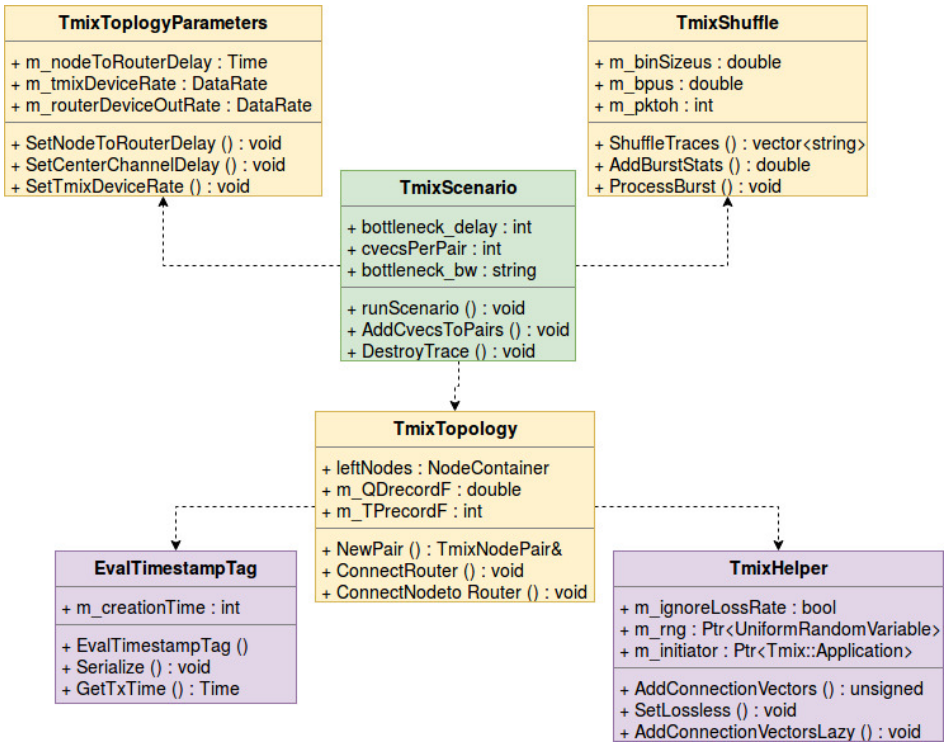


Figure 2: Class diagram of the Common TCP Evaluation Suite for ns-3

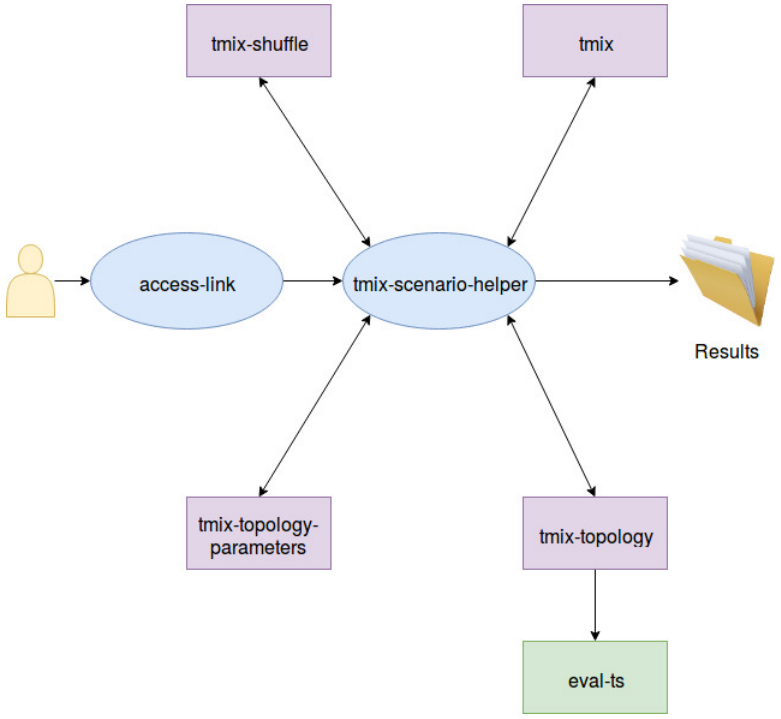


Figure 3: User interaction diagram of the Common TCP Evaluation Suite for ns-3

performance metrics is invoked. The assignment of the timestamp to the packets as they enqueue and dequeue is implemented in the `eval-ts {.h, .cc}` in the `model` subdirectory. `eval-ts` is a sub class of packet tag, and is used to record the enqueue time of packet on the `QueueDisc`. On dequeue, this timestamp is used to calculate the queue delay.

Methods to calculate the performance metrics are implemented in `tmix-topology.cc` in the `model` subdirectory of the our suite. Following is the list:

(1) **PacketEnqueue:** this method is invoked when a packet is enqueued. It adds a timestamp to the packet as a tag.
(2) **PacketDequeue:** this method is invoked when the packet is dequeued. It removes the timestamp and subtracts it from the current time, which is the dequeue time of the packet. This difference between the dequeue and enqueue time gives the queue delay of the packet. The queue delay values are collected at an interval of 10 milliseconds and the average queue delay for that interval is logged into the output file. The total queue delay and number of records are also maintained.
(3) **PacketSize:** this method is invoked when a packet arrives at the router netdevice. The packet sizes are recorded for a minimum of 10 milliseconds and the average of this over the time range is printed into the throughput file. The sum of all the throughput values and a total number of records printed are maintained. Also, average of the total packets dropped in the same interval are printed in a separate file.

Lastly, the average values of packet drop, throughput, and queue delay are stored in separate files and the method to calculate summarise results is invoked at the end of the simulation.

4 RESULTS

In this section, we compare the performance of 8 TCP extensions in AccessLink scenario by using the proposed suite. Table 1 shows the simulation setup for the same. The values selected for every parameter are based on the recommendations in the draft [6].

Table 1: Simulation Parameters

Topology Parameters	Values
Bottleneck Bandwidth	100 Mbps
Bottleneck Delay	2 ms
Edge Bandwidth	100 Mbps
Left side Edge Delays	0ms, 12ms and 25ms
Right side Edge Delays	2ms, 37ms, and 75ms
Bottleneck Buffer Size	100 ms (850 packets)

The draft [6] suggests to run each scenario with three different loads; 60%, 85%, and 110%. However, due to space constraints, only the results of 60% load are presented here. Moreover, the results are discussed only for a few selected TCP extensions which includes one each from standard TCPs, high speed TCPs, delay based TCPs, and TCPs for wireless networks. The results for other experiments can be generated using our code which is openly available at [3].

The forward traffic consists of request messages sent from initiator to acceptor, and the reverse traffic from acceptor contains responses to those requests. The acceptor is analogous to a server sending bulk of traffic. The graphs shown in Figure 4 through Figure 7 present the instantaneous throughput and queue delay for the traffic sent from the acceptor. All simulations are run for 600 seconds.

A sharp decrease in queue length and throughput is seen in all the graphs towards the end. It can be observed that the throughput is close to the link capacity for most of the simulation duration, but after this the throughput stoops as the traffic load is too much for the network to handle and as a consequence, the network collapses. This is an issue with Tmix traces as they breaks the equilibrium state of the experiment.

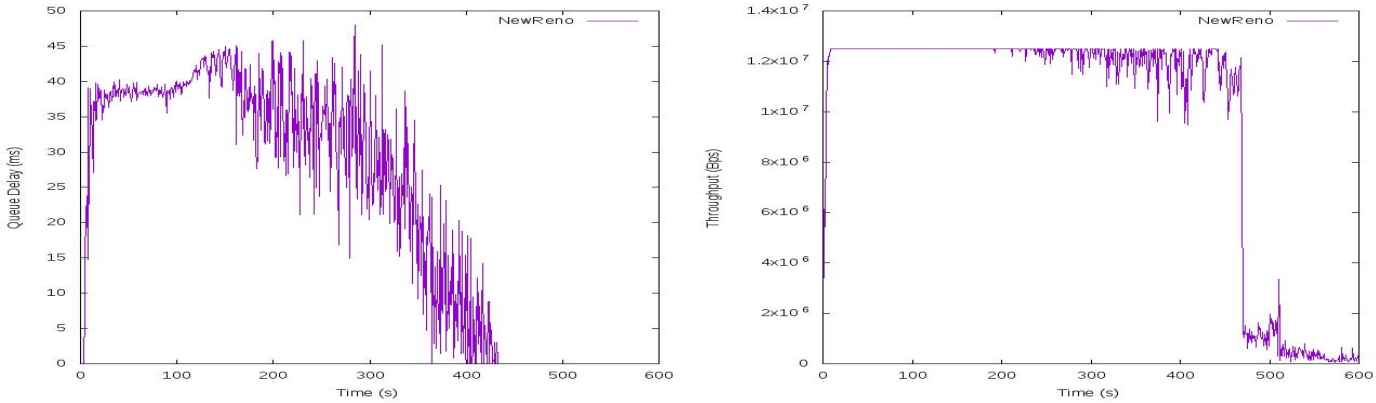
In order to be able to compare all the TCP extensions together, we have generated a plot with throughput on the Y-axis and Queue Delay on the X-axis. As shown in Figure 8, the comparison between them is depicted as an ellipse in such a way that the covariance between the queuing delay and throughput will be determined by the orientation of the ellipse. This can help us to analyze the effect of traffic load on throughput and queuing delay. The graph is generated using the following two scripts:

- `ellipsemaker`: this script was obtained from the TCP Ex Machina project [12] to generate covariance error ellipse between queuing delay and throughput.
- `generate-ellipsepoint`: the value of throughput for every recorded value of queue delay is found using this script.

Figure 4 and Figure 6 show the variation in throughput and queue delay for TCP NewReno and TCP Vegas respectively. It is evident that TCP Vegas reduces the queue delay sooner than TCP NewReno and hence, leads to lesser variations in queue delay. TCP NewReno infers congestion based on packet drops whereas TCP Vegas infers it based on the increase in the RTT. Hence, TCP Vegas doesn't wait for the packet to drop, and reduces the congestion window (*cwnd*) earlier to control congestion. However, this results in TCP Vegas having lower throughput as compared to TCP NewReno.

Similarly, Figure 8 depicts that TCP Westwood performs worse in comparison to all other TCP extensions. TCP Westwood is conservative in nature mainly because it is tailored for wireless networks which are characterised by high noise and packet drop rate. It relies on bandwidth estimation to control congestion. Bandwidth is estimated by mining the acknowledgement packets. Moreover, Westwood treats packet drops differently and does not respond to every packet drop by reducing the congestion window.

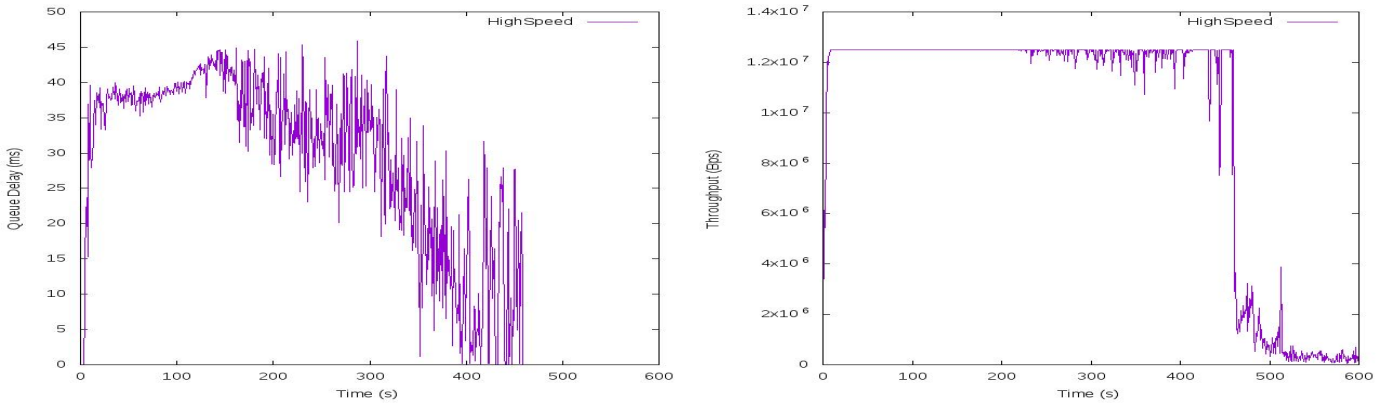
TCP HighSpeed (Figure 5) with its improved window calculations, results in faster window growth as compared to any other TCP and also recovers from losses more quickly. These features are very useful in networks with high bandwidth like the TransOceanic scenario. However, in case of scenarios like AccessLink, its performance will be similar to TCP NewReno as shown in the graph. The range of its queue delay values is lesser compared to TCP NewReno due to the improvements in the congestion window calculation.



(a) Queue Delay for TCP NewReno

(b) Throughput for TCP NewReno

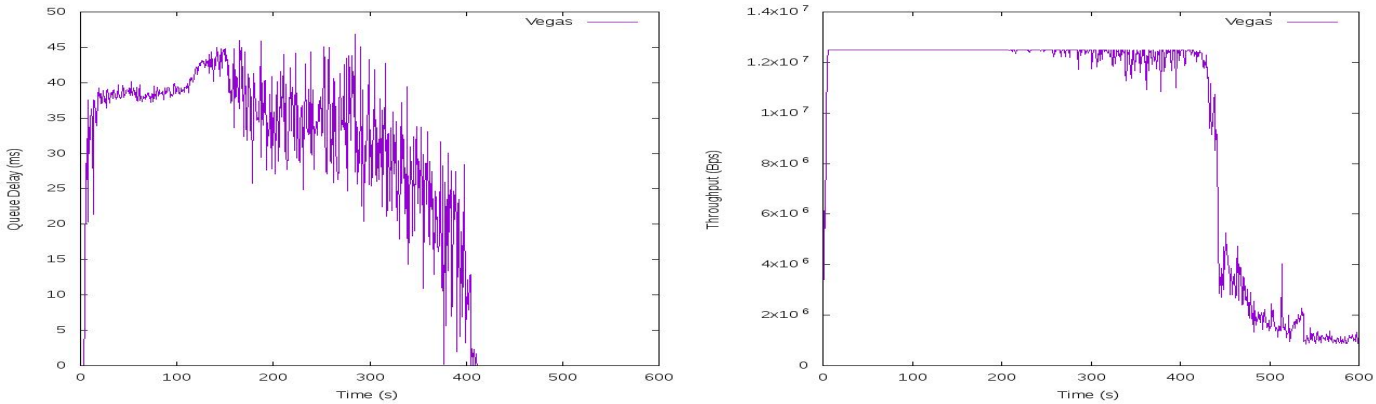
Figure 4: Throughput and Queue Delay with TCP NewReno



(a) Queue delay for TCP HighSpeed

(b) Throughput for TCP HighSpeed

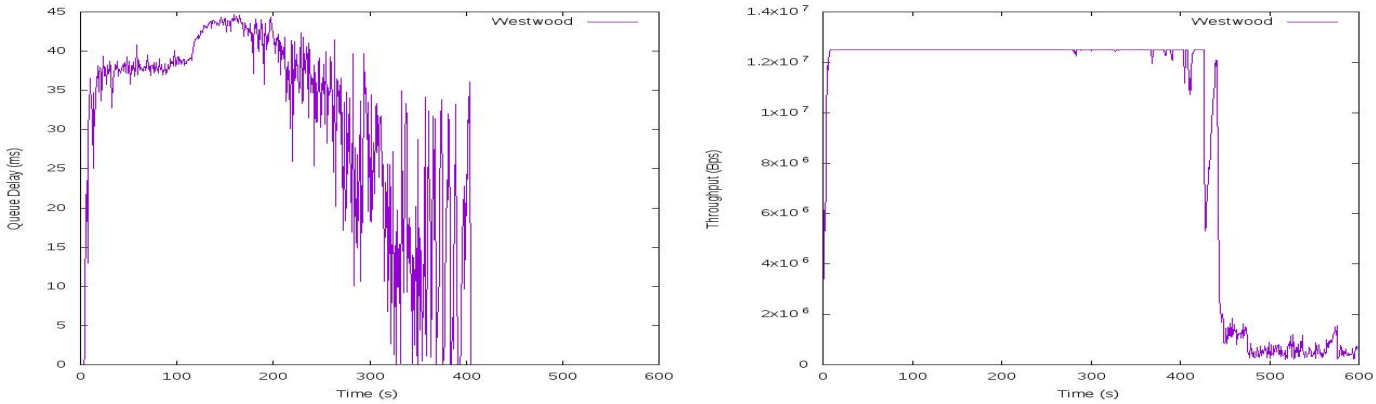
Figure 5: Throughput and Queue Delay with TCP HighSpeed



(a) Queue Delay for TCP Vegas

(b) Throughput for TCP Vegas

Figure 6: Throughput and Queue Delay with TCP Vegas



(a) Queue Delay for TCP Westwood

(b) Throughput for TCP Westwood

Figure 7: Throughput and Queue Delay with TCP Westwood

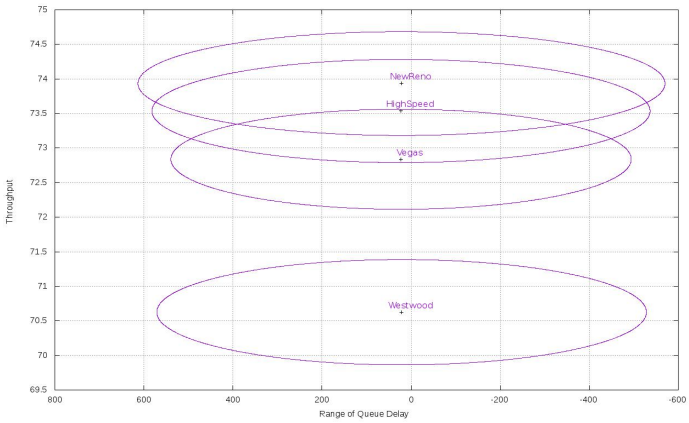


Figure 8: Combined TCP Comparison Graph

5 PRELIMINARY VALIDATION OF THE TEST SUITE

In order to verify the correctness of our suite, we chose to compare the results obtained from our evaluation suite in ns-3 to those obtained from ns-2 evaluation suite. Comparing our results to those obtained by using ns-3 traffic generators is not feasible, as the traffic composition obtained from Tmix depends on random number generators and cannot be reproduced by using the ns-3 traffic generators.

We chose the AccessLink experiment for this purpose as it is the most commonly studied scenario for TCP experimentation. Other experiments which are commonly implemented in ns-3 and ns-2 suites can also be used for validation. Our code has been made openly available at [3] along with the Internet trace files used for generating results in this section. Table 2 shows the results obtained from both evaluation suites.

Except the average queue delay for reverse traffic (sender to receiver), all the other results obtained from ns-3 evaluation suite match with those obtained from ns-2 suite. While trying to figure out the reason for this difference, it was noted that the topology creation in Tmix is flawed because it creates net devices depending on the number of sender/receiver pairs. For example, if a same sender is paired with three receivers, Tmix creates three separate net devices at the sender node, each one configured to operate at a particular data rate. Thus, if the intended data rate between each sender and router was 100Mbps, it becomes 300Mbps now because each netdevice can send packets at a rate of 100Mbps individually. Due to this behavior, the routers in ns-3 suite were far more congested than the ones in ns-2 suite, which attributed to larger queueing delay in ns-3. As a temporary hack, we tried manipulating the edge link bandwidth such that the aggregate bandwidth of all net devices on each sender turned out to be 100Mbps and compared our results again with those of ns-2. We found that the ns-3 results matched with those of ns-2 for all metrics. However, we have not included that hack in our code released at [3], as it is a critical issue and should be resolved more systematically.

Table 2: Results obtained from ns-3 and ns-2 Evaluation Suites

Evaluation metric	ns-3	ns-2
Forward Queue Delay (in ms)	0	0.023
Forward Throughput (in MB/sec)	2.011	2.28
Forward Packet Drop Rate	0	0
Reverse Queue Delay (in ms)	22.4	0.7
Reverse Throughput (in MB/sec)	8.70	8.70
Reverse Packet Drop Rate	0	0

6 OPEN ISSUES

We highlight some open issues encountered during the design and implementation of this suite in ns-3:

- This evaluation suite requires significant restructuring of the Tmix module: we had to modify the `tmix-topology {.h, .cc}` to enable the creation of a dumbbell topology with different edge delays. Also, modifications were made

to constrain the number of nodes on each side of the dumbbell, as the original Tmix code created two nodes for each initiator/acceptor pair whenever a link is created. However, as mentioned in Section 5, new net devices are still being created for each initiator-acceptor pair. This creates multiple channels from an edge node to a router node, which drastically increases the incoming bandwidth. The increase in the number of net devices and channels connected to a router is around twice the factorial of the total number of nodes connected to that router. This is still a limitation on the topology that is created and must be rectified.

- Tmix should be extended to support various other topologies like wireless dumbbell and parking lot: currently, for each connection, a point-to-point link is created between two nodes. Hence, additional support for different types of links (e.g., CSMA, WiFi, etc) must be added to the existing code to enable the usage of this suite for different network experiments. The current implementation of Tmix topology is tailor made for basic dumbbell topology and requires considerable amount of code to be added to support a complex topology like parking-lot.
- Additional modules (e.g., Explicit Congestion Notification) are required in ns-3 to support the scenarios like Data Center Networks.
- Simulating a single scenario using this suite requires long amount of time on a personal desktop configuration, ranging from days to weeks. The MPI support in ns-3 could be leveraged to reduce this time and improve the efficiency of the suite.
- For attaining the equilibrium in the experiment, the traces are shuffled such that the traffic offered in the second third of the simulation is equal to the final third. However, due to network collapse, the throughput and the queue delay reduce sharply and the traffic becomes highly unstable as evident from the graphs in Section 4.

7 CONCLUSION AND FUTURE WORK

In this paper, we have proposed a Common TCP evaluation suite for ns-3 which allows the researchers to plug in their proposed TCP extension and evaluate its performance in the presence of realistic Internet traffic. The proposed test suite can be used to compare the performance of the existing TCP variants, or to gain an in-depth understanding of the working of a particular TCP. We look forward to add more experiments to our test suite which were difficult to implement due to significant changes required in the implementation of Tmix and unavailability of a few modules in ns-3, such as the support for satellite network simulations. Finally, we aim to validate our proposed suite thoroughly by comparing the results obtained from it to those obtained from ns-2 evaluation suite.

ACKNOWLEDGEMENTS

We would like to thank Tom Henderson for helpful feedback on restructuring the previous TCP evaluation suite, and Vrushab Maitri, Vishal Rathod and Sachin D. Patil for their valuable contributions. We would also like to thank Jeny Rajan and Shashidhar Koolagudi for providing the necessary infrastructure to generate results for this project, and the entire ns-3 development team for their excellent guidance and support.

REFERENCES

[1] L. Andrew and S. Floyd. 2010. *Common TCP Evaluation Suite.* Internet-Draft draft-irtf-tmrg-tests-02. Internet Engineering Task Force. https://tools.ietf.org/html/draft-irtf-tmrg-tests-02 Work in Progress.

[2] C. Casetti, M. Gerla, S. Mascolo, M. Y Sanadidi, and R. Wang. 2002. TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks. *Wireless Networks* 8, 5 (2002), 467–479.

[3] Common TCP Evaluation Suite. 2017. https://github.com/mohittahiliani/common-tcp-eval-suite. (2017).

[4] Tmix for Network Simulator 3. 2015. https://github.com/weiglemc/tmix-ns3. (2015).

[5] D. Hayes, D. Ros, L. Andrew, and S. Floyd. 2014. Common TCP Evaluation Suite. https://bitbucket.org/hayesd. (July 2014).

[6] D. Hayes, D. Ros, L. L. H. Andrew, and S. Floyd. 2015. *Common TCP Evaluation Suite.* Internet-Draft draft-irtf-iccrg-tcpeval-01. Internet Engineering Task Force. https://tools.ietf.org/html/draft-irtf-iccrg-tcpeval-01 Work in Progress.

[7] F. Hernandez-Campos. 2006. *Generation and Validation of Empirically-Derived TCP Application Workloads.* Ph.D. Dissertation. Citeseer.

[8] Y. Li, D. Leith, and R. N. Shorten. 2007. Hamilton Institute TCP Evaluation Suite. http://www.hamilton.ie/net/eval/hi2005.htm. (2007).

[9] D. Kumar Mishra, P. Vankar, and M. P. Tahiliani. 2016. TCP Evaluation Suite for ns-3. In *Proceedings of the Workshop on ns-3.* ACM, 25–32.

[10] H. Shimonishi, M. Y. Sanadidi, M. Gerla, C. Marcondes, and P. Vasu. 2007. TCP Evaluation Suite. http://nrlweb.cs.ucla.edu/tcpsuite/index.html. (2007). Evaluating New Congestion Control Schemes and Its Impact on Standard TCP NewReno.

[11] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. D. Smith. 2006. Tmix: A Tool for Generating Realistic TCP Application Workloads in ns-2. *ACM SIGCOMM Computer Communication Review* 36, 3 (2006), 65–76.

[12] K. Winstein and H. Balakrishnan. 2013. TCP ex machina: Computer-Generated Congestion Control. In *ACM SIGCOMM Computer Communication Review*, Vol. 43. ACM, 123–134.

[13] Fisher Yates Shuffling Algorithm. 2015. http://en.wikipedia.org/wiki/Fisher-Yates_shuffle. (2015).