Design and Implementation of AQM Evaluation Suite for ns-3

Ankit Deepak, Shravya K. S. and Mohit P. Tahiliani
Wireless Information Networking Group (WiNG)
NITK Surathkal, Mangalore, India, 575025
adadeepak8@gmail.com, shravya.ks0@gmail.com, tahiliani@nitk.ac.in

ABSTRACT

Excessive buffering in network devices should be avoided because it leads to a series of performance issues such as high queuing latency and variations in delay. Active Queue Management (AQM) algorithms play a vital role in monitoring and controlling the queue length in these devices. Recently there has been a significant progress in the design and development of new AQM algorithms. However, thoroughly evaluating the performance of AQM algorithms is a nontrivial task. In an effort to simplify this, the Active Queue Management and Packet Scheduling Working Group at IETF have proposed informational guidelines in RFC 7928 to test the applicability, performance and deployment complexity of AQM algorithms. This paper presents the design and implementation of an AQM evaluation framework for ns-3 which helps to quickly study the performance of AQM algorithms based on the guidelines mentioned in RFC 7928. The proposed framework automates simulation setup, topology creation, traffic generation, program execution, results collection and their graphical representation using ns-3, based on the scenarios mentioned in the RFC.

CCS CONCEPTS

•Networks →Network Algorithms; •Control Path Algorithms →Traffic engineering algorithms; •Network Performance Evaluation →Network Simulations;

KEYWORDS

ns-3, Active Queue Management, Performance Evaluation

ACM Reference format:
Ankit Deepak, Shravya K. S. and Mohit P. Tahiliani. 2017. Design and Implementation of AQM Evaluation Suite for ns-3. In *Proceedings of the 2017 Workshop on ns-3, Porto, Portugal, June 2017 (WNS3 2017),* 8 pages.
DOI: http://dx.doi.org/10.1145/3067665.3067674

1 INTRODUCTION

Active Queue Management (AQM) has been an active area of research since past two decades, and continues to evolve as new algorithms are being designed and tested for deployment in the Internet. Several AQM algorithms have emerged recently, which include: Controlled Delay (CoDel) [7], Proportional Integral controller Enhanced (PIE) [8] [13], Curvy Random Early Detection (Curvy RED) [11], PI^2 [1], etc. Although AQM algorithms have been proven to be robust in some selected scenarios, verifying their suitability in a wide variety of Internet environments is a challenging and time consuming task. In an effort to simplify the process of screening AQM algorithms, RFC 7928 [6] explicitly provides a set of benchmark scenarios for testing and performance metrics of interest. These guidelines assist the research community to systematically compare the AQM algorithms and choose the promising ones for further evaluation.

Recently, a Linux-like traffic control subsystem has been added to ns-3 [4] wherein the AQM algorithms are implemented as queuing disciplines. A simulator with such a realistic architecture is essential to study the behavioral aspects of AQM algorithms before they are considered to be safe for deployment in the Internet. In an ongoing effort to enhance the utility of ns-3, we implement an AQM evaluation framework which closely follows the guidelines provided in RFC 7928. This paper presents the design and implementation of the proposed evaluation suite in ns-3. It automates the work cycle from setting up network simulation environment to presenting the results in graphical formats.

The outline of rest of this paper is as follows: Section 2 highlights the *requirements of testing scenarios* mentioned in RFC 7928, followed by the details of modules that we contributed to ns-3 while implementing this suite. Section 3 presents the design assumptions and architecture of the proposed evaluation suite. Section 4 demonstrates the usage of our suite to compare existing AQM algorithms in ns-3. Section 5 summarizes and concludes the paper with directions for future work.

2 MODEL REQUIREMENTS

Prior to designing the AQM evaluation suite, we carried out a feasibility study to identify whether ns-3 supports the necessary models to implement every scenario mentioned in RFC 7928. This section provides a brief description of the models required for a comprehensive implementation of the proposed suite. Besides, it also highlights the models which we contributed in the process of developing this suite.

2.1 Transport Protocol Requirements

RFC 7928 recommends the following three categories of congestion controls to be used while evaluating the performance of AQM algorithms:

(1) Standard TCP congestion control: TCP NewReno with Selective Acknowledgment (SACK) is suggested as a baseline congestion control for this category. TCP NewReno is the default congestion control in ns-3, and the support for SACK has been recently added.
(2) Aggressive congestion control: CUBIC [10] is suggested as a baseline congestion control for this category, but it is not

supported in ns-3 currently. Nevertheless, RFC facilitates using other aggressive congestion control algorithms like TCP HighSpeed [3], BIC [15], Scalable [5], etc. These are supported in ns-3.

(3) Less-than-Best-Effort (LBE) congestion control: Low Extra Delay Background Transport (LEDBAT) [12] is suggested as a baseline congestion control for this category. We implemented LEDBAT in ns-3 during the development of this suite, since this is an important congestion control category to evaluate the performance of AQM algorithms. Recently, it has been pushed to `ns-3-dev`.

Apart from the congestion control requirements listed above, it is also recommended to test the AQM algorithms in the absence of congestion control i.e., when the transport protocol is unresponsive, such as UDP. ns-3 has an inherent model for simulating UDP.

2.2 Traffic Requirements

To verify the performance of AQM algorithms in different network settings (such as varying traffic load), following traffic types have been recommended in the RFC:

(1) Bulk TCP traffic: this is generally a long lasting, non-application limited traffic. Optionally, in some scenarios, it is recommended to keep this traffic short and application limited. ns-3 has built-in modules to generate this type of traffic.
(2) Web traffic: this traffic is short lived and total number of packets transferred are usually very few. Although ns-3 does not have a web traffic generator currently, there is one which is under review and planned for inclusion in the upcoming release of ns-3.
(3) Voice over IP (VoIP): this is a bidirectional interactive voice traffic, and is highly sensitive to delay. ns-3 currently does not have a model to generate this type of traffic.
(4) Constant Bit Rate (CBR) UDP traffic: this is an unresponsive traffic. It leads to sudden increase in the queue delay, and can starve TCP flows when they coexist. ns-3 provides built-in support for this type of traffic.

2.3 Topology Requirements

All scenarios described in the RFC are based on a standard dumbbell topology. In addition, a multi-bottleneck topology is suggested as an alternate option to further investigate the performance of AQM algorithms when they are deployed at multiple points along an Internet path. We have leveraged the topology helper classes of ns-3 for setting up the above mentioned topologies in the proposed suite.

For topology configuration, RFC 7928 provides a comprehensive information about the number of senders, receivers, guidelines to set the bottleneck buffer size and the type of application to be used. However, the bandwidth and delay values for edge or central bottleneck links are not provided for most of the scenarios. Hence, we have set these parameters in our suite by referring to the corresponding literature.

2.4 Other Requirements

Instead of proactively dropping packets, AQM algorithms can choose to send congestion signals by *marking* the packets by using Explicit Congestion Notification (ECN) [9]. RFC recommends to test the performance of AQM algorithms with and without ECN marking. We have added ECN support to ns-3[1] and our code is currently under review[2] for inclusion in `ns-3-dev`.

3 DESIGN AND IMPLEMENTATION

This section describes the architecture of the proposed suite and presents the details of its implementation in ns-3. Initially, the basic requirements of the suite, its core architecture, and implementation aspects are discussed, followed by a brief overview of how a user can interact with the suite.

3.1 Basic Requirements

We assume the following requirements to frame an architecture of our evaluation suite in ns-3:

(1) The suite should automate the cycle of simulation setup to results collection.
(2) It should be implemented as a separate module in `ns-3-dev`, for example by name `aqm-eval-suite` within `src`, and should contain necessary models, helpers and examples.
(3) Scenarios recommended in RFC 7928 should be implemented in the `examples` directory within `aqm-eval-suite`.
(4) It should be possible for a user to choose a specific scenario for evaluation from the ones provided in `aqm-eval-suite/examples`.
(5) It should be possible for a user to also choose multiple or all scenarios for evaluation.
(6) In every scenario, all AQM algorithms available in ns-3 should be compared with relevant performance metrics as suggested in RFC 7928.
(7) Results for each scenario should be systematically stored in text and graphical formats in a separate directory called `aqm-eval-output`, which should be within the ns-3 root directory at the same level as `src`.

Depending on these requirements, the next section describes the core architecture of our suite.

3.2 Core Architecture

The architecture of the proposed suite in ns-3 is shown in Figure 1 as a class diagram which highlights the interactions among the various classes of the suite, and their relationship with other existing classes of ns-3.

3.2.1 Model: The model provides an implementation of the following three primary classes:

EvaluationTopology: this class has three major functionalities:

(1) Creating the topology: It sets up a point-to-point dumbbell topology by using `PointToPointDumbbellHelper` with required number of nodes, and configures the data rate and

[1]this work was completed during 2016 ns-3 summer of code. More details can be found at https://www.nsnam.org/wiki/ECN_support_for_qdiscs_in_ns-3
[2]https://codereview.appspot.com/314790043

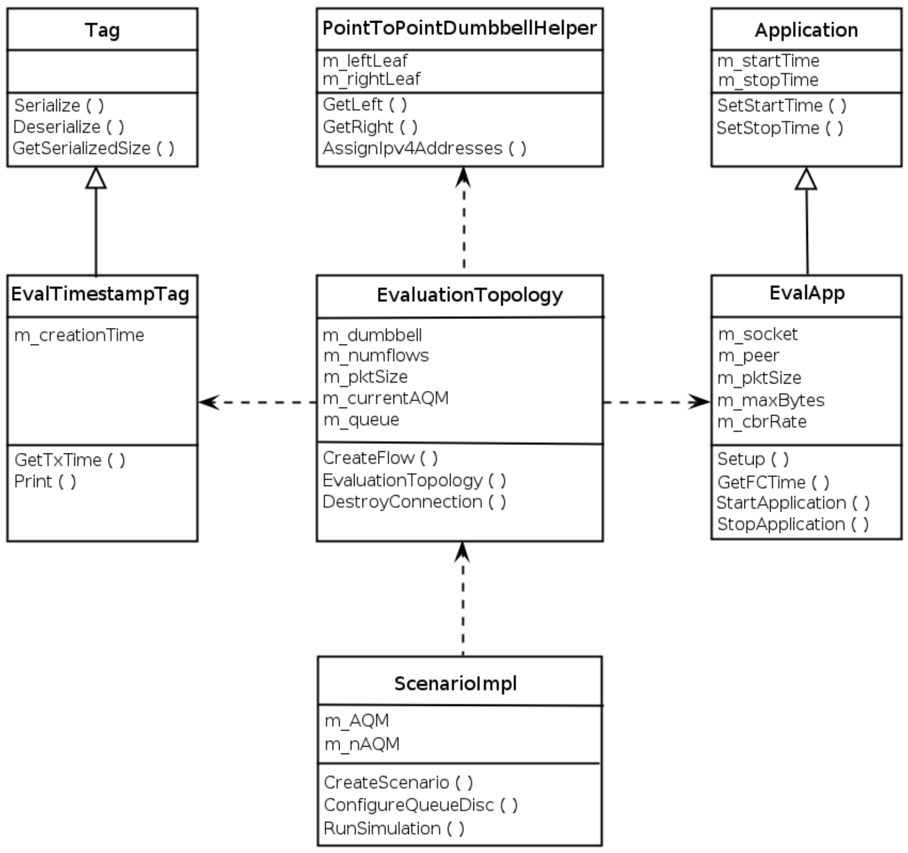


Figure 1: Class Diagram for AQM Evaluation Suite

for all the links. It also installs the desired queue discipline on the router node.

(2) Installing application on nodes: it provides an API for configuration of applications. This API takes the application parameters such as data rate, packet size, transport protocol, initial congestion window in case of TCP, maximum bandwidth and one way delay of the channels.
(3) Getting the metrics of interest from the experiment: It uses the traces sources provided by different classes of ns-3 for the calculation of metrics. The metrics recommended in the RFC are queue-delay, goodput, throughput and number of drops.
    - Queue delay: This is the average delay experienced by packets on the bottleneck queue. Enqueue and dequeue time of each packet is noted using trace sources of `QueueDisc` class and the difference between them is used to calculate the queue delay.
    - Throughput: This is the time average of data transmitted from the bottleneck router, which is calculated using the trace source of `NetDevice` class.
    - Goodput: This is the time average of data received by the sink application, which is calculated using the trace source of `PacketSink` application class.
    - Packet Drop: The number of drops is calculated by using the trace sources of `QueueDisc` class.

EvalApp: This class is based on the `OnOffApplication` which is used for generating TCP and UDP traffic. The ns-3 applications currently implemented create sockets at the start time of an application. As a consequence, there were issues in configuration of initial congestion window for TCP, since it cannot be configured before the socket is created and after the application starts. To overcome this, an application was implemented on the same principles as that of the `OnOffApplication`, in which a socket is created and the application is started only after its parameters are configured.

EvalTimestampTag: This is a subclass of `Tag` and has been developed to fetch the queue delay information from the `QueueDisc`. When a packet is enqueued by the `QueueDisc`, this tag is added with a timestamp (as the enqueue time) and when the packet is dequeued, the queue delay is computed as the difference between the dequeue time and the enqueue time.

3.2.2 Helper: The helper in the `aqm-eval-suite` consists of only one class:

ScenarioImpl: This is the superclass of all the scenarios present in the `examples` directory. It has two primary methods :

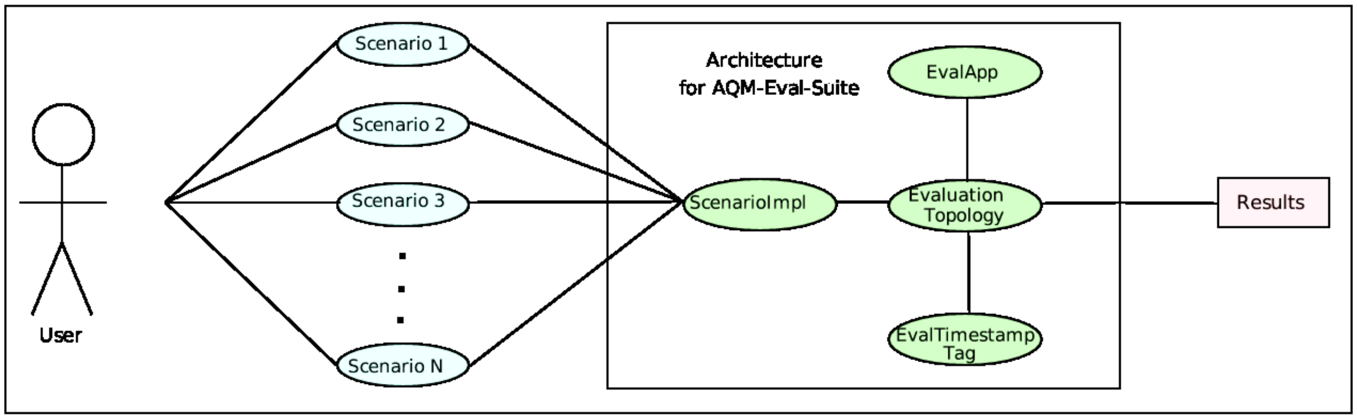


Figure 2: Interaction Diagram for the AQM Evaluation Suite

(1) `CreateScenario`: This is a purely virtual function and is implemented by each scenario according to the topology and traffic profiles mentioned in the RFC.
(2) `RunSimulation`: This method takes the scenario created by each subclass and runs them with all the queue disciplines available in ns-3.

3.2.3 Utils: `utils` directory contains scripts written in Python which take performance metrics computed by the `aqm-eval-suite` as input and generate a graph with queuing delay as the X-axis against Goodput as the Y-axis. The graph depicts an ellipse which is plotted as per the guidelines mentioned in the RFC and [14]. The co-variance between the queuing delay and goodput is determined by the orientation of the ellipse, and helps us to analyze the effect of traffic load on goodput and queuing delay. A total of four Python scripts are provided:

ellipsemaker: this script has been taken from the TCP Ex Machina project [14] to generate queuing delay and goodput covariance error ellipse.

generate-ellipsepoint: this script is used to find corresponding value of goodput for every recorded value of queue delay.

drop-process: this script calculates CDF of time difference between two successive drops of each flow.

goodput-process: this script calculates cumulative goodput of each flow.

3.3 User Interaction with the Suite

Scenarios are implemented in the `examples` directory of our suite. Figure 2 depicts the interaction of user with our model. As mentioned before, the user is given the flexibility to select a single scenario to run, or multiple scenarios at once. The `aqm-eval-runner` program in `examples` is used to run all the scenarios. Every scenario implements `ScenarioImpl` helper class and subsequently the classes provided in `model`. The results for each scenario are stored in textual and graphical formats in `aqm-eval-output`.

3.4 Scope and Limitations

In this section, we summarize the scope and limitations of our proposed model:

- The propose suite provides support for all the experiments described in Sections 5, 6 and 8 of RFC 7928.
- Experiments listed in Sections 7 and 9 are currently not supported due to the unavailability of necessary traffic generation models, like web traffic and video streaming traffic generators, in ns-3.
- Studying the interactions of AQM algorithms with ECN and scheduling algorithms is also not possible for similar reasons.
- Although RFC recommends to test the performance in multi-AQM scenarios, the exact parameters and topology descriptions are not explicitly provided. Thus, it was not possible to provide support for multi-AQM scenarios in our evaluation suite.

4 EVALUATING AQM ALGORITHMS USING THE SUITE

In this section we demonstrate the usage of our proposed suite to evaluate the performance of RED, Adaptive RED, CoDel and PIE in ns-3. Additionally, droptail mechanism is also considered. The main purpose of this section is not to discuss the performance of individual AQM algorithm and the impact of setting its respective parameters. Instead, the aim is to highlight the usage of this suite to quickly derive inferences about the performance of AQM algorithms in general.

We use the default values of the parameters related to every AQM algorithm. All the scenarios in the proposed suite are based on a simple dumbbell topology, with AQM deployed at the bottleneck router. Based on the recommendations in the RFC, the buffer size at the bottleneck routers is set equal to the Bandwidth Delay Product (BDP) in the network. Number of flows and type of traffic changes based on every scenario's requirements. Most of the TCP flows are SACK enabled NewReno flows, unless specified otherwise. Simulation duration is fixed to 300 seconds for all the scenarios.

The start times of flows are varied when multiple flows are passed through the same bottleneck link.

The next sections demonstrate how the proposed suite can be used to compare the above mentioned AQM algorithms, and present the results obtained for each scenario. Results presented highlight the tradeoff between goodput and queuing delay and are inline with those recommended in the RFC. Due to space limitations, we discuss only few scenarios and their results in this paper. Results for other scenarios can be obtained by using the proposed suite which is openly available at [2].

4.1 Interaction with Transport Protocols

This section describes experiments that capture the interaction of AQM algorithms with different classes of traffic ranging from standard TCP traffic, aggressive TCP traffic to unresponsive UDP traffic. It covers the scenarios listed in Section 5 of RFC 7928.

4.1.1 TCP-Friendly Sender: This experiment evaluates the performance of AQM algorithms with TCP-friendly traffic in different network settings. It comprises of two sub-scenarios: (a) a single, long-lived, non-application-limited, TCP NewReno flow, with an Initial congestion Window (IW) set to 3 packets, and (b) same as previous one, but with an addition of one more, application-limited, TCP NewReno flow, with an IW set to 3 or 10 packets. Other parameters are set as per the recommendations provided in Section 5.1 of RFC 7928.

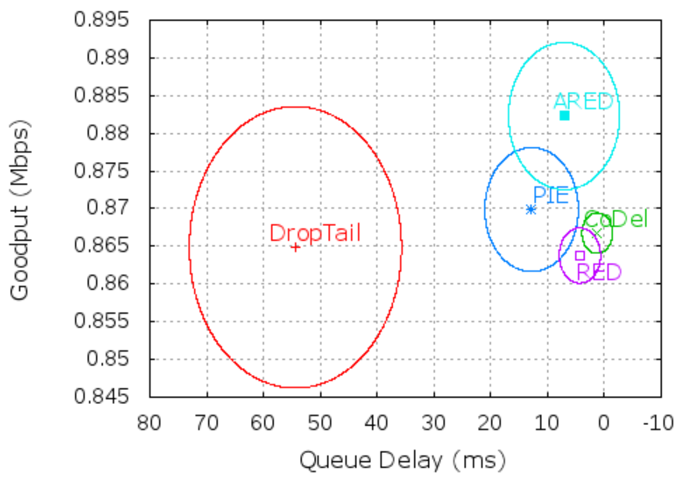


Figure 3: TCP-Friendly NewReno Sender with SACK

Figure 3 and Figure 4 present the results for (a) and (b) respectively. As expected, the addition of an extra flow in (b) affects the goodput of all AQM algorithms. The performance of DropTail is worse in both cases since it utilizes the entire buffer and induces high queuing delay.

4.1.2 Aggressive Sender: This experiment evaluates the performance of AQM algorithms with aggressive TCP traffic. RFC recommends a single, long-lived, non-application-limited, CUBIC flow for this experiment. Since a model of CUBIC is not available in ns-3, we have used High Speed TCP which is an aggressive TCP

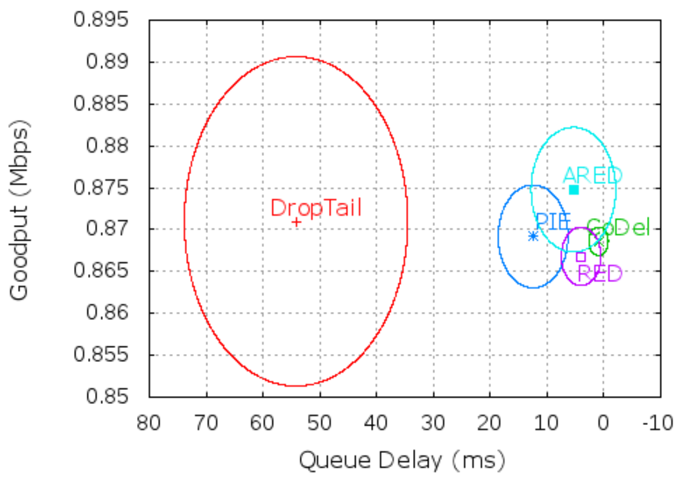


Figure 4: Two NewReno flows with different IW and different amount of data

algorithm like CUBIC. Moreover, the value of IW is not mentioned in the RFC, so we retain the default value provided in ns-3.

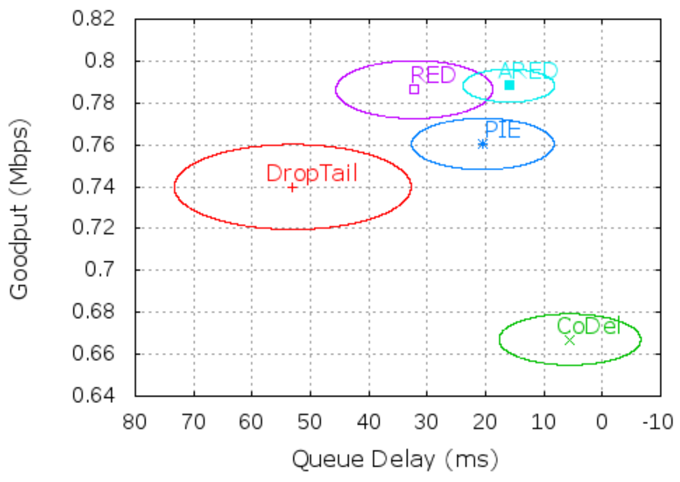


Figure 5: Aggressive Sender

Figure 5 presents the results for this experiment. In comparison to the results obtained with TCP-friendly experiments, we observe that a single aggressive flow affects the stability of the queue and incurs variations in the delay. RED with its default parameter settings has maximum variations in the queue delay. Other AQM algorithms perform relatively better by keeping the queue delay to minimum.

4.1.3 Unresponsive Sender: AQM algorithms should be robust against unresponsive traffic and also in cases where there is faulty transport implementation on an end host, or in the network that hides congestion notification in the header of packet. This experiment aims to evaluate the robustness of AQM algorithms when a

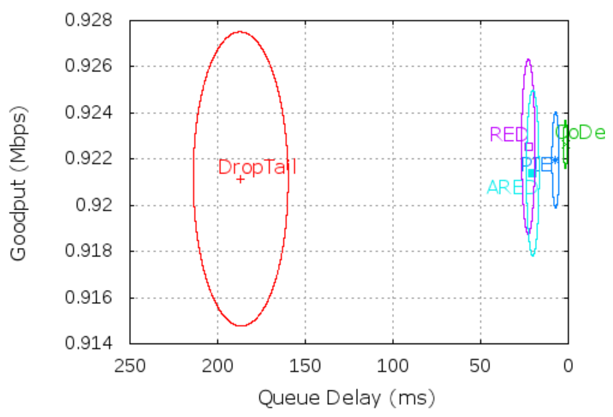


Figure 6: An unresponsive flow with a TCP NewReno flow

non-responsive UDP flow coexists with a TCP flow. UDP flow is configured to send the data at a rate less than half but greater than the bottleneck capacity, and the TCP flow is set to NewReno with SACK enabled.

Figure 6 presents the results for this experiment and it is apparent that the presence of an unresponsive flow leads to fluctuations in the link utilization and subsequently affects the stability of goodput. Nevertheless, all AQM algorithms successfully control the queue delay. DropTail incurs high queueing delay as expected.

4.1.4 Less than Best Effort Sender: This experiment is aimed at evaluating the performance of AQM algorithms with LBE sender. LBE traffic is characterized to have a smaller impact on standard TCP performance when both share the same bottleneck link. This experiment comprises of two flows, first being a LEDBAT flow and second being a TCP-friendly NewReno sender with SACK.

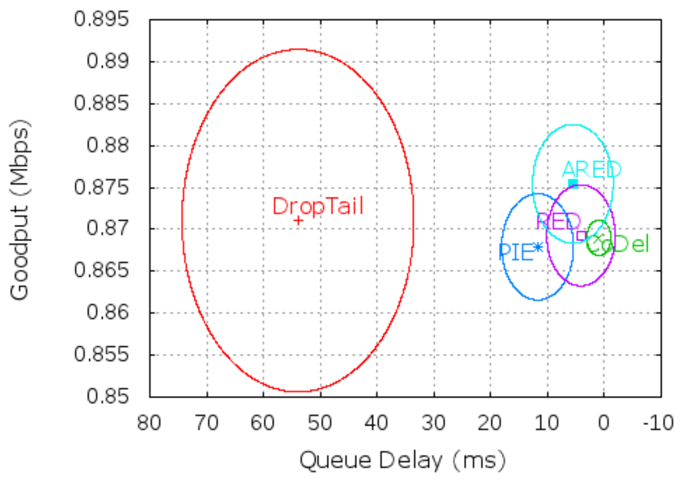


Figure 7: A LEDBAT flow with a TCP NewReno flow

Figure 7 presents the results. All AQM algorithms successfully balance the trade-off between goodput and queue delay in this experiment. The best performance is obtained with CoDel algorithm in terms of queue delay and Adaptive RED algorithm in terms of goodput.

4.2 RTT Fairness

This experiment aims to test the performance of AQM algorithms when a set of flows with different RTTs share the same bottleneck link. The main aim is to study the fairness among these different flows. The experiment features two TCP flows, first with a fixed RTT of 50ms and the second flow having a RTT varying between 5ms to 560ms in different runs. However, the results presented here belong to a specific run where RTT for second flow is set to 20ms. Results with other values of RTT for the second flow can be obtained from our suite [2].

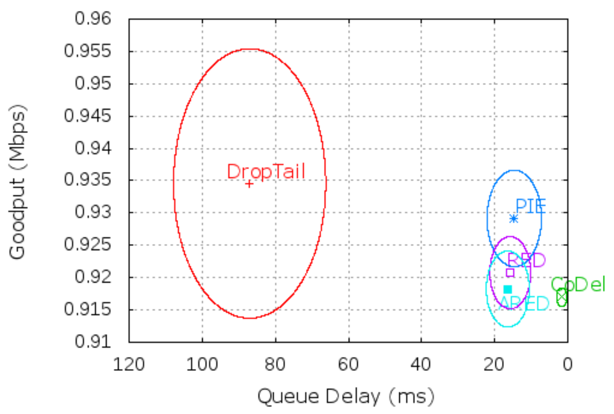


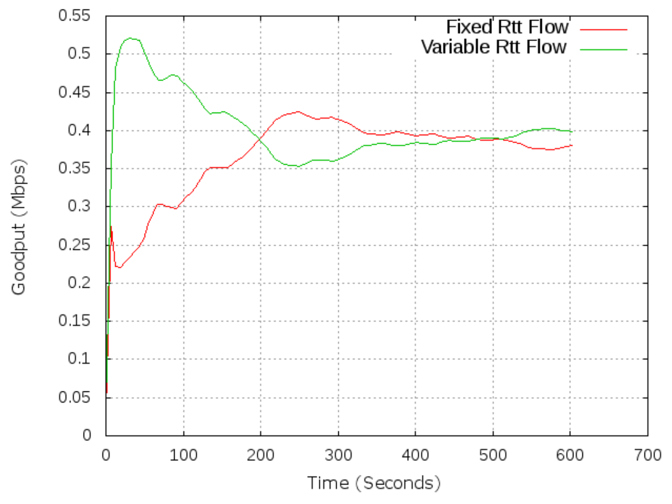
Figure 8: Two TCP NewReno flows with RTT 50ms and 20ms

Figure 8 summarizes the delay - goodput tradeoff for this experiment. As recommended in the RFC, Figure 9 presents the cumulative goodput obtained for each TCP flow with different AQM algorithms. This helps to study how soon the flows converge to a fair share of the bandwidth. In the current simulation setting, CoDel converges to a fair share much faster than other AQM algorithms. PIE fails to converge and as a result, the flow with variable RTT utilizes a larger share of the bandwidth.

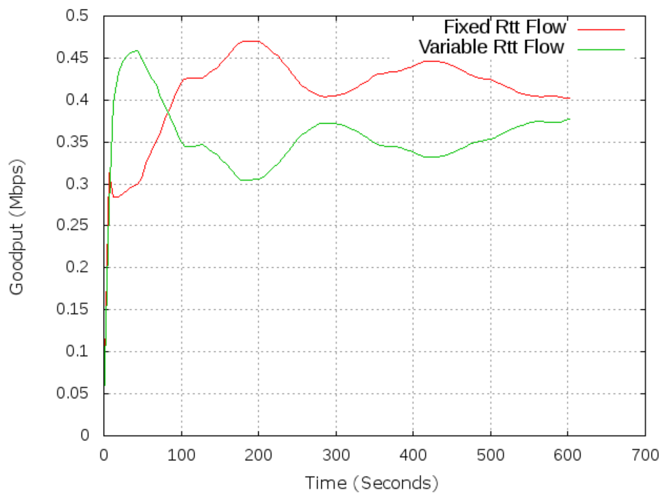
4.3 Stability Analysis

4.3.1 Varying Congestion: This experiment aims to test the performance of AQM algorithms when the level of congestion varies over a large timescale. RFC classifies the level of congestion into three: mild, medium and heavy. The number of TCP flows in mild congestion is 0.036 times S, in medium congestion is 0.081 times S and in heavy congestion is 0.114 times S, where S is the sum of BDP and maximum buffer size expressed in packets. The main aim is to study the working of AQM algorithms when the amount of traffic load varies in the network.

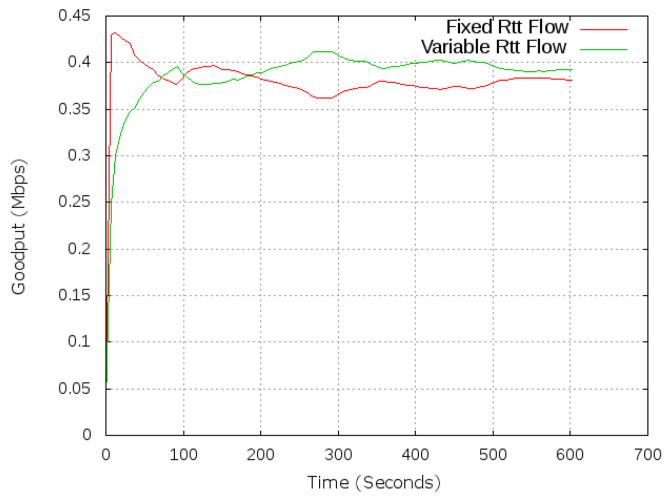
Figure 10 presents the delay - goodput tradeoff for this experiment. Results show that AQM algorithms perform reasonably well in this scenario by keeping the queue delay within the respective limits, and also handle the variations in traffic load effectively.



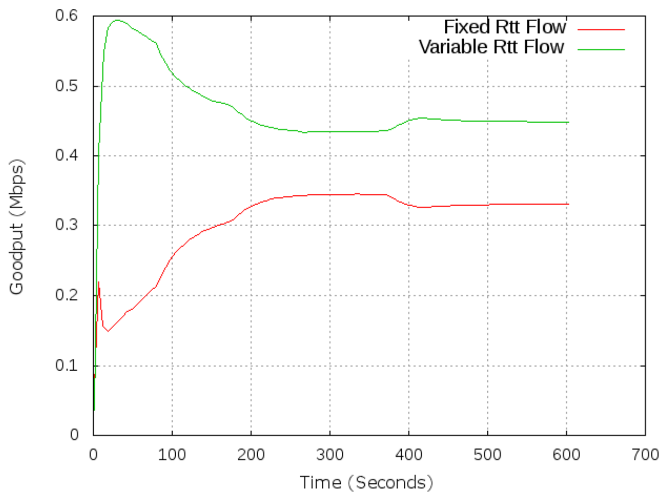
(a) RED



(b) ARED



(c) CoDel



(d) PIE

Figure 9: Goodput of two TCP NewReno flows with different AQM algorithms

4.3.2 Varying Bottleneck Bandwidth: This experiment verifies the adaptability of AQM algorithms when the bottleneck bandwidth varies sharply. Such type of variations are common in Wi-Fi environments. The bottleneck bandwidth is switched to 10Mbps and 100Mbps in every 20 seconds over a large time scale. Two TCP-friendly flows are used to generate the traffic.

Figure 11 presents the delay - goodput tradeoff for this experiment. The queue delay varies over a large scale due to sudden variations in the bottleneck capacity. When the capacity reduces to 10Mbps, the queue starts building up sharply. Similarly when the capacity increases to 100Mbps, queue drains out quickly.

5 CONCLUSIONS AND FUTURE WORK

The necessity to deploy AQM algorithms in the Internet has become apparent as time sensitive applications continue to evolve significantly. Nevertheless, thorough investigation of the working of these algorithms prior to their deployment is inevitable. In this paper, we have presented an AQM Evaluation Suite to test the performance of AQM algorithms, in line with the guidelines mentioned in RFC 7928. We believe that this suite would help the researchers to evaluate their AQM proposals quickly and comprehensively, before deploying them in a real network setup. We plan to extend this evaluation suite to provide support for ECN and other features listed in the RFC. Finally, we look forward to leverage the MPI support in ns-3 to minimize the overall execution time of our evaluation suite.

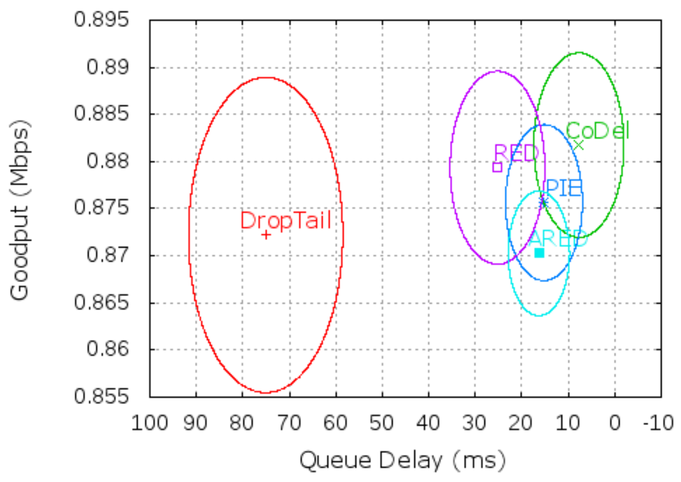


Figure 10: Varying Congestion

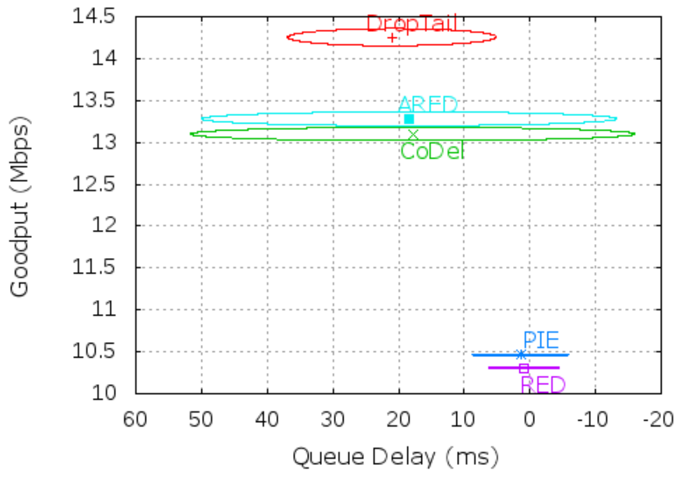


Figure 11: Varying Bandwidth of the Bottleneck link every 20s

ACKNOWLEDGMENTS

We would like to thank Tom Henderson for mentoring the ns-3 summer of code project on ECN, and for providing helpful feedback on our implementation of LEDBAT and the architecture of AQM Evaluation Suite. We would also like to thank Stefano Avallone for his timely reviews and guidance on our implementation of ECN, and Natale Patriciello for helpful suggestions on our implementation of LEDBAT and ECN. Finally, we would like to thank the entire ns-3 development team for their excellent guidance and support.

REFERENCES

[1] K. De Schepper, O. Bondarenko, J. Tsang, and B. Briscoe. 2016. PI2: A Linearized AQM for both Classic and Scalable TCP. In *Proceedings of the 12th International Conference on emerging Networking EXperiments and Technologies.* ACM, 105–119.
[2] AQM Evaluation Suite Source Repository. 2017. https://aqm-eval-suite.github.io/. (2017).
[3] S. Floyd. 2003. HighSpeed TCP for Large Congestion Windows. (2003).
[4] P. Imputato and S. Avallone. 2016. Design and Implementation of the Traffic Control Module in ns-3. In *Proceedings of the Workshop on ns-3.* ACM, 1–8.
[5] T. Kelly. 2003. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *ACM SIGCOMM computer communication Review* 33, 2 (2003), 83–91.
[6] N. Kuhn, P. Natarajan, N. Khademi, D. Ros, and others. 2016. Characterization Guidelines for Active Queue Management (AQM). (2016).
[7] K. Nichols and V. Jacobson. 2012. Controlling Queue Delay. *Commun. ACM* 55, 7 (2012), 42–50.
[8] R. Pan, P. Natarajan, C. Piglione, M. Suryanarayana Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. 2013. PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem. In *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR).* IEEE, 148–155.
[9] K. Ramakrishnan, S. Floyd, D. Black, and others. 2001. The Addition of Explicit Congestion Notification (ECN) to IP. (2001).
[10] I. Rhee and L. Xu. 2005. CUBIC: A New TCP-Friendly High-Speed TCP Variant. (2005).
[11] K. Schepper, O. Bondarenko, J. Tsang, and B. Briscoe. 2015. Data Centre to the Home: Ultra-Low Latency for All. (2015).
[12] S. Shalunov, G. Hazel, J. Iyengar, M. Kuehlewind, and others. 2012. Low Extra Delay Background Transport (LEDBAT). (2012).
[13] S. K. Srinivas, S. Murali, and M. P. Tahiliani. 2016. Implementation and Evaluation of Proportional Integral Controller Enhanced (PIE) Algorithm in ns-3. In *Proceedings of the Workshop on ns-3 (WNS3 '16).* ACM, New York, NY, USA, 9–16. DOI: https://doi.org/10.1145/2915371.2915385
[14] K. Winstein and H. Balakrishnan. 2013. TCP Ex Machina: Computer-generated Congestion Control. In *ACM SIGCOMM Computer Communication Review*, Vol. 43. ACM, 123–134.
[15] L. Xu, K. Harfoush, and I. Rhee. 2004. Binary Increase Congestion Control (BIC) for Fast Long-distance Networks. In *IEEE INFOCOM 2004*, Vol. 4. IEEE, 2514–2524.