Testing Modern AQMs on GENI-Testbed

Ish Kumar Jain
NYU Tandon School of Engineering
Advisor- Fraida Fund, Prof. Shiv Panwar

***Abstract*—Active Queue Management (AQM) algorithms have been proposed to provide both low latency (queuing delay) and high network goodput. In this project, we replicated the results of [1] to compare seven such AQM algorithms—*namely*, FIFO queue, ARED, PIE, CoDel, SFQ, fq_codel, and fq_nocodel—for different scenarios on GENI testbed. The original paper compared three aspects of modern AQMs: the good, the bad, and the WiFi.**

- **The good: steady state behaviour of AQM algorithms. Plot mean TCP goodput vs mean induced latency**
- **The bad: 1. fairness among TCP flows, and 2. Transient analysis when the flow start up.**
- **The WiFi: Analyze AQMs after adding a wireless link.**

In this report, we implemented the seven AQMs on Geni-testbed and their performance in terms of throughput, latency, and fairness index is analyzed. The experiment on Orbit for the wifi link is not implemented yet and thus omitted in this report. Our results fairly approximates to those presented in the original paper.

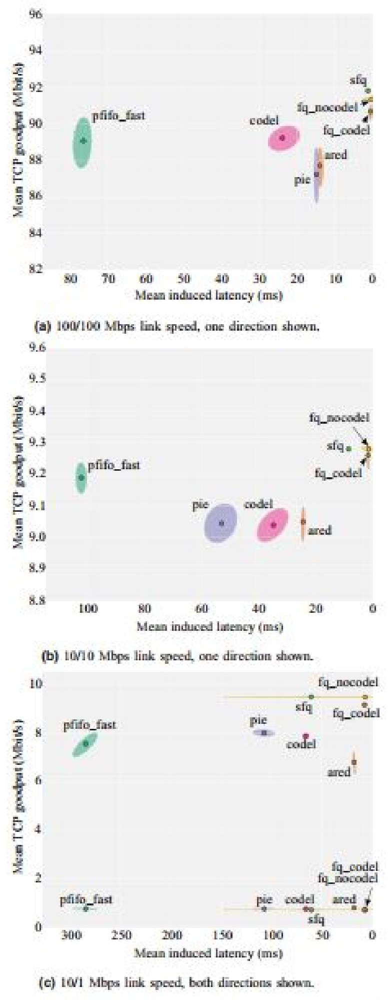
I. Explanation of Key Results

The main results are obtained in three sections: the good, the bad, and the WiFi.

A. The good about AQM: steady-state behaviour

The Fig. 1 shows latency-goodput ellipsis graphs for the seven AQM algorithms. The data was collected by real-time response under load (RRUL) test. In particular, a heavy load with four concurrent TCP flows was established in each direction and simultaneously the latency was measured for UDP and ICMP packets. Fig. 1(a, b) are plotted for symmetric traffic with 100/100 and 10/10 Mbps link speed and Fig. 1(c) was plotted for asymmetric 10/1 Mbps link speed.

Relevance: The results show that "*the default FIFO queue predictably give a high induced latency, but with high goodput*"[1]. On the other hand, the three main AQM algorithms—ARED, PIE, and CoDel—provide less latency, but with comparable goodput as FIFO queue. The best performance was given by fairness queuing algorithm which induces almost zero latency and maintain a high goodput, mainly because they prioritize the sparse ICMP flows. The subtle differences between these algorithms are explained nicely in the paper.



(a) 100/100 Mbps link speed, one direction shown.

(b) 10/10 Mbps link speed, one direction shown.

(c) 10/1 Mbps link speed, both directions shown.

Fig. 2. (a and b) The RRUL test results, showing the median values and 1-σ ellipses of the per-test-run mean goodput and mean induced latency. (c) As (a and b), but showing both upstream and downstream traffic, re-using the same latency values.

Fig. 1. Showing goodput-latency tradeoff for (a) 100/100 Mbps, (b) 10/10 Mbps and (c) 10/1 Mbps link speed for seven different algorithms [1].

[1]The quote at few places in this proposal and most of the content is taken from [1]

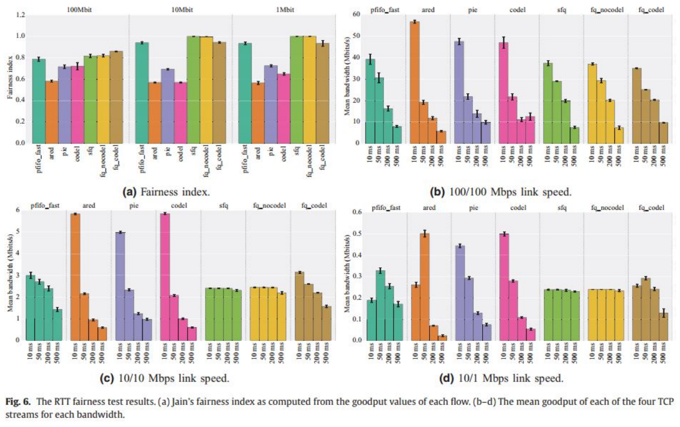


Fig. 2. RTT fairness results: (a) Jain's fairness index. (b-d) Mean goodput for four TCP streams for each bandwidth [1].

B. The bad about AQM: fairness and transient behaviour

Fig. 2 and Fig. 3 shows the results for RTT-fairness tests and transient behaviour tests respectively. As shown, the AQM algorithms are not fairer as compared to the FIFO queue. Also, the fairness queuing algorithms are undoubtedly fairer than other algorithms. However, fair queue with CoDel (*fq_codel*) is worse than other fair queue algorithms.

The result on transient behaviour in also not in the favour of AQM algorithms. Fig. 3 shows that both PIE and CoDel have severe spikes at the start of the flow. However, ARED achieves a good smooth performance.

C. The WiFi: adding a wireless link

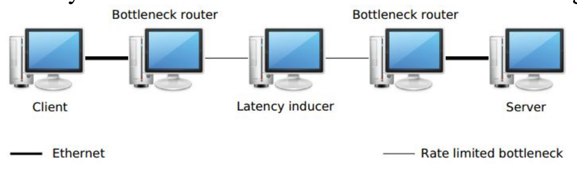
The WiFi link was shown to be the bottleneck link at 100 Mbps flow. The results however shows a similar trend in Fig. 4. The algorithms show the same trend of induced latency behaviour, "*with FIFO being worst, followed by PIE and CoDel, the ARED and the fairness queuing algorithms*", however, with different magnitudes.

II. Experimental Design

We follow the steps involved in a network experimental design from this excerpt [2].

A. Experiment-1

Wired connections: The topology consists of five nodes connected to each other in a daisy-chain manner which is commonly seen in residential Internet connection setting.



1) Goal: To compare the performance of seven different queue management algorithms

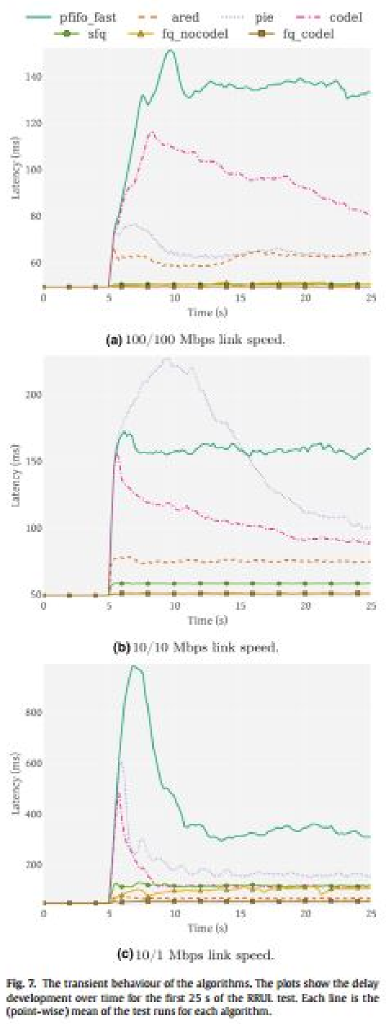


Fig. 3. Transient behaviour of the algorithms [1].

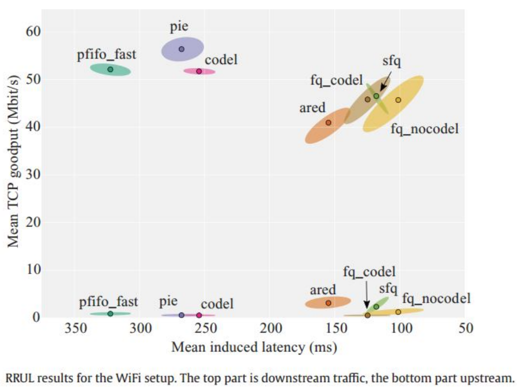


Fig. 4. RRUL results for the WiFi setup [1].

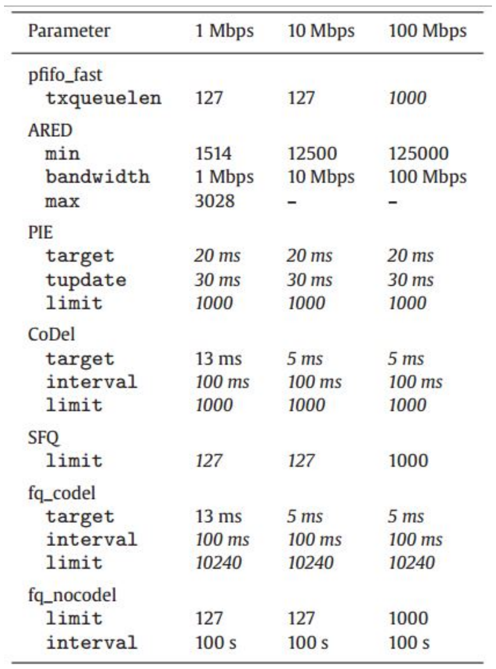
2) Services and Outcomes: The two end nodes represents client and server which are individually connected to a bottleneck router (that mimicks gateway router). The two bottleneck routers employ software rate limiting (using *tbf* rate limiter [3]) There is a middle node between the two bottleneck router that works as a latency inducer which adds latency by employing the *dummynet* emulation framework [4].

3) Metrics:

- The following metrics are used-
  - Mean TCP goodput (Mbit/s)
  - Mean induced latency (ms)
  - Fairness index
  - Mean bandwidth (Mbit/sec) per flow
  - Initial peak latency (transient behaviour)

4) Parameters:

- Since we want to measure the performance (latency, goodput) of our algorithms, other sources of latency and queuing should be turned off. The following parameters should be used **(I have not yet verified these conditions)**-
  - hardware offload feature- turned off
  - kernel Byte Queue limits- maximum of 1 packet
  - kernel compilation clock-tick frequency- highest (1000Hz)
- Flow setting using default TCP CUBIC algorithm: *Netperf*[2] tool
- Open source testing software- *Flent*[3]. **(For fairness test, I used iperf instead of Flent and Netperf)**
- Tested Algorithms: Seven queue management schemes (*qdiscs* in Linux) have to be analyzed. Mostly, the default parameters are used for these algorithms.
  - Queue length: 127 packets for 1 and 10 Mbps; 1000 packets at 100 Mbps.
  - The Qdisk parameters: (kernel defaults are shown in *italics*)



Parameter	1 Mbps	10 Mbps	100 Mbps
pfifo_fast			
`txqueuelen`	127	127	*1000*
ARED			
`min`	1514	12500	125000
`bandwidth`	1 Mbps	10 Mbps	100 Mbps
`max`	3028	-	-
PIE			
`target`	*20 ms*	*20 ms*	*20 ms*
`tupdate`	*30 ms*	*30 ms*	*30 ms*
`limit`	*1000*	*1000*	*1000*
CoDel			
`target`	13 ms	*5 ms*	*5 ms*
`interval`	*100 ms*	*100 ms*	*100 ms*
`limit`	*1000*	*1000*	*1000*
SFQ			
`limit`	*127*	*127*	1000
fq_codel			
`target`	13 ms	*5 ms*	*5 ms*
`interval`	*100 ms*	*100 ms*	*100 ms*
`limit`	*10240*	*10240*	*10240*
fq_nocodel			
`limit`	127	127	1000
`interval`	100 s	100 s	100 s

5) Factors to study and their levels:

- Seven queue management schemes using default parameters.
- Three configurations of links are used:
  - symmetrical link at 100 Mbps.
  - symmetrical link at 10 Mbps.
  - asymmetrical link with 10/1 Mbps download/upload speeds.
- For fairness result, we choose 4 RTT values for 4 flows (10, 50, 200, 500 ms).

6) Evaluation technique: GENI testbed experiment

7) Workload and Experiment Design:

- Get steady state performance- run the algorithm for $140s$ and repeat 30 times.
- For real-time response under load (RRUL) test-
  - Run 4 concurrent TCP flows in both direction.
  - Simultaneously measure latency using both UDP and ICMP packets
- Get fairness by RTT-fairness test **(I used iperf)**-
  - Run 4 concurrent TCP flows from client to server
  - Choose a different RTT value of each such flow (10, 50, 200, 500 ms)
  - Measure the TCP goodput of each stream using *Flent* tool.
  - Calculate Jain's Fairness index [5] over the total goodput of 4 flows.
- Measure the transient response
  - Measure the delay when the four bi-directional TCP streams of the RRUL test start-up.
  - Get the time sequence graph for 25 sec and repeat for 30 iterations.

B. *Experiment-2* ***(Not performed yet)***

Adding a wireless link to the setup of Experiment-1.

[2]www.netperf.org

[3]https://flent.org/

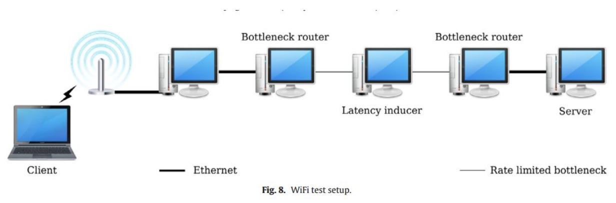


Fig. 5. Topology-1: Physical test setup [1]

1) Goal: To compare the performance of seven queue management algorithms and get goodput vs latency graph.

2) Services and Outcomes: The service is affected by the bottleneck of wireless link. This is because of the different characteristic of physical WiFi link—consider retransmit and aggregation feature of WiFi. Following outcomes have been observed by [1]-

- WiFi link is not a bottleneck at low rate (1 or 10 Mbps)
- Also, it does not show high peaks in transient behaviour.
- At 100 Mbps, the goodput-latency curve shows the same trend as wired link, but with different magnitudes of induced latency.

3) Metrics: Mean TCP goodput and Mean induced latency

4) Parameters: The WiFi access point require some WiFi driver, protocol (802.11n), and frequency spectrum (5GHz). Assume no other devices sharing the WiFi link. "*Apply the queue management algorithms to both sides of the WiFi as well as to the bottleneck link as before*".

5) Factors to study and their levels: Seven queue management schemes.

6) Evaluation technique: GENI testbed experiment. Require Orbit for wireless link.

7) Workload and Experiment design: Same as Expt.-1.

III. Practical Experiment Plan

1) Last part of experiment requires wireless link.
2) We require small network with 5-6 nodes only.
3) Traffic generating tools- *Netperf*[4], Ping ICMP packets.
4) Latency inducer tool- *dummynet* [5]
5) Queue management tool- *pfifo_fast*[6], ARED[7], PIE[8], CoDel[9], SFQ[10], fq-codel[11],
6) Measurement tools- *Flent*[12]

IV. Expending on Key Results

The following expensions of this project are possible

- The experiment can be repeated for a set of values of parameters like queue size, and link capacity and the sensitivity to these parameters can be analyzed.

[4]www.netperf.org

[5]http://info.iet.unipi.it/ luigi/dummynet/

[6]http://man7.org/linux/man-pages/man8/tc-pfifo_fast.8.html

[7]www.icir.org/floyd/papers/adaptiveRed.pdf

[8]https://tools.ietf.org/html/draft-ietf-aqm-docsis-pie-02

[9]https://tools.ietf.org/html/draft-ietf-aqm-codel-10

[10]http://lartc.org/manpages/tc-sfq.html

[11]https://tools.ietf.org/html/draft-ietf-aqm-fq-codel-06

[12]www.flent.org

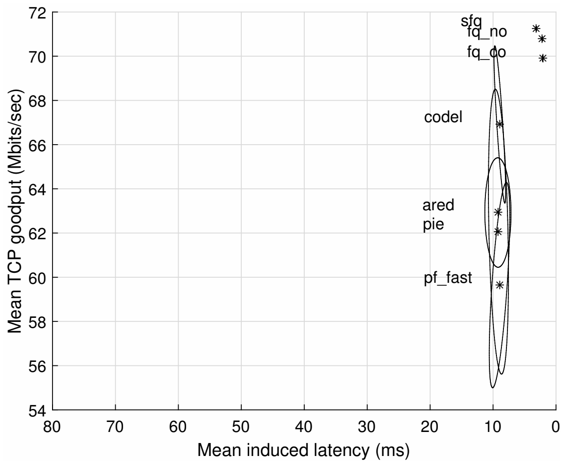


Fig. 6. RRUL test 100/100 Link

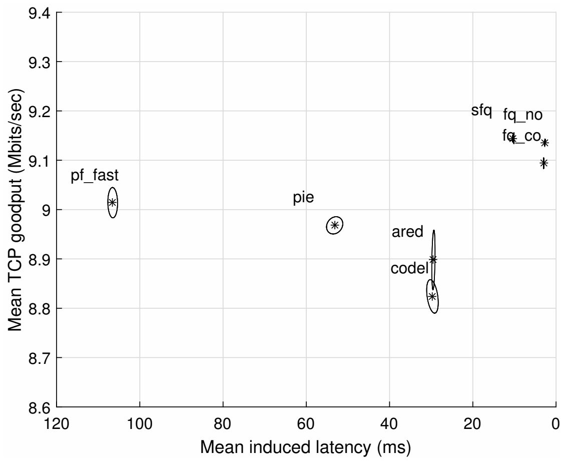


Fig. 7. RRUL test 10/10 Link

- Test the algorithms for practical flow conditions *e.g.* VoIP and Web flows (using cURL library, D-ITG, and *iperf* tools) as done in the paper [1].
- This can be implemented in more realistic home network setting [6]

V. Result

A. Results-1: RRUL

The RRUL test results for 100/100, 10/10, and 10/1 bi-directional links are shown in Fig. 6,7,8 respectively. They are obtained as an average of 5 repeated experiments (of duration 60 sec). I am getting almost similar relative performances as in the paper. However, the values are not exactly matching. The best performance graph is obtained for 10/1 and 10/10 link. However, for the 100/100 link, the magnitude of latency is lower as compared to that in the paper and most of the algorithms (except fairness queuing algorithms) gives a similar latency value. The throughput is also reduced. I am yet to find out if there is any glitch in the implementation at 100 Mbps.

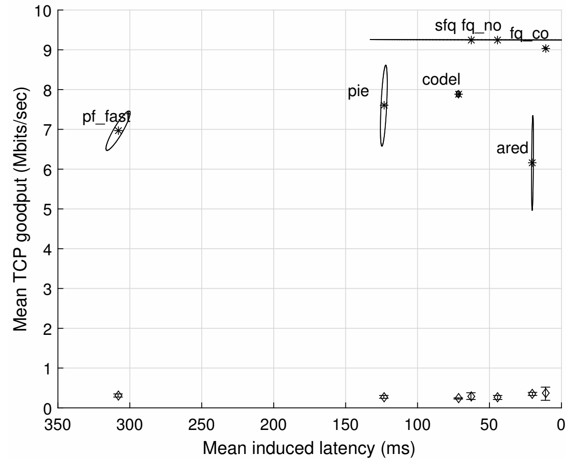


Fig. 8. RRUL test 10/1 Link

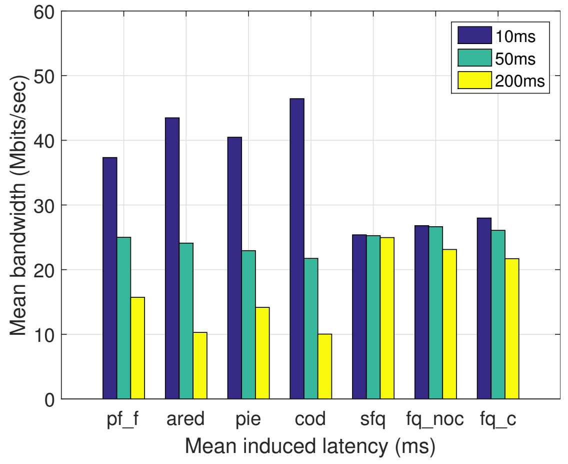


Fig. 9. Mean goodput for three concurrent TCP streams for 100/100 link

B. Result-2: Fairness

The Fairness results are shown in Fig. 9, 10, 11. I have obtained the results for 10/10 and 100/100 Mbps links. I need to fix some bugs for getting results for 10/1 link. These results are obtained by setting 3 iperf connection between client and servernon three different port numbers. On the bottleneck router, flow specific firewall rules are added using tc command. In particular, for the three tcp streams a different RTT of 10ms, 50ms, and 200ms is set. The three bar graph corresponding to each AQMs represtnts this RTT value. Please note that I could not set-up more than three firewall rules due to the limitation of Linux kernel.

Again, the results are following a similar pattern as shown in the paper. A small difference in the value can be due to a limited iteration of experiments (I repeated the experiment only 5 times due to time constraints.)

The fairness queuing algorithms are undoubtedly fairer than other algorithms(fairness index is almost 1).

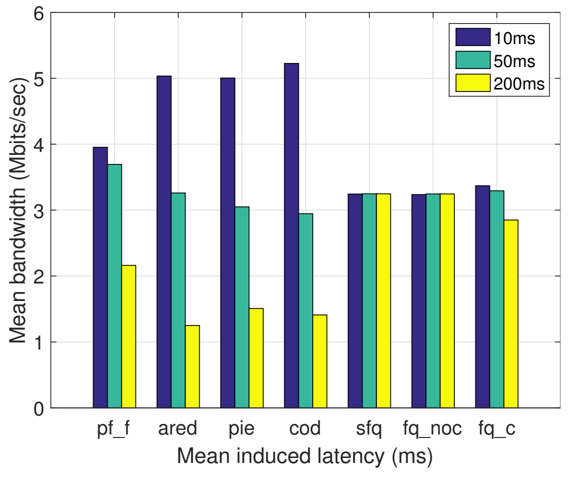


Fig. 10. Mean goodput for three concurrent TCP streams for 10/10 link

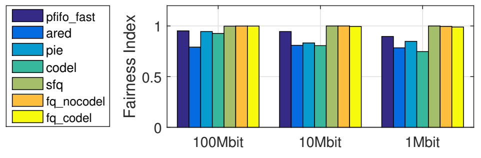


Fig. 11. Jain's Fairness Index for all 7 AQMs for 10 Mbit (left) and 100 Mbit (right)

C. Transient behaviour

The results for transient analysis are shown in Figure 12, 13, 14 for 100/100, 10/10, and 10/1 link respectively. For 10/10 link we consistently show that ARED performs the worst in Latency during start-up. For 100/100 there seems some error and for 10/1 link the results are mostly similar.

VI. Steps to replicate this experiment

First, make a topology on GENI-testbed. For RRUL test, refer to the mycode directory and for the fairness test my2code directory (I'll suggest to start from fairness test). The main

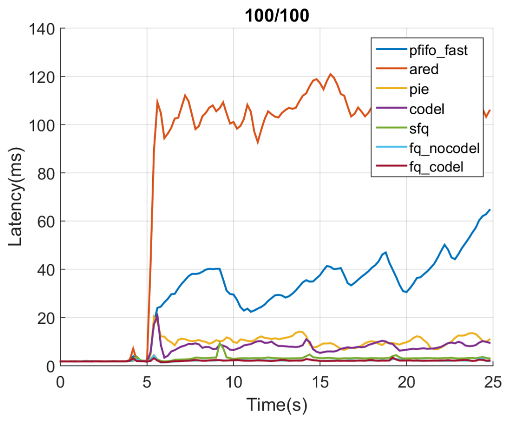


Fig. 12. Transient test 100/100 Link

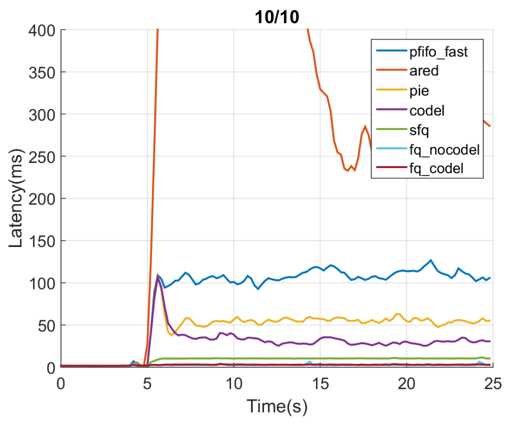


Fig. 13. Transient test 10/10 Link

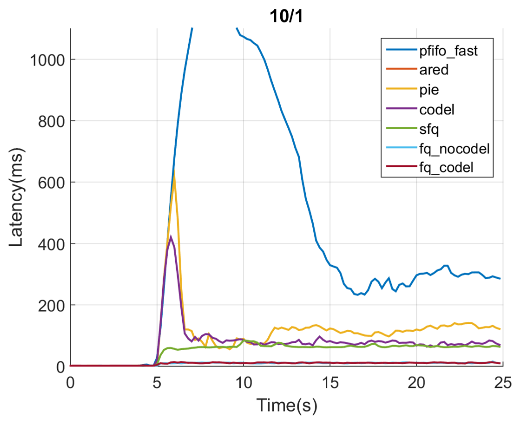


Fig. 14. Transient test 10/1 Link

file is maincode.sh which can be run by command line on a local computer. This file will automatically perform ssh to the testbed nodes and run the experiment there. Therefore, we need to update the ssh and scp commands in the code as explained in Project_steps.docx document. Once prepared, one need to run only one script on local machine that is "maincode.sh" and the results will be available in local directory. I have tested fairness experiment on a different Geni testbed and the code is working without any other modifications except updating ssh and scp information. I hope anyone can replicate it easily.

References

[1] T. Høiland-Jørgensen, P. Hurtig, and A. Brunstrom, "The good, the bad and the wifi: Modern aqms in a residential setting," *Computer Networks*, vol. 89, pp. 90–106, 2015.

[2] R. Jain, "The art of computer systems performance analysis."

[3] A. N. Kuznetsov. tc-tbf - Linux Man Pages. [Online]. Available: https://www.systutorials.com/docs/linux/man/8-tc-tbf/

[4] M. Carbone and L. Rizzo, "Dummynet revisited," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 12–20, 2010.

[5] R. Jain, D.-M. Chiu, and W. R. Hawe, *A quantitative measure of fairness and discrimination for resource allocation in shared computer system*. Eastern Research Laboratory, Digital Equipment Corporation Hudson, MA, 1984, vol. 38.

[6] C. Smith-Salzberg, F. Fund, and S. Panwar, "Bridging the digital divide between research and home networks," 2017.