Check for updates

Latest updates: https://dl.acm.org/doi/10.1145/3342280.3342340

SHORT-PAPER

Towards an AQM Evaluation Testbed with P4 and DPDK

SÁNDOR LAKI, Eötvös Loránd University, Budapest, Hungary

PÉTER VÖRÖS, Eötvös Loránd University, Budapest, Hungary

FERENC FEJES, Eötvös Loránd University, Budapest, Hungary

Open Access Support provided by:

Eötvös Loránd University

PDF Download
3342280.3342340.pdf
16 December 2025
Total Citations: 14
Total Downloads: 414

Published: 19 August 2019

Citation in BibTeX format

SIGCOMM '19: ACM SIGCOMM 2019 Conference
August 19 - 23, 2019
Beijing, China

Conference Sponsors:
SIGCOMM

SIGCOMM Posters and Demos '19: Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos (August 2019)
https://doi.org/10.1145/3342280.3342340
ISBN: 9781450368865

Towards an AQM Evaluation Testbed with P4 and DPDK

Sándor Laki
ELTE Eötvös Loránd University
Budapest, Hungary
lakis@inf.elte.hu

Péter Vörös
ELTE Eötvös Loránd University
Budapest, Hungary
vopraai@inf.elte.hu

Ferenc Fejes
ELTE Eötvös Loránd University
Budapest, Hungary
fejes@inf.elte.hu

ABSTRACT

Active Queue Management (AQM) addresses the problem arising from using unnecessarily large, unmanaged buffers and thus it aims at improving network and application performance. AQM methods introduce different drop policies to proactively drop packets according to queue states and parameters. Though we are living in an AQM renaissance, comprehensive methodology and framework for performance evaluation under realistic traffic loads are still missing. With the advent of P4, description, validation and evaluation of AQM algorithms in a generic framework have become possible since the different drop policies applied by these methods can be implemented in ingress and/or egress control blocks of a P4 program. In this demo paper, we propose an AQM evaluation framework in which AQM algorithms described in P4 language can be executed and evaluated with a modified version of our DPDK-based P4 compiler called T4P4S in a testbed with realistic traffic mixes. The framework is demonstrated with some selected AQM algorithms in a network with a 5 Gbps bottleneck.

CCS CONCEPTS

• **Networks** → **Packet scheduling**; **Network performance evaluation**; **Programmable networks**.

KEYWORDS

AQM, Drop policy, Congestion Control, P4, Evaluation

ACM Reference Format:
Sándor Laki, Péter Vörös, and Ferenc Fejes. 2019. Towards an AQM Evaluation Testbed with P4 and DPDK. In *SIGCOMM '19: ACM SIGCOMM 2019 Conference Posters and Demos, August 19–23, 2019, Beijing, China.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3342280.3342340

1 INTRODUCTION

Active Queue Management (AQM) has gained attention in the recent years, since handling of overly large buffers and thus large queueing delays, known as the bufferbloat problem is needed in future access networks. In addition to classical AQM schemes [2, 3] like RED, WRED, delay-aware approaches like CoDel, PI controller-based approaches like PIE and PI2 and many others have also been proposed in the past decade. Active Queue Management (AQM) addresses the problem arising from using unnecessarily large and unmanaged buffers and thus it aims at improving network and application performance. These methods introduce different drop policies to proactively drop packets according to queue states and parameters. The drop policies of AQM methods can be implemented in the ingress and/or egress control blocks of a P4 program [1], assuming that required queue states are exposed to the P4 program (e.g. though metadata fields). Because of the number of congestion control algorithms, novel transport protocols like QUIC and further techniques like packet pacing applied in nowadays networks the evaluation of AQM algorithms has become very challenging. With the advent of P4 [1], description of different AQM algorithms in a generic way have become possible.

In this demo paper, we present the first steps towards a comprehensive AQM evaluation framework that relies on our DPDK-based P4 compiler and software switch called T4P4S [4] and traffic generator containers using iperf. The proposed framework can be used to generate responsive and unresponsive traffic with a wide range of congestion control algorithms and network settings (settings that are supported by the current Linux kernel). To support AQM evaluation the T4P4S compiler and software switch have been modified. Its run-to-completion execution model has been replaced by splitting the pipeline into two parts: 1) from parsing to ingress control and 2) egress control to deparsing. The two parts can be assigned to different CPU cores interconnected with a queue (ring buffer). In addition, the well-known *v1model* architecture of P4-16 have been extended with new standard metadata fields for providing information on the queue state: instantaneous queue latency, average queue size, current time. This meta information needs for the drop decisions of various AQM algorithms.

2 DEMO SCENARIOS

In the demo, we present three different scenarios, using the testbed depicted in Fig. 1. It consists of two separate machines (AMD Ryzen Threadripper 1900X 8C 3.8GHz, 128GB RAM): `TrafficGenerator` (TG node) and `P4Switch` (P4 node). Each one is equipped with a 1 Gbps NIC for management purposes and a dual 10 Gbps NIC (Intel 82599ES) for the measurements. On both machines, the two 10 Gbps interfaces are used with Intel DPDK drivers. The test traffic is generated by the iperf3 tool whose server and client components run in two docker containers on TG node. These containers are directly linked to the 10 Gbps physical interfaces and emulate the traffic source and sink nodes. In the sink container, a 5 ms propagation delay is emulated (Linux *tc* command) in the uplink direction. Though iperf can generate both unresponsive UDP and responsive TCP traffic, in this demo we only focus on responsive TCP flows. The number of active flows are varied from 10 to 40 (other settings are possible) while the applied congestion control algorithm can also be configured. In the demo we show results with Reno, Vegas and Cubic, but others like DCTCP with ECN marking or BBR can

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGCOMM '19, August 19–23, 2019, Beijing, China
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6886-5/19/08…$15.00
https://doi.org/10.1145/3342280.3342340

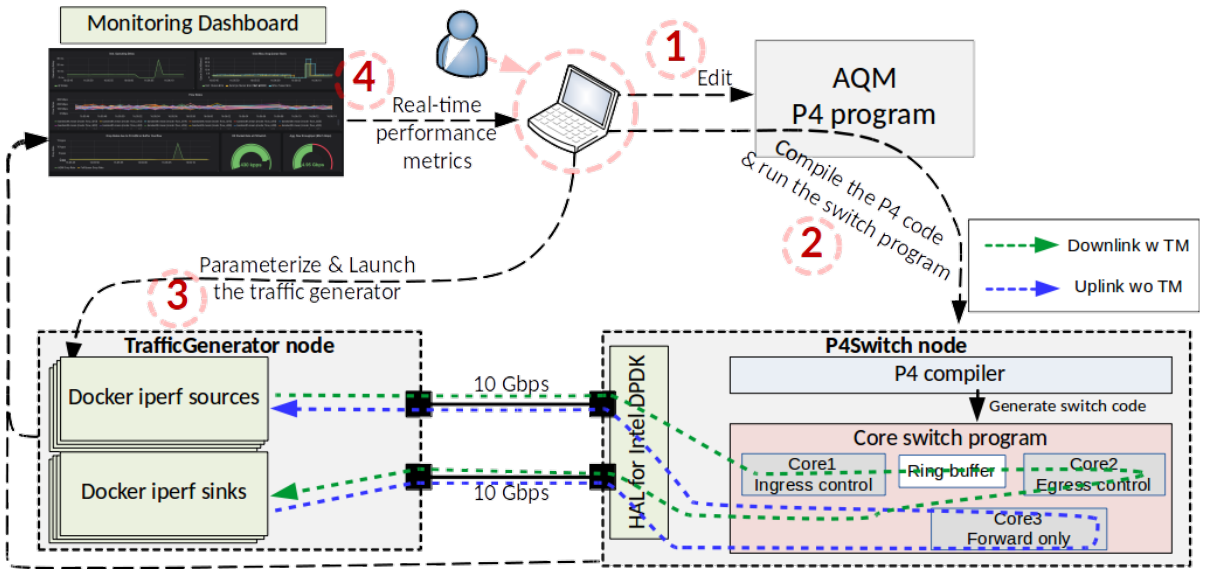

Figure 1: Demo architecture

also be examined with ease. The P4 node uses our modified T4P4S compiler and execute the compiled P4 AQM program.

For demonstration, we prepared two selected AQM programs (RED and PIE) and a simple FIFO as reference in P4-16 language using the extended v1model architecture[1]. The compiled T4P4S switch applies a rate limiter on the outgoing link in uplink direction (on Core 2) to emulate a bottleneck of 5 Gbps. Note that this parameter can be varied in the framework. As mentioned previously, P4 node applies the given AQM method in the uplink direction and a simple forwarding on the reverse direction (no bottleneck). The queue statistics are collected and stored in every one second continuously. All the performance metrics provided by the T4P4S core switch program and the throughput values of flows given by iperf are visualized in real-time on a dashboard. In the demo we cover three scenarios: a reference FIFO, PIE AQM and RED AQM. In all cases, the maximum buffer size is 16384 packets. Accordingly, we distinguish two kind of drops: ingress drop caused by the drop policy of the applied AQM method and drop caused by a full buffer.

Scenario 1 (Reference). In the model described in Sec. 1, a P4 program where the egress port is only set can emulate a simple FIFO queue discipline. The buffer is only limited by its maximum size. This scenario demonstrates how an unmanaged buffer behaves in the presence of large number of flows; the TCP sources fill the queue, resulting in large queueing delay. Compared to other scenarios much more packet drops (caused by full buffer) can be observed.

Scenario 2 (PIE AQM). The PIE AQM method [3] uses a PI controller and additional heuristics for controlling queue latency by periodically updating the drop probability to be applied. To this end, the P4 program uses the instantaneous queueing delay exposed by the T4P4S switch, previous states and parameters. States are stored in registers and the main parameters include the target delay that is set to 5ms in our demo, the update interval determining the frequency of the drop probability updates (20 ms in our case). The effect of AQM policy is already visible with small number of flows (e.g. 10 flows). The experienced queueing delay meets the target value in most of the cases during the course of the experiment (less than the delay in FIFO case). Temporal deviations can only be seen when new flows arrive in the system. One can also observe a fair resource share of the bottleneck capacity among flows.

Scenario 3 (RED AQM). The RED AQM method [2] uses the average queue size to determine the drop probability to be applied. In contrast to PIE, this probability is calculated for each packet. In the P4 implementation we use an exact match-action table that maps queue sizes given in KBytes to a range of 0-255 representing the drop probability (255 denotes probability 1.0). The RED profile stored in this table is filled by the control plane. Note that the control plane can update the RED profile in run-time by inserting/removing elements into/from the given table. In the demo, the drop probability is zero up to 3000 KBytes (min_{th}); it starts linearly increasing to $max_p = 0.1$ between 3000 and 8000 KBytes (max_{th}); above max_{th} all the packets are dropped with probability 1.0. Similarly to PIE, experienced queueing delay is much smaller, and the average queue size mostly meets min_{th}. Limited number of packet loss can only be experienced. The fairness among flows is ensured.

Though in this demo we only consider two AQM algorithms, the proposed framework can evaluate any AQM solutions that can be described in P4 language. The framework also supports the analysis of how AQM methods behave under various high-speed network traffic: 1) unresponsive and responsive flows, 2) various congestion control algorithms and 3) other mechanisms like packet pacing.

Acknowledgement. The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013). The authors also thank the support of Ericsson. S. Laki thanks the support of the ÚNKP-18-4 New National Excellence Program of the Ministry of Human Capacities.

[1]The P4 programs of the selected AQM algorithms are available at http://lakis.web.elte.hu/aqmdemo

REFERENCES

[1] Pat Bosshart et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM CCR* 44, 3 (2014), 87–95.
[2] Sally Floyd and Van Jacobson. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking (ToN)* 1, 4 (1993), 397–413.
[3] Rong Pan et al. 2013. PIE: A lightweight control scheme to address the bufferbloat problem. In *IEEE HPSR*. 148–155. https://doi.org/10.1109/HPSR.2013.6602305
[4] Péter Vörös et al. 2018. T4P4S: A Target-independent Compiler for Protocol-independent Packet Processors. (2018), 1–7.