A Congestion Control Independent L4S Scheduler

Szilveszter Nádas
Ericsson Research
Budapest, Hungary
szilveszter.nadas@ericsson.com

Gergő Gombos, Ferenc Fejes and Sándor Laki
ELTE Eötvös Loránd University
Budapest, Hungary

ABSTRACT

In the recent years a new Internet service called Low-Latency Low-Loss Scalable-Throughput (L4S) has been proposed to provide scalable sources with ultra-low latency over the Internet, while keeping the delay and resource sharing reasonable for flows with classic loss-based congestion control. At the same time both classic and scalable congestion controls are being evolved resulting in unfair resource share among flows using different congestion controls, even within the same congestion control family. The fairness is further degraded by heterogeneous RTTs over the real-world Internet. In this paper, we show that existing L4S schedulers are not able to handle this level of heterogeneity we forecast for the future Internet. To overcome this limitation, we propose Virtual Dual Queue CSAQM that extends the Core Stateless AQM (CSAQM) concept to support L4S requirements while keeping its precise in-network control of resource sharing. We have implemented the algorithm in our DPDK-based AQM evaluation testbed and demonstrated by measurements that it performs well even in those heterogeneous cases where mainstream L4S schedulers are sub-optimal.

CCS CONCEPTS

• **Networks → Packet scheduling**.

ACM Reference Format:
Szilveszter Nádas and Gergő Gombos, Ferenc Fejes and Sándor Laki. 2020. A Congestion Control Independent L4S Scheduler. In *Applied Networking Research Workshop (ANRW '20), July 27–30, 2020, Online (Meetecho), Spain.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3404868.3406669

ACM ISBN 978-1-4503-8039-3/20/07…$15.00
https://doi.org/10.1145/3404868.3406669

1 INTRODUCTION

In the past ten years, novel applications such as AR/VR, remote rendering, HD or holographic video conferencing, remote control and remote presence have emerged, requiring high bandwidth, low latency or both to provide good quality of experience. The protocols used over the Internet nowadays were not designed for fulfilling all these requirements. Latency is affected by many different factors, but queuing delay is its most varying component, influenced by the actual traffic load. Though different Active Queue Management (AQM) methods have been introduced in the past decade [22, 24] to avoid buffer bloating [14] and thus reduce queueing latency, the resulting queueing delay is still in the order of the typical round trip time (RTT). Most of these methods assume greedy behavior of classic congestion controls (CCs) like TCP Reno or CUBIC [16] that by design require buffers proportional to the bandwidth-delay product, and thus cannot provide ultra-low latency without sacrificing the bottleneck utilization. In addition to classic solutions, with Data Center TCP (DCTCP) [3], a new "scalable" class of congestion control was introduced for high-speed environments. It exploits the benefits of Explicit Congestion Notification (ECN) mechanism and reacts proportional to the congestion level, leading to much faster control as rate scales. It achieves good utilization without filling buffers in the routers. Google's novel BBRv2 CC [9, 18] reacts to packet loss similarly to classic CCs; and it also implements a Scalable "DCTCP-inspired ECN" mechanism.

To support capacity-seeking applications that want both full link utilization and low queuing delay, a new Internet service called Low-Latency Low-Loss Scalable-Throughput (L4S) has been proposed [7] to provide scalable sources with ultra-low latency over the Internet, while coexisting classic flows experience reasonable delay and fair throughput share at the same time. [26] lists the requirements L4S sources must satisfy. To support the incremental deployment of L4S, the DualQ Coupled AQM concept has been proposed [11]. It maintains separate queues for L4S and Classic traffic, fairness among different flows is achieved by a coupled packet dropping/marking mechanism. DualPI2 AQM [2, 11, 25] has been introduced as a possible realization of this concept. It uses PI2 [10] as the coupled AQM and simple RED [13] as L4S AQM. It's performance has been shown in a thorough evaluation with DCTCP as L4S and Cubic as Classic sources. [23] further evaluates DualPI2 and identifies issues with large RTTs for the DCTCP implementation of Linux kernel 5.3. They perform experiments in the bandwidth range of 4-200 Mbps and RTT range of 5-100 ms.

In this paper, we show that though DualPI2 can provide fair resource sharing between L4S and Classic flows in several cases, it cannot cope with the wild conditions we forecast in the future Internet. We predict that in the Access-Aggregation Network (AAN), the CCs used by the flows and the RTTs will be much more heterogeneous than in data centers and other closed enterprise networks, where the whole infrastructure is controlled by a single entity. This is a challenging scenario even for different classic CCs [12]. It has been shown in [12, 20] that Core-Stateless AQM (CSAQM) can harmonize CCs, but it only supports a single queue (with shared delay) and thus cannot satisfy the L4S requirements. In addition to DCTCP and BBRv2, TCP Prague [6] is also a candidate to comply with the L4S requirements. We plan to extend our analysis to it in a later paper.

In this paper, we extend the CSAQM concept [20] to dual queues. The proposed Virtual Dual Queue-CSAQM (VDQ-CSAQM) relies on the Virtual Queue (VQ) concept of [4], but extends it with 1) maintaining two VQs to achieve L4S goals, 2) a coupling mechanism between L4S and Classic queues, and 3) storing packet value history needed for core-stateless resource sharing. We demonstrate that VDQ-CSAQM is an efficient and versatile solution fulfilling L4S requirements. Through its in-network resource sharing mechanism, it can also improve fairness among flows with heterogeneous RTTs and heterogeneous CCs, while supporting a rich set of resource sharing policies, not only fair sharing.

2 PPV OVERVIEW

The Per Packet Value (PPV) based resource sharing [21] extends the idea of core stateless resource sharing solutions like [8, 27] by marking each packet with a continuous value called Packet Value (PV). The main goal in a network element is to maximize the total aggregate PV of delivered packets. The resource sharing procedure is composed of two phases: **1)** Packet marking at network edge; **2)** Packet scheduling and dropping based on the PV at the bottlenecks.

1) The goal of packet marking is to assign a PV to each packet based on the operator policy and the measured traffic rate R of the *flow* it belongs to. The PV represents the potential of the packet to get through the network, but the transmission probability also depends on the congestion level of the network. If the network is highly congested packet with high PVs might be dropped, while in case of moderate congestion even packets with low PV get through.

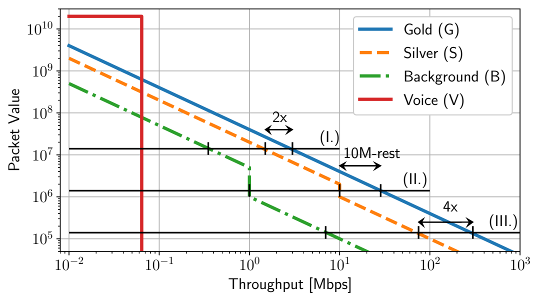


Figure 1: TVF examples, log-log scale.

To achieve this goal, packets are marked at the edge of the network by using the resource sharing policy of the operator described by a Throughput-Value Function (TVF) (denoted by $TVF(.)$). The marker assigns PVs to packets, such that the rate of packets of the given flow with PV larger than x is $TVF(x)$. The packet marker is implemented as follows. The flow rate R is measured (on a short timescale), the assigned PV is $TVF(x)$, where x is a uniformly distributed random number in $[0, R]$. The same packet marking algorithm is applied for all flows independently.

Fig. 1 shows examples for different operator policies expressed as TVFs that describe conditional weighted resource sharing between three classes, Gold, Silver, and Background while Voice has strict priority up to 64 Kbps. The intersection of the TVFs with horizontal lines representing different congestion levels (i.e. CTVs, see below) defines the desired throughput of the classes at the given congestion level. Until Silver flows reach 10 Mbps, Gold flows get twice the throughput of Silver ones (I.). When the throughput of Silver flows is above 10 Mbps, Gold flows get four times the throughput (III.). In between, Silver flows get 10 Mbps, and Gold flows get the rest (which will be between 20 Mbps and 40 Mbps) (II.). For this range of congestion levels, the Silver policy implements a rate limiter at 10 Mbps.

2) Bottleneck AQM treats packets solely relying on the carried PVs. It does not rely on any information about the flows, not even the number of flows. Each such node aims at maximizing the total amount of PV transmitted over the shared bottleneck. To this end, scheduling algorithms of different complexity can be used, including algorithms that drop the packet with the smallest PV (even from the middle of the buffer) when the buffer length is too long [21] or a congestion control independent AQM algorithm called CSAQM [20]. CSAQM maintains a Congestion Threshold Value (CTV) that reflects the actual congestion level and ECN-marks or drops all packets with PV smaller than CTV. Note that the amount of high and low PV packets in different flows determines the resource share between them. As a result, flows with larger share of high PV packets (above the CTV) receive more throughput.

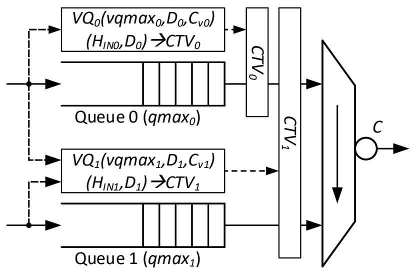

$VQ_0(vqmax_0, D_0, C_{v0})$
$(H_{IN0}, D_0) \rightarrow CTV_0$
CTV_0
Queue 0 ($qmax_0$)
CTV_1
C
$VQ_1(vqmax_1, D_1, C_{v1})$
$(H_{IN1}, D_1) \rightarrow CTV_1$
Queue 1 ($qmax_1$)

Figure 2: Architecture of VDQ-CSAQM Scheduler

3 VIRTUAL DUAL QUEUE CSAQM

Fig. 2 provides an overview of the proposed Virtual Dual Queue Core-Stateless AQM (VDQ-CSAQM). It maintains two physical FIFO and two virtual FIFO queues. In contrast to traditional virtual queues (VQs) we do not only count the number of bits but also store packet information in FIFO order in the VQs. Each incoming packet is first classified as L4S or Classic traffic that can be done by, e.g., using ECT(0) or Non-ECT markings for Classic packets, while ECT(1) to identify L4S traffic as in [7]; or using the DSCP field. After classification, a packet arrives either to the low delay *L4S Queue* (Queue 0) or the higher delay *Classic Queue* (Queue 1). Packet information including size and packet value is also enqueued to VQ_0 and/or VQ_1. One can observe that VQ_0 only stores information about L4S traffic while VQ_1 about both L4S and Classic flows, coupling the two traffic classes. The VQs have a slightly smaller rate than the physical bottleneck rate (C).

If the relevant Physical Queue or any related VQ is full, the packet is dropped upon arrival. Based on the packet sizes and PVs stored in the VQs, the Congestion Threshold Values CTV_0 and CTV_1 are periodically computed and then applied as simple filters to drop or ECN mark packets with PVs below any relevant CTV, at the dequeue phase. The algorithm is detailed below.

We encode PV logarithmically to 2 octets ($v = \lfloor \log(PV)/\log(PV_{max}) \times 65535 \rfloor$), $PV = 0$ is coded to $v = 0$. The bottleneck capacity is C, the service rate of the VQs are C_{v0} and C_{v1}, where $C_{v0} \leq C_{v1} \leq C$ ($C_{v0} = 0.9C$ and $C_{v1} = 0.984C$ are used in this paper). The length of Queue 0 and Queue 1 are $qmax_0$ ($C \cdot 50$ ms) and $qmax_1$ ($C \cdot 200$ ms), while the length of the VQs are $vqmax_0$ ($=qmax_0$) and $vqmax_1$ ($=qmax_0 + qmax_1$). The target delay for the VQs are D_0 (1 ms) and D_1 (20 ms). We also maintain the PV composition of each VQ_i in a histogram called H_{INi}, counting the number of packet bits in VQ_i for all possible coded PVs.

Packet Arrival: Let p be the packet arrived; v its coded PV; ps its size in bits; and after classification it shall be enqueued in Queue i (0-L4S;1-Classic traffic). If there is room in the physical Queue i ($qlen_i + ps \leq qmax_i$) and in all the related virtual queues ($vqlen_j + ps \leq vqmax_j, \forall j \geq i$) to store the incoming packet, the packet is enqueued, otherwise it is dropped[1]. In the normal case when all flows are responsive and the maximum size of the buffer is reasonable, packet drop is not triggered by packet arrival, and packet ECN-marking/dropping is likely only happens at dequeue phase, based on the calculated CTV(s). When a packet is enqueued, its metadata (v, ps) is also added to $VQ_j, \forall j \geq i$. The VQ length ($vqlen_j$ += ps) and the associated PV histogram ($H_{INj}(v)$ += ps) are also updated accordingly.

Dequeue Phase: The dequeue step is initiated when the bottleneck link becomes idle. VQD-CSAQM applies strict priority scheduling between the two physical queues: if there is a packet in Queue 0 we dequeue that first, otherwise we try to dequeue from Queue 1. Assuming that packet p with PV v is dequeued from Queue i, we check whether packet p is to be acted upon by the AQM: if $v \geq CTV_j, \forall j \geq i$, we transmit p unchanged. Otherwise, if packet p is ECT, we mark it as ECN CE and transmit it; if it is Non-ECT, we simply drop it and dequeue the next packet (if any). Then the related VQs also need to be updated.

Virtual Queue maintenance: VQs contain packet metadata (v, ps) in a FIFO buffer (see *Packet Arrival*). Whenever we transmit a packet, we remove exactly $ps_{vi} = ps \cdot C_{vi}/C$ bits from all VQ_is. We may have to remove more than one packet data record and/or remove a fraction of a packet by decreasing its size if needed (instead of dropping all packet information). The PV histogram and the VQ length are then updated as $H_{INi}(v')$ −= ps' and $vqlen_i$ −= ps' for all the packet records removed/decreased from VQ_i, where v' is the PV of the record, and ps' is its size or the amount it is decreased by. We continue serving non-empty VQs by C_{vi} rate, even when the Physical Queues are empty.

Maintain CTVs for AQM action: In every $T_{update,ctv}$ (10 ms by default) interval the CTVs are recalculated from the PV histograms associated to the virtual queues. The queue length threshold for VQ_i is calculated as $l_i = C_{vi} \times D_i$. If the actual queue length is smaller than or equal to the threshold ($vqlen_i \leq l_i$), CTV_i is set to 0. Otherwise, starting from the maximum coded PV ($V_{max} = 65535$) and decreasing the coded PV (v), the bits of H_{INi} are accumulated until we reach l_i. The smallest v, where l_i is not yet exceeded, determines the (coded) CTV_i, i.e. $CTV_i = \text{argmin}_v \sum_{j=v}^{V_{max}} H_{INi}(j) \leq l_i$.

VDQ-CSAQM extends CSAQM [20] and it provides the L4S problem with a novel solution that ensures fair resource sharing among flows whose packets may be stored in different physical queues, depending on their traffic class. It supports ECN in both queues, packet marking/dropping at the head

[1]In contrast to CSAQM, PV-based arrival drop [20] is not possible with VDQ-CSAQM, because the packet to be dropped from the VQ might have been already transmitted. However, we have not seen arrival drops except in extreme cases (neither for CSAQM nor for VDQ-CSAQM).

of the queue is based on two CTVs calculated from internal states of the two virtual queues (e.g., stored bits, packet value distribution). AQM decision is now implemented at dequeue phase, but it can easily be preformed upon arrival instead, if necessary. In addition to these improvements, it keeps the advantages of the original CSAQM: 1) It has no assumption on congestion control (in neither queue) other than responding to the respective congestion signals. 2) It has a simple, clean control loop by design and much less (and much more general) parameters to tune than (Dual Queue) PI2. 3) It works well in environments with heterogeneous RTTs and CCs. 4) Furthermore, it can implement a rich set of resource sharing policies beyond equal sharing as it keeps the resource sharing vocabulary of PPV. 5) It performs especially well with short flows, it does not drop/mark packets until the fair share of a flow is reached.

Our design goal in VDQ-CSAQM is to keep the computational complexity of the AQM logic low (only comparing PV to CTVs), enabling its implementation in the packet forwarding fast path, e.g., using P4 [5]. Though the update and maintenance of CTVs result in more complex computations, they are only executed periodically and can be offloaded to the slow path (e.g., a background process, switch CPU, remote SDN controller). Note that VQ implementations in programmable switches may require simplifications and approximate algorithms, while the strict priority scheduling applied in VDQ-CSAQM is supported by most switches and possible targets. Deployment aspects are discussed in [12].

4 EVALUATION

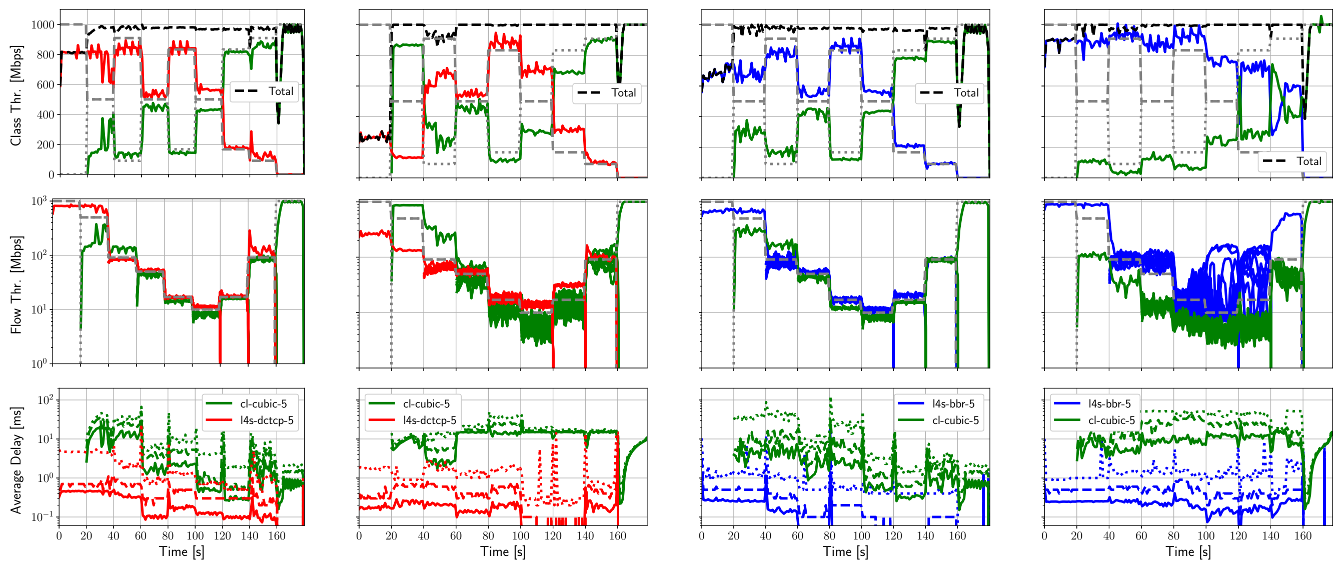
We implemented both DualPI2 AQM and VDQ-CSAQM into our DPDK-based AQM evaluation framework and bottleneck emulator. Our testbed consists of three Xeon E5-1660 v4@3.2GHz servers, with 64Gb RAM and Intel XL710 40GbE Ethernet NIC. They run Ubuntu Server 18.04 and are connected in a chain topology. The two ends, the sender and the receiver, use the BBRv2 alpha kernel (5.4.0-rc6) [1] including the latest improvements for BBRv2 ECN support. The middle node acts as a router, executes the DPDK-based AQM implementation and emulates the bottleneck. For generating BBRv2, DCTCP and CUBIC traffic we use iperf2 [15]. The propagation RTT is emulated with Linux tc netem [17] by delaying ACKs on the receiver side. We use the default DualPI2 parameters [11] (RTT_{max}=100ms, RTT_{typ}=15ms, $target_{PI2}$=15ms, RED_{minth}=475us, RED_{maxth}=1ms and the coupled size of the queue is 250ms). We use the defaults in Section 3 for VDQ-CSAQM. ECN is always enabled for BBRv2, but in the classic queue we configured the AQM to drop BBRv2 packets (instead of marking them), to trigger classic CC response by BBR there.

In all the scenarios, the bottleneck capacity is set to 1 Gbps. The propagation RTT is 5 ms and the applied TVF is Gold by default. We continuously collect the following statistics in the AQM implementation; per flow the total transmitted bits, ECN marked bits, and dropped bits; per connection class the histogram of the packet sojourn times (0.1 ms resolution); and the bottleneck utilization. We log these every 1 s, which allows creating time-series plots.

We analyze selected results in this article, all results are available at [19]. The analyzed results looked similar even when re-running the test cases.

Dynamic traffic, equal RTT (5 ms): We compare the performance of the two AMQs under changing traffic intensities, using various CCs. In case of VDQ-CSAQM all flows use Gold TVF, meaning equal desired resource sharing. The experiment consists of nine phases varying the number of L4S and Classic flows. Each phase lasts 20 seconds and the number of L4S-Classic flows are the following in order: 1-0;1-1;10-1;10-10;50-10;50-50;10-50;1-10;0-1. Classic flows use CUBIC CC, while the L4S CC is either BBRv2 or DCTCP. The investigated scenarios are displayed in Fig. 3. The top graphs depict the total (black dashed), the aggregated L4S (DCTCP-red, BBRv2-blue) and Classic (Cubic-green) throughput curves over time. The desired throughput for the two classes is illustrated by gray curves (L4S-dashed, Classic-dotted). The throughput curves of each TCP flow are shown in the middle, the fair share is also denoted by gray. The bottom graphs illustrate the average (solid), the 90th percentile (dashed) and the maximum (dotted) of observed queueing delays for both L4S and Classic buffers.

In Figs. 3a and 3b DCTCP is used as L4S congestion control. One can observe that both AQMs provide good flow fairness even among different traffic classes if the number of flows is large (60s-160s). In the first phase, however, a single DCTCP flow is not able to fully utilize the bottleneck link. With VDQ-CSAQM, 0.8Gbps flow throughput can be achieved, but DualPI2 results in even worse utilization of appox. 30%. This phenomenon does not appear with a single Classic flow (160s-180s). With one L4S and one Classic flows (20s-40s), both AQM methods show significant unfairness. DualPI2 results in an 1:6 L4S:Classic ratio, giving larger share to Classic traffic. In contrast, VDQ-CSAQM favors DCTCP, showing a slightly better 4:1 share. We beleive that this is mainly due to the difference in stric prio (VDQ-CSAQM) vs. time-shifted FIFO (DualPI2) scheduling. The 90th percentile and the average queuing delays are similar for both methods. DualPI2 results in slightly smaller delays for L4S traffic, but VDQ-CSAQM also provides average delays in sub-millisecond order. Except some temporal peaks, significant maximum delay can only be observed if the number of flows is limited (e.g., 0-40s), as the more flows arrive in the system, the maximum delay also goes below 1-2 ms with VDQ-CSAQM. On the other hand,



(a) DCTCP-Cubic; VDQ-CSAQM **(b) DCTCP-Cubic; DualPI2** **(c) BBRv2-Cubic; VDQ-CSAQM** **(d) BBRv2-Cubic; DualPI2**

Figure 3: Performance of VDQ-CSAQM (a,c) and DualPI2 (b,d) with dynamic traffic and equal RTT (5 ms).

the Classic traffic experiences larger delays with DualPI2 than with our core-stateless approach. DualPI2 keeps the delay around its target of 15 ms (see PI2 settings), while VQ-CSAQM can reduce the Classic delay when the number of flows increases.

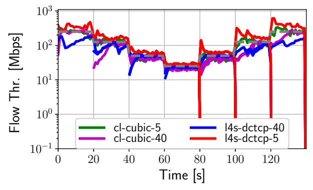
Fig. 3c and Fig. 3d show the same scenario with BBRv2 as L4S CC. One can observe that DualPI2 can control BBRv2 L4S traffic less, leading to increased queueing delays for both L4S and Classic classes, and significant unfairness between the two traffic classes. In general, if the number of BBRv2 flows is large, Classic traffic experiences a very low throughput share. With VDQ-CSAQM almost perfect fairness can be seen in most experiment phases, the largest deviance (7:3 L4S:Classic ratio) is shown for 1-1 L4S-Classic flows (20s-40s). In this case, DualPI2 results in 1:9 Classic:L4S share. In contrast to DualPI2, VDQ-CSAQM adapts ECN marking and drop rates for traffic with different CC behaviors, since it can automatically be learned from the observed packet value distributions. Note that VDQ-CSAQM can improve the fairness between various CCs, but the applied CC still impacts the observed resource share. For example, if a flow releases some throughput (e.g., due to a congestion signal), another flow using a CC with faster adaptation can occupy the freed resource, leading to higher average throughput than its ideal.

Dynamic traffic, heterogeneous RTTs: Since the propagation RTT on the Internet is heterogeneous, we consider a scenario where flows with different RTTs (5 ms and 40 ms) coexist in both classes. The experiment consists of 7 phases where the number of flows for each traffic class-RTT pair is varied: 1;2;5;10;5;2;1 (representing 1 L4S-5ms, 1 L4S-40ms, 1-Classic-5ms and 1-Classic-40ms flows in the first phase, etc.). Each phase lasts 20 sec. In Fig. 4a and Fig. 4b L4S flows use DCTCP CC. By comparing the results of DualPI2 and VDQ-CSAQM one can clearly see that VDQ-CSAQM provides better but not perfect fairness while DualPI2 cannot handle the heterogeneity in RTTs well. With VDQ-CSAQM, L4S traffic with 5 ms RTT occupies more bandwidth than the fair share, which is more visible when flows start leaving the system (red curve after 80s-140s). DCTCP with small RTT can adapt to the new conditions much faster, occupying the available capacity.

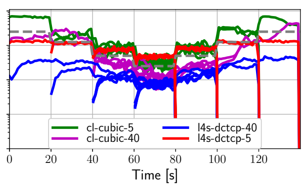
The use of BBRv2 as L4S CC results in even worse performance for DualPI2 (Fig. 4d). The unfairness between L4S and Classic classes is more significant than with DCTCP, and BBRv2 almost fully suppresses Cubic flows. On the other hand, VDQ-CSAQM provides good performance (Fig. 4c), as with DCTCP. Though BBRv2 flows occupy faster the freed resource than Classic ones, but also allow Cubic flows to increase their rate (80s-140s).

Though DualPI2 was designed to DCTCP-like L4S behavior, it is not prepared for heterogeneous RTTs, and the different behavior and faster reactions of BBRv2. Though VDQ-CSAQM does not provide perfect fairness, it is reasonable and can adapt to different CC behaviors by design. Also, DualPI2 has RTT dependent parameters, VDQ-CSAQM does not, which makes it more generally applicable.

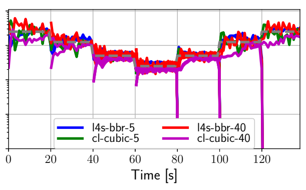
Resource sharing policies: VDQ-CSAQM relies on the PPV core-stateless resource sharing concept and has the advantage that in addition to fair share it supports more



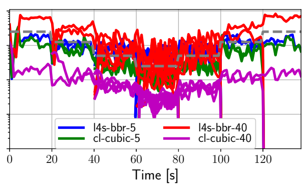
(a) DCTCP-Cubic; VDQ-CSAQM



(b) DCTCP-Cubic; DualPI2



(c) BBRv2-Cubic; VDQ-CSAQM



(d) BBRv2-Cubic; DualPI2

Figure 4: Performance of VDQ-CSAQM (a,c) and DualPI2 (b,d) with heterogeneous RTTs (5 ms and 40 ms).

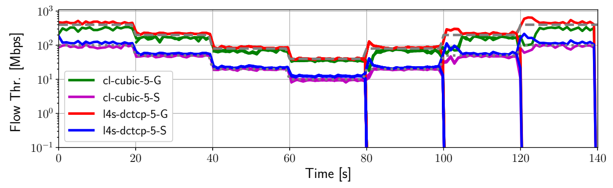


Figure 5: DCTCP vs. Cubic; Gold and Silver policies.

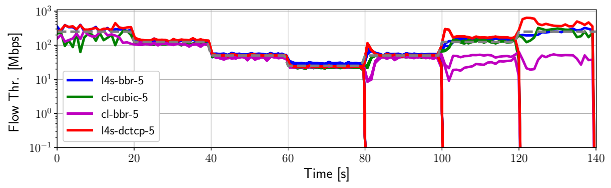


Figure 7: Full CC mix; vdq-csaqm.

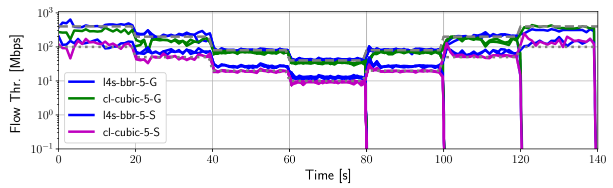


Figure 6: BBR vs. Cubic; Gold and Silver policies.

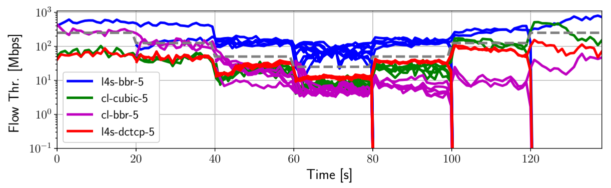


Figure 8: Full CC mix; DualPI2.

complex resource sharing policies. Figs. 5 and 6 illustrate a case with different resource sharing policies: Gold and Silver (see Section 2). We have 1-1-1-1 (L4S Gold-L4S Silver-Classic Gold-Classic Silver) flows in the beginning, then it is changed every 20 s to 2, 5, 10, 5, 2, 1 each. In the region when Silver flows experience at least 10 Mbps this means 1:4 (Silver:Gold) desired resource sharing (dashed gray curves). The desired resource share is realized well in both cases. For the DCTCP vs. Cubic case (Fig. 5), there is a somewhat larger deviation in the end (100-140s), while in the BBRv2 vs. Cubic case there is a somewhat larger deviation in the beginning (0-40s), which disappears as the number of flows increases. We conclude that VDQ-CSAQM keeps the rich resource sharing capabilities of CSAQM for dual queues as well.

Heterogeneous CCs: Figs. 7 and 8 depict a scenario with 4 different CCs coexisting in the same system. We change the traffic mix from 1-1-1-1 (L4S BBR-L4S DCTCP-Classic BBR-Classic Cubic) to 2-2-2-2, 5-5-5-5, 10-10-10-10 and back to 5, 2, and 1 each. For VDQ-CSAQM, one can observe reasonable fairness during the whole measurement (Fig. 7). Interestingly, the Classic BBRv2 flow cannot restore its throughput, when load decreases (between 100-140s). For DualPI2, the experienced fairness is much worse, since L4S BBR flows take the most throughput (Fig. 8), ten times more than the desired fair share in some cases. The behavior of Classic BBRv2 flows is again very interesting; at the start they take a high share, while in the end they have the lowest share. It is especially interesting to compare the 0-20s and 120-140s intervals, when the traffic mix is the same (1 flow from each class), but the resource share is significantly different. We think that BBR's model-based CC cannot tolerate the highly changing conditions of this scenario. The in-network resource sharing mechanism of VDQ-CSAQM has proved to work well even in this wild environment.

5 DISCUSSION

We have demonstrated that the Virtual Dual Queue Core Stateless AQM (VDQ-CSAQM) performs well even in cases with heterogeneous congestion controls and RTTs, where the existing DualPI2 scheduler has proved to be suboptimal. Google's novel BBRv2 congestion control changes the CC landscape, and CCs used over the Internet are being evolved with unprecedented speed. In this wild environment, compatibility of CCs even within the same CC family (either classic or scalable) cannot be expected. CSAQM provides good coexistence even for incompatible congestion controls. VDQ-CSAQM keeps this property and it extends CSAQM for the dual queue L4S case. By utilizing virtual queues of different speeds, it can provide very small delay for L4S flows, while keeping the bottleneck utilization reasonable. We propose to deploy VDQ-CSAQM in Access Aggregation Networks.

ACKNOWLEDGEMENT

The authors thank the support of Ericsson Hungary Ltd. and the following grant: Project no. ED_18-1-2019-0030 (Application-specific highly reliable IT solutions) has been implemented with the support provided from the NRDI Fund of Hungary, financed under the Thematic Excellence Programme funding scheme. The research of S. Laki was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences and the ÚNKP-19-4 New National Excellence Program of the Ministry for Innovation and Technology. The authors thank the anonymous reviewers for their suggestions on how to improve clarity of the paper.

REFERENCES

[1] [n.d.]. TCP BBR v2 Alpha/Preview Release, In https://github.com/google/bbr/tree/v2alpha. *checked 2020-01.*

[2] Olga Albisser, Koen De Schepper, Bob Briscoe, Olivier Tilmans, and Henrik Steen. 2019. DUALPI2 - Low Latency, Low Loss and Scalable (L4S) AQM. In *Proc. Netdev 0x13*. https://www.files.netdevconf.org/f/febbe8c6a05b4ceab641/?dl=1

[3] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *ACM SIGCOMM 2010* (New Delhi, India). ACM, 63–74.

[4] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. 2012. Less is more: trading a little bandwidth for ultra-low latency in the data center. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 19–19.

[5] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.

[6] B Briscoe, K De Schepper, O Albisser, J Misund, O Tilmans, M Kuehlewind, and A Ahmed. 2019. Implementing the TCP Prague Requirements for Low Latency Low Loss Scalable Throughput (L4S). *Proc. Linux Netdev 0x13, March* (2019).

[7] Bob Briscoe, Koen De Schepper, Marcelo Bagnulo, and Greg White. 2019. *Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture*. Internet-Draft draft-ietf-tsvwg-l4s-arch-04. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-l4s-arch-04 Work in Progress.

[8] Zhiruo Cao, Ellen Zegura, and Zheng Wang. 2005. Rainbow fair queueing: theory and applications. *Computer Networks* 47, 3 (2005), 367 – 392. https://doi.org/10.1016/j.comnet.2004.07.018

[9] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van son. 2016. BBR: Congestion-Based Congestion Control. *ACM Queue* 14, 5, Article 50 (Oct. 2016), 34 pages. https://doi.org/10.1145/3012426.3022184

[10] Koen De Schepper, Olga Bondarenko, Ing-Jyh Tsang, and Bob Briscoe. 2016. PI2: A Linearized AQM for Both Classic and Scalable TCP *(ACM CoNEXT '16)*. ACM, 105–119.

[11] Koen De Schepper, Bob Briscoe, and Greg White. 2020. *DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)*. Internet-Draft draft-ietf-tsvwg-aqm-dualq-coupled-11. IETF. https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-aqm-dualq-coupled-11 Work in Progress.

[12] Ferenc Fejes, Gergő Gombos, Sándor Laki, and Szilveszter Nádas. 2019. Who will Save the Internet from the Congestion Control Revolution?. In *Workshop on Buffer Sizing*.

[13] Sally Floyd and Van son. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking* (1993), 397–413.

[14] J. Gettys. 2011. Bufferbloat: Dark Buffers in the Internet. *IEEE Internet Computing* 15, 3 (May 2011), 96–96. https://doi.org/10.1109/MIC.2011.56

[15] Vivien GUEANT. 2017. Iperf-Iperf3 And Iperf2 User Documentation. *Iperf. fr. Np* (2017).

[16] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Oper. Syst. Rev.* 42, 5 (July 2008), 64–74. https://doi.org/10.1145/1400097.1400105

[17] Stephen Hemminger et al. 2005. Network emulation with NetEm. In *Linux conf au*. 18–23.

[18] N. Cardwell et. al. 2019-03. An update on BBR. slides-104-iccrg-an-update-on-bbr-00. https://datatracker.ietf.org/doc/slides-104-iccrg-an-update-on-bbr/

[19] Szilveszter Nádas, Gergő Gombos, Ferenc Fejes, and Sándor and Laki. 2020-06. A Congestion Control Independent L4S Scheduler - Measurement results. In *http://ppv.elte.hu/cc-independent-l4s*.

[20] Szilveszter Nádas, Gergő Gombos, Péter Hudoba, and Sándor Laki. 2018. Towards a Congestion Control-Independent Core-Stateless AQM. In *ANRW '18* (Montreal, QC, Canada). 84–90. https://doi.org/10.1145/3232755.3232777

[21] Szilveszter Nádas, Zoltán Richárd Turányi, and Sándor Rácz. 2016. Per Packet Value: A Practical Concept for Network Resource Sharing. In *IEEE Globecom 2016*.

[22] Kathleen Nichols and Van son. 2012. Controlling Queue Delay. *Commun. ACM* 55, 7 (July 2012), 42–50. https://doi.org/10.1145/2209249.2209264

[23] Dejene Boru Oljira, Karl-Johan Grinnemo, Anna Brunström, and Javid Taheri. 2020. Validating the Sharing Behavior and Latency Characteristics of the L4S Architecture. *ACM SIGCOMM Computer Communication Review* 50, 2 (2020).

[24] Rong Pan, Preethi Natarajan, Chiara Piglione, Mythili Suryanarayana Prabhu, Vijay Subramanian, Fred Baker, and Bill VerSteeg. 2013. PIE: A lightweight control scheme to address the bufferbloat problem. In *IEEE HPSR*. 148–155. https://doi.org/10.1109/HPSR.2013.6602305

[25] Koen De Schepper, Olga Bondarenko, Ing-Jyh Tsang, and Bob Briscoe. 2015. *'Data Center to the Home': Ultra-Low Latency for All*. Technical report. RITE Project. http://riteproject.eu/publications/

[26] Koen De Schepper and Bob Briscoe. 2020. *Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay (L4S)*. Internet-Draft draft-ietf-tsvwg-ecn-l4s-id-10. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-ecn-l4s-id-10 Work in Progress.

[27] Ion Stoica, Scott Shenker, and Hui Zhang. 2003. Core-stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High-speed Networks. *IEEE/ACM Trans. Netw.* 11, 1 (Feb. 2003), 33–46. https://doi.org/10.1109/TNET.2002.808414