Towards a Congestion Control-Independent Core-Stateless AQM

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

Gergő Gombos, Péter Hudoba and Sándor Laki
ELTE Eötvös Loránd University
Budapest, Hungary
{ggombos,hudi89,lakis}@caesar.elte.hu

ABSTRACT

In this paper, we propose CSAQM, a novel approach of Active Queue Management (AQM) for network traffic with multiple Congestion Control algorithms. Our goal is similar to that of PI2 AQM that supports both Classic and Scalable TCP in a single queue. In contrast to existing solutions, CSAQM has two key advantages: 1) arbitrary congestion controls (including unresponsive flows) can also be handled and 2) it supports a rich set of resource sharing policies beyond equal sharing. CSAQM is based on the core-stateless Per Packet Value resource sharing concept and extends it with a congestion control-independent AQM. The performance of the proposed CSAQM and its behavior under various network conditions have been analyzed through extensive simulations.

CCS CONCEPTS

• **Networks → Packet scheduling**;

KEYWORDS

Congestion Control; PPV; AQM; Resource Sharing

ACM Reference Format:
Szilveszter Nádas and Gergő Gombos, Péter Hudoba and Sándor Laki. 2018. Towards a Congestion Control-Independent Core-Stateless AQM. In *ANRW '18: Applied Networking Research Workshop, July 16, 2018, Montreal, QC, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3232755.3232777

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 the author(s) 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.
ANRW '18, July 16, 2018, Montreal, QC, Canada
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-5585-8/18/07…$15.00
https://doi.org/10.1145/3232755.3232777

1 INTRODUCTION

In the recent years several new Congestion Control (CC) and Active Queue Management (AQM) algorithms have been proposed to further improve both the throughput and latency of Internet applications. As the number of AQMs and CCs increases, it is more and more challenging to keep them compatible, especially when AQMs assume specific CC behaviors.

Most CCs used over the Internet are TCP-friendly, sharing bottleneck capacity in a fair way among flows. CUBIC [7] CC is the most commonly used one among them. Data Center TCP (DCTCP) [1] is not TCP friendly, but it provides much faster throughput convergence and smaller queuing delay utilizing Explicit Congestion Notification Congestion Experienced (ECN CE) signal in a (previously) non-standard compliant way. It is though too aggressive to co-exist with TCP-friendly flows of today. The recently introduced BBR CC can be very unfriendly when there are multiple competing CUBIC flows [4].

We are living in an AQM renaissance mainly due to overly large buffers and thus large queueing delays, known as the bufferbloat problem [6]. In addition to classical AQM schemes like RED [5], delay-aware approaches like CoDel [11], PIE [12] and PI^2 [3] have been proposed in the past decade. Among these PIE and PI^2 were developed for shared bottleneck buffers, where dropping is only possible upon arrival. PIE assumes classical TCP CC and it determines packet drop/mark probability based on this assumption. PI^2 extends PIE by handling DCTCP and CUBIC CCs in the same buffer, though it explicitly knows which flow is using DCTCP CC (using ECT classification) and which is not.

The Per Packet Value (PPV) concept [10] provides a core-stateless way to control resource sharing among flows thereby it allows previously incompatible CCs share the same bottleneck. In addition to providing bounded delays like the above AQMs, it also allows throughput differentiation of different flows. It is based on packet drop though, so it is not trivial to apply it for ECN marking. PVPIE [9] combines PIE AQM with the PPV concept. It has been demonstrated to harmonize incompatible CCs and it is trivial to extend

RIGHTSLINK

its behavior to ECN marking. Though, it still keeps PIE's assumptions on CC behavior and PIE's control loop, which is hard to tune for all scenarios and new CCs.

In this paper, we propose CSAQM, a simpler, controllerless alternative for PVPIE, which can also perform ECN CE marking. It can react much faster to bursts, it has a single, easy to set parameter. Its only assumption on congestion control is that when the CC marks ECN Capable Transport (ECT) on the packets, it keeps the ratio of ECN CE marked packets reasonable. (The CCs are not required to keep the ratio of dropped packets small.) Even when the above assumption is not true, which is clearly a hostile behavior, CSAQM can still maintain a stable behavior and control responsive ECT flows by ECN CE signal only. Finally, CSAQM keeps the resource sharing property of the PPV concept, also supporting a wide range of flexible operator policies.

2 SYSTEM MODEL

The PPV concept [10] extends the idea of core stateless resource sharing solutions like [2, 13] by marking each packet with a continuous value called Packet Value (PV). The network aims at maximizing the the total aggregate PV delivered. The system model of PPV is split into two phases: 1) Packet marking at network edge; 2) Packet scheduling and dropping based on the Packet Value at resource nodes in the middle of the network.

First, packets are marked at the edge of the network by using the resource sharing policy of the operator. Operator policies are described by Throughput-Value Functions (TVFs) (marked by $V(.)$), that basically defines the PV distribution of a flow for any sending rate. Specifically, for any throughput value b, the traffic up to b shall receive a PV of $V(b)$ or higher. Accordingly, at high congestion only packets with high PVs are transmitted, more precisely packets with PV above a given Congestion Threshold Value (CTV). Note that the amount of high and low PV packets determines the resource share between various flows, resulting in that at high congestion, flows with larger share of high PV packets receive more throughput.

Fig. 1 shows 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) 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 4 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

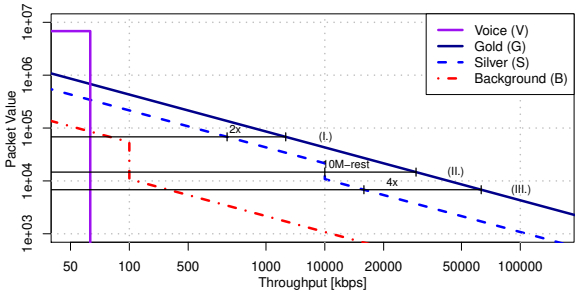


Figure 1: TVF examples, log-log scale.

the Silver policy pretends the behavior of a rate limiter at 10 Mbps. The figure also shows a background traffic class that has a small share of moderate PV to keep connectivity going, but receives larger bandwidth only if there is little or no congestion.

Second, resource nodes in the middle of the network treat packets without maintaining flow-states, solely relying on the carried PVs. Each such node aims at maximizing the total amount of value transmitted over the shared bottleneck. To this end, [10] proposes a simple scheduling algorithm that drops the packet with the smallest PV (even from the middle of the buffer) when the buffer length is too long. In [9] the PVPIE AQM is introduced which adapts PIE behavior to the PPV concept. Similarly to PIE and PI^2, it also uses a controller to determine the drop probability from the current and target queuing delay values, but instead of applying packet drops uniformly at random, the calculated probability is transformed to a CTV by taking the empirical probability distribution of PVs carried by the packets observed in a given time frame. Then all the packets with PV less than this CTV are dropped, resulting in the expected drop probability and resource sharing in stable periods.

3 CORE-STATELESS AQM

In this paper we propose Core-Stateless AQM (CSAQM) that typically marks/drops packets at the head of a single FIFO queue based on a maintained CTV. In contrast to PI^2 this behavior does not require any flow classification, but still results in different empirical mark/drop probabilities for different CCs. Furthermore, CSAQM can implement a rich set of resource sharing policies beyond equal sharing as it keeps the resource sharing vocabulary of PPV without the need of complex control loop of PVPIE, having much simpler and cleaner design.

Fig. 2 provides an overview of the proposed CSAQM method. At *Packet Arrival* the method first decides whether the incoming packet is enqueued or dropped, based on the *maximum queue length* and the Packet Value composition

RIGHTSLINK

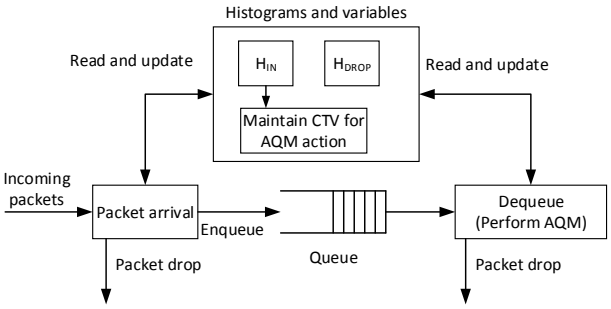


Figure 2: Architecture of CSAQM Scheduler

stored in the histogram H_{IN} which maintains the total number of packet bits in the queue for all possible PVs. If the queue is full, packets already in the buffer with smaller PV than the one of incoming packet needs to be dropped to make space, if possible. This is achieved by marking these packets as dropped by moving bits from histogram H_{IN} to a drop histogram H_{DROP} that represents the packets to be dropped at dequeue phase. 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 only happens at dequeue, based on the CTV. At *Dequeue* phase, CSAQM first drops packets based on H_{DROP}, then based on the maintained CTV it decides whether to mark/drop on the outgoing packet. CTV is updated periodically based on a *queue length threshold* and H_{IN}. If the current queue size is below the *queue length threshold*, all the packets are sent out without any modification (no ECN marking/dropping). One can observe that the meaning of this threshold parameter is different than the target delay of PIE, PI^2 and PVPIE.

The detailed operation of the resource node is as below. We code PV logarithmically to 2 octets ($v = \lfloor \log(PV)/\log(PV_{max}) \times 65535 \rfloor$), v denotes the unsigned integer representing the coded PV, $PV = 0$ is coded to $v = 0$. The bottleneck capacity is C. The *maximum allowed queuing delay* is D_{max} and the *queueing delay threshold* is D. Let p be the packet arrived; v its coded PV and ps its size in bits. We calculate the actual queue length as $q = \sum_i H_{IN}(i)$. The free space in the queue is calculated as $s = C \times D_{max} - q$.

Packet Arrival: Upon packet arrival, if there is enough space, i.e. $s \geq ps$, the packet is inserted to the end of the queue and H_{IN} is updated: $H_{IN}(v)$ += ps. If there is not enough room to store p in the queue ($s < ps$), CSAQM *marks for dropping* $ps - s$ bits with the smallest coded PVs (w where $w < v$) until either $s = ps$ or we cannot drop more (as all $H_{IN}(w) = 0$, where $w = 0 \ldots v-1$). If we succeed, p is enqueued (H_{IN} is also updated: $H_{IN}(v)$ += ps), otherwise it is dropped.

Marking for dropping: When maximum b bits of coded PV v is to be dropped from the queue, it is checked whether $H_{IN}(v) > 0$ and if it is, $d = \min(H_{IN}(v), b)$ amount of bits are marked for dropping by $H_{DROP}(v)$ += d and $H_{IN}(v)$ −= d.

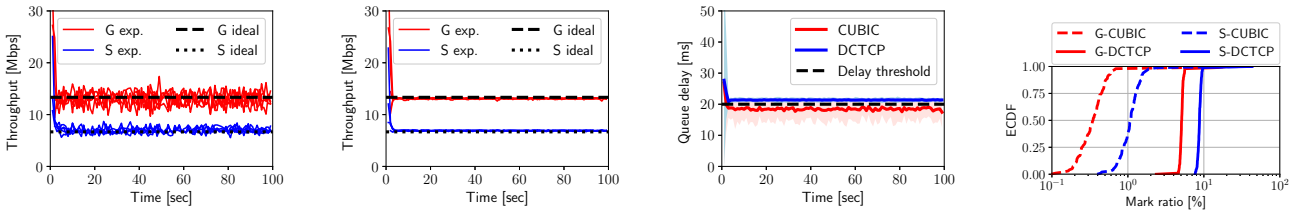
Dequeue: Dequeue is initiated by the bottleneck node. The first packet p is dropped until $H_{DROP}(v) = 0$. When packet p is dropped, it is removed from the queue and $H_{DROP}(v)$ −= ps is updated. Then if $H_{DROP}(v) < 0$, the histograms are updated: $H_{IN}(v)$ += $H_{DROP}(v)$ and then $H_{DROP}(v) = 0$. After that the head packet p is removed from the queue and H_{IN} is updated: $H_{IN}(v)$ −= ps. Then CSAQM check whether p is to be acted upon by the AQM, by checking whether $v < ctv$. If it is not we transmit it unchanged. If it is, then if the packet is ECT we mark it as ECN CE or drop it otherwise. Upon packet drop by the AQM we rerun the *Dequeue* phase until either we can transmit a packet or the queue becomes empty.

Maintain CTV for AQM action: Upon every packet dequeue, it is checked whether this function was run in the last $T_{update,ctv}$ time. If not then ctv is re-evaluated as follows. The queue length threshold is calculated as $l = C \times D$. If the actual queue length is smaller than or equal to the threshold ($q \leq l$), ctv is set to 0. Otherwise, starting from the maximum coded PV ($V_{max} = 65535$) and decreasing the coded PV (v), the bits of H_{IN} are accumulated until we reach l. The smallest v, where l is not yet exceeded determines the (coded) ctv, i.e. $ctv = \text{argmin}_v \sum_{i=v}^{V_{max}} H_{IN}(i) \leq l$.

4 EVALUATION

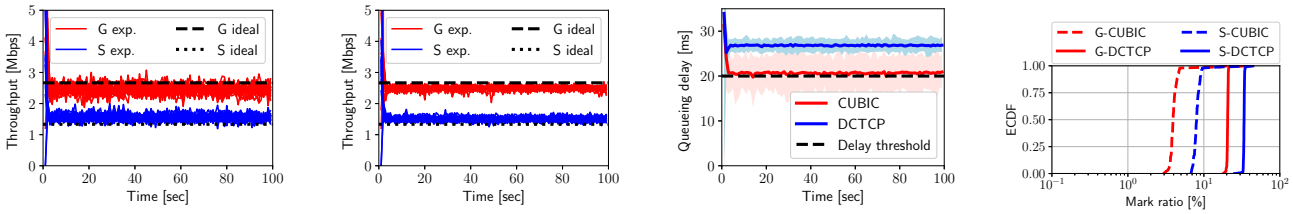
CSAQM described in Section 3 has been implemented in NS3 simulator [8]. In all the examined simulation scenarios, we consider a single downlink bottleneck where the same bottleneck buffer is shared by a number of *flows*. We also assume that there is no uplink bottleneck. The term *flow* refers to a piece of traffic to which an operator policy (expressed by a specific TVF) is applied. We perform packet marking per *flow* according to the Gold or Silver TVFs on Fig. 1. Sources use either CUBIC TCP [7] or Data Center TCP (DCTCP)[1] using the implementation of Linux Kernel 4.4 through NS-3 Direct Code Execution [14]. By default there is a single TCP connection per *flow*. Both TCP CCs are configured to be ECN capable. Simulations have been carried out with different bottleneck link capacities, C=40 Mbps being the default. The end-to-end RTT was also varied, the default is 10 ms. The default CSAQM parameters are the following: queueing delay threshold $D = 20$ ms, $D_{max} = \infty$, and $T_{update,ctv} = 1$ *ms*. In the rest of the chapter we only mention non-default simulation settings. We measure throughput, packet marking ratio, and average/minimum/maximum queueing delay in 1 s long intervals. Several simulation cases were inspired by and comparable to the Evaluation section in [3].

RIGHTSLINK



(a) Throughput - CUBIC (b) Throughput - DCTCP (c) Queueing delay (d) Packet marking ratio

Figure 3: Homogeneous CC (CUBIC or DCTCP): 5 Gold and 5 Silver TCP flows, C=100 Mbps, RTT=10 ms



(a) Throughput - CUBIC (b) Throughput - DCTCP (c) Queueing delay (d) Packet marking ratio

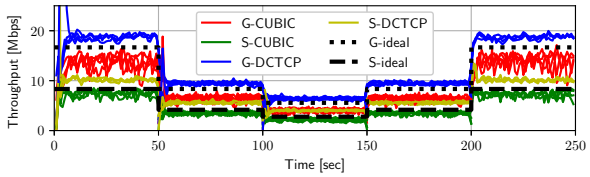
Figure 4: Homogeneous CC (CUBIC or DCTCP): 25 Gold and 25 Silver TCP flows, C=100 Mbps, RTT=10 ms

TCP flows with the same Congestion Control. We simulated four cases where a C = 100 Mbps bottleneck is being shared by TCP flows of the same CC. Fig. 3 depicts the results for 5 Gold and 5 Silver TVF flows. We run two separate simulations with CUBIC and DCTCP CCs. The figure displays throughput during the different simulations on separate subfigures for CUBIC and DCTCP, and aggregate the queueing delay and the packet marking ratios on single figures. The queueing delay range (between the min and max observed values) is also marked by a lighter shade (Fig. 3c). Fig. 4 depicts the results when 50 Gold and 50 Silver flows share the bottleneck in the same network. The throughput is close to the ideal in all four cases (Figs. 3a, 3b, 4a, and 4b) and it oscillates somewhat more for CUBIC CC. The queueing delay is close to the threshold after the initial transient, except on Fig. 4c for DCTCP flows, because DCTCP CC requires high marking rate for the lower per flow throughput experienced in this scenario. As depicted in Fig. 3d and 4d, DCTCP flows naturally experience higher marking probability than CUBIC (about the square root of CUBIC), without any assumption on CC behavior (in contrast to PI^2). In summary for these cases the system works as expected, providing proportionally fair resource sharing according to the predefined TVF-based operator policies.

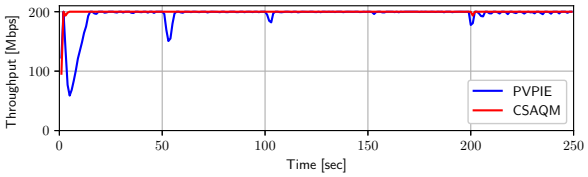
Note that it would be possible to adapt D with a slow controller to smaller values if one wants the delay to be closer to the threshold in all stable cases and the update speed of CTV can still be kept fast. That is outside our current scope though.

Varied traffic intensity with the coexistence of different CCs. In this case, TCP flows with different CCs share the C = 200 Mbps bottleneck. The RTT is 100 ms, each flow has 5 simultaneous TCP connections. There are 4 flow groups: Gold-CUBIC, Silver-CUBIC, Gold-DCTCP and Silver-DCTCP. The simulation starts with 4 flows in each group (4×4 total), then the number of flows is increased to 8×4 and 12×4 at 50 s and 100 s. We decrease it to 8×4 and 4×4 again at 150 s and 200 s. Fig. 5 depicts the experienced per flow and total throughput, and the queueing delay. It can be seen that DCTCP flows get slightly more throughput than CUBIC flows using the same TVF, though both are still close to the ideal. Note that similar results have also been presented for PI^2 in [3] where the resource sharing between competing CUBIC and DCTCP flows is closer to equal, but (as opposed to CSAQM) PI^2 is designed according to the CCs used and it has information about which packet belongs to which CC (used for determining the drop/mark probability to be applied). This scenario was also evaluated with PVPIE where the per flow throughput curves were slightly closer to the ideal with much worse transient behavior. Fig. 5b compares the utilization of CSAQM and PVPIE. There are large over- and under-shots for PVPIE (and utilization loss), while for CSAQM the utilization is consistent. Figs. 5c and 5d compare the queueing delay for the two AQMs and also illustrate

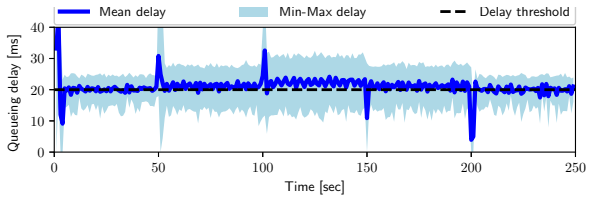
RIGHTSLINK



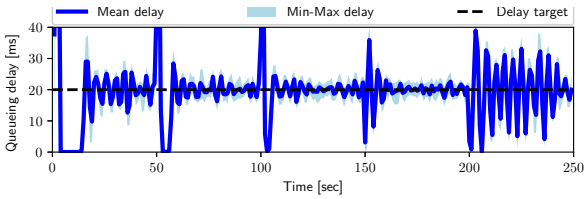
(a) Throughput with CSAQM



(b) Utilization with CSAQM and PVPIE



(c) Queueing delay with CSAQM



(d) Queueing delay with PVPIE

Figure 5: CSAQM and PVPIE comparison, varied traffic, mixed CCs, C=200 Mbps, RTT=100 ms

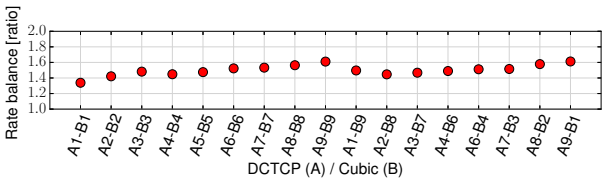


Figure 6: Throughput balance, C=40 ms, RTT=10 ms

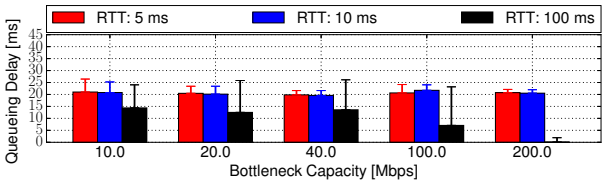


Figure 7: Average and maximum queuing delay.

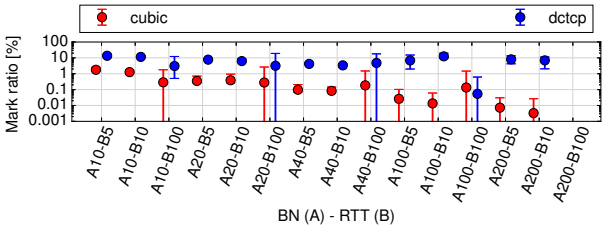


Figure 8: Marking ratio (25th perc., mean, 99th perc.)

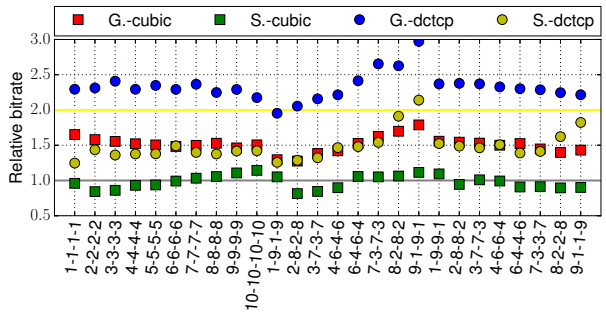


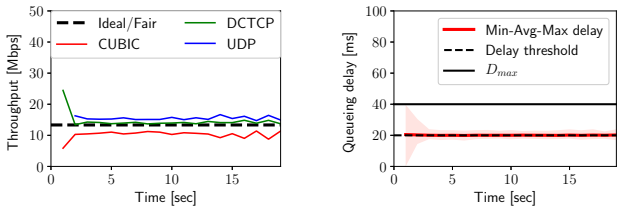
Figure 9: Relative bitrate. C=40 ms, RTT=10 ms.

the longer and higher oscillations for PVPIE. We believe that CSAQM is a better solution than PVPIE, because the improvement in the transient behavior is more important in practical cases than the drawback of the slightly worse steady state fairness.

Different CC flows, combination of different settings Fig. 6 depicts the DCTCP/CUBIC throughput ratio for 19 simulation cases. The number of Gold DCTCP (A) and Gold CUBIC (B) flows is varied as shown on the x axis. One can observe that the resource share is still reasonable with the DCTCP flows getting somewhat more throughput. Fig. 7 depicts the queueing delay for different bottleneck capacities and RTTs for 1 Gold DCTCP and 1 Gold CUBIC flows. Fig. 8 is the packet marking probability for 1-1 flows and varied bottleneck capacities (A) and RTTs (B). The results for both figures show good CSAQM algorithm behavior. (The C=200 Mbps and RTT=100 ms case was similarly degenerative in [3].)

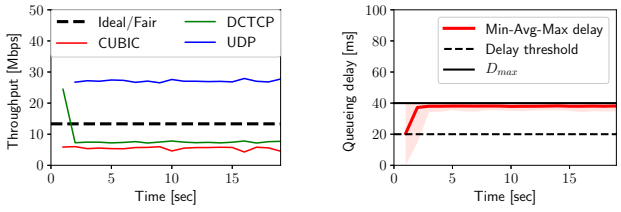
Different CCs and TVFs Fig. 9 shows the average relative bitrate per flow group for flows with different CCs and TVFs coexisting in the same network. The relative bitrate is the average throughput of a flow divided by the ideal throughput

RIGHTSLINK



(a) Throughput **(b) Queuing delay**

Figure 10: A system with unresponsive UDP flow



(a) Throughput **(b) Queuing delay**

Figure 11: A system with ECN-cheater UDP flow.

of a Silver flow. Thus the ideal value for Silver flows is 1 (gray line) and for Gold flows, 2 (yellow line). Each case is identified by a four tuple A-B-C-D representing the number of Silver-CUBIC, Silver-DCTCP, Gold-CUBIC and Gold-DCTCP flows in the simulation, respectively. One can observe that for cases where the number of different source types is the same, some deviation from the ideal curves can be recognized; both Gold and Silver DCTCP flows receive more from the shared resources than their ideal. This phenomenon is similar to the one seen in Fig. 5a, caused by the non-scalable property of CUBIC CC. CUBIC CC simply reduces its sending rate too aggressively in the presence of ECN CE and the freed capacity is then exploited by the DCTCP sources. For asymmetric cases, deviation from the ideal throughput is increasing as DCTCP flows are getting under and CUBIC flows are over represented in the traffic (e.g. 9-1-9-1 where the single Gold and Silver DCTCP flows get 1.5 and 2 times more throughput than their ideal, resp. However, it does not affect more the 9-9 Silver and Gold CUBIC flows that are still close to their ideal throughput.). Though the coexistence of scalable and classic CCs affects the resource sharing accuracy, resulting in deviations from the ideal share, but these differences are mainly caused by the conservative decrease policy of classic CCs.

Non-responsive flows We evaluated cases with an unresponsive UDP flow with Gold TVF included in the system where C=40 Mbps and RTT=10 ms. The sending rate of the UDP flow is 60 Mbps. There are 1-1 Gold CUBIC and DCTCP flows. Fig. 10 depicts a case with high D_{max}=40 ms, when the unresponsive UDP flow reports itself as Non-ECT. In this case, CSAQM drops the packets of the UDP flow with PV below CTV. The resource sharing is similar to what we have seen for cases with mixed CCs, the UDP share being close to the DCTCP share. Fig. 11 depicts an ECN-cheating scenario with D_{max}=40 ms where the UDP flow states that it is ECT capable, but it does not react on ECN CE. Consequently, it can grab a significant part of the capacity, because it fills the buffer space between D and D_{max} with its packets, which is a significant part of the bandwidth-delay product. Still it is the only flow experiencing actual packet loss after the initial transient (there is a small packet loss for the DCTCP flow in the first second). The TCP flows can still be congestion controlled by ECN marking only, because when dropping happens due to that the queue length has reached D_{max}, the smallest PV has never been associated to TCP packets. In this case, CSAQM based on D still marks the packets of both DCTCP and CUBIC flows below the calculated CTV, and both CCs can operate based solely on ECN markings. This property of the CSAQM is intriguing, but it still cannot provide the desired sharing when the aggressive and hostile UDP flow is present, though it can relax its effects on the system. In this case the results are quite sensitive to the setting of D_{max}. If D_{max} is too low then there are more and more losses also to TCP flows, even when there is no cheating flows. If it is set too high, the resource share of the TCP flows will be smaller and smaller in the presence of cheating flows.

5 CONCLUSIONS

In this paper we propose Core-Stateless AQM (CSAQM) that, in contrast to PI^2, is unaware of the congestion control used and it still works for both DCTCP and CUBIC TCP, even when these share the same bottleneck. CSAQM is based on the Per Packet Value core-stateless resource sharing framework and thus it can also implement a rich set of resource sharing policies beyond equal sharing for flows (even with multiple congestion controls coexisting in the same network). Based on the presented simulation results, CSAQM provides a controller-less alternative to PVPIE, converges much faster in transient situations and results in smaller queing delay fluctuations. We have also demonstrated that even when non-responsive flows fill the bottleneck queue, it can still provide loss-less congestion signal (ECN) to responsive flows. Based on these results we believe that CSAQM is applicable for a wide set of congestion control algorithms even future ones and its simple design enables practical implementations.

RIGHTSLINK

ACKNOWLEDGEMENT

The research was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013) and Ericsson Hungary Ltd. G. G. also thanks the support of the ÚNKP-17-3 New National Excellence Program of the Ministry of Human Capacities. S.L. also thanks the support of the ÚNKP-17-4 New National Excellence Program of the Ministry of Human Capacities.

REFERENCES

[1] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM '10)*. ACM, 63–74.

[2] 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

[3] Koen De Schepper, Olga Bondarenko, Ing-Jyh Tsang, and Bob Briscoe. 2016. PI2: A Linearized AQM for Both Classic and Scalable TCP. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies (CoNEXT '16)*. ACM, 105–119.

[4] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. PCC Vivace: Online-Learning Congestion Control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 343–356. https://www.usenix.org/conference/nsdi18/presentation/dong

[5] Sally Floyd and Van Jacobson. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking (ToN)* 1, 4 (1993), 397–413.

[6] 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

[7] 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

[8] Thomas R Henderson, Mathieu Lacage, George F Riley, C Dowell, and J Kopena. 2008. Network simulations with the ns-3 simulator. In *Sigcomm (Demo)*, Vol. 14.

[9] Sándor Laki, Gergő Gombos, Szilveszter Nádas, and Zoltán Turányi. 2017. Take Your Own Share of the PIE. In *Proceedings of the Applied Networking Research Workshop (ANRW '17)*. ACM, 27–32.

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

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

[12] 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

[13] 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

[14] Hajime Tazaki, Frédéric Uarbani, Emilio Mancini, Mathieu Lacage, Daniel Camara, Thierry Turletti, and Walid Dabbous. 2013. Direct code execution: Revisiting library os architecture for reproducible network experiments. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 217–228.

RIGHTSLINK