Interactions between Congestion Control Algorithms

Belma Turkovic*, Fernando A. Kuipers† and Steve Uhlig ‡
* † Delft University of Technology, Delft, The Netherlands
‡ Queen Mary University of London, London, United Kingdom
Email: {*B.Turkovic-2, †F.A.Kuipers}@tudelft.nl, ‡Steve.Uhlig@qmul.ac.uk,

***Abstract*—Congestion control algorithms are crucial in achieving high utilization while preventing overloading the network. Over the years, many different congestion control algorithms have been developed, each trying to improve over others in specific situations. However, their interactions and co-existence has, to date, not been thoroughly evaluated, which is the focus of this paper. Through head-to-head comparisons of loss-based, delay-based and hybrid types of congestion control algorithms, we reveal that fairness in resources claimed is often not achieved, especially when flows sharing a link have different round-trip times or belong to different groups.**

I. Introduction

In the wake of the growing demand for higher bandwidth, higher reliability, and lower latency, novel congestion control algorithms have been developed. For example, in 2016, Google published its bottleneck bandwidth and round-trip time (BBR) congestion control algorithm, claiming it was able to operate without filling buffers [1]. Around the same time, TCP LoLa [2] and TIMELY [3] were proposed, focusing on low latency and bounding of the queuing delay. Moreover, new transport protocols such as QUIC allow the implementation of algorithms directly in user space, which facilitates quick development of new transport features. However, *congestion control algorithms have been typically developed in isolation*, without thoroughly investigating their behaviour in the presence of other congestion control algorithms, which is the goal of this paper.

In this paper, we first divide existing congestion control algorithms into three groups: loss-based, delay-based, and hybrid. Based on experiments in a testbed, we study the interactions over a bottleneck link among flows of the same group, across groups, as well as when flows have different Round-Trip Times (RTTs). We find that flows using loss-based algorithms are over-powering flows using delay-based, as well as hybrid algorithms. Moreover, when flows using loss-based algorithms fill the queues, increase in queuing delay of all the other flows sharing the bottleneck is determined by their presence. Non-loss-based groups thus cannot be used in a typical network, where flows typically rely on a loss-based algorithm. In addition, we observe that convergence times can be large, which may surpass the flow duration for many applications. Finally, we find that hybrid algorithms, such as BBR, not only favour flows with a higher RTT, but they also cannot maintain a low queuing delay as promised.

In Section II, we provide an overview and classification of congestion control mechanisms. In Section III, we (1) identify a set of key performance metrics to compare them, (2) describe our measurement setup, and (3) present our measurement results. Additional measurements are given in an extended version [4].

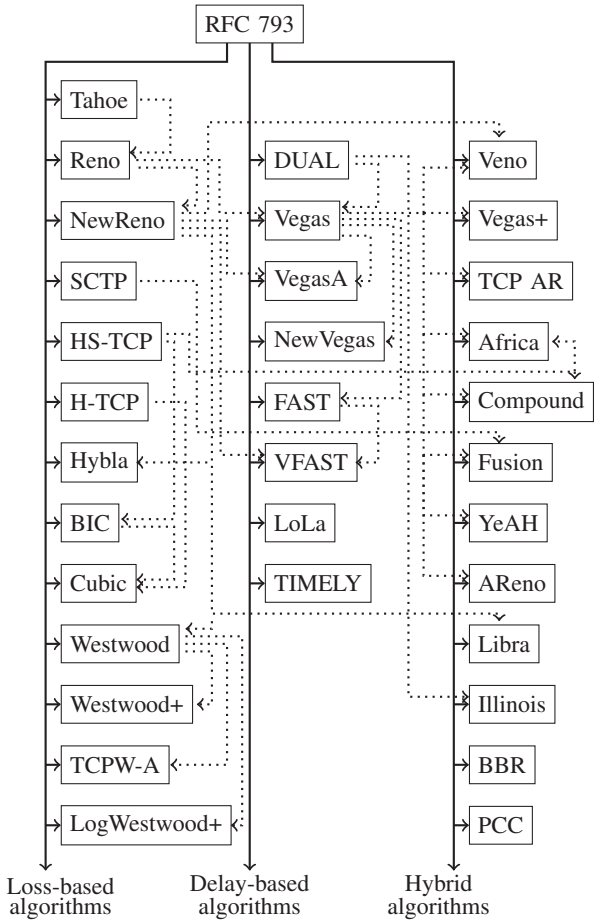

Fig. 1: Classification of different congestion control algorithms. Dotted arrows indicate that one was based on the other.

II. Background

Since the original TCP specification (RFC 793 [5]), numerous congestion control algorithms have been developed. In this paper, we focus mostly on algorithms designed for wired networks. The algorithms we consider can be used both by QUIC and TCP and can be divided into three main groups (see Fig. 1): (1) loss-based algorithms that detect congestion when buffers are already full and packets are dropped, (2) delay-based algorithms that rely on RTT measurements and detect congestion by an increase in RTT, indicating buffering,

978-3-903176-17-1 / © 2019 IFIP

and (3) hybrid algorithms that use some combination of the previous two methods.

A. Loss-based algorithms

The original congestion control algorithms from [5] were loss-based algorithms with TCP Reno being the first widely deployed one. With the increase in network speeds, Reno's conservative approach of halving the congestion window became an issue. TCP connections were unable to fully utilize the available bandwidth, so that other loss-based algorithms were proposed, such as NewReno [6], Highspeed-TCP (HS-TCP [7]), Hamilton-TCP (H-TCP [8]), Scalable TCP (STCP [9]), Westwood (TCPW [10]), TCPW+ (TCP Westwood+ [11]), TCPW-A [12], and LogWestwood+ [13]. They all improved upon Reno by including additional mechanisms to probe for network resources more aggressively. However, they also react more conservatively to loss detection events, and discriminate between different causes of packet loss.

However, these improvements did not address any of the existing RTT-fairness issues, but introduced new ones [14], [15]. Indeed, when two flows with different RTTs share the same bottleneck link, the flow with the lowest RTT is likely to obtain more resources than other flows. To resolve this issue, BIC [14] and Hybla [15] were proposed. Hybla modified NewReno's Slow Start and Congestion Avoidance phases and made them semi-independent of RTT. However, the achieved RTT-fairness meant that flows with higher RTTs behaved more aggressively. The main idea of BIC was to use a binary search algorithm to approach the optimal congestion window size. However, later evaluations showed that BIC can still have worse RTT-fairness than Reno [16]. In response, Cubic was proposed in [16]. Since **Cubic** is the current default algorithm in the Linux kernel, we will use it as a reference for loss-based algorithms throughout this paper.

B. Delay-based algorithms

In contrast to loss-based algorithms, delay-based algorithms are proactive. They try to find the point when the queues in the network start to fill, by monitoring the variations in RTT. An increase in RTT, or a packet drop, causes them to reduce their sending rate, while a steady RTT indicates a congestion-free state. Unfortunately, RTT estimates can be inaccurate due to delayed ACKs, cross traffic, routing dynamics, and queues in the network [3], [17].

The first algorithm that used queuing delay as a congestion indicator was TCP Dual. The first improvement to this algorithm was Vegas [18]. It focuses on estimating the number of packets in the queues and keeping it under a certain threshold. However, several issues were identified. First, when competing with existing loss-based algorithms, Vegas flows suffer from a huge decrease in performance [19], [20]. Second, it has a bias towards new flows and, finally, interprets rerouting as congestion [20]. To address these issues several modifications to Vegas were proposed, including VegasA [20], Vegas+ [19], FAST [21], VFAST [22], and NewVegas [23].

Recently, as low latency became important, several new algorithms have been proposed. Hock et al. designed LoLa [2], focusing on low latency and convergence to a fair share between flows. To improve performance in datacenter networks, Google proposed TIMELY [3], which relies on very precise RTT measurements. Since **Vegas** is used as the base algorithm by many other delay-based and hybrid algorithms, we use it as a reference for delay-based algorithms.

C. Hybrid algorithms

Hybrid algorithms use both loss and delay as congestion indicators. The first hybrid algorithm was Veno [24]. It is a modification of the Reno congestion control that extends the additive increase and multiplicative decrease functions by also using queuing delay as the secondary metric. To efficiently utilize the available bandwidth in high-speed networks, many algorithms use similar modifications based on the Vegas or Dual network state estimations. Some of the most important ones are Africa [25], Compound [26], and YeAH [27]. Other algorithms modify the congestion window increase function to follow a function of both the RTT and the bottleneck link capacity, such as Illinois [28], AR [29], Fusion [30], TCP-Adaptive Reno (AReno) [31], and TCP Libra [32].

In 2016, Google developed the bottleneck bandwidth and round-trip time (BBR) algorithm. However, several problems, mostly related to the Probe RTT phase, were discovered: (1) bandwidth can be shared unfairly depending on the timing of new flows and their RTT, and (2) unfairness towards other protocols, especially Cubic [33], [34], [35].

At the same time, a new approach to congestion control using online learning was proposed in PCC [36]. We use **BBR** as our representative for hybrid algorithms, since it is actually deployed (in Google's network) and implemented in the Linux kernel (since v4.9).

III. Evaluation

Using the metrics described in Sec. III-A and via the set-up described in Sec. III-B, in Sections III-C and III-D we evaluate the representatives of the three algorithm groups (Cubic, Vegas and BBR). Additional measurements and results of all other algorithms that have been implemented in the Linux kernel can be found in the extended version of this paper [4].

A. Performance metrics

Sending rate represents the bit-rate (incl. data-link layer overhead) of a flow generated by the source, per time unit.

Throughput measures the number of bits (incl. the data-link layer overhead) received at the receiver, per time unit.

RTT (round-trip time) represents the time between sending a packet and receiving an acknowledgement of that packet.

Goodput measures the amount of useful data (i.e., excl. overhead) delivered by the network between specific hosts, per time unit. This value is an indicator of the application-level QoS experienced by the end-users. Additionally, we use the **goodput ratio**, i.e., the amount of useful data transmitted divided by the total amount of data transmitted.

Fairness describes how the available bandwidth is shared among multiple users. We consider three different types of fairness: (1) **intra-fairness** describes the resource distribution between flows running the same congestion control algorithm; (2) **inter-fairness** describes the resource distribution between flows running different congestion control algorithms, and (3) **RTT-fairness** describes the resource distribution between flows having different RTTs. Fairness is represented by Jain's index [37]. This index is based on the throughput and indicates how fair the available bandwidth at the bottleneck is shared between all flows present. This fairness index ranges from $1/n$ (worst case) to 1 (best case), where n is the number of flows.

B. Experiment setup

Each server in our testbed has a 64-bit Quad-Core Intel Xeon CPU running at 3GHz with 4GB of main memory and has 6 independent 1 Gbps NICs. Each server can play the role of a 6-degree networking node. All nodes run Linux with kernel version 4.13 with the txqueuelen set to 1000, and were connected as shown in Fig. 2 with degree $1 \leq n \leq 4$ (consequence of the limited number of NICs per server in the testbed). Given that the performance of congestion control algorithms is affected by the bottleneck link on the path, such a simple topology is sufficient for our purposes. The maximum bandwidth and the bottleneck (between s1 and s2) was limited to a pre-configured value ($100Mbps$ in the case of TCP and $10Mbps$ in the case of QUIC to make sure that the sending rate of the end-user applications is enough to saturate the bottleneck link) with the use of ethtool. To

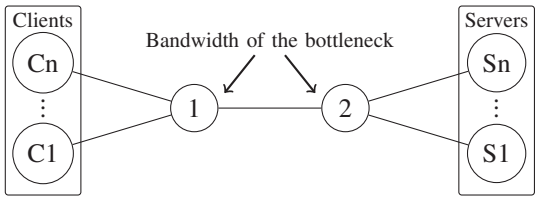

Fig. 2: Experiment topology.

perform measurements, we rely on tshark, iperf, QUIC client and server (available in the Chromium project [38]) and socket statistics. From traffic traces (before and after the bottleneck), we calculate the metrics described in Sec. III-A. All the values are averaged per flow, using a configurable time interval. We consider the following two scenarios:

BW scenario. Each analyzed algorithm is compared to itself and all others. Host C_i generates TCP flows towards servers running at S_i using different congestion control algorithms.

RTT scenario with flows having different RTTs. The purpose of this scenario is to test the RTT-fairness of different congestion control algorithms. In addition to the setup of the previous scenario, the delay at links between S_i and node 2 is artificially increased using Linux TC (adding $0 - 400ms$).

We ran these scenarios five times. For all of them, the results we observe lead to qualitatively similar interactions, as presented in Sections III-C and III-D.

C. Results: BW scenario

Intra-Fairness. Delay-based and loss-based algorithms have the best intra-fairness properties, with an average fairness index within $0.94 - 0.95$ (Table I). Fig. 3 shows that Jain's index is always close to 1, indicating that all present flows receive an equal share of the resources. In addition, delay-based algorithms operate without filling the buffers, in contrast to the loss-based algorithms that periodically fill the buffers and drop packets (Fig. 3). Further, the convergence time of loss-based algorithms is higher ($\approx$ 20 s, compared to $5s$ needed for 2 Vegas flows) and their throughput oscillates the most from all the evaluated approaches (Fig. 3). When the number of Cubic flows increases to 4, bandwidth oscillations increase as well, and fairness decreases to 0.82 [4].

In contrast, hybrid-based algorithms (BBR) unexpectedly had the worst intra-fairness properties. Fig. 3 shows that they rarely converge to the same bandwidth, but oscillate between 30 $Mbps$ and 70 $Mbps$ (every probeRTT phase), even in scenarios in which they claim a similar share of the available resources on average. The flow that measures a higher RTT adopts a more aggressive approach and claims more resources, even if the measured RTT difference is very small ($\leq 0.5ms$). Hence, they are not particularly stable. Unexpectedly, when the number of flows increases to 4, the fairness index improves, and although oscillations go down they are still present.

Inter-Fairness. As expected, flows that use delay-based algorithms experience a huge decrease in throughput if they share the bottleneck with loss-based flows (Fig. 4). This is because they detect congestion earlier, at the point when the queues start to fill. Loss-based algorithms on the other hand continue to increase their sending rate as no loss is detected. This increases the observed RTT (Fig. 3) of all flows, triggering the delay-based flow to back off [19], [20].

A similar behaviour is observed when a bottleneck is shared between flows from a hybrid and a delay-based algorithm: BBR outperforms Vegas. However, the difference in the throughput is less significant than the one observed in the previous scenario, with the Vegas flow claiming almost $40Mbps$ on average (Table I). When we increase the number of Vegas or BBR flows at the bottleneck to four, the new flows increase their bandwidth at the expense of the BBR flow, reducing its share from $50Mbps$ down to $20Mbps$, and increasing the fairness index to $0.9 - 0.94$ [4]. This is a consequence of the fact that BBR tries to operate without filling the queues, allowing the delay-based algorithm to grow and claim more bandwidth. Thus, we conclude that, in contrast to loss-based algorithms, delay-based algorithms can co-exist with hybrid-based ones.

When the bottleneck is shared between a hybrid and a loss-based algorithm, Cubic outperforms BBR, reducing its share of resources to as little as 8% on average (Table I), confirming results from [39]. The fairness index at the start of the connection is very low as Cubic claims all the available bandwidth at the expense of the BBR flow. After the Cubic flow fills the buffers, BBR measures an increased RTT and

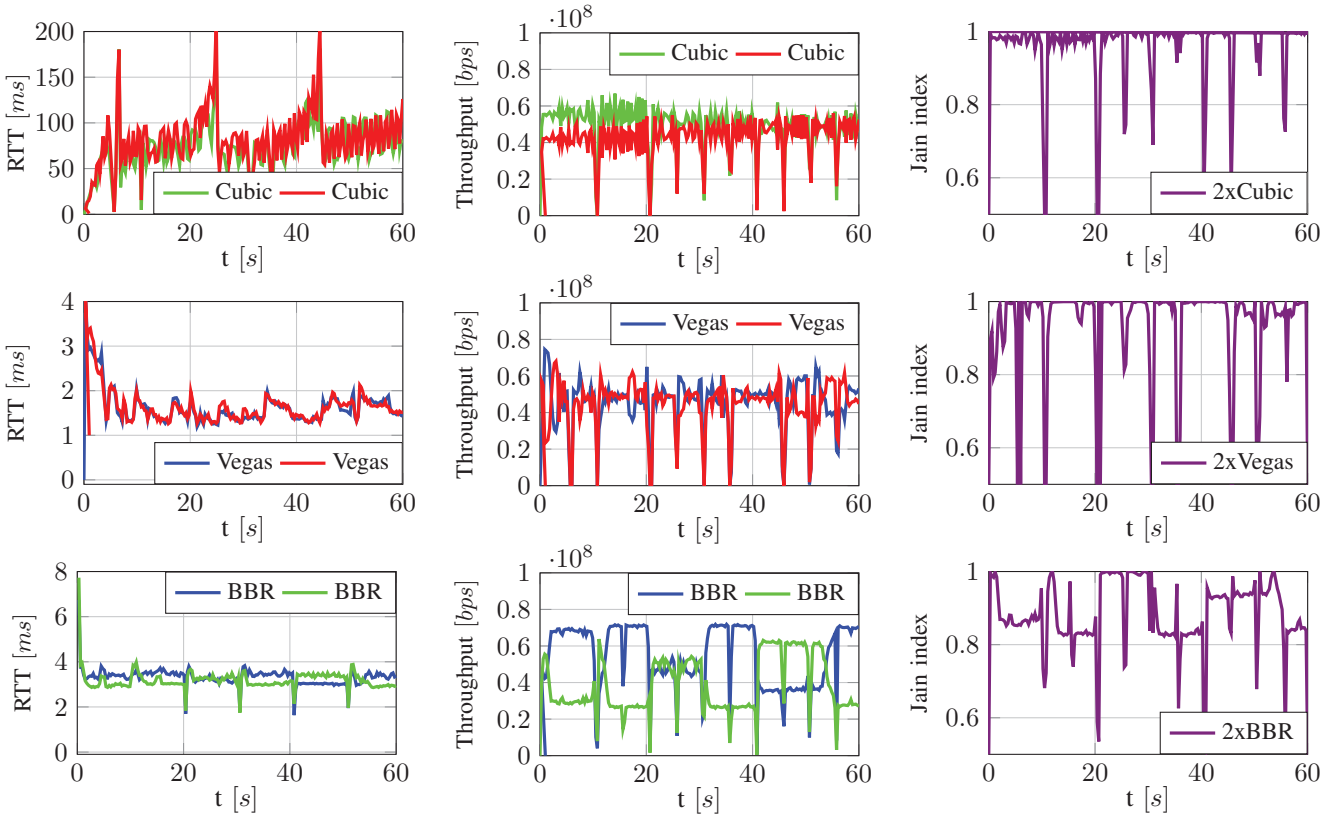

Fig. 3: BW scenario: Comparison of average RTT, average throughput, and fairness index for representatives of the congestion control algorithm classes groups in case the link is shared by 2 flows using the same algorithm (time unit 300ms).

TABLE I: BW scenario with 2 flows: Different metrics for representatives of the three congestion control algorithm groups (calculated for 5 different runs).

Protocol	Group	Algorithm	Average goodput $[Mbps]$	Average goodput ratio $[\%]$	Average RTT $[\#packets]$	Average sending rate $[ms]$	Average throughput $[Mbps]$	Average Jain index $[Mbps]$
TCP	Loss- vs. Loss-based	Cubic Cubic	44.98 43.15	93.57 93.78	76.65 78.32	48.77 50.98	46.59 46.59	0.95
	Delay- vs. Delay-based	Vegas Vegas	43.81 42.72	94.81 94.76	1.66 1.68	48.65 49.79	45.47 44.38	0.94
	Hybrid vs. Hybrid	BBR BBR	44.98 42.72	92.32 94.39	3.21 3.24	52.18 46.89	46.70 44.36	0.86
	Loss-based vs. Hybrid	Cubic BBR	82.29 7.56	94.27 88.86	70.37 174.38	90.91 8.87	85.05 7.89	0.59
	Loss- vs. Delay-based	Cubic Vegas	87.73 1.74	94.34 91.57	67.16 139.79	97.30 2.00	90.66 1.82	0.52
	Delay-based vs. Hybrid	Vegas BBR	38.37 48.56	94.34 94.68	4.55 4.25	37.31 61.65	39.83 50.37	0.84

adopts, as a consequence, a more aggressive approach (Fig. 3). However, packet loss triggers Cubic's back-off mechanism, allowing BBR to measure a lower RTT estimate. Consequently, BBR reduces its rate, allowing the Cubic flow to claim more bandwidth again. Moreover, when we increase the number of Cubic flows to three, the throughput of the BBR flow drops close to zero. Similarly, even three BBR flows are not able to compete with one Cubic flow, with each of them claiming approximately 5% of the total bandwidth on average [4].

Delay. Even if one loss-based algorithm is present at the bottleneck, the observed delay is determined by it, nullifying the advantages of delay-based and hybrid algorithms, namely the prevention of the queue buildup. BBR, as well as Vegas, which claim to be able to operate with a small RTT, suffer from a huge increase in average RTT (by more than $100\ ms$, Table I) when competing with Cubic (compared to $1 - 5ms$ without Cubic). However, when a link is shared between a hybrid and a delay-based flow, both of them are able to maintain a low RTT. In such scenarios, hybrid algorithms, such as BBR, due to their more aggressive approach compared to

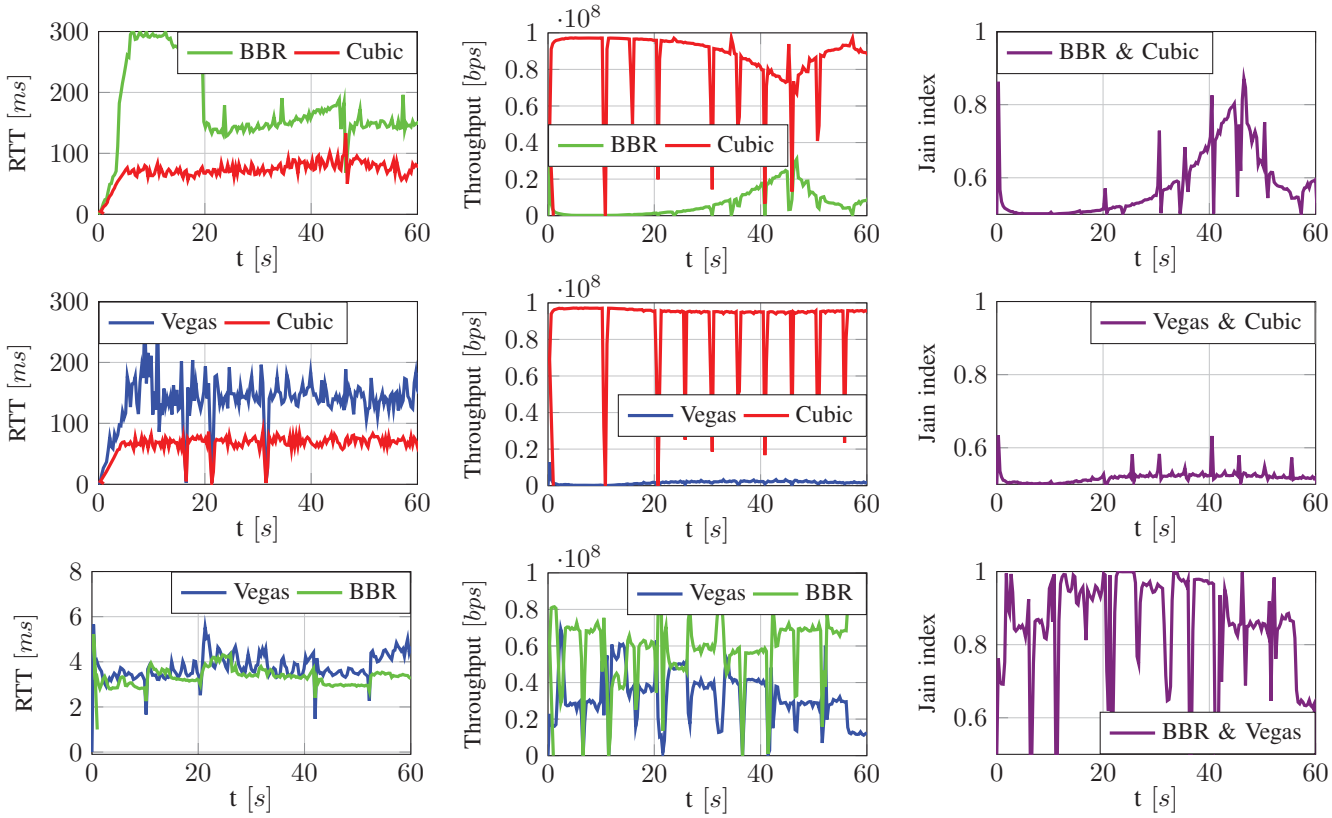

Fig. 4: BW scenario: Comparison of average RTT, average throughput and fairness index for representatives of the congestion control algorithm groups in case the link is shared by 2 flows using different algorithms (time unit 300ms).

delay-based algorithms, determine the RTT. Vegas flows, as a consequence, suffer from a small increase in RTT (from 1.68ms to 4.55ms, Table I).

Summary. In terms of fairness, the only combination that works well together is delay and hybrid algorithms. In such a scenario, delay is low and the throughput fairly shared, the more flows the fairer the distribution of resources. Hybrid, as well as delay-based algorithms, suffer from a huge increase in the observed delay if even one loss-based algorithm is present at the bottleneck making them unusable in typical networks consisting of many different flows. We observe that the most popular TCP flavour, Cubic, is prone to oscillation and has a high convergence time ($\approx 20s$). Further, we observe that BBR is not stable, reacting to very small changes in the observed RTT, which was not previously reported in the literature.

D. Results: RTT scenario

We observe RTT-fairness issues for all three groups of algorithms. Even though loss-based algorithms such as Cubic claim good RTT-fairness properties, they favour the flow with a lower RTT [40]. This is most noticeable when analyzing two Cubic flows in Fig. 5. Even when the number of flows increases to 4 (Fig. 6), the flow with the lowest RTT immediately claims all the available resources, leaving less than half to the other flows in the first 30 s. Several improvements addressing this problem, such as TCP Libra [32] have been proposed. However, current kernel implementations do not capture these improvements.

The fairness index for delay-based algorithms slowly increases over time, but due to a very conservative congestion avoidance approach of Vegas, even after $60s$, flows do not converge (Fig. 6). When we increase the number of Vegas flows to four, the dynamics at the bottleneck becomes more complex with the newest flow (with the highest RTT) claiming the largest share of resources at the end (Fig. 6). Moreover, contrary to the previous scenarios, in the slow start phase, Vegas flows fill the bottleneck queue and the observed queuing delay increases to $70ms$. However, after $30s$ the queues are drained, fairness improves, and the observed queuing delay is very low for all flows ($2-3ms$, Fig. 6).

Hybrid-based algorithms, such as BBR, favour the flow with the higher RTT, confirming results from [33], [39]. The flow with a higher RTT overestimates the bottleneck link, claiming all the available resources and increasing the queuing delay (Fig. 5) by a factor of more than 10 (from $\approx 4ms$ to $\approx 50ms$). Moreover, when we increase the number of BBR flows to four, contrary to expectations, the average RTT increases significantly (by a factor of almost 30) reaching values comparable to the ones observed by the loss-based algorithms in the same scenario although only BBR flows were present at the bottleneck (Fig. 6, Table III).

Summary. We observe that RTT-fairness is poor for all

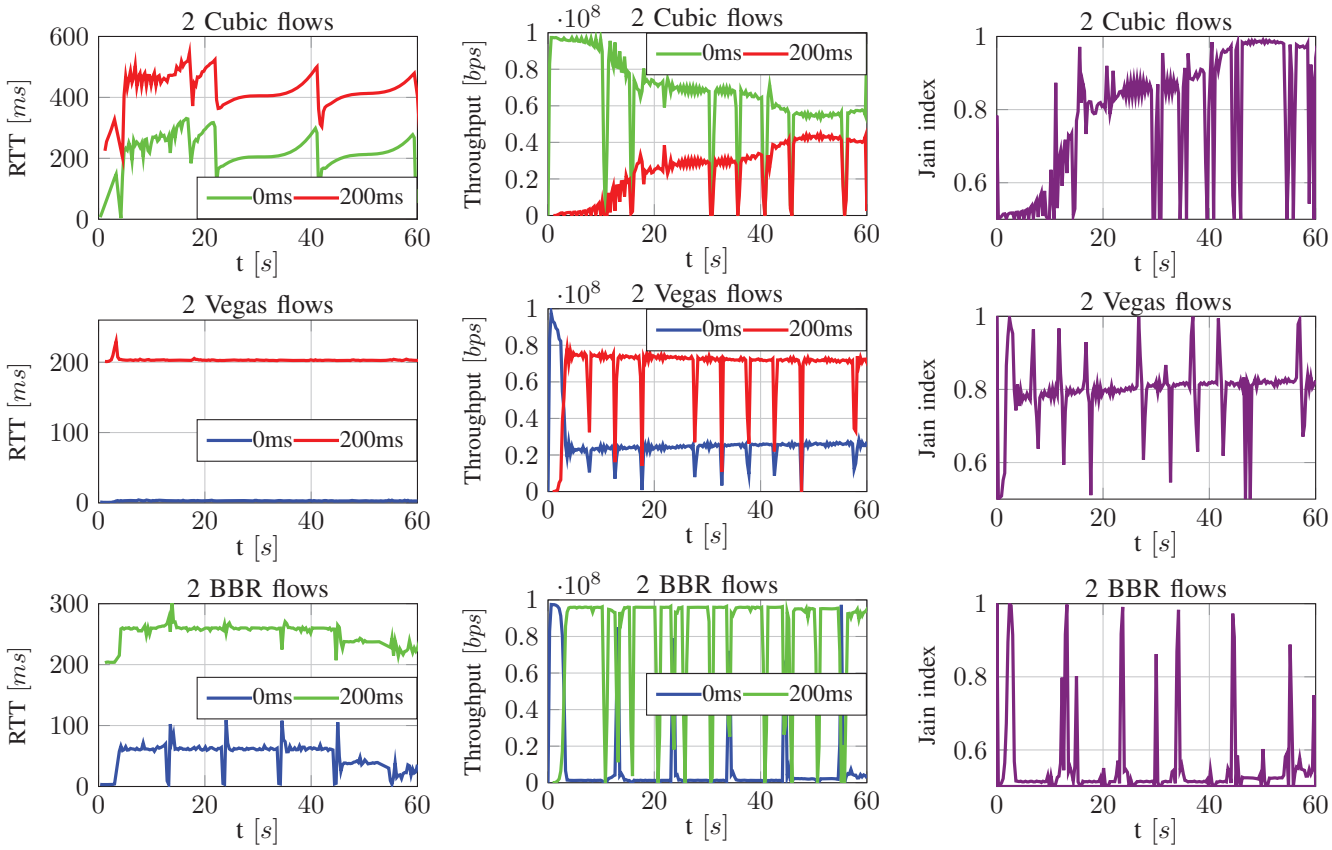

Fig. 5: RTT scenario: Comparison of average RTT, average throughput, and fairness index for representatives of the congestion control algorithm groups in the case the link is shared by 2 flows using the same algorithm (time unit 300ms).

TABLE II: RTT scenario: Different metrics for representatives of the congestion control algorithm groups in case the link is shared by two flows using the same algorithm (calculated for 5 different runs).

Protocol	Group	Algorithm	Average goodput [$Mbps$]	Average goodput ratio [%]	Average RTT [$\#packets$]	Average sending rate [ms]	Average throughput [$Mbps$]	Average Jain index [$Mbps$]
TCP	Loss- vs. Loss-based	Cubic(0ms)	65.67	94.07	233.09	75.47	67.88	0.76
		Cubic(200ms)	21.88	93.80	435.53	25.36	22.92	
	Delay- vs. Delay-based	Vegas(0ms)	14.99	94.21	32.03	18.91	15.62	0.66
		Vegas(200ms)	72.60	94.31	228.96	81.48	75.08	
	Hybrid vs. Hybrid	BBR(0ms)	8.90	91.98	50.08	9.87	9.24	0.56
		BBR(200ms)	79.54	94.39	249.56	90.97	82.1	

groups of algorithms. Delay-based algorithms are the only ones that can maintain a low delay compared to the other two groups. However, they still do not converge towards their fair share. Loss-based algorithms such as Cubic perform poorly, contrary to expectations and their own claims, favouring flows with lower RTTs. When loss-based algorithms converge to a fair share, the convergence time is so slow that the average fairness index is still low (0.69 on average). Finally, hybrid algorithms such as BBR suffer from significant dynamics in the sharing among its own flows, favoring those with higher RTT and significantly increasing the queuing delay. Hence, we observe that even when only BBR flows are present on the bottleneck, the claim of being able to operate without filling the buffers is not true.

E. Results: QUIC

When QUIC is used with different congestion control algorithms, we observe similar interactions as earlier. With BBR, we observe the same RTT-unfairness properties as with the TCP BBR, which always favours the flows with a higher RTT (with an average fairness index of 0.59). Similarly, QUIC with Cubic always favours the flow with a lower RTT. However, the difference between the throughput of the two QUIC Cubic flows is much smaller than the one observed for the TCP equivalent, with an average fairness index of 0.93. In all our QUIC scenarios where hybrid (BBR) and loss-based (Cubic) flows compete, Cubic outperforms BBR. Over time, as QUIC BBR flows detect a higher RTT and adopt a more aggressive

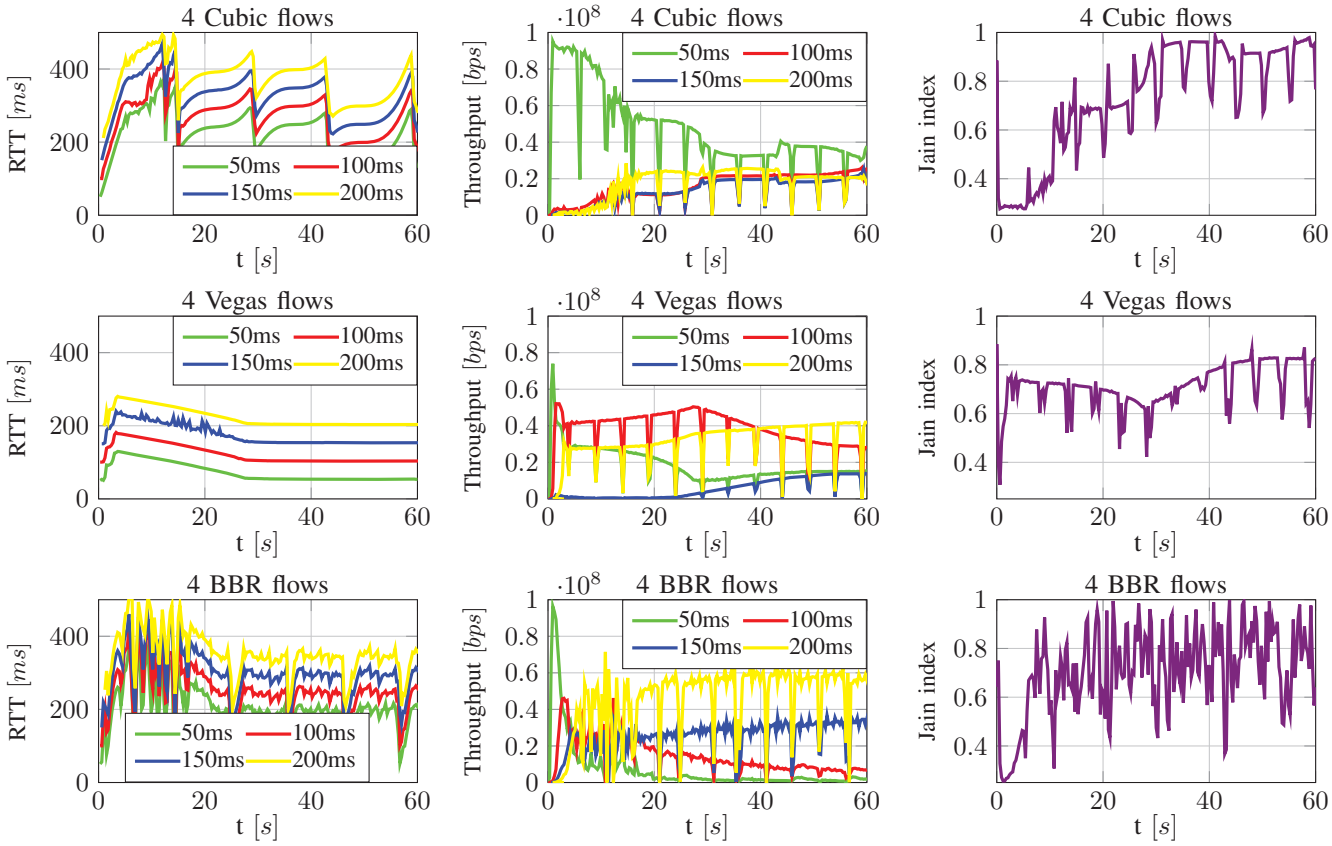

Fig. 6: RTT scenario: Comparison of average RTT, average throughput, and fairness index for representatives of the congestion control algorithm classes in case the link is shared by 4 flows using the same algorithm (time unit 300ms).

TABLE III: RTT scenario: Different metrics for representatives of the congestion control algorithm classes in case the link is shared by four flows using the same algorithm (calculated for 5 different runs).

Protocol	Group	Algorithm	Average goodput [$Mbps$]	Average goodput ratio [%]	Average RTT [$\#packets$]	Average sending rate [ms]	Average throughput [$Mbps$]	Average Jain index [$Mbps$]
TCP	Loss-based	Cubic(50ms)	47.48	93.86	216.60	53.66	49.59	0.69
		Cubic(100ms)	15.32	92.39	264.99	17.71	16.09	
		Cubic(150ms)	11.70	91.62	316.87	13.62	12.32	
		Cubic(200ms)	13.68	92.33	368.14	15.78	14.39	
	Delay-based	Vegas(50ms)	27.32	92.98	94.50	31.09	28.54	0.62
		Vegas(100ms)	41.85	93.88	144.11	47.13	43.63	
		Vegas(150ms)	7.50	90.80	196.87	8.62	7.90	
		Vegas(200ms)	11.57	91.47	245.18	13.22	12.16	
	Hybrid	BBR(50ms)	7.11	88.01	203.63	42.56	7.44	0.63
		BBR(100ms)	15.23	91.61	253.49	21.43	16.06	
		BBR(150ms)	22.20	93.59	302.70	18.81	23.45	
		BBR(200ms)	42.39	94.18	353.22	15.97	44.70	

approach, BBR grabs more bandwidth at the expense of the Cubic flows. However, this process is slow and the throughput of the BBR flow remains low. Detailed measurements of QUIC can be found in the extended version of this paper [4].

IV. Conclusion

After dividing existing congestion control algorithms into three groups (loss-based algorithms, delay-based algorithms, and hybrid algorithms), we studied their interactions.

We observed multiple fairness issues, among flows of the same group, across different groups, as well as when flows having different RTTs were sharing a bottleneck link. We found that delay-based, as well as hybrid algorithms, suffer from a decrease in performance when competing with flows from the loss-based group, making them unusable in a typical network where the majority of flows will rely on a loss-based algorithm. Not only do they get an unfair share of the available bandwidth, but they also suffer from a huge increase in the observed delay when the loss-based algorithms fill the queues.

The only combination that worked well together was delay and hybrid algorithms: the observed RTT was low and resources shared fairly (the more flows the fairer the distribution of resources). Finally, we found that hybrid algorithms, such as BBR, are very sensitive to changes in the RTT, even if that difference is very small ($\leq 0.5ms$). They not only favour the flow with a higher RTT at the expense of the other flows, but they also cannot maintain a low queuing delay as promised even if they are the only flows present in the network.

Our work therefore shows that to support applications that require low latency, a good congestion control algorithm on its own won't be enough. Indeed, guaranteeing that flows of a given group (in terms of type of congestion control) will receive their expected share of resources, requires that resource isolation be provided between the different groups.

REFERENCES

[1] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[2] M. Hock, F. Neumeister, M. Zitterbart, and R. Bless, "TCP LoLa: Congestion Control for Low Latencies and High Throughput," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, 2017, pp. 215–218.

[3] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based Congestion Control for the Datacenter," pp. 537–550, 2015.

[4] B. Turkovic, F. A. Kuipers, and S. Uhlig, "Fifty shades of congestion control: A performance and interactions evaluation," *arXiv preprint arXiv:1903.03852*, 2019.

[5] J. Postel, "Transmission control protocol specification," *RFC 793*, 1981.

[6] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," RFC 5681 (Draft Standard), Internet Engineering Task Force, September 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5681.txt

[7] S. Floyd, "HighSpeed TCP for large congestion windows," Tech. Rep., 2003.

[8] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long-distance networks," in *Proceedings of PFLDnet*, vol. 2004, 2004.

[9] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," *ACM SIGCOMM computer communication Review*, vol. 33, no. 2, pp. 83–91, 2003.

[10] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proceedings of the 7th annual international conference on Mobile computing and networking*. ACM, 2001, pp. 287–297.

[11] L. A. Grieco and S. Mascolo, "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 25–38, 2004.

[12] K. Yamada, R. Wang, M. Y. Sanadidi, and M. Gerla, "TCP westwood with agile probing: dealing with dynamic, large, leaky pipes," in *2004 IEEE International Conference on Communications (IEEE Cat. No.04CH37577)*, vol. 2, June 2004, pp. 1070–1074 Vol.2.

[13] D. Kliazovich, F. Granelli, and D. Miorandi, "Logarithmic window increase for TCP Westwood+ for improvement in high speed, long distance networks," *Computer Networks*, vol. 52, no. 12, pp. 2395–2410, 2008.

[14] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 4. IEEE, 2004, pp. 2514–2524.

[15] C. Caini and R. Firrincieli, "TCP Hybla: a TCP enhancement for heterogeneous networks," *International journal of satellite communications and networking*, vol. 22, no. 5, pp. 547–566, 2004.

[16] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.

[17] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-host congestion control for TCP," *IEEE Communications surveys & tutorials*, vol. 12, no. 3, pp. 304–342, 2010.

[18] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, *TCP Vegas: New techniques for congestion detection and avoidance*. ACM, 1994, vol. 24, no. 4.

[19] G. Hasegawa, K. Kurata, and M. Murata, "Analysis and improvement of fairness between TCP Reno and Vegas for deployment of TCP Vegas to the Internet," in *Proceedings 2000 International Conference on Network Protocols*, Nov 2000, pp. 177–186.

[20] K. Srijith, L. Jacob, and A. L. Ananda, "TCP Vegas-A: Improving the performance of TCP Vegas," *Computer communications*, vol. 28, no. 4, pp. 429–440, 2005.

[21] C. Jin, D. Wei, S. H. Low, J. Bunn, H. D. Choe, J. C. Doylle, H. Newman, S. Ravot, S. Singh, F. Paganini, G. Buhrmaster, L. Cottrell, O. Martin, and W. chun Feng, "FAST TCP: from theory to experiments," *IEEE Network*, vol. 19, no. 1, pp. 4–11, Jan 2005.

[22] S. Belhaj and M. Tagina, "VFAST TCP: An improvement of FAST TCP," in *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*. IEEE, 2008, pp. 88–93.

[23] J. Sing and B. Soh, "TCP New Vegas: improving the performance of TCP Vegas over high latency links," in *Network Computing and Applications, Fourth IEEE International Symposium on*. IEEE, 2005, pp. 73–82.

[24] C. P. Fu and S. C. Liew, "TCP Veno: TCP enhancement for transmission over wireless access networks," *IEEE Journal on selected areas in communications*, vol. 21, no. 2, pp. 216–228, 2003.

[25] R. King, R. Baraniuk, and R. Riedi, "TCP-Africa: An adaptive and fair rapid increase rule for scalable TCP," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3. IEEE, 2005, pp. 1838–1848.

[26] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for High-Speed and Long Distance Networks," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, 2006, pp. 1–12.

[27] A. Baiocchi, A. P. Castellani, and F. Vacirca, "YeAH-TCP: yet another highspeed TCP," in *Proc. PFLDnet*, vol. 7, 2007, pp. 37–42.

[28] S. Liu, T. Başar, and R. Srikant, "TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks," *Performance Evaluation*, vol. 65, no. 6-7, pp. 417–440, 2008.

[29] H. Shimonishi and T. Murase, "Improving efficiency-friendliness tradeoffs of TCP congestion control algorithm," in *Global Telecommunications Conference, 2005. GLOBECOM'05. IEEE*, vol. 1. IEEE, 2005, pp. 5–pp.

[30] K. Kaneko, T. Fujikawa, Z. Su, and J. Katto, "TCP-Fusion: a hybrid congestion control algorithm for high-speed networks," in *Proc. PFLDnet*, vol. 7, 2007, pp. 31–36.

[31] H. Shimonishi, T. Hama, and T. Murase, "TCP-adaptive reno for improving efficiency-friendliness tradeoffs of TCP congestion control algorithm," in *Proc. PFLDnet*. Citeseer, 2006, pp. 87–91.

[32] G. Marfia, C. Palazzi, G. Pau, M. Gerla, M. Sanadidi, and M. Roccetti, "Tcp libra: Exploring rtt-fairness for tcp," in *International Conference on Research in Networking*. Springer, 2007, pp. 1005–1013.

[33] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a Deeper Understanding of TCP BBR Congestion Control," in *IFIP Networking 2018*, Zurich, Switzerland, May 2018.

[34] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of BBR congestion control," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, Oct 2017, pp. 1–10.

[35] S. Ma, J. Jiang, W. Wang, and B. Li, "Towards RTT Fairness of Congestion-Based Congestion Control," *CoRR*, vol. abs/1706.09115, 2017. [Online]. Available: http://arxiv.org/abs/1706.09115

[36] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "{PCC}: Re-architecting congestion control for consistent high performance," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 395–408.

[37] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, "A Quantitative Measure of Fairness and Discrimination," *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 1984.

[38] "The chromium projects: Chromium," https://www.chromium.org/Home, accessed: 04-03-2019.

[39] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of bbr congestion control," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE, 2017, pp. 1–10.

[40] T. Kozu, Y. Akiyama, and S. Yamaguchi, "Improving rtt fairness on cubic tcp," in *2013 First International Symposium on Computing and Networking*, Dec 2013, pp. 162–167.