Congestion Control in the Wild - Investigating Content Provider Fairness

Ike Kunze[1], Jan Rüth[1], Oliver Hohlfeld[2]
[1]*RWTH Aachen University,* [2]*Brandenburg University of Technology*
{kunze, rueth}@comsys.rwth-aachen.de, hohlfeld@b-tu.de

***Abstract*—Congestion control (CC) is an indispensable component of transport protocols to prevent congestion collapse as it distributes the available bandwidth among all competing flows, ideally in a fair manner. It thus has a large impact on performance and there exists a constantly evolving set of CC algorithms, each addressing different performance needs. While the algorithms are commonly tested regarding the problems underlying their implementation, the interaction with existing algorithms is often not considered. Additionally considering the fact that content providers (CPs) such as content distribution networks (CDNs) are known to tune TCP stacks for performance gains, the large assortment of algorithms opens the door for custom parametrization and potentially unfair bandwidth sharing.**

In this paper, we thus empirically investigate if current Internet traffic generated by CPs still adheres to the conventional understanding of fairness. For this, we compare fairness properties of testbed hosts to actual traffic of six major CPs subject to different queue sizes and queueing disciplines in a home-user setting. Additionally, we investigate how mice and elephant flows from the different CPs interact. We find that some employed CC algorithms lead to significantly asymmetric bandwidth shares and very poor flow completion times for mice flows. Fortunately, AQMs such as FQ_CoDel are able to alleviate such unfairness.

***Index Terms*—TCP congestion control, Measurements, Content distribution networks, BBR, BBR2, Cubic**

I. Introduction

The Internet has grown way beyond its original purpose of being a research network. Today, thousands of autonomous systems connect and exchange data. The fundamental principles governing this data exchange are well established since decades and defined in IETF RFCs. To this end, the current best-effort Internet relies on CC to *i)* not collapse the network, and to *ii)* achieve fairness for flows competing for bandwidth at a bottleneck. For TCP, RFC 5681 [1] requires the implementation of slow start, congestion avoidance, fast retransmit, and fast recovery (generally known as TCP Reno). Other algorithms improve on certain aspects of Reno, e.g., to enable higher performance over large bandwidth-delay product (BDP) networks. Usually, when a new or modified CC algorithm is proposed, it is rated in terms of TCP fairness when competing with Reno or Cubic as Linux's default CC algorithm. While fairness is generally a hard to define property for Internet flows and flow-rate fairness is a controversial metric [2], it is still widely used. In 2005, Medina et al. [3] showed that *most* Internet flows halve their congestion window on loss and are thus TCP conform, leading to an expected flow-rate fairness [4].

Since then, the Internet landscape has drastically changed, end-users use the Internet with increasing access speeds [5] and content such as videos is causing a substantial fraction of Internet traffic [6], [7]. These increasing demands have led to a logical centralization of the content-serving Internet where a few big players serve the majority of the content [8], [9]. In previous work [10], we have shown that CDNs specialize in serving such content by tuning their TCP stacks beyond RFC-recommended values in hope for higher performance and user satisfaction. Fundamentally, such observations raise the question of fairness, and in fact, from an economic standpoint being unfair to a competing CP might be advantageous (e.g., by being able to deliver data with more than a fair bandwidth share). While identifying a CP's CC algorithm (e.g., via [11]) helps in understanding its principal behavior, these works do not take into account the actual parameterization of the algorithms which have the potential of drastically changing the fairness. Transport protocol evolution with QUIC has the potential to further lower the hurdle for modification in the future, given its realization in userspace for flexible customization.

In light of these historical changes, we have already started to investigate the interaction of large CPs regarding fairness in our previous work [12]. More precisely, we shed light on current practices and evaluate if actual Internet traffic adheres to the conventional understanding of fairness. To this end, we devise a methodology that enables us to compare testbed results with Internet traffic. This work extends our previous investigations and provides the following contributions:

- We present a testbed methodology using RTT-fairness to study actual TCP traffic by major CPs to account for a broad set of TCP optimizations used in practice.
- We repeat and extend our previous work [12] by extensive measurements from July to September 2019.
- We compare fairness properties of testbed hosts to actual traffic by six major CPs subject to different RTTs, queue sizes, and queueing disciplines in a home-user setting.
- We provide a fairness evaluation of BBRv2.
- We also evaluate the performance of the CPs in a mouse vs. elephant scenario.
- We finally show that AQMs like CoDel and FQ_CoDel can improve fairness as well as flow completion times.

Structure. We first discuss flow-rate fairness and its importance for the Internet in Sec. II and present related work in Sec. III. We then introduce our testbed methodology in Sec. IV and validate it in Sec. V. After that we describe how we investigate CPs in the Internet in Sec. VI before discussing the results of our fairness study in semi-controlled settings (Sec. VII) and in the Internet (Sec. VIII). Finally, we conclude the paper.

This is the author's version. The original version appeared in *IEEE Transactions on Network and Service Management*, https://doi.org/10.1109/TNSM.2019.2962607

II. Background

One of the key challenges in the Internet is the decentralized resource allocation of bandwidth. However, TCP's initial design only prevented overloading single end-points and did not consider the possibility that the network itself could become overloaded and collapse upon this congestion. As centralized algorithms are not deployable in the Internet, decentralized CC was soon added to TCP's design. However, it quickly showed that there are scenarios where the early CC often yields less than optimal performance which has led to a plethora of research for evolved and optimized algorithms. With more and more algorithms, questions about their bandwidth sharing arose, and hence, research has been investigating the *fairness* of CC.

Different Notions of Fairness. Most fairness investigations base on flow-rate fairness with well-studied measures being the intra-protocol fairness, i.e., how do two instances of the same algorithm share the available bandwidth, the RTT-fairness, i.e., what happens if they have different RTTs, and the inter-protocol fairness, where two different algorithms are investigated. While there is a consensus in that intra- and inter-protocol fairness should be defined such that a high level of fairness is achieved if both flows get the same amount of bandwidth if they have the same RTTs (refered to as same-RTT fairness in the rest of this paper), there is a long-lasting debate regarding RTT-fairness [13]. On the one hand, people argue that all flows should have an equal share of bandwidth even if their RTTs are different. Consequently, they identify significant levels of RTT-unfairness in current CC algorithms. On the other hand, people argue that flows with different RTTs should also have different bandwidth shares as flows with high RTTs occupy more resources within the network, and hence, should only get a smaller share of the bandwidth at all intermediate hops. A recent proposal [14] advocates the study of harm rather than bandwidth equality when comparing CC algorithms.

RTT-fairness in Practice. Most loss-based CCs have an *inverse-proportional* RTT behavior, i.e., flows with a shorter RTT get a larger share of the bandwidth, corresponding to the argument mentioned above of a larger resource claim within the network. Some CC algorithms go in the opposite direction and opt for a *proportional* RTT behavior, i.e., flows with a larger RTT get a larger share of the bandwidth, which is particularly visible in BBRv1. This behavior boils down to the optimal operation point of TCP (which each CC targets), given by the BDP, BDP = RTT $\cdot$ C with physical round-trip time (RTT) and link bandwidth (C), and thus, directly depends on the RTT [15]. As BBRv1 flows try to estimate the BDP, flows with larger RTTs will naturally arrive at higher bandwidth estimates and thus outperform flows with smaller RTTs. Which of both behaviors is more desirable is out of scope of this work, we just remark that both are present in today's algorithms.

BBRv2. In 2019, BBRv1's successor BBRv2 was presented [16], [17] but its fairness properties are not yet known or at least not yet reported. BBRv2 incorporates a mechanism to react to packet loss and Data Center TCP-style Explicit Congestion Notification (ECN), setting it apart from its predecessor. The general state machine of BBRv2 is similar to the original BBRv1, i.e., after the startup and subsequent drain phases, cyclic ProbeBW and ProbeRTT phases are used to estimate the true BDP. The packet loss and ECN detections are used as additional signifiers during the different phases. Using an explicit loss rate target, BBRv2, e.g., leaves the initial startup phase (slow start) if the marking or loss rate exceeds a target threshold. There are further changes in v2, like measuring an aggregation level (e.g., ACK compression), a maximum in-flight estimation on long and short time-scales using loss or markings, and less throughput variation during ProbeRTT. All these features, should lead to a better coexistence with Reno and Cubic flows according to the authors.

Some Thoughts on Fairness. While the bandwidth is a general property of the bottleneck, the RTT is a highly volatile flow-specific property which means that it naturally has to have an impact on the achievable bandwidth share. It is thus difficult to judge to what extent TCP variants should be fair. One could argue that it is perfectly fine to tune a TCP implementation to work better under various transient conditions as long as it is fair under a reasonably good condition. Another view on fairness does not regard the bandwidth but the congestion that the algorithms cause. Thus, one could argue that the RTT variation and losses should be minimized for competing flows. Fairness becomes an even tougher notion when we regard short flows that compete with bulk flows. Due to TCP's probing, many short flows never leave slow start, and thus, comparing bandwidths is insufficient. Here it might simply be enough to judge how fast the short flow can finish, when it is forced to leave slow start, or how the bulk flow reduces its rate.

In the following, we discuss how related work has up to now investigated fairness in CC and in the Internet.

III. Related Work

Well-studied measures are the intra-protocol flow-rate fairness, i.e., how well do two instances of the same algorithm share the available bandwidth, the RTT-fairness, i.e., what happens if the flows have different RTTs, and the inter-protocol fairness, where two different algorithms are investigated.

Intra-Protocol and RTT-Fairness. For Cubic, research has commonly found decent intra-protocol fairness and an inverse-proportional RTT-fairness, meaning that instances with smaller RTTs get a larger share of the overall bandwidth. These findings have been confirmed for a large set of different network characteristics, ranging from small (10 Mbps, e.g., [18]) to large bottleneck bandwidths (10 Gbps, e.g., [19]) or short (16 ms, e.g., [20], [21]) to long (324 ms, e.g., [20], [21]) RTTs.

For BBRv1, less research exists and the available studies partly disagree on the properties of BBRv1. This is especially true for intra-protocol fairness, as Cardwell et al. [22] claim a high degree of fairness across the board, while Hock et al. [23] identify scenarios where the fairness is significantly impaired. Regarding RTT-fairness, it is commonly found that BBRv1 has a proportional RTT-fairness property, i.e., a flow with a larger RTT gets a larger share of the available bandwidth [22], [24], [25]. Hock et al. [23] generally confirm the findings but by investigating two different bottleneck queue sizes, they find that in scenarios with a smaller queue size (0.8× BDP) flows with a smaller RTT have a slight advantage, while in large

buffer scenarios (8×BDP) the inverse is true and flows with larger RTTs have a significant advantage.

Inter-Protocol Fairness. While the intra-protocol and RTT-fairness of CC is important for a large scale-out of the algorithms, the inter-protocol fairness property shines a light on the coexisting use of different CC algorithms in the Internet. Unfortunately, several groups of researchers have found that BBRv1 and Cubic do not cooperate well, as Cubic flows dominate BBRv1 flows in scenarios with larger buffers (generally above 1×BDP) while the opposite is true for small buffer scenarios [22], [23], [25].

While many studies investigate how certain algorithms affect each other, there is missing up to date research on which are actually used in the Internet. Moreover, many studies neglect the parameterization and tuning potential of these algorithms that are used in practice. To address this, this study explores if actual Internet traffic of large content providers—which carry the bulk of today's Internet traffic—still adheres to the conventional understanding of TCP fairness.

IV. Methodology

CC research traditionally involves *simulation* or *testbed* studies, which give researchers *complete control* over the investigated scenarios. While this is desirable for controlled experiments, the involved abstractions and assumptions do not allow to completely cover real-world settings. For example, the employed algorithms and their parameterization in real-world systems are typically unknown. To study CC fairness in practice, we, therefore, contact real-world Internet systems with a testbed setup. This enables us to still control *some* parameters (e.g., bottleneck bandwidth and delay) while studying the CC algorithms as run by real systems. This way we can study if Internet traffic by CPs still adheres to the conventional understanding of TCP flow-rate fairness.

A. Home User (Residential Access) Scenarios

The fundamental design choice of our study is to investigate fairness from the perspective of an end-user accessing the Internet from home. Even though peering links have been identified as possible points of congestion [26], it is still widely believed that access links form the bottlenecks and thus congestion happens at the network edges, more specifically at the end-user's access link [27]. We model this scenario in the form of a simple dumbbell topology which we illustrate in Fig. 1. The user—represented by the client—is connected to the testbed network via a dedicated machine serving as a configurable bottleneck (via Linux's traffic control (TC) subsystem). In general, the client can request traffic from all kinds of sources, from within the testbed and from Internet sources. For our study, we focus on three distinct scenarios.

In Scenario (a) (testbed only), we investigate the out-of-box performance of CC by simultaneously requesting traffic from two testbed machines. This, above all, establishes a baseline and identifies potential influencing factors on the overall interaction of CC. Building upon this baseline, Scenario (b) (testbed & Internet) then replaces one of the two testbed flows with a flow originating from the Internet. Thus, we compare how

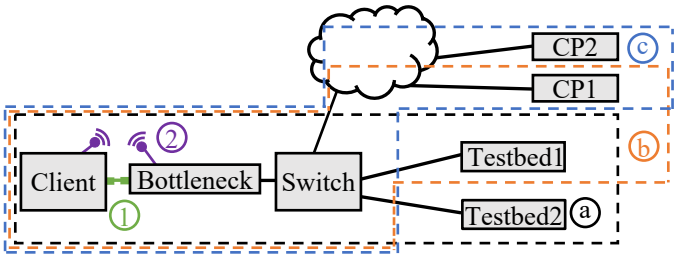

Fig. 1: Testbed topology. Scenario (a) (testbed-only), Scenario (b) (testbed & Internet), Scenario (c) (Internet-only). Ethernet (1) and Wifi (2) bottleneck connections.

the Internet flows interact with the out-of-box CC algorithms. Finally, Scenario (c) (Internet-only) considers the case where both flows originate from the Internet to investigate how and whether their interactions differ from the previous scenarios.

The common goal of all three scenarios is to judge the bandwidth sharing behavior of different CC algorithms in different network settings for which we consider four network characteristics. The bottleneck *bandwidth* and the overall *RTT* hereby give hard upper bounds (in terms of available data rate) and lower bounds (in terms of responsiveness) on the overall performance, while the bottleneck queue, characterized by its *queue size* and *queuing discipline*, introduces jitter, and loss.

B. Testbed Setup

The core of our testbed consists of one machine that represents the end-user and hence serves as the client throughout the scenarios and another machine which represents the user's access link and hence the bottleneck of the overall connections. The latter is then used to model all connection-specific properties like delay or bandwidth. For the scenarios where we create flows from within our network, we deploy one machine for each flow that is involved and configure server-side parameters like the deployed CC algorithm on them. All machines within the testbed use a Linux 5.2 kernel modified by Google to support BBRv2 [28] and they are interconnected via Gigabit Ethernet ((1) in Fig. 1) to ensure that the physical links never become a bottleneck. To account for the common setting that end users access the Internet via a WiFi connection to their router, we can additionally select a wireless 802.11n link between client and bottleneck ((2) in Fig. 1).

Limiting Bandwidth. Most configurations, like rate-limiting, are done on the bottleneck's egress queues. Here, we configure the bandwidth and queue using a token bucket filter with a burst size of a single frame while using different queuing techniques. Even though Internet access links are often asymmetrical, we disregard this fact as we are not interested in reverse-path congestion and use the same bandwidth in both directions.

Ensuring Same-RTT Scenario. To reason about the main question of this work, i.e., the bandwidth sharing properties of Internet flows, we generally employ the same-RTT setting in which intra- and inter-protocol fairness dictate that two flows should share the bandwidth equally if they have the same RTT (see Sec. II). We add delay to our testbed by using TC to perform ingress packet processing at our bottleneck. There, we redirect traffic to an intermediate queue disc enabling us to

Setting	Parameter Space
Bandwidth	10 Mbps
RTT	50 ms, real
Buffer sizes	$0.5 \times$ BDP, $2 \times$ BDP
Queueing discipline	drop-tail, CoDel, FQ_CoDel

TABLE I: The study parameter space.

use NetEm to add a delay before we release the packet for forwarding to the actual egress queue. We do not configure any artificial jitter using NetEm as this causes packet reordering; the additional delay and jitter are thus only caused by the egress queue. To have symmetric delays, we add half of the configured delay to each ingress of the bottleneck. While care needs to be taken in sizing the NetEm queue to not cause artificial packet loss, this approach has the advantage that the end-host stacks are not involved in the delay which is known to badly interfere with CC when Linux detects queuing pressure (TCP small queues) [29]. Further, in Scenarios (b) and (c) we even have no control over all end-hosts. To ensure that we can investigate the same-RTT fairness, we set different delays for each flow to harmonize their RTTs. To this end, we measure the *minimum* RTT through our testbed (using TCP pings) when not using any artificial delays for each flow. We then use each flow's min RTT to configure delays such that all flows experience the same artificial min RTT. Note that our measurements in Sec. VIII-B are an exception to this as we study the CPs' behavior using their real RTTs.

Limitations. Our testbed has several limitations that need consideration. We must ensure that our traffic shaper is the actual bottleneck of the path from the CP to our client. Since we do not have full control over all involved entities, we can only configure bandwidths that are sane given our interconnection. Our testbed uses Gigabit Ethernet, our Institute is then connected via 10 Gbps to our University's backbone, which in turn connects to the German research network (DFN) via 40 Gbps which then peers at DE-CIX with all CPs investigated in this study. Thus, shaping traffic for typical end-user access links should render the bottleneck to our traffic shaper. Further, we need to artificially bump up the RTTs at least to the largest minimum RTT measured. For us, the CPs typically show RTTs around 5 ms to 20 ms.

Additionally, to ensure repeatability and independence, we take several precautions to avoid undesired side-effects. First, to investigate the interaction of CC, we must be actually limited by the congestion window which is why we advertise an initial flow-control receive window of 200 segments. In the same way, we ensure that send and receive buffers are large enough to fully utilize the available bandwidth and do not introduce an undesired new bottleneck. Finally, we clear all TCP caches after each measurement to ensure that cached metrics such as ssthresh do not affect future measurements (testbed only).

C. Parameter Space

Selecting reasonable parameters for our testbed is challenging. We must adhere to the testbed's limitations while seeking to replicate a reasonable end-user environment. Table I summarizes the parameter space which we discuss next.

Bandwidth. To ensure that the bottleneck is within our testbed, we have to set the bottleneck link bandwidth accordingly. For this, we perform a number of bandwidth tests to determine which data rates are reliably offered by the individual CPs and find the lowest data rates to be around 60 Mbps. We choose 10 Mbps as a lower bound to represent a low-end connection.

RTTs. We choose 50 ms as the lower bound for the minimum RTT even though we expect typical CPs to usually have much lower RTTs to their customers. However, these increased RTTs make it possible to reduce the relative error when we pad up the RTTs to ensure the same level of delay between connections. Additionally, we perform one experimental series where we use the real RTTs to the CPs. In our previous work, we also investigate the performance for bandwidths of 50 Mbps and RTTs of 100 ms, but find qualitatively similar results [12].

Buffer Sizes. For the bottleneck buffer, we experiment with different queue sizes since we know of no study that investigates typical last-mile buffer sizes. While the potential for overly large buffers (bufferbloat) is known [30], less than 1% of the end-user flows were observed to experience RTT variations larger than 1 sec by a major CDN [31]. Therefore, we choose one overly large buffer in the order of 2×BDP and, inspired by research advocating new buffer sizing rules ($\sqrt{num\_flows}$ [32] and $\log win\_size$ [33]), one smaller buffer size of 0.5×BDP which, for our investigated bandwidths and delays, yields queue sizes between both proposed sizing rules.

AQM. In addition to these parameters, we also change the queuing discipline between a regular drop-tail queue and (FQ-) CoDel [34] to investigate the impact of AQM on fairness.

D. Capturing Traffic (And Measuring Retransmissions)

We rate the fairness by capturing the traffic that the client receives for each flow and observe the queue size of the bottleneck buffer by using a simple eBPF program on the bottleneck machine. Additionally, we monitor the number of retransmissions of the different flows using two distinct mechanisms. For the flows generated in the testbed, we use an eBPF program on the corresponding testbed machine which directly yields the retransmission counts as reported by the TCP stack. As this approach is not possible for traffic originating from CPs, we deploy tcptrace [35] as an additional tool to derive approximate retransmission counts from our general traffic captures. More specifically, we use packets classified as out of order as a reasonable approximation for the actual retransmission counts. This is suitable in our special scenario, because apart from the retransmissions we do not expect any packet reordering as we have a close connection to the IXP and as the CPs themselves will not induce packet reordering. Still, using out of order packets as an indicator for retransmissions comes with the disadvantage that tail loss, i.e., packet loss that occurs at the end of traffic bursts cannot be detected because the corresponding retransmissions will look like normal transmissions at the beginning of the next traffic burst in our captures and thus tcptrace will not detect them.

E. Traffic Scenarios

We distinguish between two different traffic scenarios.

Long Flow Scenario. In the first scenario (identical to our previous work), the client requests a flow from one machine and after 5 s a second flow from another machine. Both flows then continue to transmit data for another 40 s before shutting down. Of the overall 45 s, we investigate 35 s starting 5 s after the second flow entered. While this methodology is intended to focus on the long term fairness between the two flows, we also examine whether it is important which flow is started first by including experiments with a flipped starting order.

Short Flow Scenario. Our second scenario is motivated by the fact that a significant portion of Internet traffic is relatively short Web traffic which is often forced to compete with bulk transfers similar to those in the first scenario. We project this observation into our testbed in that the client first requests a bulk transfer from one machine (similar to the first scenario) and after 5 s a second (short) flow from another machine which transmits 2.5 Mbyte of data. We then analyze the complete transmission period of the second, i.e., shorter, flow as we are particularly interested in how well the short flow can claim bandwidth and in how well the bulk flow releases bandwidth.

In both scenarios, we repeat each measurement 30 times to investigate the stability of our results.

F. Fairness and Performance Metrics

We employ different fairness and performance metrics to judge the behavior of the flows in the two traffic scenarios. To rate the fairness, we look at the ratio of transmitted bytes (over the same timespan of the shorter flow) and define our fairness measure as:

$$fness(a,b) = \begin{cases} 1 - \frac{bytes(a)}{bytes(b)} & \text{if bytes(b)} \geq \text{bytes(a)} \\ -1 + \frac{bytes(b)}{bytes(a)} & \text{if bytes(a)} > \text{bytes(b)} \end{cases}$$

Intuitively, *fness*(A,B) maps the fairness behavior of the two flows into the range of [-1, 1] with zero indicating absolute fairness, -1 that Flow A absolutely dominates Flow B and a value of 1 the opposite. In between, the measure depicts the ratio of bytes actually transmitted, e.g., 0.5 indicates that Flow B transmitted twice the bytes compared to Flow A.

While this ratio-based metric is well-suited for our long flow scenario, we refrain from applying it to the short vs. bulk flow scenario, because there the short flow will probably never get to the fair share of the bottleneck as its transmission volume is simply too small. Instead, we use the flow completion time (FCT) of the short flow to judge the bandwidth acquisition and release mechanisms of both the short and bulk flow. For this, we measure the time needed by the short flow to complete its transmission, starting with the first received fragment *after* the initial handshake and ending with the last transmitted byte of payload *before* the connection is terminated.

V. CC in Controlled Environments

We perform the first part of our measurement campaign in Scenario (a) (testbed-only). We thereby seek to validate that our testbed produces meaningful results by testing whether the performance of out-of-box CC algorithms in our testbed is similar to the findings of related work (cf. Sec. III). As related work considers a wide range of parameter settings and variations of dumbbell topologies, their exact replication is beyond the scope of our testbed validation. Further, we use our local setting to present a study of BBRv2 fairness (Sec. V-B). Last, we investigate the impact of a wireless connection to the users' bottleneck (with potentially higher error rates) on fairness (Sec. V-C).

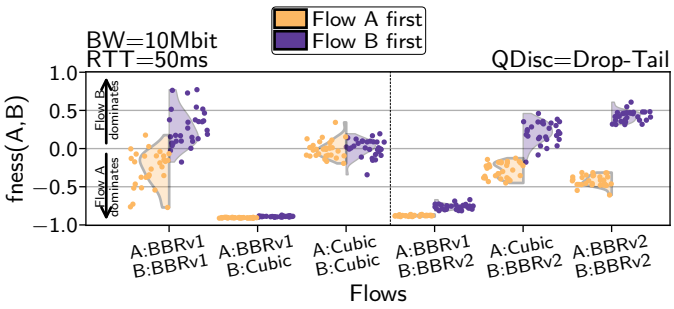

Fig. 2: Results from Scenario (a) for different CC algorithm combinations (BW: 10 Mbps, RTT: 50 ms, Buffer: 0.5×BDP).

A. Testbed Validation

We validate our testbed in a setting with 10 Mbps, a min RTT of 50 ms, and a buffer of 0.5×BDP with BBRv1 and Cubic testbed flows. The left side of Fig. 2 visualizes our corresponding *fness* metric in the form of a scatterplot. We show all of our measured values together with a kernel density estimate to better visualize the location of the majority of our measured data. For each combination of algorithms, we plot the results when flow A starts first (yellow) side by side with the switched setting when flow B starts first (violet). When a CC algorithm performs against itself, switching which flow starts first consequently only mirrors the data at the 0-axis.

For Cubic, our results show expected values as it shows a large degree of fairness to itself (*intra*-protocol fairness). BBRv1 is also fair in *most* cases, although there are also very unfair situations where the flow that is started first dominates the other. This behavior is not visible in our previous work [12] where we find BBRv1 to have a significantly higher level of fairness although the high variance is also present. When comparing the *inter*-protocol fairness, we observe that BBRv1 clearly monopolizes the bandwidth regardless of which flow starts first. This confirms related work on BBRv1 in low-buffer scenarios [22], [23], [25]. These experiments hence validate that our testbed yields meaningful results, because they confirm various known findings which encourages us to employ it for real Internet flows. Before we discuss these settings in more detail, we first use our local setting to present an early evaluation of BBRv2 and examine if a WiFi connection to the bottleneck has a noticeable impact on fairness for the user.

B. BBRv2

The measurement results for BBRv2 are shown on the right side of Fig. 2 and allow us to make three interesting observations. First, BBRv2 has the lowest level of *intra*-protocol fairness of the three investigated CC algorithms. Second, BBRv2 is dominated by the BBRv1 flow. Third,

		BBRv1			BBRv2			Cubic		
		QSize	Retransm. 1st	Retransm. 2nd	QSize	Retransm. 1st	Retransm. 2nd	QSize	Retransm. 1st	Retransm. 2nd
2nd flow	BBRv1	76%	1966	1414	65%	993	254	60%	860	195
	BBRv2	70%	497	1085	60%	197	80	55%	190	73
	Cubic	63%	217	745	57%	122	102	61%	100	50

TABLE II: Avg. queue size (QSize) and avg. retransmissions of first (1st, column) and second (2nd, row) flow for different CC combinations where the second flow joins a running first flow (BW: 10 Mbps, RTT: 50 ms, Buffer: 0.5×BDP).

BBRv2 achieves the highest level of fairness when competing against Cubic. We suspect that the latter two observations stem from the fact that BBRv2, in contrast to BBRv1, incorporates features of a loss-based CC algorithm.

To better judge the impact of CC on the overall network, we further consider key characteristics of the bottleneck buffer and the participating end-hosts by observing the queue size of the bottleneck buffer and the number of retransmissions of the two flows. In this scenario, we use our eBPF-based utility to measure the retransmissions as we have full control over all end hosts. Table II shows these characteristics averaged over the 30 measurements. As can be seen, BBRv1 induces the highest number of retransmissions of all investigated CC algorithms, especially against itself. This is caused by BBRv1 slightly overestimating the BDP for performance reasons which causes a small standing queue and, in the small buffer scenario, already suffices to cause packet loss [36]. BBRv2 is able to drastically reduce the number of retransmissions, which we attribute to its loss-based features, while Cubic as a pure loss-based algorithm causes the fewest retransmissions.

Due to our special setting, the loss can be solely attributed to congestion on the bottleneck link. In reality, however, a significant portion of end-users accesses the Internet using mobile devices like smartphones or laptops and the wireless communication medium as such typically comes with higher error rates. To rule out that transmission errors on the MAC layer have an impact on the overall performance, we additionally investigate if the use of a wireless connection to the bottleneck influences the performance from the end-users perspective in an office environment.

C. The Effect of WiFi on Fairness

For the investigation of the wireless link, we deploy Scenario (2) in Fig. 1 switching to an 802.11n WiFi link while leaving the rest of the parameterization as is, i.e., traffic is still shaped but is subsequently affected by the Layer 2 medium access and retransmissions of the wireless link.

Fig. 3 compares our earlier measurement (Fig. 2) for Ethernet connections with those in the WiFi setting. As can be seen, the effect of WiFi medium access on fairness is negligible in our investigated setting as we observe almost the same result distribution as in the wired setting. This confirms the general intuition that the bottleneck is the defining property of the overall connection. Additional iterations of our later experiments also in the WiFi setting generally produce similar results in that the WiFi connection does not seem to have a significant effect on the fairness.

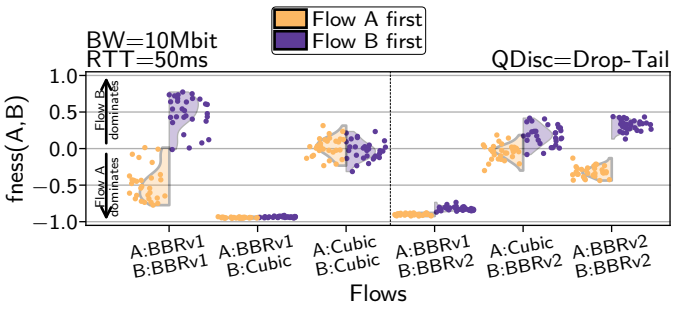

Fig. 3: Results from Scenario (a) for different CC algorithm combinations using a WiFi connection to the bottleneck (BW: 10 Mbps, RTT: 50 ms, Buffer: 0.5×BDP).

Takeaway. *By confirming known findings, namely a high intra-protocol fairness for Cubic and a low inter-protocol fairness of Cubic and BBRv1, we first validate that our testbed yields meaningful results. We further find that the loss-based features of BBRv2 improve BBRv1's inter-protocol fairness with Cubic and additionally reduce the number of retransmissions as well as the queue size caused by BBRv1 in small buffer settings. Finally, we find that the choice of connection medium, i.e., either Ethernet or WiFi, to the bottleneck has a limited impact on the level of fairness for the end-user.*

We next turn to Scenarios (b) and (c) to study how CPs, and thus possibly non-standard algorithms from the Internet, perform against our known CC algorithms and against each other. For this, we first describe how we select the targets for our study in Sec. VI before we focus on the semi-controlled settings in Sec. VII and CC in the wild in Sec. VIII.

VI. Selecting Internet Targets

We base our evaluation of TCP fairness on actual Internet traffic by six major CPs (Akamai, Amazon, Cloudflare, Edgecast, Fastly, and Google) in two settings: *i)* lab vs. CP and *ii)* CP vs. CP. Studying actual Internet traffic is motivated by the observation that CC research often neglects the complex parameterization possibilities. For example in a previous study [10], we found that CDNs use different initial window configurations and some utilize pacing. To this end, we suspect that not only the initial windows might be different, thus we choose two URLs for Akamai (named AkamaiA (then using IW32) and AkamaiE (then using IW16)) mapping to these different settings. Furthermore, Amazon, Cloudflare, and Google have all publicly announced to utilize BBRv1. Thus, we investigate the performance of actual Internet traffic originating from these six different CPs when competing against our testbed flows in Scenario (b) and against themselves in Scenario (c).

For the long flow experiments, we collect target CP-hosted URLs generating large responses (the smallest being 25 MB) by analyzing the HTTPArchive. Since they can still be too small to cover our 45 s measurement period, we make use of HTTP/2 multiplexing, i.e., we request the same resource multiple times (in parallel, on the same connection) enabling us to prolong the transmission by a multiple of the original size. The h2load tool in nghttp2 [37] provides this functionality.

Finding suitable URLs for the short flow experiments is more difficult. On the one hand, the response sizes have to be

representative for a "typical" website; on the other hand, the responses of the different CPs should be very similar in size so that we can easily compare their FCTs. The HTTPArchive reports median (average) website sizes of about 2 MB (3 MB) in July 2019, and thus, we aim for a website size roughly in between those two values. Also accounting for the goal that all CPs should have about the same response size, we finally arrive at 2.474 MB for the largest website and 2.470 MB for the smallest website. The results in this work base on our measurement campaign from July to September 2019; results from February 2019 are presented in our initial paper [12].

VII. CC in Semi-Controlled Settings

We start with Scenario (b) where traffic from our testbed machines (BBRv1, BBRv2, Cubic) competes with Internet traffic, i.e., the algorithms deployed by the CPs. Using our knowledge from Sec. V, we aim to interpret and characterize the behavior of the CPs and group the CPs accordingly. For this, we once again configure our testbed with 10 Mbps and a min RTT of 50 ms and additionally vary between small (0.5×BDP) and large (2×BDP) buffer sizes. We plot the measurements results using our *fness* metric: here, results < 0 indicate a dominance for our testbed flow while results > 0 favor the CP.

A. Cubic

Small Buffer. Fig. 4 shows the results for a local Cubic flow over a small (top) and large (bottom) buffer. In the small buffer scenario, Amazon[1], Cloudflare, and Google clearly dominate the traffic in all instances giving little bandwidth to the Cubic flow (unfair setting). In contrast, Edgecast and Fastly struggle against our Cubic flow even when their flow starts first (unfairness by our testbed flow). The two Akamai flows offer differing behavior as AkamaiA is similar to Amazon, Cloudflare, and Google in that it completely dominates our testbed Cubic flow while the local Cubic flow dominates AkamaiE similarly to Edgecast and Fastly. This difference in behavior of the two Akamai flows supports our initial guess that Akamai uses different configuration parameters.

Large Buffer. When looking at the large buffer setting in the plot below we observe a different picture. The Amazon, Cloudflare, and Google flows do not dominate anymore and the fairness heavily depends on which flow was initiated first. Edgecast and Fastly also see a fairer bandwidth distribution, at least if their flows are started first as they still struggle to gain enough bandwidth when the testbed initiates the first flow. What is very interesting to see is that both Akamai flows are completely dominated by the testbed Cubic flow, no matter which flow is started first.

B. BBRv1

Small Buffer. Things start to significantly differ when we configure our testbed flow with BBRv1 as shown in Fig. 5. As can be seen in the upper plot (small buffer), the testbed BBRv1 flow dominates the flows of Edgecast and Fastly as well as

[1]Note that Amazon behaved similarly to Edgecast and Fastly in our previous study [12], but has since rolled out BBRv1.

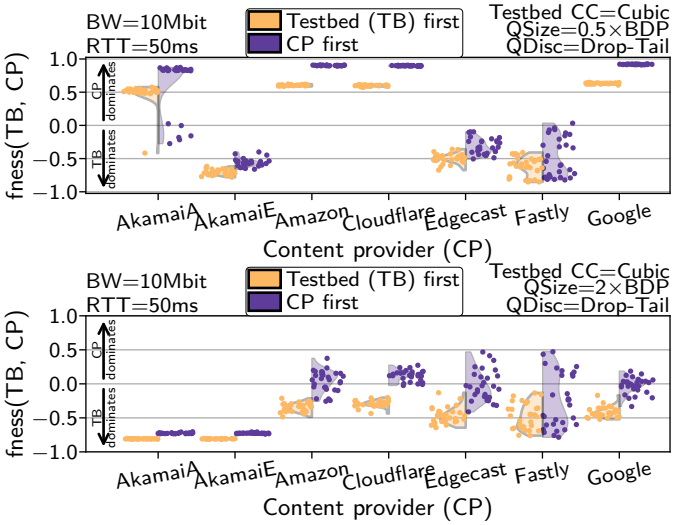

Fig. 4: A testbed Cubic flow competes with the CPs for traffic using a small buffer (top) and a large buffer (bottom).

AkamaiE. AkamaiA shows an interesting behavior, as part of the measurements also show the clear dominance of the testbed flow while about half of the iterations either show a much fairer result (testbed started first) or even a dominance of the Akamai flow (Akamai started first). The observed characteristics hereby seem to be stable in that the behavior seems to switch between two distinct states. Google shows a wide range of observed fairness ratios although it dominates the testbed flow in most cases. For Amazon and Cloudflare, we observe a high level of fairness when the CP flows are started first and a slight advantage for the testbed flows when they are started first.

Large Buffer. Similar to our observations for Cubic, we also find a higher level of fairness when the local BBRv1 flow operates over a larger buffer. Amazon, Cloudflare, and Google hereby achieve the highest fairness values while Edgecast and Fastly have a significantly improved fairness, although the latter comes with larger fairness ranges. Finally, both Akamai flows are still dominated by the local BBRv1 flow.

C. BBRv2

Small Buffer. When switching our testbed flow to BBRv2, interesting observations can be made from our results illustrated in Fig. 6. In the small buffer setting (top) the BBRv2 flow is dominated by Amazon, Cloudflare, and Google as well as AkamaiA which is similar to the behavior of our Cubic testbed flow. At the same time, there is a high level of fairness when competing against AkamaiE, Edgecast, and Fastly with slight advantages for the flow that is started first. Overall, the behavior of BBRv2 much more resembles Cubic than BBRv1 which we trace back to the fact that BBRv2 incorporates feedback in the form of ECN and packet loss.

Large Buffer. For larger buffers, we observe the same behavior when competing against Edgecast and Fastly, although there is a slightly larger variance compared to the small buffer setting. AkamaiA and AkamaiE behave similar to each other in that there is a very high level of fairness when the CP flows are started first and a slight domination of the testbed if the BBRv2 flow is started first. Regarding Cloudflare and Google, one can

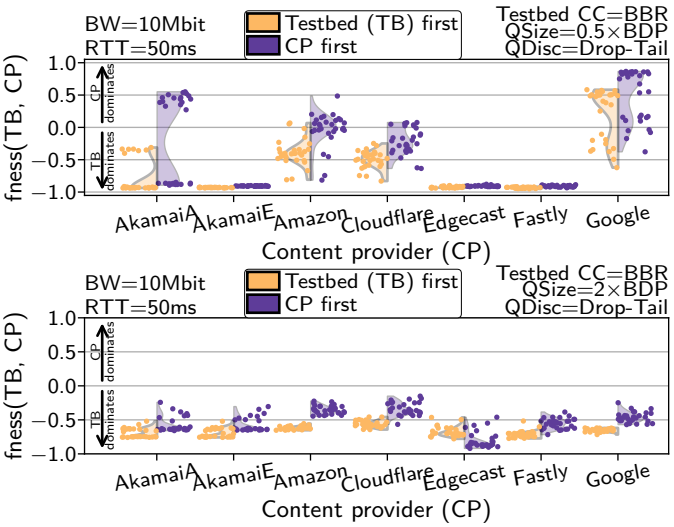

Fig. 5: A testbed BBRv1 flow competes with the CPs for traffic using a small buffer (top) and a large buffer (bottom).

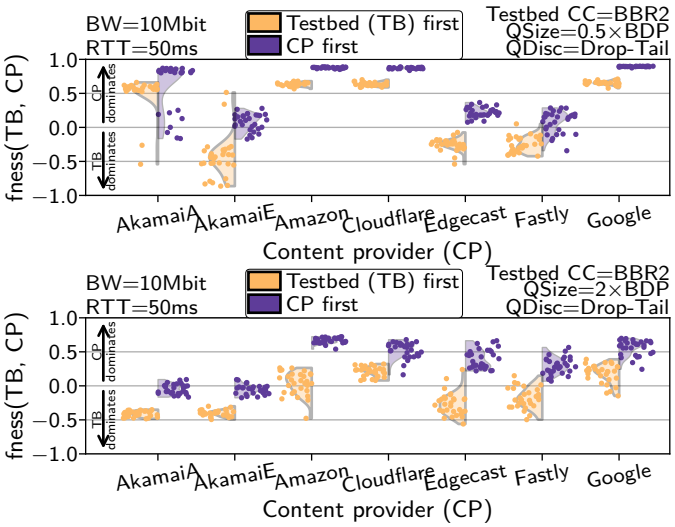

Fig. 6: A testbed BBRv2 flow competes with the CPs for traffic using a small buffer (top) and a large buffer (bottom).

		BBRv1@2BDP			BBRv1@.5BDP			BBRv2@2BDP			BBRv2@.5BDP			Cubic@2BDP			Cubic@.5BDP		
		QSize	Retransm. TB	Retransm. CP	QSize	Retransm. TB	Retransm. CP	QSize	Retransm. TB	Retransm. CP	QSize	Retransm. TB	Retransm. CP	QSize	Retransm. TB	Retransm. CP	QSize	Retransm. TB	Retransm. CP
2nd flow by	AkamaiA	55%	240	109	61%	1014	655	33%	54	12	71%	176	389	72%	23	13	75%	78	238
	AkamaiE	56%	254	110	55%	847	254	34%	57	14	39%	113	210	71%	22	21	51%	39	147
	Amazon	52%	61	64	66%	1364	1656	46%	37	189	58%	208	931	67%	39	63	59%	153	880
	Cloudflare	58%	189	229	67%	1386	1759	52%	36	126	61%	208	1104	70%	38	156	60%	148	954
	Edgecast	64%	326	265	56%	761	181	49%	47	290	47%	77	69	77%	34	200	57%	41	57
	Fastly	64%	357	137	55%	685	217	53%	42	123	49%	75	116	78%	31	77	57%	37	125
	Google	52%	31	37	69%	1026	1563	47%	43	110	58%	224	896	66%	39	49	59%	177	747

TABLE III: Average queue size (QSize) and average retransmissions of the testbed (TB, column) and of the CP (CP, row) flow where the CP flow joins a running TB flow (BW: 10 Mbps, RTT: 50 ms).

say that the CP flows have the advantage in all settings, but it is the most pronounced when the CP flows also start first. Finally, for Amazon, there is a high level of fairness when the testbed flow is started first while Amazon clearly dominates if it is started first.

D. CC Interpretation

Based on our observations in this section and the queue size and retransmission statistics shown in Table III we now attempt to characterize the behavior of the different CPs. Note that we here measure the retransmissions of both flows using our tcptrace methodology which only gives us approximate results although we could achieve more accurate results for the testbed flow by using the eBPF-based tool. This, however, would make a comparison between the retransmission counts of the testbed and the CP flows impossible.

Group A: Amazon, Cloudflare, Google. For Amazon, Cloudflare, and Google, it is publicly known that they use BBRv1 and our results confirm these public announcements. First, we can see that the three CPs dominate our local Cubic flow in the small buffer scenario while the interaction is fairer in the larger buffer scenario. We further find a large range of fairness values when interacting with our local BBRv1 flow. Finally, the three CPs induce the largest queue sizes as well as the most retransmissions in the small buffer scenario which we have previously seen by BBRv1 flows in our purely local testbed. Still, differences in performance between the three CPs, e.g., regarding the local BBRv1 flow in the small buffer scenario, indicate that the CPs use different configurations for their CC.

Group B: Edgecast, Fastly. Edgecast and Fastly show clear indicators for using a loss-based CC. They are dominated by the local BBRv1 flow and achieve a medium level of fairness when competing against the local Cubic flow in the small buffer scenario. Furthermore, we find comparably small queue sizes and few retransmissions in the low buffer scenario which suggests that they react to loss at the bottleneck.

Group C: AkamaiA, AkamaiE. The two Akamai flows are difficult to characterize as they do not present a clear behavior throughout all scenarios. They sometimes behave similarly e.g., when competing against Cubic or BBRv1 in the large buffer setting, and sometimes differ, e.g., when AkamaiA dominates the local Cubic flow over small buffers where AkamaiE is dominated by Cubic. From this observation, we conclude that the difference in configuration that we have noted in previous work [10] indeed has an impact on the fairness behavior. We additionally find that AkamaiA shows an interesting bi-modal behavior when competing against BBRv1 in the small buffer setting. This further leads us to believe that AkamaiA might in fact also change configuration between connections. Regarding the concrete CC variant, we speculate that both Akamai versions might use a loss-based component as they significantly struggle against BBRv1.

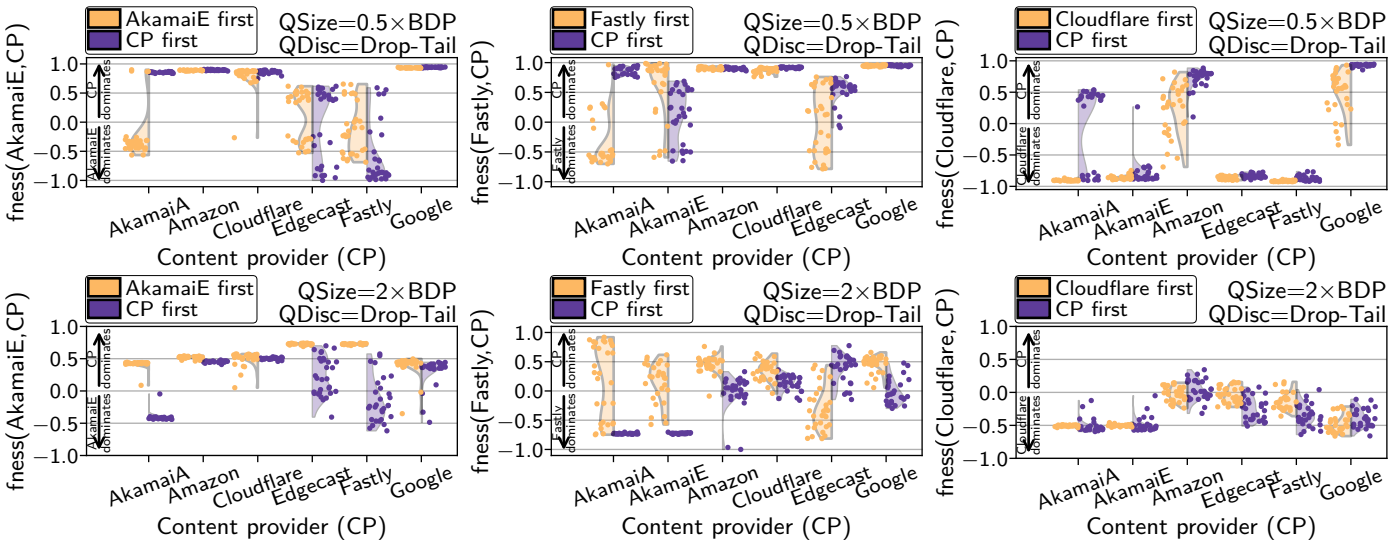

Fig. 7: AkamaiE (Group C), Fastly (Group B), and Cloudflare (Group A) competing against the other content providers. (BW: 10 Mbps, RTT: 50 ms, Buffer: 0.5×BDP (top) / 2×BDP (bottom)).

Takeaway. *We find some degree of heterogeneity regarding the CC algorithms deployed by our CPs. Amazon, Cloudflare, and Google use BBRv1, Edgecast and Fastly use some loss-based CC and the Akamai flows use different algorithms, potentially even on a per-connection basis. We further find that the CC algorithms employed by the CPs generally seem to achieve better fairness with off-the-shelf algorithms when more buffer size is available. However, large buffers can cause jitter and generally inflate the latency. Yet in small buffer settings, BBRv1 currently claims nearly all bandwidth and shows a large variability in fairness when competing with other BBRv1 flows.*

While it might seem advantageous at first glance that algorithms like BBRv1 claim more bandwidth, it could actually be bad for CPs. In the web, CPs often compete with 3rd party resources loaded on the same website. When the CP claims all bandwidth, it may negatively affect the web page loading behavior since competing flows of the other resources could suffer from reduced performance. Thus, CPs should interact fair with their competitors which is the focus of the next part of our study, i.e., how two CP flows interact.

VIII. CC in the Wild

To investigate the interaction between the different CPs, we deploy Scenario (c) where both flows are requested from the CPs. As we are interested in examining the interactions from many angles, we vary several testbed parameters to get a detailed picture of the influencing factors on fairness. We first perform a baseline experimental series where we investigate how the case of two CP flows compares to the setting with one CP and one local flow (VIII-A). The experiments thereafter all base on this fundamental testbed configuration and each only varies *one additional* parameter. More precisely, we investigate if and how our observations change if we no longer artificially bump up the RTTs and instead use the CPs' real RTTs (VIII-B), how short and long flows of the different CPs interact (VIII-C) and, finally, if and how AQMs can improve the observed fairness in the different settings (VIII-D). Throughout this section, we perform all experiments using a 10 Mbps bottleneck link and a min RTT of 50 ms, except for Sec. VIII-B where we use the CPs' real RTTs. We further restrict our presentation to the results of AkamaiE, Fastly, and Cloudflare as representatives of the groups described in Sec. VII-D.

A. Baseline Experiments (CP vs. CP)

In our first experimental series, we are interested in how AkamaiE, Fastly, and Cloudflare perform against the other CPs. Fig. 7 shows the corresponding results for a scenario with a small (top) or large (bottom) buffer size. Recall our fairness measure, where results < 0 indicate a dominance of the explicitly mentioned CP while results > 0 favor the competing CP mentioned on the x-axis.

Small Buffers. Starting with the upper row, i.e., the small buffer scenario, we observe that the CPs generally interact as expected based on the results of the previous sections. Cloudflare's BBRv1 flows dominate the loss-based flows of Group B (right) and Fastly is also dominated by the rest of group A (middle). The BBRv1 versions of Amazon and Google hereby seem to be more aggressive as they both dominate Cloudflare if they are started first while there is a large fairness range when Cloudflare is the first flow (right). Compared to their interaction with group A, Edgecast and Fastly together achieve a higher level of fairness although there is a large range of fairness values when Fastly is started first (middle). The most interesting observation can be made for AkamaiA when it is started first and competes with the BBR flows of Cloudflare (right), Amazon, and Google (not shown) as the bi-modal behavior is again visible. As AkamaiE also struggles against the flows of group A, we find another hint for our guess that Akamai uses some loss-based CC algorithm which detects congestion earlier than BBRv1. Finally, we observe that there is a large range of fairness values when AkamaiE

2nd flow by	AkamaiE@2BDP QSize	AkamaiE@2BDP Retransm. 1st	AkamaiE@2BDP Retransm. 2nd	AkamaiE@.5BDP QSize	AkamaiE@.5BDP Retransm. 1st	AkamaiE@.5BDP Retransm. 2nd	Fastly@2BDP QSize	Fastly@2BDP Retransm. 1st	Fastly@2BDP Retransm. 2nd	Fastly@.5BDP QSize	Fastly@.5BDP Retransm. 1st	Fastly@.5BDP Retransm. 2nd	Cloudflare@2BDP QSize	Cloudflare@2BDP Retransm. 1st	Cloudflare@2BDP Retransm. 2nd	Cloudflare@.5BDP QSize	Cloudflare@.5BDP Retransm. 1st	Cloudflare@.5BDP Retransm. 2nd
AkamaiA	47%	2	1	50%	194	208	85%	80	110	53%	132	174	57%	51	16	58%	1052	266
AkamaiE	-	-	-	-	-	-	85%	78	61	69%	170	239	58%	31	21	57%	1109	222
Amazon	58%	31	154	63%	286	987	75%	149	342	63%	241	805	59%	88	80	70%	1804	1573
Cloudflare	61%	47	277	62%	289	1035	75%	133	433	62%	241	872	-	-	-	-	-	-
Edgecast	75%	21	93	50%	186	76	82%	65	54	60%	132	73	69%	138	120	61%	943	184
Fastly	73%	22	20	48%	190	79	-	-	-	-	-	-	71%	91	34	61%	925	182
Google	57%	43	126	69%	356	1132	75%	168	446	69%	252	1018	56%	50	37	77%	1800	1671

TABLE IV: Average queue size (QSize) and average retransmissions for the first (1st, column) and second (2nd, row) where the second flow joins the running first flow (BW: 10 Mbps, RTT: 50 ms).

competes against the loss-based Group B, which is different to its interaction with our out-of-the-box Cubic in Sec. VII-A.

Large Buffers. Focusing on the bottom row, i.e., the larger buffers, we can similarly predict several interaction schemes based on our previous observations. For the interaction between group A and the other CPs, we find that larger buffers generally improve the fairness which we attribute to the fact that larger buffers create enough headroom so that BBRv1's small standing queue does not completely fill up the real queue. The headroom is, e.g., used by the loss-based flows of Group B which are no longer completely dominated by Group A (middle). A similar observation can be made for the interaction of Cloudflare with both, Amazon and Google. For the interaction of Group B, we do not see a significant impact of the buffer size on the fairness which supports our findings in Sec. VII-A where we see a similar effect when Group B competes with Cubic. When the two Akamai flows interact, the flow starting second always dominates the first flow.

Retransmissions and Queue Size. Table IV shows the average queue size (QSize) and the retransmission statistics for the case where Flow 1 starts first, which is the CP shown in the top row of the table and in yellow in Fig. 7. As expected, large buffers reduce the loss rate and thus the number of retransmissions while we generally see high retransmission counts for the small buffer scenario. This is especially true for interactions with flows of Group A as these cause very high retransmission counts, both for themselves and their competitors while the loss-based Group B causes smaller numbers of retransmissions. The inverse is true for the queue sizes, because here Group B causes larger queues than Group A in the large buffer setting while they all cause quite full queues in the small buffer scenario. What is interesting to see is that the largest queues are caused when the loss-based CPs Edgecast and Fastly compete with the Akamai flows. This again indicates a potential loss-based component included in the CC of Akamai.

Combining previous findings, the fairness levels shown in Fig. 7 and the statistics in Table IV, it can be seen that larger buffers, in most cases, also increase fairness and reduce the number of retransmissions for the interaction of CPs. However, larger buffers come with the risk of higher queue delays which is why model-based approaches like BBR try to estimate the available BDP to not overly fill the buffer. For this, BBRv1 and BBRv2 critically depend on the RTT which, in contrast to the bottleneck bandwidth, is a highly volatile, flow-specific property. In this regard, our testbed abstracts from reality as we pad the RTTs that we observe from the different CPs to the same level so that we can properly judge fairness in the same RTT case. In reality, however, users will experience different RTTs to the different CPs and even in our special setting, we observe RTTs ranging from 5 ms to 20 ms. In fact, we have an RTT of slightly below 5 ms to all CPs except for Fastly to which we have an RTT of around 20 ms which is explained by the fact that the URL under investigation is actually hosted in London and not in Frankfurt. Thus, to examine the performance of the CPs in a more realistic setting, we loosen our RTT padding for the following experiment.

B. Behavior with real RTTs

To investigate how the CPs interact when they are not subject to our RTT padding, we repeat our baseline experiment described in Sec. VIII-A but no longer artificially adjust the RTTs. This in turn has an impact on choosing the buffer sizes as we compute them based on the BDP and thus on the RTT. As we no longer balance the RTTs, we no longer have a common RTT which we can use for these computations. Instead, we choose the highest RTT, i.e., 20 ms, for our computations to reflect the generally assumed over buffering in the Internet.

Fig. 8 shows the measurement results for the small buffer scenario. Starting with Fastly, it can be seen that it gets dominated by all other CPs no matter in which order the flows are started. This is most likely due to the already mentioned fact that Fastly has by far the largest RTT. Apart from that we observe expected behavior as *Group A* once more dominates the competition and as Amazon and Google are more aggressive than Cloudflare. What is interesting to see is that under real conditions, AkamaiE achieves almost perfect fairness when competing against AkamaiA and Edgecast.

In our experiments with larger buffers (2× BDP, results not shown), we once again find much fairer settings and even Fastly now achieves a rough level of fairness with all CPs and is at most dominated by a factor of 2. Consequently, we do not find significant differences to the investigation scenario with a balanced RTT for most of the CPs as we have similar RTTs to them anyway as most of them have a presence in Frankfurt. However, we find that Fastly, which has by far the highest RTT, is substantially dominated by the other CPs in the small buffer scenario, but larger buffers help to ameliorate this.

While our long flow scenario is well-suited to investigate the general interaction of different CPs as CC algorithms take some time to negotiate appropriate sending rates and, in fact, continuously do so, a significant portion of Internet

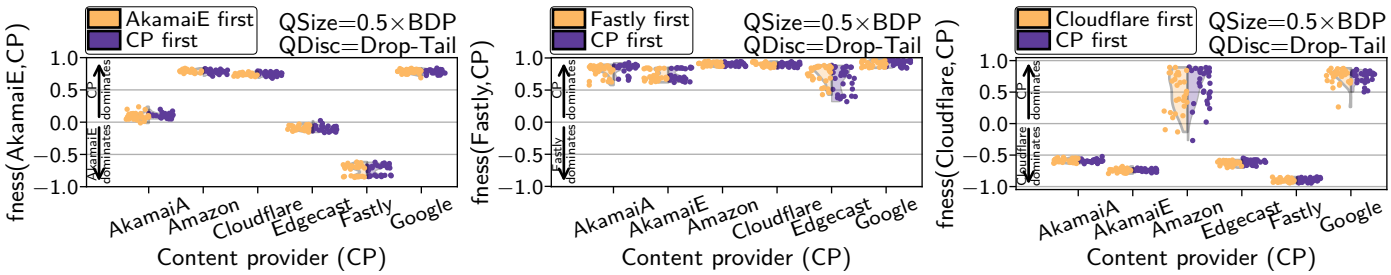

Fig. 8: AkamaiE (Group C), Fastly (Group B), and Cloudflare (Group A) competing against the other content providers. (BW: 10 Mbps, RTT: real RTT, Buffer: 0.5×BDP calculated for an RTT of 20ms).

traffic is Web traffic which is characterized by relatively short transmissions and thus not yet captured. Hence, we next investigate how the different CPs behave on shorter time scales.

C. The Behavior of Short Flows

In the Internet, short flows carrying small amounts of data (mice) are often forced to compete with long-running bulk transfers (elephants). To emulate this setting, we let a short flow join a link that is already occupied by a bulk transfer. Thus, the short flow has to quickly capture bandwidth while the bulk transfer has to release it. Given the limited transfer volume of the short flow, it is rare that they actually achieve a fair share of the bandwidth as the transmission is typically already completed before reaching that level. We map the mice vs. elephant scenario to our testbed by replacing one of the long-running CP flows with a short flow. For this, we use URLs generating responses of around 2.47 MB; the client consequently first requests the long-running flow, and after 5 s the short flow. We judge the behavior of the two flows by analyzing the time needed by the short flow to complete its transmission, i.e., its flow completion time (FCT). This metric serves as a proxy for how well the flow can capture bandwidth and how often the long flow gave it transmission opportunities. Using the FCT has also been proposed as a useful metric for CC [38] and is most suited to compare the behavior of the short flows as there is no general notion on how to define fairness for short flows (cf. Sec. II).

Fig. 9 shows the FCTs for the CPs competing against each other with 0.5 (top) and 2 (bottom) × BDP buffers and RTTs once again equalized to 50 ms. For each combination of algorithms, we plot the results when the selected CP is the short flow (yellow) side by side with the switched setting where the selected CP represents the long flow (violet).

Small Buffers. As shown in the upper row of Fig. 9, short flows of most CPs struggle against long flows of Group A, i.e., Amazon, Cloudflare, and Google, while they finish earlier when competing against long flows of AkamaiE, Edgecast, and Fastly. We suspect that the latter are better at quickly releasing bandwidth due to their loss-based features while BBRv1, and with that Group A, generally holds the transmission rate steady and only drastically reduces it every 10 s during the ProbeRTT phase. Regarding their short flow behavior, Group A flows generally seem to have the fastest FCTs, even if they compete against other Group A long flows while the loss-based flows especially struggle against such long flows.

Large Buffers. In most combinations, deploying larger buffers (bottom) shortens the transmission times. The largest gain can be identified in settings where the short flows compete against Group A long flows as they can now finish much faster. This might once again be explained by the fact that BBRv1 overestimates the BDP on purpose which can be enough to degrade performance in low buffer scenarios while it leaves larger headroom in larger buffers which can then be claimed by the short flows. It is also a valid explanation for the observation that interacting Group A flows have a more focused fairness range than in the small buffer setting. On the other hand, the FCT for interacting Group B flows deteriorates which we attribute to our previous observation that loss-based CC algorithms cause highest queue size in the large buffer scenario and consequently delay the transmissions of the short flow.

Interpretation. These inferences base on the findings of the previous sections and are backed by further statistical data about these experiments. Table V lists the average queue size (QSize) as well as the average number of retransmissions for the long flow (flow in the row) and the short flow (flow in the column) for the experiments visible in Fig. 9. A first observation is that Group A short flows induce the highest number of retransmissions which we believe is the reason why they are so effective in claiming bandwidth to quickly finish their transmissions: the retransmissions suppress loss-based algorithms and free up bandwidth which can then be captured by the short flows themselves. This, however, means that they generally buy higher performance with (drastically) increased retransmission counts. The completely opposite can be seen for a short AkamaiE vs. a long AkamaiA and a long Google flow as they operate completely lossless (top/bottom left of the table) at least as long as the buffer is large enough. This, however, also shows that few retransmissions may not go hand in hand with a fast finish as the AkamaiE flow is about 2 times slower when competing against Google than when competing against AkamaiA.

In the small buffer scenario, Group A long flows cause significant retransmissions, both to themselves and to the short flows, which consequently suffer from longer FCTs. We can also see that Group B long flows, by far, cause the largest queue sizes in the large buffer scenario which thus adds significant queueing delay to their short flows and suffices to degrade performance when two Group B flows interact. This, above all, nicely illustrates that larger buffers always come with the risk of increased queue delays although they seem to improve fairness

Long flow by	AkamaiE@2BDP QSize	AkamaiE@2BDP Retransm. Short	AkamaiE@2BDP Retransm. Long	AkamaiE@.5BDP QSize	AkamaiE@.5BDP Retransm. Short	AkamaiE@.5BDP Retransm. Long	Fastly@2BDP QSize	Fastly@2BDP Retransm. Short	Fastly@2BDP Retransm. Long	Fastly@.5BDP QSize	Fastly@.5BDP Retransm. Short	Fastly@.5BDP Retransm. Long	Cloudflare@2BDP QSize	Cloudflare@2BDP Retransm. Short	Cloudflare@2BDP Retransm. Long	Cloudflare@.5BDP QSize	Cloudflare@.5BDP Retransm. Short	Cloudflare@.5BDP Retransm. Long
AkamaiA	31%	0	0	80%	112	197	35%	5	1	80%	45	106	41%	187	23	79%	314	219
AkamaiE	-	-	-	-	-	-	35%	4	2	43%	17	68	42%	227	25	49%	264	77
Amazon	44%	1	3	54%	122	514	45%	2	7	58%	92	621	49%	120	66	60%	273	623
Cloudflare	55%	12	64	57%	114	553	50%	2	46	57%	78	620	-	-	-	-	-	-
Edgecast	79%	14	170	57%	76	36	78%	17	172	54%	8	24	77%	359	192	53%	297	44
Fastly	77%	49	63	57%	80	82	-	-	-	-	-	-	76%	388	86	58%	255	89
Google	44%	0	0	60%	198	549	46%	2	3	60%	118	529	49%	106	67	69%	422	724

TABLE V: Average queue size (QSize) and average retransmissions for the short (columns) and long (rows) flows where the short flow joins the running long flow (BW: 10 Mbps, RTT: 50 ms).

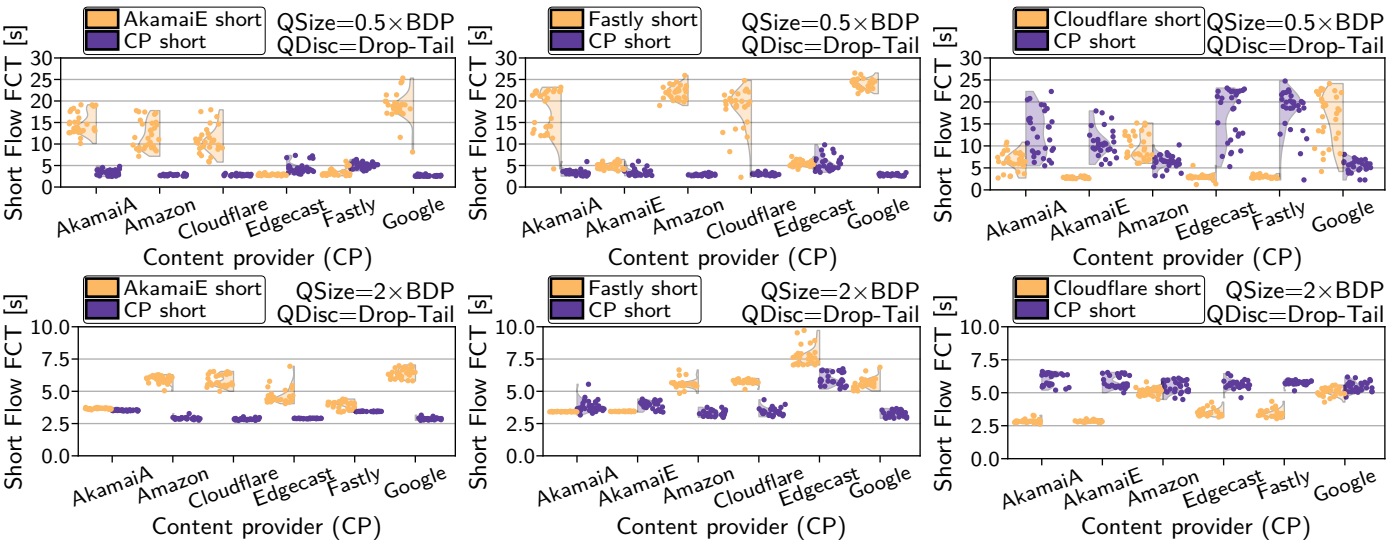

Fig. 9: Short flow scenario: AkamaiE (Group C), Fastly (Group B), and Cloudflare (Group A) competing against the other content providers. (BW: 10 Mbps, RTT: 50 ms, Buffer: 0.5×BDP (top) / 2×BDP (bottom)).

and FCTs in most cases. Ideally, we would have a scenario with a smaller queue but still the high level of fairness and quick flow completions. As AQMs like CoDel are especially designed to keep the delay (and hence the queue) small, we are interested in whether they can help to achieve the desired combination of small delays on one side and a high level of fairness as well as fast FCTs on the other side. This is why we investigate the effect of AQMs in our base setting, in the real RTT scenario and in the short flow setting in the next section.

D. Can CoDel Improve Fairness?

AQMs inherently change the behavior of a queue which is why they have a significant impact on the overall performance. Generally, they have two possible forms of feedback to which flows might respond: *i)* dropping packets and *ii)* using ECN. In our work, we only consider the first case of feedback, i.e., packet drops because it requires no end-to-end support, which is outside of our control at CPs. We repeat the experiments from before but activate CoDel and its flow-queuing variant FQ_CoDel on the intermediate bottleneck machine. In the following, we further concentrate on the case with a queue size of 2×BDP because CoDel's effect on small queues is likely to be diminishing. Fig. 10 exemplarily shows the fairness results in the otherwise unchanged scenarios previously used in Sec. VIII-A, i.e., for 10 Mbps and a min RTT of 50 ms, while Table VI further shows the average queue size (QSize) and the average retransmissions for both flows where the second flow joins the first ongoing flow.

CoDel. Our main observation is that CoDel generally achieves its goal of reducing the queue sizes. Compared to the drop-tail case in Table IV where the queue has an occupancy of 50% to 85%, CoDel lowers the maximum average queue size to 28%. What is interesting to see is that while in the large buffer settings without AQM, the loss-based Group B causes the largest queue sizes, it is now Group A using BBRv1 which causes larger queues as their CC does not react to the intentional loss induced by CoDel. CoDel (top of Fig. 10) further seems to achieve a very high level of fairness when Fastly competes with Edgecast and when Cloudflare competes with Amazon and Google. Similarly, AkamaiA and AkamaiE interact in a fair manner in half of the settings while they each dominate in half of the cases when they are started first, thus once more showing their bi-modal behavior. Consequently, one can say that CoDel seems to achieve high fairness values in the intra-protocol setting, i.e., when both parties use the same CC algorithm. In the inter-protocol setting, we find very bad fairness values when Group B competes against the remaining CPs. We thus suspect that their CC quickly reacts to the drops of CoDel and consequently adjusts their rates while the other CPs seem to ignore the loss feedback. This is somewhat surprising for the Akamai flows as we have previously speculated about potential

2nd flow by	AkamaiE+CoDel QSize	AkamaiE+CoDel Retransm. 1st	AkamaiE+CoDel Retransm. 2nd	AkamaiE+FQ_CoDel QSize	AkamaiE+FQ_CoDel Retransm. 1st	AkamaiE+FQ_CoDel Retransm. 2nd	Fastly+CoDel QSize	Fastly+CoDel Retransm. 1st	Fastly+CoDel Retransm. 2nd	Fastly+FQ_CoDel QSize	Fastly+FQ_CoDel Retransm. 1st	Fastly+FQ_CoDel Retransm. 2nd	Cloudflare+CoDel QSize	Cloudflare+CoDel Retransm. 1st	Cloudflare+CoDel Retransm. 2nd	Cloudflare+FQ_CoDel QSize	Cloudflare+FQ_CoDel Retransm. 1st	Cloudflare+FQ_CoDel Retransm. 2nd
AkamaiA	17%	2894	882	16%	2418	955	13%	161	2801	10%	97	2092	17%	1513	489	18%	1690	1189
AkamaiE	-	-	-	-	-	-	14%	152	2488	12%	90	2276	18%	1675	1273	19%	1584	1376
Amazon	19%	2705	1772	16%	2545	847	12%	142	1020	11%	84	1439	28%	1746	1555	19%	1208	870
Cloudflare	19%	2538	1754	16%	2763	859	12%	146	1122	12%	88	1511	-	-	-	-	-	-
Edgecast	14%	2982	154	16%	2779	633	4%	72	58	13%	91	1411	16%	1170	132	20%	1151	771
Fastly	14%	2787	126	13%	2629	65	-	-	-	-	-	-	14%	1299	126	16%	1716	85
Google	18%	2797	1439	18%	2694	712	11%	143	930	11%	88	1241	25%	1840	1296	15%	1141	716

TABLE VI: Average queue size (QSize) and average retransmissions for the first (1st, column) and second (2nd, row) where the second flow joins the running first flow (BW: 10 Mbps, RTT: 50 ms).

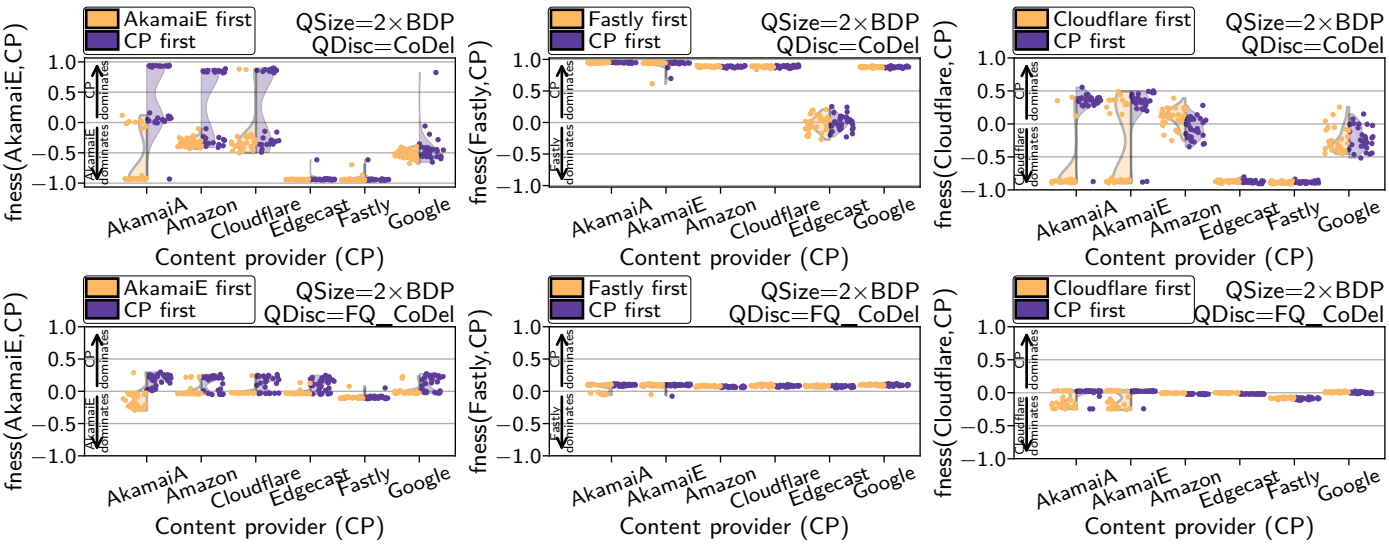

Fig. 10: AkamaiE (Group C), Fastly (Group B), and Cloudflare (Group A) competing against the other content providers. (BW: 10 Mbps, RTT: 50 ms, Buffer: 2×BDP, queueing discipline: CoDel (top) / FQ_CoDel (bottom)).

loss-based components. The findings here rather point to a model- or delay-based CC.

Overall, it can be said that CoDel achieves lower queue delays at the cost of higher retransmission counts. Moreover, it seems to have a negative impact on the fairness between CC algorithms using different indicators for identifying congestion and currently disadvantages purely loss-based CC algorithms.

FQ_CoDel. Shifting towards the flow-queuing variant (bottom of Fig. 10), which is designed to produce a fair queuing, we observe a tremendous increase in fairness. Throughout all measurements, fairness is close to the equilibrium and we only observe slight variations. These variations are mostly visible for the two Akamai flows which once again show a tendency towards a bi-modal pattern when competing against Group A. FQ_CoDel further reduces the queue sizes in the Group A settings and overall achieves a much more uniform queue size distribution as queue sizes now only range from 10% to 20% across all investigated settings while the retransmission counts stay roughly in the same order of magnitude.

Impact on performance. We also investigate the effects of FQ_CoDel in the other scenarios and our results generally confirm the high level of fairness in the real RTT setting (results not shown). While occasional loss to keep the delay low is certainly acceptable for long-running connections, we have previously seen that the performance of short flows can be degraded when they are affected by loss. We thus also revisit the short vs. long flow scenario and find that for short flows competing against an AkamaiE long flow, FCTs almost double to 4-5 seconds compared to around 2.5 seconds in Fig. 9. On the other hand, rather long FCTs, e.g., in scenarios with Group B long flows, are now reduced to the same level. Thus, it can be concluded that FQ_CoDel successfully builds upon the original idea of CoDel, namely reducing delay, by adding a component to achieve flow fairness. This fairness is not only achieved regarding the overall throughput of long flows, but different combinations of short and long flows all roughly yield the same FCT. Unfortunately, FQ_CoDel still causes high retransmission counts, especially for algorithms such as BBRv1 that ignore such loss, showing that the fairness leaves a bitter aftertaste given the amounts of wasted bandwidth.

Takeaway. *Summarizing, it can be said that the original CoDel above all seems to improve the intra-protocol fairness as evidenced by high fairness values for the interaction of the CPs within their own groups. This is bad news for the heterogeneous Internet, as scenarios with different algorithms suffer from severe unfairness. Luckily, the flow-queuing variant enables a large degree of fairness even in heterogeneous settings. It also balances the expected FCT of short flows and thus, at the cost of some CPs losing performance, creates an even playground for all CPs in the Internet which is especially good*

news for web content which is often distributed across several CPs. Thus, it seems to again stand that the technologies to enable a fair and performant Internet are available and only need to be deployed at the bottlenecks.

IX. Conclusion

In this work, we empirically investigate the fairness of content providers in the Internet to inform network management practices. Using our testbed, we examine the behavior of actual Internet traffic when competing under lab-controlled properties of a bottleneck. Generally, we only find limited fairness in the Internet today. Some content providers interact well with each other, while others do not which is likely reflected in their choice of CC algorithm. Customizations of the CPs hereby seem to significantly impact the fairness and can easily help them to get an unfair advantage. The bottleneck itself also has a strong effect on the fairness as its buffer size often inverts observations when going from small to large. This demands research to shine a light on actual configurations of bottleneck buffer sizes in the Internet and on customizations of CPs to then investigate, e.g., the impact on web performance when content is served from a diverse set of content providers. Still, there is a silver lining: network management can put the fairness back into the network operator's hand by utilizing state-of-the-art AQMs such as FQ_CoDel, although this requires deployment on millions of devices at the edges of the network.

Acknowledgment. Funded by the Deutsche Forschungsgemeinschaft (DFG) as part of project B1 within the Collaborative Research Center (CRC) 1053 – MAKI and under Germany's Excellence Strategy — EXC-2023 Internet of Production — 390621612.

References

[1] M. Allman *et al.*, "TCP Congestion Control," RFC 5681, 2009.
[2] B. Briscoe, "Flow Rate Fairness: Dismantling a Religion," *SIGCOMM CCR*, vol. 37, no. 2, 2007.
[3] A. Medina *et al.*, "Measuring the Evolution of Transport Protocols in the Internet," *SIGCOMM CCR*, vol. 35, no. 2, 2005.
[4] S. Floyd *et al.*, "Comments on the Usefulness of Simple Best-Effort Traffic," Internet Requests for Comments, RFC 5290, 2008.
[5] J. Thompson *et al.*, "Q1 2017 State of the Internet - Connectivity Report," Akamai, Tech. Rep., 2017.
[6] M. Trevisan *et al.*, "Five Years at the Edge: Watching Internet from the ISP Network," in *ACM CoNEXT*, 2018.
[7] J. Erman *et al.*, "Over the Top Video: The Gorilla in Cellular Networks," in *ACM IMC*, 2011.
[8] E. Carisimo *et al.*, "Studying the Evolution of Content Providers in the Internet Core," in *IEEE/IFIP TMA*, 2018.
[9] C. Labovitz *et al.*, "Internet Inter-domain Traffic," in *ACM SIGCOMM*, 2010.
[10] J. Rüth *et al.*, "Demystifying TCP Initial Window Configurations of Content Distribution Networks," in *IEEE/IFIP TMA*, 2018.
[11] P. Yang *et al.*, "TCP Congestion Avoidance Algorithm Identification," in *IEEE ICDCS*, 2011.
[12] J. Rüth *et al.*, "An Empirical View on Content Provider Fairness," in *IEEE/IFIP TMA*, 2019.
[13] S. Floyd, "Metrics for the Evaluation of Congestion Control Mechanisms," Internet Requests for Comments, RFC 5166, 2008.
[14] R. Ware *et al.*, "Beyond jain's fairness index: Setting the bar for the deployment of congestion control algorithms," 2019.
[15] N. Cardwell *et al.*, "BBR: Congestion-Based Congestion Control," *ACM Queue*, vol. 14, Sept.-Oct., 2016.
[16] N. Cardwell *et al.*, "BBR v2 A Model-based Congestion Control," https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr, 2019.
[17] N. Cardwell *et al.*, "BBR v2: A Model-based Congestion Control IETF 105 Update," https://datatracker.ietf.org/meeting/105/materials/slides-105-iccrg-bbr-v2-a-model-based-congestion-control.pdf, 2019.
[18] D. J. Leith *et al.*, "Experimental evaluation of Cubic-TCP," in *Workshop on PFLDNet*, 2007.
[19] L. Xue *et al.*, "A study of fairness among heterogeneous TCP variants over 10 Gbps high-speed optical networks," *Optical Switching and Networking*, vol. 13, Jul. 2014.
[20] S. Ha *et al.*, "CUBIC : A New TCP-Friendly High-Speed TCP Variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, 2008.
[21] D. Miras *et al.*, "Fairness of High-Speed TCP Stacks," in *IEEE AINA*, 2008.
[22] N. Cardwell *et al.*, "BBR Congestion Control," https://www.ietf.org/proceedings/97/slides/slides-97-iccrg-bbr-congestion-control-01.pdf, 2016.
[23] M. Hock *et al.*, "Experimental Evaluation of BBR Congestion Control," in *IEEE ICNP*, 2017.
[24] S. Ma *et al.*, "Fairness of Congestion-Based Congestion Control: Experimental Evaluation and Analysis," 2017. [Online]. Available: http://arxiv.org/abs/1706.09115
[25] D. Scholz *et al.*, "Towards a Deeper Understanding of TCP BBR Congestion Control," in *IFIP Networking*, 2018.
[26] A. Dhamdhere *et al.*, "Inferring Persistent Interdomain Congestion," in *ACM SIGCOMM*, 2018.
[27] S. Bauer *et al.*, "The Evolution of Internet Congestion," in *TPRC*, 2009.
[28] https://github.com/google/bbr/tree/v2alpha.
[29] N. Cardwell, "BBR evaluation with netem," https://groups.google.com/d/msg/bbr-dev/8LYkNt17V_8/xyZZCwcnAwAJ.
[30] J. Gettys *et al.*, "Bufferbloat: Dark Buffers in the Internet," *Communications of the ACM*, vol. 55, no. 1, Jan. 2012.
[31] O. Hohlfeld *et al.*, "A QoE Perspective on Sizing Network Buffers," in *ACM IMC*, 2014.
[32] G. Appenzeller *et al.*, "Sizing Router Buffers," in *ACM SIGCOMM*, 2004.
[33] M. Enachescu *et al.*, "Routers with Very Small Buffers," in *IEEE Infocom*, 2006.
[34] K. Nichols *et al.*, "Controlling Queue Delay," *ACM Queue*, vol. 10, no. 5, May 2012.
[35] http://www.tcptrace.org/.
[36] Y. Cao *et al.*, "When to use and when not to use BBR: An empirical analysis and evaluation study," in *ACM IMC*, 2019.
[37] https://github.com/nghttp2/nghttp2.
[38] N. Dukkipati *et al.*, "Why Flow-Completion Time is the Right Metric for Congestion Control," *ACM SIGCOMM CCR*, vol. 36, no. 1, 2006.

Ike Kunze is a researcher and Ph.D. student at the Chair of Communication and Distributed Systems (COMSYS) at RWTH Aachen University. He received his M.Sc. degree (with honors) in Computer Science from RWTH Aachen University in late 2018. His research interests include in-network computing, transport protocols, and active queue management.

Jan Rüth received his B.Sc. and M.Sc. in Computer Science from RWTH Aachen University. In 2014, he joined the Chair of Communication and Distributed Systems at RWTH Aachen University as a Ph.D. student and researcher. His research interests include transport protocols, internet measurements, and performance evaluations in general.

Oliver Hohlfeld is a professor of computer science and heads the Chair of Computer Networks at Brandenburg University of Technology. Before, he was at RWTH Aachen University and at TU Berlin / Deutsche Telekom Innovation Laboratories. He was a visiting scholar at the group of Paul Barford at the University of Wisconsin - Madison, USA. He studied computer science at Darmstadt University of Applied Sciences, Institute Eurecom (Sophia Antipolis, France), and Darmstadt University of Technology and holds a B.Sc. and M.Sc. degree. He obtained a Ph.D. (Dr. rer. nat.) from TU Berlin in 2013.