Stateful-BBR – An Enhanced TCP for Emerging High-Bandwidth Mobile Networks

Lingfeng Guo
Department of Information Engineering
The Chinese Univeristy of Hong Kong
Hong Kong
Email: gl016@ie.cuhk.edu.hk

Yan Liu
Cloud ARCH & Platform Dept
Tencent
China
Email: rockyanliu@tencent.com

Wenzheng Yang
Cloud ARCH & Platform Dept
Tencent
China
Email: wzyhktk@gmail.com

Yuming Zhang
Department of Information Engineering
The Chinese Univeristy of Hong Kong
Hong Kong
Email: zy219@ie.cuhk.edu.hk

Jack Y. B. Lee
Department of Information Engineering
The Chinese Univeristy of Hong Kong
Hong Kong
Email: jacklee@computer.org

2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS) | 978-1-6654-1494-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/IWQOS52092.2021.9521358

***Abstract*— With the progressive deployment of 5G networks around the world, mobile networks are entering a new era where bandwidth will be breaking through the Gbps barrier. In this work, we investigate the performance of current TCP designs in such high-bandwidth networks, demonstrating the potential bottleneck due to TCP's Slow-Start mechanism which is an integral component in most TCP designs. For example, transferring a file of 1 MB size in a first-generation 5G network using Linux's default TCP-Cubic and Google's TCP-BBR resulted in average throughputs of 18.2 Mbps and 32.8 Mbps, respectively. Compared to the mean available bandwidth of 180 Mbps, the gap is significant. To tackle this problem, we developed an enhanced Stateful-TCP technique to transform BBR into a new S-BBR to accelerate its startup performance to narrow the gap. Results from trace-driven emulated 5G network experiments show that S-BBR could improve BBR's throughput performance by 50% to 100% while maintaining similar delay performance. This is further validated by an independent competitive benchmark using over 500 clients where S-BBR raised BBR's throughput by 69%. S-BBR is sender-based and thus can be readily deployed in Internet servers without any requirements from the client side, it retains BBR's desirable features and so offers a promising solution to enhance mobile applications' performance in the emerging high-bandwidth mobile and wireless networks.**

Keywords—slow-start, stateful, BBR, BBRv2, mobile, Wi-Fi, networks.

I. INTRODUCTION

With the introduction of 5G mobile networks, mobile communications are poised to rival the performance of their wired counterparts. The first-generation 5G networks could offer bandwidths up to 1 Gbps under good network conditions. Even in normal environments, e.g., inside an office with a stationary 5G smartphone, the measured bandwidth can easily exceed 300 Mbps, with a mean bandwidth of 180 Mbps. Moreover, further advances in 5G will soon breakthrough the Gbps barrier, surpassing even many wired networks connected via GBE [1].

With the leaps in mobile network bandwidth, it is now up to the end-systems, i.e., servers and mobile devices, to exploit it to improve the performance of mobile applications and services. In this study, we focus on enhancing the Transmission Control Protocol (TCP) to realize the benefits from the vastly increased bandwidth in the emerging high-speed mobile networks.

There have been continuous innovations in the design and optimization of TCP over the years due to its central importance to most Internet services and applications. With three decades' of research, modern TCP designs are all very efficient, able to achieve very high bandwidth efficiency if the flow is sufficiently long [2-17]. However, this last condition is increasingly challenged by two factors.

First, TCP's flow size is primarily determined by the specific application utilizing it. For applications such as large software download/update or movie download, the large amount of data transferred will enable TCP to ramp up its throughput to take advantage of the bandwidth available. On the other hand, there are many other applications which transmit data in sporadic short bursts, e.g., images in social media. Moreover, even video streaming has evolved away from RTP/RTSP-based streaming [18] to DASH-based streaming [19] where a video is divided into small segments of data, each delivered in separate HTTP transactions.

This latter evolution is significant as video now accounts for 73% of all Internet traffics and is projected to increase further in the future [20]. Consider a video encoded at a medium bitrate of 1 Mbps. If each video segment is 2 seconds then the mean segment size (i.e., flow size) will be 250 KB only. Previous work [21-24] have demonstrated that at these flow sizes, TCP will likely complete the data transfer before it can ramp up its transmission rate to fully utilize the bandwidth available.

Second, the rapid increase in mobile network bandwidth further compounds the problem. While mobile network bandwidth has increased by one order of magnitude, e.g., from 4G's 100 Mbps to 5G's 1 Gbps, the network propagation delay is only reduced slightly, e.g., from 30~50 ms in 4G to 10~20 ms in 5G. The relevance of this is that TCP's transmission rate ramp-up speed is inversely proportional to the path RTT as

978-0-7381-3207-5/21/$31.00 ©2021 IEEE

978-1-6654-1494-4/21/$31.00 ©2021 IEEE

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on November 23,2021 at 06:28:57 UTC from IEEE Xplore. Restrictions apply.

most TCPs' congestion control algorithms are driven by acknowledgement (ACK) packets returned from the receiver. Thus, compared to the bandwidth increase in 5G networks, the network delay will present another hurdle to TCP's ability to take full advantage of the abundant bandwidth available.

We tackle this challenge in this work by applying the recently-introduced Stateful-TCP paradigm [11] to the TCP-BBR developed by Google [5, 12]. Stateful-TCP aims at speeding-up TCP's initial transmission rate by learning the network parameters such as path bandwidth and propagation delay from a previous flow to the same peer, and then apply it to configure the transmission rate in the new flow's startup phase. It complements TCP's congestion control and error control algorithms and thus could be applied to most existing TCP designs. It was first applied to Cubic in a recent work [11] which raised Cubic's throughput performance by over 50% at flow size of 1 MB.

This work focuses on BBR because it was designed to be more resilient to non-congestion losses than Cubic, and thus offers higher performance especially in mobile and wireless networks [5]. Moreover, excluding Cubic, BBR is among the more widely-deployed TCP designs in the Internet and is gaining popularity among many service providers.

In the previous work [11], Stateful-TCP has been shown to work well in emulated networks, cloud VM environments, as well as a competitive benchmarking platform. However, actual performance results from deploying Stateful-TCP in production Internet services are not yet available. In this work, we collaborated with a tier-1 CDN service provider which offered us a rare opportunity to conduct comparative performance benchmarking in production servers hosting a very large scale real-world Internet service. This not only directly verifies the feasibility of Stateful-TCP, but also provides, for the first time, its actual performance gains achievable in real Internet services.

Our experimental results obtained from trace-driven emulation of 5G network showed that Stateful-BBR (or S-BBR for short) could achieved substantially higher throughput than BBR, ranging from 50% to 100% depending on flow size and random loss rate. Moreover, in spite of the higher throughput, S-BBR exhibited packet queuing delays comparable to the original BBR. A further validation experiment conducted by an independent benchmarking company using over 500 clients also produced similar throughput performance gains (69%) in wired and Wi-Fi networks. Last but not least, we were able to deploy Stateful-BBR in Tencent's production servers and the initial results suggested that S-BBR could offer substantially higher throughput performance (+27%) as well as reduced low-throughput cases (-6.6%) compared to BBR.

The rest of the paper is organized as follows: Section II reviews some previous related works; Section III presents experimental results to validate Stateful-TCP's assumptions in production Internet servers; Section IV presents the design and implementation of Stateful-BBR; Section V evaluates and compares S-BBR to current TCP designs; Section VI summarizes the paper and outlines some future work.

II. Previous Related Wrok

Three decades of research in TCP has produced many novel designs. An exhaustive review of them is beyond the scope of this paper. Below we first briefly review some of the more well-known designs and then review the related works on accelerating TCP's initial transmission rate.

Although no official statistics are available, the most widely-deployed TCP design is very likely to be TCP-Cubic [2] as it is the default TCP in both Linux and Microsoft Windows. Cubic, and its ancestor BIC [6], were designed to perform well in large-BDP networks. However, its congestion control algorithm is relatively sensitive to random packet losses. This motivates research in loss-resilient TCPs. Notable examples include Westwood [3], Veno [4], Vivace [9], Sprout [8], and BBRv1/v2 [5,12], of which Google's BBR has gained considerable deployment in and out of Google's services.

Another class of TCP designs were aimed at achieving low packet latency which is critical to many delay-sensitive applications. Notable examples include C2TCP [13], Copa [10] and ExLL [14]. More recently, researchers have also begun to explore the use of machine generated and machine learning approaches to the design of TCP, e.g., Taova [16], Indigo [15], and Orca [17].

Most previous works focused on TCP's long-term performance (i.e., tens of seconds). While that is an important performance goal, TCP's short-flow performance is also significant in practice as many Internet applications transfer data in sporadic short bursts. To this end, TCP's short-flow performance is primarily constrained by its Slow-Start phase which always begins with a conservative transmission rate and then ramps it up exponentially until it exits the Slow-Start phase. To mitigate the Slow-Start bottleneck, researchers have proposed novel ways to set the initial congestion window (CWnd) size or sending rate, e.g., based on explicit feedback from routers [21], based on receiver's advertised window (AWnd) size [22, 23], or using learning-based approach [24].

In a closely-related work, Guo and Lee [11] developed a Stateful-TCP paradigm to accelerate TCP's Slow-Start using network parameters obtained from the previous flow to the same peer. They applied it to Cubic to form S-Cubic where the initial CWnd is set according to the path bandwidth-delay-product (BDP) estimated in the previous flow, and apply pacing in the first RTT to smooth out the initial transmission. S-Cubic was shown to achieve substantial performance gain over Cubic and in one of the experiments it even outperformed BBRv1. This motivates us to investigate the application of Stateful-TCP to BBR which potentially could push the performance boundary even further.

III. An Experimental Validation of Stateful-TCP

Taking advantage of our access to the tier-1 CDN service provider's production servers, we first conduct a set of experiments to validate the assumptions behind Stateful-TCP in the context of a real Internet service. The service itself is an app store for downloading and updating mobile apps all over the world.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on November 23,2021 at 06:28:57 UTC from IEEE Xplore. Restrictions apply.

We modified the TCP implementation in the chosen servers to log various statistics of all TCP flows served. The logging module has been carefully designed and tuned to minimize impact to the server's performance. In the following, we analyze the collected TCP statistics to validate two key assumptions behind Stateful-TCP.

A. Stateful-TCP Hit Rate

Stateful-TCP is designed to make use of flow information obtained from a previous flow to the *same* peer to configure the initial transmission rate of the new flow. Therefore, it could offer performance gains only if the same client connects to the server multiple times. Intuitively, most Internet applications are likely to initiate multiple TCP connections to the server in a single session, there is no quantitative data on the exact extent of it. Moreover, CDN service providers often employ server load balancer to distribute the incoming requests to the server farm so subsequent requests from the same client may not necessarily be diverted to the same server, thus reducing the opportunity for Stateful-TCP to activate.

We set out to investigate this question by logging the client IP addresses in a production server. The first time a client IP connects to the server it will log its IP address in a hash table and count the connection as a miss. If the same client IP initiates additional TCP connections afterwards then those connections will be counted as a hit, representing the case where Stateful-TCP can take effect. We ran the experiment continuously for one week with a server table size of 64M entries.

Fig. 1 plots the accumulated number of TCP connections and the Stateful-TCP hit rate over time. There are two notable observations. First, the hit rate grew with time as one would expect, reaching 90% in one week's time. This implies that repeated connections to even a single server is significant. Second, the server's Stateful-TCP hit rate increased sharply in the first 122 minutes, reaching a level of 65%. This suggests that the ramp-up period for Stateful-TCP to take effect is reasonably short and thus could begin offering performance gains soon after a server is started. This is also consistent with the intuition that many Internet applications inherently generate multiple TCP connections in an application session so that it does not take long for repeated connections to the same client IP to appear.

It is also worth noting that the production service under study is a mobile app store. Intuitively, an app store session is typically one-off rather than multiple app downloads. Therefore, we expect other services such as web, video streaming, etc., will exhibit even more rapid ramp-up of the Stateful-TCP hit rate.

B. Path BDP

Once activated, Stateful-TCP makes use of information obtained from the previous flow, e.g., throughput, minimum RTT, and BDP in S-Cubic [11], to bypass TCP Slow-Start to accelerate the initial transmission rate. A key assumption here is that TCP's Slow-Start is the bottleneck during TCP's startup phase. While this is widely-recognized and can easily be

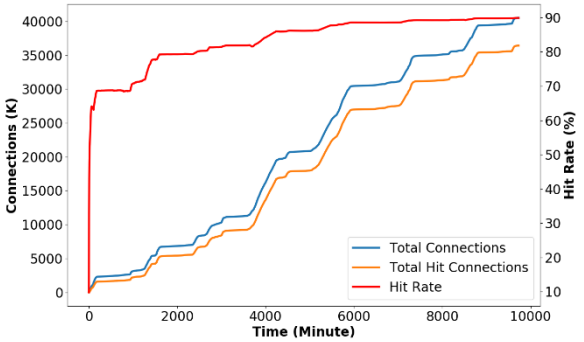


Fig. 1. One-week connection statistics from a production server.

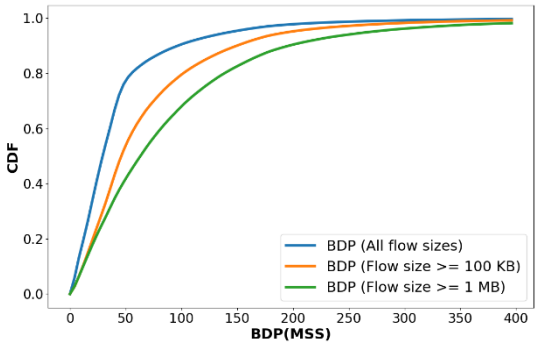


Fig. 2. Distribution of BDP from production servers.

demonstrated in network testbeds, there is little quantitative evidence in the literature on its extent in real Internet services.

To fill the gap, we conducted experiments in 10 production app-store servers over a period of one week to measure and log each TCP flow's BDP. BDP is estimated from the product of minimal RTT measured in a TCP flow and the TCP flow's mean throughput. In particular, if the BDP is equal to or smaller than the initial CWnd (i.e., default of 10 MSS in Linux), then Slow-Start will not be a bottleneck. In contrast, the larger the difference between BDP and initial CWnd, the more performance gain will be attainable by Stateful-TCP.

Fig. 2 plots the cumulative distribution function (CDF) for the BDP of all TCP flows recorded. It is evident that a significant proportion of flows, i.e., 80.4%, have estimated BDP exceeding 10 MSS. Note that some of the flows were very short (e.g., less than 100 KB) and thus would underestimate the BDP (again due to Slow-Start). If we exclude flows shorter than 100 KB then the proportion of flows with BDP larger than 10 MSS increased further to 89.5%. Thus even if one increases the initial CWnd to a larger value such as 50 MSS, the proportion at ~40% (≥100 KB) or ~60% (≥1 MB) is still significant. This confirmed that TCP's Slow-Start is indeed a significant bottleneck in practice.

IV. Stateful-BBR

In this section we first briefly revisit the design of BBRv1 and BBRv2, and then presents the design and implementation of S-BBR.

A. Recap of BBR

BBR has attracted much attentions since its introduction by Google in 2016 [5] (now known as the BBRv1). It has since

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on November 23,2021 at 06:28:57 UTC from IEEE Xplore. Restrictions apply.

TABLE I. COMPARISION OF CUBIC, BBRV1, AND BBRV2.

	Cubic	BBRv1	BBRv2
Congestion signal	Pkt losses	N/A	Pkt losses / ECN
Bandwidth probing	Cubic curve	+25% burst every 8 RTTs	inflight_probe grows exponentially per round
Inflight packets cap	None	2 BDPs	1.25 BDPs
Min RTT Probing	None	Cut packets inflight to 4 pkts	Cut packets inflight by half

been deployed in some of Google's services. With the availability of its Linux implementation, it has also been increasingly deployed by others as well.

BBRv1 differs from Cubic in that it is no longer pure credit-based, i.e., transmissions driven by ACK arrivals and CWnd availabilities. Instead, BBRv1 employs pacing throughout to maintain a transmission rate commensurate with the estimated path bandwidth. BBRv1 still maintains CWnd to control the packets inflight so as to reduce the bufferbloat issue that can occur in Cubic [25]. More importantly, unlike Cubic which cuts the CWnd by 30% upon a loss event [2], BBRv1 does not do so and rely on its bandwidth estimator to regulate its transmission rate. This is also one of the reasons why BBRv1 is more resilient to packet losses and outperforms Cubic in lossy networks.

Google released the version 2 of BBR (BBRv2 for short) in 2019 [12]. BBRv2 was designed to address some potential issues observed in the deployment of BBRv1. Specifically, BBRv1 probes for more bandwidth by raising its transmission rate momentarily by 25% every 8 RTTs. This could cause more packet losses in shallow buffer networks [12] and will become more severe when multiple BBRv1 flows compete at the same bottleneck.

BBRv2 mitigated the problem through three means: (i) it will reduce the transmission rate upon packet loss / ECN (instead of not reacting to them as in BBRv1); (ii) it probes bandwidth more slowly once the packets inflight has reached the estimated BDP; and (iii) it reduces the maximum CWnd from BBRv1's 2 BDP to 1.25 BDP.

In addition, BBRv2 also fine-tuned the probing mechanism for minimum RTT by adjusting the cut in inflight packets from 4 (as in BBRv1) to half. This allows more packets to fill the pipe to achieve better bandwidth utilization, especially in networks with varying bandwidth, i.e., mobile and wireless networks.

Table I summarizes the key differences between Cubic, BBRv1, and BBRv2. Interested readers are referred to literature [2, 5, 12] for more details.

B. Application of Stateful-TCP

The key idea behind Stateful-TCP is that most Internet applications generates many TCP flows in an application session. For example, in short-video services such as TikTok, Likee, Kuaishou, etc., which have seen explosive growth in the past few years. Users of these short-video services often watch many videos in succession. Therefore, the network condition experienced by the TCP flows will likely to be highly correlated. Stateful-TCP exploits this by using network information obtained in a previous flow to configure the initial configuration of the subsequent flow so that the latter does not need to begin its transmission rate conservatively as is currently implemented by Slow-Start.

To apply Stateful-TCP to BBR – S-BBR, we need to address three questions: (i) what network information, i.e., *stateful metrics*, to carry over and how to obtain them; (ii) how to configure the startup phase of a new flow; and (iii) how to transit from the startup phase to the normal congestion control phase. We present one design in the following as the starting point for exploration. We emphasize that it is only an initial design and there are other possible designs and many open problems remain which warrant further investigations.

On stateful metrics, there are three network parameters central to the operation of BBR (henceforth we will use BBR to include both BBRv1 and BBRv2), namely estimated path bandwidth, minimum RTT, and estimated path BDP. They are not independent so one only needs to know any two to calculate the remaining one. We chose the first two, i.e., estimated path bandwidth and minimum RTT, as the metrics in Stateful-TCP as these two are already estimated by BBR as part of its normal operation. Therefore, by piggyback on them, no additional processing overhead will be incurred.

One potential issue we found is that BBR's bandwidth estimator works by measuring the amount of data acknowledged by ACKs over around one RTT. For long flows this work well as the pipe is filled with inflight packets during the measurement interval. It may result in underestimation, however, if the flow size is smaller than the path BDP[1]. As an illustration, in a 5G network with 1 Gbps bandwidth and 20 ms round-trip propagation delay, the BDP will be 2.5 MB. Thus, if the flow size is smaller than that then the flow will complete within one RTT which could lead to underestimation of the path bandwidth.

This is undesirable as an application session could be interleaved by long (e.g., images) and short (e.g., texts) flows. Consequently, bandwidth underestimation in the short flows will degrade the performance of subsequent flows (c.f. Fig.2). Therefore, we introduce a new constraint to discard the stateful metrics if the flow size is smaller than the cached BDP. Otherwise the stateful metrics {path bandwidth, minimum RTT} will be cached in a table hashed by the peer's IP address.

In the startup phase of the new flow (i.e., after three-way handshake), S-BBR will first check if cached stateful metrics for the peer IP is available, and if so, will use it to directly configure the startup phase, bypassing Slow-Start altogether. Otherwise, S-BBR will revert back to BBR with normal Slow-Start where it begins with an initial CWnd size of 10.

[1] In BBR the bandwidth (called deliveryRate) is estimated from the amount of packets delivered since the ACKed packet was sent divided by the time elapsed since *the last ACK was received when the ACKed packet was sent*. The latter is equal to or longer than one RTT.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on November 23,2021 at 06:28:57 UTC from IEEE Xplore. Restrictions apply.

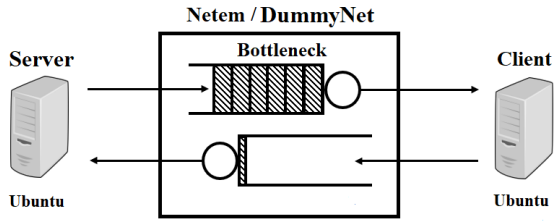


Fig. 3. The topology used in emulated mobile network experiments.

By contrast, with cached stateful metrics, S-BBR will override Slow-Start by setting the pacing rate to the cached path bandwidth and the initial CWnd to path BDP which is computed from the product of cached path bandwidth and cached minimum RTT from the previous flow. This enables S-BBR to transmit at a data rate matching the previously-estimated path bandwidth right from the beginning, thereby improving its throughput especially for short flows. Note that in S-BBR, the AWnd is suppressed unless it is zero [26]. In other words, the maximum inflight packets allowed is determined solely by CWnd (instead of min{CWnd, AWnd}). This prevents the receiver's initial AWnd, which like CWnd is typically small, from restricting the sender to send a full BDP's worth of packets to fully utilize the bandwidth available.

The startup phase completes once the first ACK is received and S-BBR will pass the control back to BBR's congestion control algorithm for the rest of the flow. Upon flow termination, the latest bandwidth and minimum RTT estimates are then stored into the stateful table entry for the peer.

C. Linux Implementation

Implementations for BBR have evolved over the years. To support our experiment platforms we implemented three S-BBR versions: (i) S-BBRv1 based on Linux kernel 5.4 – this represents the more recent Linux kernel version and is used for trace-driven emulated experiments in Section V-A; (ii) S-BBRv1 based on an earlier kernel version 4.14 – this is the kernel version required by the competitive benchmarking platform used in Section V-B; and (iii) S-BBRv2 based on the BBRv2 Alpha codebase [27] for Linux kernel version 5.4. Just like BBR, S-BBR is entirely sender-based so no modification to the receiver TCP is required. Moreover, the Stateful-TCP logics can be implemented entirely within TCP's pluggable congestion module so that no further kernel modification is needed. The source codes of the S-BBR implementations are available at github[2].

To enable the stateful startup phase, S-BBR must maintain an internal table to cache the stateful metrics for completed flows to be looked up later by subsequent flows. The congestion control module implements this by allocating a fixed-size table at module registration time for use throughout the runtime of the module. The table will be deallocated during module deregistration, e.g., switching to a different TCP congestion control module, and so cached stateful metrics are not persistent in that regard. A further optimization would be to store the table to persistent storage upon module deregistration for use the next time the module is started (e.g., after server reboot).

A key component of S-BBR is its stateful metrics table. To reduce runtime overhead, the current implementation adopts a simple hashing function to map the peer's IP address to a hash table entry. Collision detection is supported by storing the peer's IP address alongside the stateful metrics. In addition to IPv4, the S-BBR module also supports IPv6 but only the lower 64 bits of the address is used for hashing and storage to reduce memory consumption.

Altogether, each table entry consumes 16 bytes of memory which is relatively modest. Larger table is desirable as collision rate will decrease, enabling more effective activation of S-BBR. The exact table size needed is likely to dependent on many factors such as the client population's geographical as well as topological distribution, the types of applications/services provisioned at the server, and the rules adopted in the server farm's load-balancer. This is an open problem that warrants further investigation.

V. Performance Evaluation

In this section, we take a first look at the performance of S-BBR and compare it to BBR as well as other TCP designs. In particular, we focus on S-BBR's performance in mobile and wireless networks. Three experimental platforms were employed in this study. The first one employed network emulator, either netem [28] or a modified version of dummynet [3], using bandwidth trace data captured from production mobile networks to recreate their bandwidth variations in the topology depicted in Fig. 3.

The second one is a commercial competitive benchmarking platform. This platform is mainly used by content and service providers to conduct independent competitive testing to inform their selection of CDN, data center, and network providers. The main advantages of this platform is their scale (500+ clients), scope (covering 9 provinces through the top three ISPs), and its design to capture realistic performance as experienced by end-users.

The third and the most important platform is the production app-store servers of our tier-1 CDN service provider. We present and discuss their performance results in the following sections.

A. Emulated Mobile Network Experiments

In the first set of experiments, we employed bandwidth trace data captured from a production 5G network using a stationary 5G-smartphone as the receiver. The server is located in a data center. The bandwidth capture was done by flooding the 5G link with UDP datagrams sent from the server to the smartphone. The network trace is then captured using a custom software running in the smartphone which were later processed into a format for use in the network emulators.

The network emulator emulates the 5G link by recreating the bandwidth variations from the trace data. The emulated

[2] Implementations of S-BBRv1 and S-BBRv2 are available at https://github.com/mclab-cuhk/Stateful-BBR

[3] The modified dummynet is available at https://github.com/mclab-cuhk/netmap-ipfw

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on November 23,2021 at 06:28:57 UTC from IEEE Xplore. Restrictions apply.

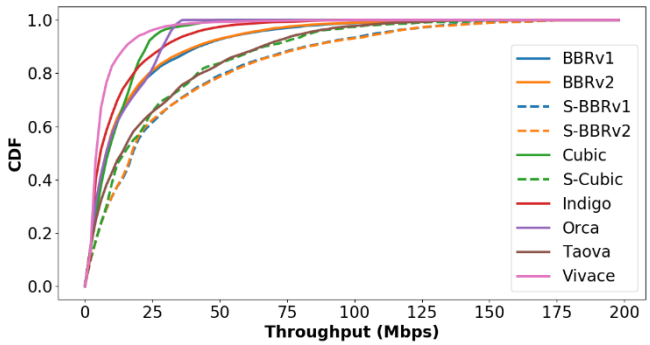


Fig. 4(a). Distribution of per-flow mean throughput in an emulated 5G network with 0.1% random packet loss.

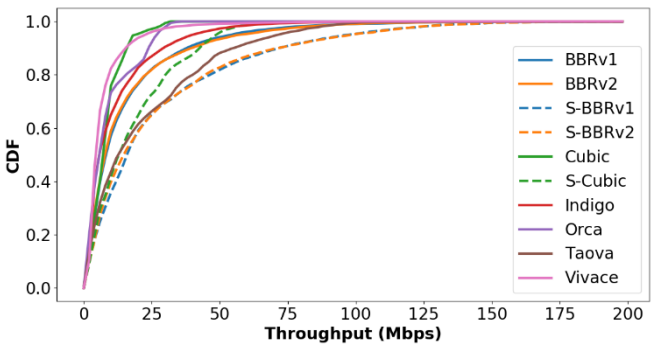


Fig. 4(b). Distribution of per-flow mean throughput in an emulated 5G network with 1% random packet loss.

TABLE II. THROUGHPUT (IN MBPS) COMPARISON IN AN EMULATED 5G NETWORK WITH 0.1% RANDOM PACKET LOSS.

	Mean Flow Size			
TCP	64 KB	128 KB	512 KB	1024 KB
BBRv1	4.5	7.5	20.5	32.8
BBRv2	4.4	7.2	19.7	31.7
S-BBRv1	9.0	15.3	39.7	59.3
S-BBRv2	8.9	15.1	39.9	59.8
Cubic	4.5	7.0	14.3	18.2
S-Cubic	7.8	13.0	32.8	48.8
Indigo	3.5	5.5	14.3	22.3
Orca	4.3	6.9	15.8	20.8
Taova	6.6	11.8	31.8	46.9
Vivace	2.9	3.9	8.3	13.3

TABLE III. THROUGHPUT (IN MBPS) COMPARISON IN AN EMULATED 5G NETWORK WITH 1% RANDOM PACKET LOSS.

	Mean Flow Size			
TCP	64 KB	128 KB	512 KB	1024 KB
BBRv1	4.5	7.2	18.9	30.1
BBRv2	4.4	7.1	19.1	30.6
S-BBRv1	8.1	13.7	35.3	52.9
S-BBRv2	7.9	13.4	34.8	52.0
Cubic	4.1	6.1	10.3	11.3
S-Cubic	5.5	9.0	22.3	30.6
Indigo	3.5	5.5	14.3	22.3
Orca	3.5	5.5	11.5	14.3
Taova	6.4	11.3	28.9	41.0
Vivace	2.8	3.8	8.2	13.0

TABLE IV. QUEUING DELAY (IN MS) COMPARISON IN AN EMULATED 5G NETWORK WITH 0.1% RANDOM PACKET LOSS.

	Mean Flow Size			
TCP	64 KB	128 KB	512 KB	1024 KB
BBRv1	1	1	6	10
BBRv2	1	2	8	13
S-BBRv1	1	1	5	10
S-BBRv2	1	2	7	13
Cubic	1	1	1	1
S-Cubic	1	1	1	1
Indigo	1	1	1	1
Orca	1	1	1	1
Taova	1	1	1	1
Vivace	1	1	1	1

TABLE V. QUEUING DELAY (IN MS) COMPARISON IN AN EMULATED 5G NETWORK WITH 1% RANDOM PACKET LOSS.

TCP	Mean Flow Size			
	64 KB	128 KB	512 KB	1024 KB
BBRv1	1	1	5	9
BBRv2	1	2	5	8
S-BBRv1	1	1	5	8
S-BBRv2	1	1	4	7
Cubic	0	0	0	0
S-Cubic	1	1	0	0
Indigo	1	1	1	1
Orca	1	1	1	1
Taova	1	1	1	1
Vivace	0	0	1	1

link's propagation delay is set to 20 ms with a link buffer size of 10 MB in accordance with our measurement of the actual 5G network. Two random packet loss settings, 0.1% and 1% corresponding to low and high loss scenarios, were tested. To offer a broader perspective on the performance of S-BBR, we included six other recent TCP designs in the experiment. This includes Indigo [15], Orca [17], Taova [16], Vivace [9], Cubic [2], as well as S-Cubic [11] – the stateful version of Cubic.

In the experiments, TCP flows were generated with flow size drawn from the Pareto distribution (with $\alpha = 2.5$) which exhibits long-tail characteristics resembling Internet flows-size distribution [29-30]. Four distributions with mean flow sizes of 64 KB, 128 KB, 512 KB, and 1024 KB, were tested, each comprising 1,000 flows.

We first evaluate the TCPs' overall performance in Fig. 4 which plots the CDF of the average throughput achieved by individual flows from all four flow-size distributions (i.e., 4,000 flows per TCP). Throughput is calculated from the amount of data transferred divided by the duration of the flow, including connection-setup time.

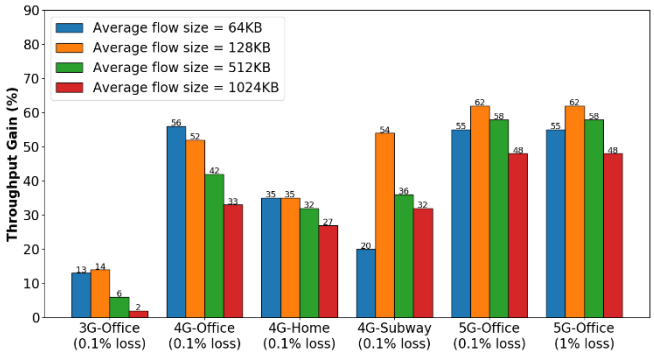
We note that the curves for BBRv1 and BBRv2 almost overlap completely, suggesting that both achieved similar performance in this experiment. Their stateful counterparts, namely S-BBRv1 and S-BBRv2, achieved noticeably higher throughput in both random loss settings. Compared to S-Cubic which also adopts Stateful-TCP, S-BBR generally achieved higher throughput performance, more so in the 1% loss setting. This is due to Cubic's congestion control algorithm's sensitivity to random packet losses [5]. The other TCP designs are generally less sensitive to random packet losses as many of them have taken random packet loss into consideration in their design. Overall, S-BBR achieved the best throughput performance in this experiment.

We further analyze the TCPs' throughput performance in Table II and III by separating the results according to the four flow size distributions. Here, we can clearly see the impact of flow size on the achievable throughput where most TCPs

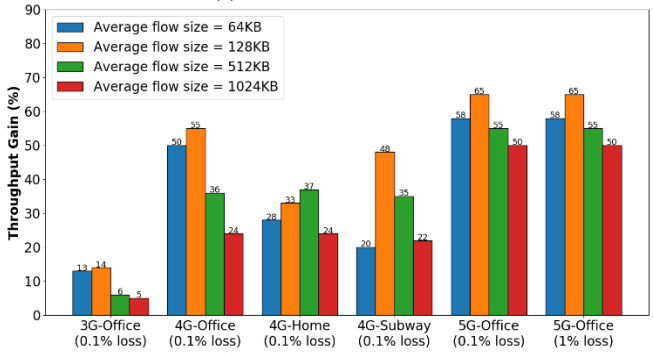
Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on November 23,2021 at 06:28:57 UTC from IEEE Xplore. Restrictions apply.

TABLE VI. MOBILE TRACE DATASET CHARACTERISTICS.

Mobile Networks	Mean Bandwidth	RTT	Mean BDP	Link Buffer Size	Data Source
3G (office)	~5 Mbps	50 ms	~21 MSS's	1280 KB	Own
4G (office)	~20 Mbps	50 ms	~84 MSS's	5 MB	Own
4G (home)	~16 Mbps	50 ms	~67 MSS's	5 MB	[17]
4G (subway)	~11 Mbps	50 ms	~46 MSS's	5 MB	[17]
5G (office)	~180 Mbps	20 ms	~300 MSS's	10 MB	Own



(a) S-BBRv1 over BBRv1



(b) S-BBRv2 over BBRv2

Fig. 5. Throughput gains of S-BBRv1 over BBRv1 and S-BBRv2 over BBRv2 in six mobile networks.

needed flow size of at least 512 KB to reach mean throughputs over 10 Mbps. Note that the average bandwidth of the 5G trace data used is ~180Mbps so the gap is substantial. Part of the reason is due to TCP's connection setup time which is unavoidable and becomes more significant at smaller flow sizes. The rest of the gap is due to TCP's ramp-up time as discussed earlier. Among the non-stateful TCP designs, Taova achieved considerably higher throughput at the two lower flow size distributions. This is because Taova does not adopt TCP Slow-Start and hence its initial transmission rate is not constrained by it.

Overall, S-BBR achieved substantially higher mean throughput than the other TCPs, especially at the smaller flow sizes. For example, at the mean flow size of 128 KB, S-BBRv2 achieved 110% higher throughput than BBRv2 in the 0.1% loss setting. Even at 1 MB mean flow size, S-BBRv2 still achieved 89% higher throughput than BBRv2, and 23% higher than the second best TCP, i.e., S-Cubic. The performance gap widens further in the 1% loss setting as S-Cubic's performance is degraded by its sensitivity to random packet losses. In the 1 MB mean flow size case, S-BBRv2 achieved 70% higher throughput than BBRv2 and 27% higher throughput than the second best TCP, i.e., Taova.

Next, we investigate their delay performance. We recorded the packet queueing delay at the network emulator and summarized the mean packet queueing delay in Table IV and V for the two loss settings. We observe that BBR and S-BBR exhibited longer queueing delay than other TCPs, especially for the longer flow sizes. This is a result of their higher throughput and their bandwidth probing mechanisms. Remarkably, compared to BBR, S-BBR's higher throughput did not incur higher queueing delay. This is because the stateful startup phase allows a substantial portion of the flow to be transmitted at a rate close to the path bandwidth, thereby reducing exposure to subsequent periodic bandwidth probing which could cause packet queueing.

The results thus far focused on 5G network. In the second set of experiments, we expanded the network emulation to cover 4G and 3G networks. In addition to trace data collected by the authors, additional dataset from Orca [17] were included as it contains trace data collected from home and subways, two scenarios that were not covered in our trace data. Table VI summarizes properties of the five trace datasets used in the experiments.

In this second set of experiments, we focus on the performance gains of S-BBR compared to BBR over different types of mobile networks ranging from 3G to 5G. To ease comparison, we employ the metric *throughput gain* – defined as the throughput gained by S-BBR compared to BBR under the same network condition, in the following comparisons.

Fig. 5(a) and 5(b) plot the throughput gained by S-BBRv1 and S-BBRv2, respectively, over the five mobile network traces with two random packet loss settings. Compared to the 5G network case, their throughput gains are lower in 4G networks as their mean BDP sizes are far smaller than 5G (c.f. Table VI). Comparing the three 4G network traces, we can observe that the gains are largely in line with the trace's mean BDP size although the correlation is weakened in the 1% loss setting. Finally, the gains under the 3G trace is lowest as one would expect from its much smaller BDP. This result strongly suggests that the performance gains are likely to increase in future networks where bandwidth is only going to go up.

B. Independent Benchmarking

In this section, we report experimental results obtained from Bonree [31], a benchmarking platform specializes in competitive performance benchmarking of networks and services. The benchmarking setup comprises over 500 client hosts distributed across nine provinces in China. Unlike the previous experiments, these clients are connected via either wired or Wi-Fi networks, thereby offering an additional perspective to evaluate S-BBR's performance.

One limitation is that BBRv2 is not part of the Linux kernel supported by the platform and it does not allow kernel recompilation (which BBRv2 requires) due to security restriction so the benchmarking experiment is limited to BBRv1 only. The experiment lasted for 24 hours, with BBRv1 and S-BBRv1 tested in a round-robin manner to download a file of 1 MB size. In total around 2,000 downloads were

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on November 23,2021 at 06:28:57 UTC from IEEE Xplore. Restrictions apply.

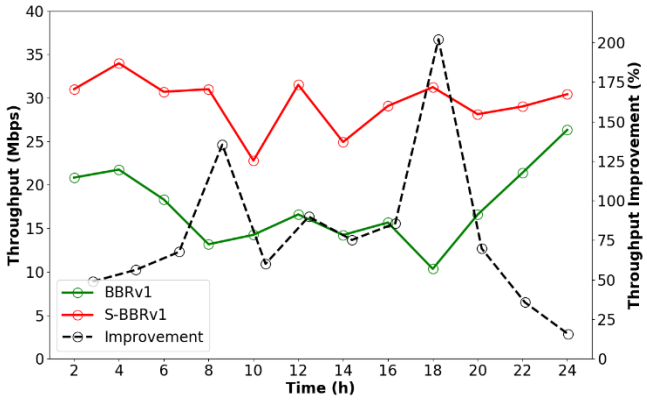


Fig. 6. Comparison of average throughput for BBRv1 and S-BBRv1 in a benchmarking platform. Each data point is an average of 2-hour samples.

TABLE VII. PERFORMANCE COMPARISON IN CDN PRODUCTION SERVERS.

A/B Comparison	TCP Variant	Mean Throughput	Success Rate	Low-speed Ratio
#1	BBRv1	6.0 Mbps	98.9%	18.7%
	S-BBRv1	7.6 Mbps	99.1%	12.1%
#2	Cubic	6.2 Mbps	98.4%	14.2%
	S-BBRv1	8.4 Mbps	98.4%	9.8%
#3	S-Cubic	7.1 Mbps	98.8%	8.8%
	S-BBRv1	8.7 Mbps	98.7%	9.3%

completed for each TCP, with a success rate of 99.9% in both cases.

Fig. 6 compares the 2-hr mean throughput achieved by BBRv1 and S-BBRv1. As expected, S-BBRv1 achieved higher throughput in all 12 timeslots, with an overall throughput gain of 69%. Among the ~2,000 downloads, S-BBRv1 has more high throughput (≥100 Mbps) cases than BBR (4.4% vs. 2.5%) as well as fewer low throughput (≤10 Mbps) cases (5.3% vs. 16.3%). Compared to the mobile network experiments in the previous section, the throughput gains here are even higher even though the throughputs are lower than the 5G network trace. It is not apparent what the reason is as the competitive benchmarking platform is not open to users so the exact network conditions are not known. Nonetheless, the results show that S-BBR's performance gains do carry over to wired and Wi-Fi networks

C. *Production App-Store Servers*

In this section, we report experimental results obtained from production app-store servers of a tier-1 CDN service provider. As this is a production Internet service it is not possible to conduct controlled experiment. Instead, we selected two production servers with the same hardware and OS kernel version, located inside the same data center to minimize differences due to server hardware/OS and server-side network connectivity.

In our experiment, one of the two servers ran S-BBR while the other server ran the comparing algorithms, i.e., BBRv1, Cubic and S-Cubic. We were not able to test S-BBRv2/BBRv2 because the production servers' Linux kernel does not yet support BBRv2. Nevertheless, based on the results from the previous sections, we expect their performances to be similar. In each A/B comparison, we started both servers at the same time and ran each comparison for two days, recording the throughput of all mobile app downloads. Each experiment generated around one million download records.

We summarize in Table VII three of the most critical performance metrics in the CDN industry, namely mean throughput, low-speed ratio (<1 Mbps), and success rate. In the first comparison, the overall mean throughput of S-BBRv1 and BBRv1 were 7.6 Mbps and 6.0 Mbps respectively, representing a 27% gain in throughput by S-BBRv1. This is a significant performance gain as it reduces the app download time, thereby improving the user experience. In addition to overall mean throughput, Stateful-TCP could also reduce the likelihood of poor connections, S-BBRv1 cuts down the low-speed ratio by 6.6%, compared to BBRv1. While in the second comparison, S-BBRv1's mean throughput gain of 35% over Cubic is even higher. We conjecture that this is due to Cubic's more sensitive performance in lossy networks (e.g., 4G and Wi-Fi).

Next, we compare S-BBRv1 against the Stateful-TCP version of Cubic, i.e., S-Cubic proposed by Guo and Lee [11]. At 8.7 Mbps versus S-Cubic's 7.1 Mbps overall mean throughput, S-BBRv1 achieved 23% higher throughput performance than S-Cubic, therefore offering an attractive option for improving today's production Internet services.

Last but not least, we also want to verify if Stateful-TCP exhibits any compatibility issue with mobile client's TCP stack in a production Internet service. As summarized in Table VII, all three TCP designs share a similar download success rate of ~99%, demonstrating that S-BBRv1 has the same compatibility performance as the unmodified TCP in the Linux kernel.

VI. SUMMARY AND FUTURE WORK

Through experiments in both emulated and production environments, this work demonstrated the feasibility and performance gains achievable by applying Stateful-TCP to BBR. This is an important milestone as the ultimate test of any new TCP design is in actual deployment, and this work is the first step in this direction. Clearly, it is still merely one data point and much work remains to be done to fully explore and exploit the potentials of Stateful-TCP, even beyond its application to Cubic and BBR. Therefore, we are releasing the full source codes of both S-BBRv1 and S-BBRv2 to facilitate and encourage the research community as well as the industry to experiment with them and possibly to deploy them in other production Internet services to further investigate their performance and to uncover any limitations that will lead to even better designs.

ACKNOWLEDGMENT

The authors wish to thank the anonymous reviewers for their comments and suggestions in improving this work, and Tencent for their generous support of this work by authorizing their production servers for the experiments, and providing access to the Bonree benchmarking platform. This work was partially funded by CUHK Direct Grant for Research Project No. 4055156.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on November 23,2021 at 06:28:57 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] A. Gupta and R. Jha, "A Survey of 5G Network: Architecture and Emerging Technologies," *IEEE Access*, vol. 3, pp. 1206–1232, 2015.

[2] I. Rhee and L. Xu, "Cubic: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS Operating System Review*, New York, USA, Jul.2008, pp. 64-74.

[3] C. Casetti, M. Gerla, S. Mascolo, M.Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," *Proc. ACM Mobicom.* Rome, Italy, July 2001.

[4] C. Fu and S. Liew, "TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks," *IEEE J. Sel. Areas Commun*, vol. 21(2), Feb. 2003, pp. 216–228.

[5] N. Cardwell, Y. Cheng, C. S. Gunn, V. Jacobson, and S. Yeganeh, "BBR: Congestion-Based Congestion Control," *Proc. ACM Queue*, Sep. 2016.

[6] L. Xu, K. Harfoush and I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-distance Networks," *Proc. IEEE INFOCOM*, vol.4, Hong Kong, 2004, pp.2514-2524.

[7] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," *Proc. SIGCOMM*, New Delhi, India, Aug 2010, pp.63-74.

[8] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks," *Proc. NSDI*, Lombard, IL, Apr. 2013, pp 459–471.

[9] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "Vivace: Online-learning Congestion Control," *Proc NSDI*. Washington, USA, Apr 2018.

[10] Venkat Arun and Hari Balakrishnan, "Copa: Practical Delay-Based Congestion Control for the Internet," *Proc. NSDI*. Renton, WA, 2018.

[11] L.F. Guo and J.Y.B. Lee, "Stateful-TCP - A New Approach to Accelerate TCP Slow-Start, " to appear in *IEEE Access.*

[12] Neal Cardwell, Yuchung Cheng, "BBR Congestion Control Work at Google - IETF 101 update," *IETF*, Tech. Rep., 2019.

[13] Soheil Abbasloo, Y. Xu, and H. Jonathan Chao, "C2TCP: A Flexible Cellular TCP to Meet Stringent Delay Requirements," *IEEE J. Sel. Areas Commun*, vol. 37(4):918–932, 2019.

[14] Shinik Park, Jinsung Lee, Junseon Kim, Jihoon Lee, Sangtae Ha, and Kyunghan Lee, "ExLL: An Extremely Low-latency Congestion Control for Mobile Cellular Networks," *Proceedings of ACM CoNEXT*, 2018.

[15] F. Y. Yan, J. Ma, G. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein, "Pantheon: the Training Ground for Internet Congestion Control Research," *Proc NSDI*. Boston, USA, Jul 2018.

[16] K. Winstein and H. Balakrishnan, "TCP Ex Machina: Computer-generated Congestion Control," *Proc. SIGCOMM*, HongKong, Aug 2013.

[17] S. Abbasloo, C.Y. Yen, H.J. Chao, "Classic Meets Modern: A Pragmatic Learning-based Congestion Control for the Internet," *Proc. SIGCOMM*, Aug, 2020.

[18] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP – Design Principles and Standards," *Proc. MMSys*, NY, USA, 2011.

[19] A. Aloman, A. I. Ispas, P. Ciotirnae, R. Sanchez-Iborra and M. D. Cano, "Performance Evaluation of Video Streaming Using MPEG DASH, RTSP, and RTMP in Mobile Networks," *Proc IFIP (WMNC)*, Munich, 2015, pp. 144-151.

[20] I. Sandvine, "Global Internet Phenomena Report. 2016," 2015.

[21] S. Hauger, M. Scharf, J. K¨ogel, and C. Suriyajan, "Quick-Start and XCP on A Network Processor: Implementation Issues and Performance Evaluation," *Proc. IEEE HPSR*, Shanghai, China, May 2008, pp 703-714.

[22] D. Liu, M. Allman, S. Jiny, and L. Wang, "Congestion Control without A Startup Phase," *Proc. Int. Workshop PFLDnet*, Feb. 2007, pp. 61–66.

[23] Qingxi Li, Mo Dong, and Brighten Godfrey, "Halfback: Running Short Flows Quickly and Safely," *Proc. ACM CoNEXT*, Germany. Dec 2015.

[24] X. Nie, Y. Zhao, G. Chen, K. Sui, Y. Chen, D. Pei, M. Zhang, and J. Zhang, "TCP Wise: One Initial Congestion Window is not Enough," *Proc. IPCCC*, San Diego, USA, Dec 2017, pp. 1–8.

[25] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *Proc. ACM Queue*. 2011

[26] K. Liu and J.Y.B. Lee, "Mobile Accelerator: A New Approach to Improve TCP Performance in Mobile Data Networks," *Proc. IWCMC*, Istanbul, Turkey, Jul. 2011, pp. 2174 - 2180.

[27] BBRv2 source code, [online] https://github.com/google/bbr/blob/v2alpha/net/ipv4/tcp_bbr2.c

[28] Netem, [Online] https://github.com/akamai/cell-emulation-util

[29] K. Avrachenkov, "Differentiation between Short and Long TCP Flows: Predictability of the Response Time," *Proc. INFOCOM*, 2004.

[30] A. B. Downey, "Lognormal and Pareto Distributions in the Internet," *Computer Communications.* vol. 28(7), 2005, pp. 790–801

[31] Bonree. Accessed: Sep. 1, 2020. [Online]. Available: https://www.bonree. Com

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on November 23,2021 at 06:28:57 UTC from IEEE Xplore. Restrictions apply.