BBRx: Extending BBR for Customized TCP Performance

Jae Won Chung*, Feng Li†, Beomjun Kim†

*Viasat, 300 Nickerson Rd, Marlborough, MA, 01752
{jaewon.chung}@viasat.com
†Verizon Labs, 60 Sylvan Rd, Waltham, MA, 02145
{feng.li, beomjum.kim}@verizon.com

***Abstract*—The most dominant Internet Protocol, TCP, has its performance tied to the attributes of the network it deployed on; in the past decades many congestion detection and estimation techniques have been proposed to improve TCP performance over various network links. These techniques have depended on packet loss or delay sensing to autonomously balance between fairness and peak performance. More recent congestion detection/estimation approaches incorporate online and/or offline learning of the congestion states to achieve their end goals. This paper introduces a practical way to apply online-learning on TCP congestion control domain without tightly coupling them together to minimize the risk of deploying pre-tuned learning algorithm in the fast path. BBR was presented in 2016 to avoid congestion by sending TCP segments at the estimated bottleneck bandwidth rather than reacting to packet drops. We extended BBR, referred to as BBRx, to react to the path RTT as well as the bottleneck bandwidth to help finding an optimal throughput-delay operation point in wireless environment. In order to ensure BBRx is continuously operating in an optimal point in ever changing network conditions, TCP flow statistics are monitored in real-time through Netlink socket by a user-space application which tunes BBRx parameters with online learning algorithms. Preliminary results show that BBRx effectively manages the bottleneck queue yielding a low packet loss rate on links with shallow buffers. Also, the auto-tuned BBRx finds an optimal throughput-delay operation point in LTE environment to maximize per-flow throughput within a bounded loss rate and delay.**

I. Introduction

Advent of high-bandwidth high-delay networks with multi 10s of Gbps link capacity imposed a new performance requirement to TCP that is difficult to satisfy with Additive Increase Multiplicative Decrease (AIMD) [9] or CUBIC [8] congestion avoidance. Due to the slow transmission rate ramp up following a multiplicative decrease, a small number of TCP flows cannot effectively consume the available link bandwidth. This link utilization inefficiency is even subtle for high speed mobile networks since significant amount of packet losses are caused by bit errors in wireless transmissions. To overcome this performance barrier, research community seeks to find new congestion control approaches different from AIMD/CUBIC.

In one stream, efforts are made to enhance the entire end-to-end congestion control system including network switches as well as TCP sender/receiver. DCTCP [1] optimizes the multiplicative decrease behavior of AIMD/CUBIC congestion avoidance mechanism with the help of AQM and ECN enabled network switches. As requiring both the TCP sender and receiver modifications in addition to AQM, DCTCP is suggested for intra-datacenter communications. ABC [7] proposes a new congestion control framework where bottleneck switches govern the behavior of the end-system. In order to suppress the queue length below a target, an ABC switch uses a Proportional Integral (PI) controller to estimate the target aggregated data rate, compute and notify the fair share to each ABC sender using an ECN extension method. ABC requires modification to network switch AQM as well as the TCP sender, and may be better suited for a new datacenter or a new mobile network bounded by L4 Performance Enhancement Proxies (PEPs).

Sender-side congestion control approaches to improve the TCP performance on high-bandwidth high-delay networks without a support from the network branches into two different directions. The first is to apply machine learning (ML) techniques. PCC [6] applies gradient ascendant ML technic on flow utility computed with estimated goodput and RTT to find optimal transmission rates for the connection. Approaches to use deep learning approaches to determine the transmission rate dynamics at the TCP sender include Remy [10]. In general, deep learning approaches are computationally expensive since continuous training is required to adapt to changing network conditions. Also, debugging neural networks is not a trivial process. Practical adoption of deep learning on network congestion control may take a little more time and effort.

Sender-side congestion control approaches to use different network performance attributes than packet drop events to control the transmission rate includes BBR [3]. BBR sender estimates the bottleneck bandwidth and uses it as the steady state transmission rate while periodically probe for extra bandwidth to ensure fairness among the contending flows. While BBR congestion control has been presented as an improvement, with the promise of higher throughput and lower delay as compared to other TCP congestion control algorithms, BBR can cause a large packet drops for a bottleneck with shallow buffers [2], [6], [11].

This paper analyzes BBR control deficiency and provides a solution to mitigate the packet drop issue. We extended BBR, referred to as BBRx hereafter, to use PI control logic such that TCP senders can respond to RTT dynamics as

well as the bottleneck bandwidth to suppress the bottleneck queuing delays. Under mobile wireless environments where only a small number of flows from a device contend for the bandwidth allocated to the device at an eNodeB, BBRx can effectively keep the queuing delay around an optimal throughput-delay operation point.

BBRx offers an array of control parameter sets for the senders to choose from based on the encountered network conditions. Based on the estimated bottleneck bandwidth and minimum RTT ranges, each BBRx sender chooses a control parameter set suited for the environment. In order to ensure that each control parameter set provides an optimal TCP performance, TCP flow statistics are continuously monitored through Netlink socket by Learning Agent. The agent applies gradient ascendant algorithm on the average utility of the monitored flows to calibrate the parameter set for flows in similar network conditions. The loosely coupled TCP tuning feedback control loop provides a novel way to monitor and adjust TCP parameters per the performance goal in real time while minimizing the risk of deploying pre-tuned learning algorithm in the fast path.

Performance of auto-tuned BBRx is evaluated in LTE mobile environment as well as in wired environments compared with BBR. The preliminary results show the following: BBRx effectively manages the bottleneck queue yielding a low packet loss rate especially on links with shallow buffers; the auto-tuned BBRx finds an optimal throughput-delay operation point in LTE environment to maximize per-flow throughput within a bounded loss rate and delay.

II. BBRx Design

This section shows the design of BBR extension (BBRx) to use Proportional Integral (PI) control logic. Algorithm 1 shows an implementation of Rate-Based PI Controller for AQM used by network nodes to control TCP senders transmission rate by computing the congestion notification (ECN marking) probability [4], [5].

Algorithm 1 Rate-Based PI Controller for AQM

Every δ seconds (epoch):
  $p \leftarrow p + \alpha(\delta\gamma C - b) - \beta(q - q0)$;
  $b \leftarrow 0$;
Every packet arrival:
  $b \leftarrow b + sizeof(packet)$;
  notify(packet, p);
Variables:
  p: congestion notification probability
  q: queue length in bytes
  b: total bytes received this epoch
Parameters:
  C: link capacity (bytes per second)
  γ: target link utilization ($0 < \gamma \leq 1$)
  $q0$: target queue length in bytes
  δ: measurement interval
  α: virtual queue control constant
  β: queue control constant

When this PI control logic (shown in 1) is adopted by TCP senders that can estimate C and q of the bottleneck link, it is not necessary to calculate the congestion notification probability (p), since the senders can control the number of bytes (b) to transmit during each epoch (δ). Replacing p with b of the PI control logic gives:

$$b \leftarrow b + \alpha(\delta\gamma C - b) - \beta(q - q0) \tag{1}$$

Lets divide both end of the equation by epoch (δ) to convert bytes to transmission rate (r). Then, we have:

$$r \leftarrow \{(1-\alpha)r + \alpha\gamma C\} - \frac{\beta}{\delta}(q - q0) \tag{2}$$

Let $0 < \alpha < 1$, then α becomes a weight factor to estimate/average the target link capacity at the TCP sender, e.g., $(1-\alpha)r + \alpha\gamma C = \gamma C_{avg} \approx \gamma C$. Thus, with the queue length in bytes (q) and the target queue length in bytes ($q0$) converted to queuing delay (d) and the target queuing delay ($d0$), Equation 2 is simplified to:

$$r \leftarrow \gamma C - \frac{\beta}{\delta}C(d - d0) \tag{3}$$

Equation 3 is the PI control logic used also in [7] for ABC switches to estimate optimal transmission rates for TCP senders. Here, the queuing delay offset from the target ($d - d0$) can be expressed in terms of RTT (R) estimated at the TCP sender and the target RTT ($R0$), assuming a single point of bottleneck in the network path. Thus, Equation 3 becomes:

$$r \leftarrow \gamma C - \frac{\beta}{\delta}C(R - R0) \tag{4}$$

BBRs steady state transmission rate set to the estimated bottleneck capacity (C) can be seen as a special case of the PI control with $\gamma = 1$ and $\beta = 0$. In other words, BBR ignores the path RTT dynamics and does not even attempt to manage bottleneck link buffer length. Therefore, BBR can cause buffer overflows especially when flows are contending for a bottleneck with shallow buffer. By extending BBR to adapt the full PI control logic in Equation 4, BBRx can alleviate this issue.

Due to control noises introduced during queuing delay offset estimation at the TCP sender, it may be difficult for BBRx senders to accurately control the bottleneck queuing delay. In its simplest form, BBRx may use minimum RTT (R_{min}) as the target RTT ($R0$). When a number of BBRx flows are contending for the bottleneck bandwidth, the senders may have R_{min} measurement errors. The sum of deltas virtually increases the target queuing delay over $d0$ at the bottleneck link, and causes aggregated traffic load at the bottleneck to converge to a point greater than one required to keep the queuing delay at the original target ($d0$).

The control noise can also be amplified when two or more nodes in the network path experience congestion, because queuing delay variations added by the congested non-bottleneck nodes increase jitter in the R_{min} measurement at the sender. Nevertheless, BBRx does improve bottleneck buffer overflow in all network scenarios, and sustain low the bottleneck queuing delays in such environments as wireless

mobile networks, where only a small number of flows contend for bottlenecks.

III. Configuration

In order to maximize the performance of BBRx, the reduced PI control parameter (β) in Equation 4 requires a tuning. Given the upper bound of the system RTT and the bottleneck capacity (C), the stable operating range of β/δ can be found using Bode frequency response analysis [4]. Once the stable range of β/δ is found, β can be determined by choosing a δ (epoch) sufficiently small but greater than or equal to R_{min}.

A. Empirical Auto-Tuning Method

Although a stable range of β can be found via a frequency response analysis, it is difficult to predict which β value yields an optimal TCP performance for a given network condition. This section introduces an empirical method to tune the control parameters. To simplify tuning process, the target RTT (R0) in Equation 4 is expressed in terms of minimum RTT (R_{min}) and an integer ($k \geq 4$) as follow.

$$r \leftarrow \gamma C - \frac{\beta}{\delta} C(R - \frac{k}{4} R_{min}) \tag{5}$$

As fixing $\gamma = 1$ (or 0.98) and $\delta = R_{min}$ respectively, BBRx tuning process is an optimization problem of finding a minimum k and the corresponding β that gives the best utility (U). When the utility is a convex function of β as shown in Figure 1, gradient descendant/ascendant method can be used to find an optimal β for each k starting from $k = 4$. When the optimal utility stops growing for a k increment, the previous k and the corresponding β is elected as the optimal control parameter set for the environment.

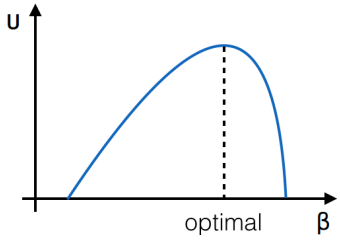


Fig. 1: Convex Utility Function

The parameter tuning process starts by initializing $k = 4$ and $\beta = 0.45$, and monitoring the TCP perforamnce. We developed a simple application that registers for TCP flow termination events to obtain per-flow TCP stats from the kernel via Netlink socket, [1] and relays the stats to Learning Agent daemon. The Learning Agent uses utility function from [6] to compute U_i for each flow, and update the average utility (U) as shown below for the current β and k set.

$$U_i(\beta_{|k}) = \mathrm{T}_i Sigmoid(\mathrm{p}_i - 0.05) - \mathrm{T}_i \frac{\mathrm{p}_i}{1 - \mathrm{p}_i} \tag{6}$$

[1]This tool also supports polling mode to periodically poll per-flow TCP BBR statistics from the kernel.

$$U(\beta_{|k}) = \frac{1}{N} \sum_{i=1}^{N} U_i(\beta_{|k}) \tag{7}$$

When a sufficiently large number of flow samples (N) are collected, $U(\beta_{|k})$ is given to gradient ascendant algorithm to find the optimal β value for the current k. Note that throughput (T_i) and packet loss rate (p_i) in Equation 6 are function of β given k respectively. It is difficult to mathematically prove the convex nature of the utility function due to the lack of $\mathrm{T}_i(\beta_{|k})$ and $\mathrm{p}_i(\beta_{|k})$ models. Yet, empirical measurements with different β in Figure 2 and Figure 3 clearly demonstrate that there is only one global optimal utility within the β range.

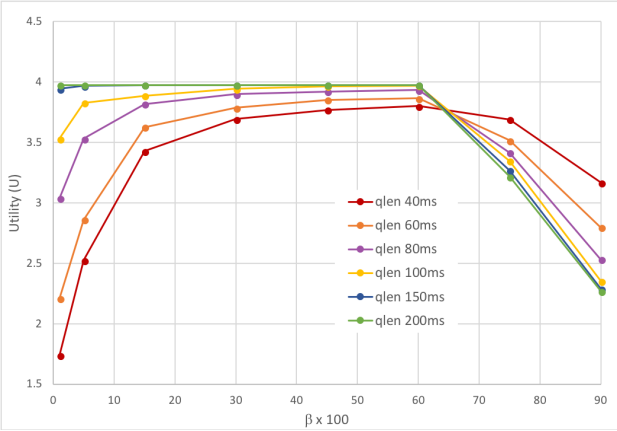
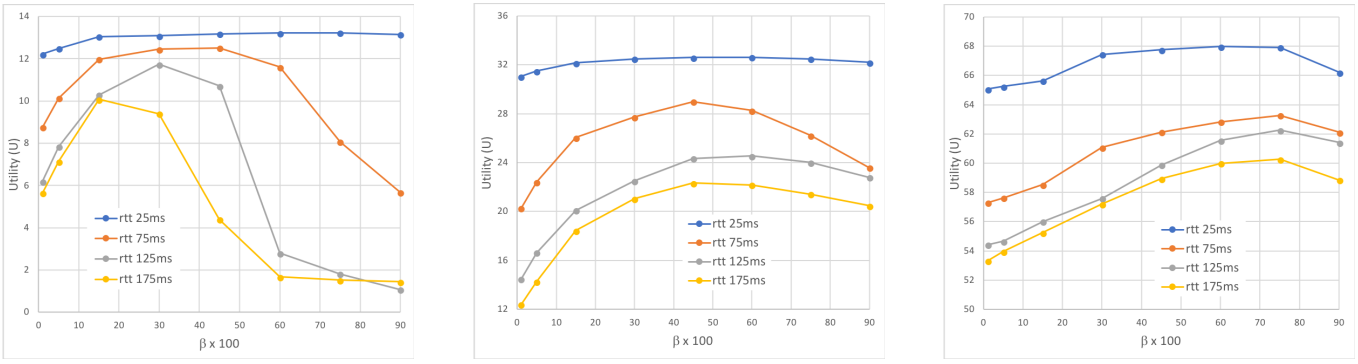


Fig. 2: Bottleneck Queue Length vs Optimal β Range

Figure 2 shows the average utility as function of β given $k = 6$ from our emulation dumbbell network setup with a single BBRx flow. The bottleneck queue length varies from 40ms to 200ms while fixing the bottleneck bandwidth (C) and the round-trip link delay (R_{min}) to 5Mbps and 25ms respectively. First, it shows that BBRx becomes BBR for small $\beta = 1$, and yields a lower flow utility due to overflows as the bottleneck buffer size is reduced. Second, a larger β beyond the optimal value ($\beta = 0.6$) decreases the utility due to control instability, i.e., results in a larger magnitude sinusoidal pattern of high queuing delay followed by link under-utilization. Third, the system has a wide range of stable β [0.2, 0.6] providing a large margin of configuration freedom. In the next section, we show that the degree of stable β margin reduces as the system RTT increases, and suggest to use different control parameter sets based on network conditions.

B. Multi-Range Configuration

Figure 3 plots the average utility as function of β given $k = 6$ from our emulation dumbbell network setup with 6 concurrent BBRx flows. We vary the bottleneck bandwidth (C) and the round-trip link delay (R_{min}) to understand the impact of C and R_{min} on the stable β margin. Figures show a trend that the stable margins are wide for systems with a small RTT (25ms) and move right to [0.5, 0.7] range as C increases. In addition, Figure 3(a) shows that the stable β margins reduce significantly and shifts left to [0.2, 0.4] range as R_{min} grows over 100ms for a small bottleneck capacity ($C = 15Mbps$).



(a) BBW (C) = 15Mbps (b) BBW (C) = 35Mbps (c) BBW (C) = 75Mbps

Fig. 3: Bottleneck Bandwidth vs minimum RTT vs Optimal β Range ($k = 6$, $flows = 6$)

		C (Mbps)			
		[0,3)	[3,10)	[10,1k)	[1k,∞)
R_{min} (ms)	[0,50)	$\beta = 0.75$ $k = 4$	$\beta = 0.75$ $k = 4$	$\beta = 0.75$ $k = 4$	$\beta = 0.75$ $k = 4$
	[50,100)	$\beta = 0.45$ $k = 4$	$\beta = 0.45$ $k = 4$	$\beta = 0.75$ $k = 4$	$\beta = 0.75$ $k = 4$
	[100,∞)	$\beta = 0.25$ $k = 4$	$\beta = 0.45$ $k = 4$	$\beta = 0.75$ $k = 4$	$\beta = 0.75$ $k = 4$

TABLE I: BBRx Configuration Table (recommended)

The fact that there is no single optimal range to cover all networking conditions but the stable margins are wide makes network condition-based configuration approach sensible. BBRx module uses multiple configuration bins, where C and R_{min} range determines a bin for β and k values. Table I shows the recommended default for each bin based on empirical results (except the last column). BBRx senders refers to this kernel module table when they enter STARTUP or PROBE_BW state to continue using optimal control parameters for the connection duration.

The BBRx module table bins are updated separately by Learning Agent when it has a new β and/or k update for the bin. This forms a loosely coupled TCP tuning feedback control loop that provides a novel way to monitor and adjust TCP parameters per the performance goal expressed by the utility function of choice in real time while minimizing the risk of deploying pre-tuned learning algorithm in the fast path.

IV. Preliminary Evaluation

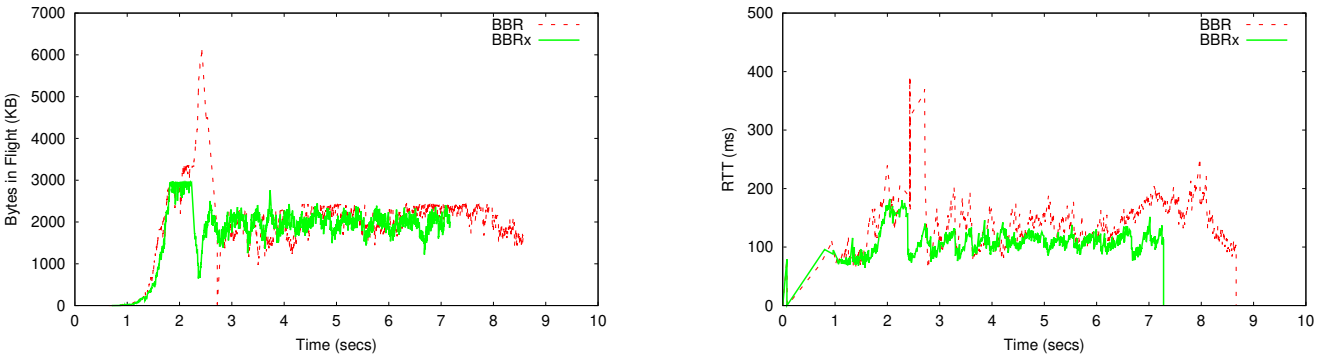
We perform preliminary measurements on a US tier-1 wireless carrier's production network in June 2018. Before starting, we setup two separate ESXi virtual machines – one for BBR, the other for BBRx – each VM is pinned with two Intel Xeon E5-2660 v3 2.60GHz CPUs, and allocated with 32GB memory. Both servers are hosts on the same ESXi 6.5 server running with a HP 460c Gen8 blade which connects to the Internet through a HPE 6120XG 10 Gbps switch. Because both of servers are dedicated to performance measurement studies and only visible for User Equipment (UE) from a small IP pool, we can assume that there is no any performance difference between a virtual host and a bare-metal HP 460c Gen8 host.

The both servers are configured with the same parameters, except for the Linux kernel version and default congestion control algorithms. BBRx extends `struct inet_connection_sock` to hold more variables. Therefore, we build BBRx with customized 4.15 kernel, while BBR is the off-shelf version packaged within 4.15 kernel. Except for the different size of struct inet_connection_sock, the BBRx kernel uses the same parameters as the default 4.15 kernel. Based on recommendations by Cardwell et al. [3], we enable fair queuing and pacing using Linux Traffic Control (`tc`) utilities on both servers. To reduce the number of random variables, we choose $\beta = 0.5$ and $k = 6$ for BBRx during this preliminary study. Both servers run Apache 2.4.7 with PHP 7. A custom PHP script dynamically generates 100 MB files with random content (to avoid any possible caching along the data path) for the smart phone to download. `Tcpdump` captures packet traces, setup to record 200 bytes per packet to provide complete TCP headers. Tests show the PHP script and `tcpdump` have less than 1% CPU load on each server.

The client smart phone is a Samsung Galaxy S7 with 4GB RAM and a 64-bit Qualcomm Snapdragon 820 Quad-Core CPU, running Android 6.0.1 Marshmallow. To avoid any performance degradation, we enable its performance mode and continually fully charge it with a power brick. The phone measures baseline round-trip times via ICMP, and throughput via HTTP download. Because the cellular network provides LTE-Advance with Carrier Aggregation (CA), we expect to observe more than 80 Mbps throughput in *good* RF conditions (e.g. `SINR` $\geq$ `25dB`), with a single TCP connection.

For illustration, Figure 4 compares a single trial of BBRx and BBR over time. Both trials ran as "stationary test" at a fix location where the SINR is greater than 25dB. Thus, neither flow experienced handover nor a large wireless drops. In Figure 4, the left figure (Fig 4) compares the "bytes in flight" (the as-yet unacknowledged transmitted bytes), while the right figure (Fig 4(b))shows the round-trip times (RTTs) measured via TCP ACKs. BBRx flow averaged 110.2 Mbps while BBR flow averaged 92.6 Mbps.

From Figure 4(a), both BBRx and BBR behaves aggressively during their initial bandwidth probing phase. However, the initial probing phase of BBR is more aggressive, and its



(a) Bytes in Flight (b) RTT

Fig. 4: Single Trial to Compare BBR and BBRx over LTE links

`cwnd` keeps growing more than 3 MBytes. Therefore, the BBR flow experiences a burst of duplicate ACKs at the end of initial start up phase, around 14% ACKs in the BBR flow are duplicated ACKs. On the contrary, the BBRx flow exists from its initial start up phase gracefully, and it does not trigger the burst of duplicate ACKs as the BBR flow does. Meanwhile, as Figure 4(b) shows, the burst of duplicate ACKs at the end of start up phase of the BBR flow also triggers a RTT spike up to 350 ms. Note, the large RTT spike along with a large amount of duplicated ACKs triggers the retransmission of two TCP segments and cwnd reduction, which slowing down of the transmission of the BBR flow.

As Figure 4 shows, both BBRx and BBR behave similarly during their steady phase while the BBRx flow maintains a slightly lower RTTs than the BBR flow. With the benefits of the lower RTT and no retransmission, the BBRx flow archives 20 Mbps higher average throughput than the BBR flow under similar wireless condition.

V. Conclusions

The paper presents a BBRx, an online learning enhanced TCP congestion control module based on TCP BBR and a user space program which allows automatically tuning BBRx parameters to react network condition changes during runtime. BBRx minimizes the operation risk of deploying pre-tuned TCP congestion control algorithm over fast path, and relieves the "parameter tuning" burden from performance engineers. To our best knowledge, BBRx is the first effort to improve BBR by combining the control theory (PI controller) with online-learning technology. BBRx provides a practical way to apply on-line learning algorithm for congestion control modules.

Our preliminary evaluation over a US tier-1 cellular networks show that BBRx has potential to achieve an optimal throughput and delay operation point in especially LTE environment to maximize per flow throughput without introducing high queuing delay.

References

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *ACM SIGCOMM Computer Communication Review*, volume 40, pages 63–74. ACM, 2010.

[2] V. Arun and H. Balakrishnan. Copa: Practical Delay-Based Congestion Control for the Internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*, April 2018.

[3] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. BBR: Congestion-Based Congestion Control. *ACM Queue*, 14, September-October, 2016.

[4] J. W. Chung. *Congestion Control for Streaming Media*. PhD thesis, Worcester Polytechnic Institute, Worcester, MA, July 2005.

[5] J. W. Chung and M. Claypool. Aggregate Rate Control for TCP Traffic Management. In *Proceedings of ACM SIGCOMM (poster)*, September 2004.

[6] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira. PCC Vivace: Online-Learning Congestion Control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, April 2018.

[7] P. Goyal, M. Alizadeh, and H. Balakrishnan. Rethinking Congestion Control for Cellular Networks. In *Sixteenth ACM Workshop on Hot Topics in Networks (HotNets)*, Palo Alto, CA, November 2017.

[8] S. Ha, I. Rhee, and L. Xu. CUBIC: a New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[9] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of ACM SIGCOMM*, August 1988.

[10] K. Winstein and H. Balakrishnan. TCP ex Machinea: Computer-Generated Congestion Control. *ACM SIGCOMM Computer Communication Review*, 43(4), 2013.

[11] 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. Technical report, Stanford University, 2018.