Deprecating The TCP Macroscopic Model

Matt Mathis
Google, Inc
mattmathis@google.com

Jamshid Mahdavi
WhatsApp, Inc
jamshid@whatsapp.com

This article is an editorial note submitted to CCR. It has NOT been peer reviewed.
The authors take full responsibility for this article's technical content. Comments can be posted through CCR Online.

ABSTRACT

The TCP Macroscopic Model will be completely obsolete soon. It was a closed form performance model for Van Jacobson's landmark congestion control algorithms presented at Sigcomm'88. Jacobson88 requires relatively large buffers to function as intended, while Moore's law is making them uneconomical.

BBR-TCP is a break from the past, unconstrained by many of the assumptions and principles defined in Jacobson88. It already out performs Reno and CUBIC TCP over large portions of the Internet, generally without creating queues of the sort needed by earlier congestion control algorithms. It offers the potential to scale better while using less queue buffer space than existing algorithms.

Because BBR-TCP is built on an entirely new set of principles, it has the potential to deprecate many things, including the Macroscopic Model. New research will be required to lay a solid foundation for an Internet built on BBR.

CCS CONCEPTS

• **Internet performance**;

KEYWORDS

TCP Performance, BBR-TCP, Performance Modeling

1 INTRODUCTION

The TCP Macroscopic Model[1] estimates that Reno TCP performance is proportional to one over the square root of the loss probability. It was derived from the algorithms described in Van Jacobson's landmark paper "Congestion Avoidance and Control"[2] This paper, and the principles of window based congestion control defined within, form the foundation of most congestion control work for the operational Internet over the last three decades. It is so well known that it is often cited without a formal reference as Jacobson88.

Jacobson88 describes the Congestion Avoidance algorithm (from [3]) and Slowstart algorithms. Our model estimated an upper bound on the average performance of the Additive Increase / Multiplicative Decrease behavior of the Congestion Avoidance algorithm. Others have provided more refined estimates which also modeled other aspects of TCP behavior, such as receiver window limits and timeouts[4].

Over the years, the congestion control algorithms in Jacobson88 evolved into the modern Reno[5], Compound[6] and CUBIC[7] TCP variants which are commonly seen in the Internet today.

At the time our interests at the Pittsburgh Supercomputing Center were focused on maximizing the performance of TCP over high-speed networks[1]. In pursuit of this, we standardized TCP Selective Acknowledgment[11], which permitted TCP to maintain its self clock across round trips with multiple packet drops. In the years since then, a lot more work has been done but now we argue that Jacobson88 and the Macroscopic Model have been pushed as far as they can take us.

In this editorial, we examine the impact of "Bottleneck Bandwidth and Round-trip propagation time" (BBR) TCP[12, 13] on past research on congestion control. BBR is a new approach to congestion control based on an entirely new set of principles. In a very real sense BBR is a "redo" of Jacobson88. BBR makes the Macroscopic Model irrelevant, and calls into question three decades of research that is explicitly or implicitly based on the assumptions in Jacobson88.

2 CLOCKING IN TCP

In addition to the congestion algorithms described above, Jacobson88 introduced two key design principles: packet conservation and self clock.

"Packet conservation" is the explicit constraint that packets could only be sent into the network when earlier packets have left the network. A data packet sent by the sender would: traverse the network to the data receiver; the receiver generated an acknowledgement (ACK); which traversed back through the network; granting permission for the sender to release additional packets. The sender usually sent the same number of packets as the receiver reported were delivered, maintaining an approximately fixed number of packets in flight. The congestion control algorithm periodically adjusted the number of packets in flight by sending fewer or extra packets in response to some ACKs.

Packet conservation naturally tends to implement a self clock. While TCP is successfully maintaining a full network, there will typically be a standing queue of packets at the dominant bottleneck. As the packets pass through the bottleneck and traverse the network they maintain an approximately fixed-sized queue at the bottleneck. This flow of packets provides a "clock" for the entire communication system, such that the bottleneck is busy and everything else happens just in time to keep the bottleneck busy.

Jacobson88 worked well in a world where there were no delay-sensitive applications and memory was cheap enough that most

[1]For perspective, at that time our highest speed network was a 800 Mbps point to point connection between two supercomputers in a data center[8]. Our "high-speed" LAN was 100 Mbps FDDI[9] and our shared connection the vBNS[10] was OC-3 (155 Mbps), which was later upgraded to OC-12 (622 Mbps).

network devices had large enough queue buffers at the bottleneck to provide a good, stable self clock.

The key parameter of a network queue is drain time: how long does it take for a full queue to completely drain through the bottleneck if no additional packets arrive? If the drain time is slightly larger than the path round trip time it turns out to be fairly easy for Jacobson88 style congestion control to maintain a queue at the bottleneck and attain 100% utilization across a wide range of conditions. All of the old TCP variants could do this. Newer algorithms, such as CUBIC[7], substantially improve on Jacobson88 under less ideal conditions, but don't generally change the underlying self-clock and packet conservation principles.

However, Moore's law dooms large buffers deep in the interior of the Internet. The problem is that maintaining constant drain time in the presence of ever increasing interface (link) speeds requires that the pace of progress for the queue buffer memory exceeds Moore's law. Colloquially Moore's law states that the product of speed (clock rate) and complexity (device count) doubles every 18 months. Consider the following thought experiment: Internet data rates have been doubling roughly every 2 years (slightly slower than Moore's law). To maintain constant drain time the queue buffer memory has to double in size every 2 years. It also has to double in speed every 2 years. With data rates doubling every 2 years, the speed-complexity product for buffer memory has to double every year to maintain constant drain time. There is no cost effective way to do this, and as a consequence over a span of decades the available drain times have been dropping for the fastest devices on the network.

For the fastest network switches (100Gb/s per port and above)[2] drain times are now sub millisecond[14]. These queues are not large enough to shift packets in time by as much as a millisecond, and as a consequence self clock is unable to smooth out colliding bursts from different senders. Furthermore, the use of packet loss as the primary congestion signal requires sufficient buffering to allow for multiplicative backoff without wasting link bandwidth.The investigation of buffering requirements has been an active research area for 25 years, without any convincing resolution[15–19]. The underlying assumption in Jacobson88, that the network has sufficient buffering, is not sustainable given the economic realities of scaling network buffer sizes.

At the same time the Internet edge has developed the opposite problem: in home networks where data rates are relatively low (typically 1 Gb/s and below), it is easy to build large buffers with plenty of bandwidth. These devices can have queue drain times in the seconds or tens of seconds range, a problem known as BufferBloat[20].

There is a common theme here: Jacobson88 does not manage bottleneck queue occupancy, and in fact controls against queue full, rather than the onset of queueing. Although the research in alternatives goes back a long way[21], none have met the bar for wide deployment.

3 BBR

BBR's genesis came from Van Jacobson himself, reflecting on the failings of Jacobson88 in modern networks.

[2]There are routers on the market that have long drain times at these rates but they are extremely expensive.

BBR takes a substantially different approach to congestion control. It uses measurements to construct an explicit model of the network bottleneck and the rest of the path. This model predicts when the bottleneck is about to become idle (i.e. have available capacity); it aims to transmit new data just-in-time to keep bottleneck utilization high without causing excessive queue occupancy. This approach can eventually optimize both bandwidth and buffer utilization across the entire Internet performance spectrum.

The algorithms, code and testing experience have been widely published[12, 13, 22] and are not described in detail here. The BBR algorithms are based on four broad principles:

- With a few exceptions, transmissions are paced: scheduled by some mechanism such as tcp_fq[23] or a Carousel timing wheel[24]. Pacing eliminates the need for the network to preserve a self clock and reduces variance in the data rates at all network bottlenecks, which in turn reduces the transient queues everywhere, making more efficient use of precious network buffer memory. Carousel also improves the efficiency of resident kernel buffer memory, because it enables TCP to deliver data to the network layer just-in-time. Pacing also improves transactional flows (short on/off flows) because after a pause TCP can be restarted at its prior rate. Without pacing, every transaction must either slowstart or restart at line rate.
- BBR builds an explicit model of the network and bottleneck. The primary parameters are maximum delivered data rate (*max_BW*) of the forward path and minimum Round Trip Time (*min_RTT*) of the entire path. There are also parameters that characterize ACK aggregation and jitter, ECN markings, etc.
- The BBR framework emphasizes model parameters that have concrete definitions and are calculated from direct measurements of the packet stream. They can also be approximated by passive observations of sent and received packets. Contrast this to *cwnd* and *ssthresh* which are heuristics that do not reflect well defined network properties and can only be indirectly inferred from observing packets.
- The key innovation behind BBR is using different experiments to measure *min_RTT* and *max_BW* at disjoint times. BBR introduces new assumptions that the network usually has relatively stable properties and that non-overlapping measurements can accurately estimate the optimal operating point.

The core algorithm in BBR is to dither the paced sending rate above and below the maximum observed received data rate, *max_BW*. The maximum receive rate is probed by sending at 125% of *max_BW*. If the network is already full and flows have reached their fair share, the observed *max_BW* won't change. *min_RTT* is observed every time the network is underfilled, either due to application pauses (transactions) or by deliberately reducing the sending rate for one round trip every 10 seconds. These measurement experiments are managed by a collection of heuristics that are really what determine BBRs personality and how it interacts with other flows. BBR provides a modular and extensible framework for heuristics, with all parameters specified in standard units. Although we believe that

BBR's core algorithms form a solid framework, the suite of measurement heuristics is still evolving, and presents many opportunities for future work.

Google is already using BBR for both internal and external traffic, and can confirm that BBR performs better (higher data rate and/or lower queue occupancy) than CUBIC for web, video, and RPC traffic[25]. WhatsApp is also using BBR with measurable improvements to performance. The lower queue occupancy indicates that it is not generally taking capacity away from other transport protocols and effectively increases the Internet's capacity, because network buffers now have more headroom to accept incast flows and bursts from legacy stacks[3]. BBR substantially raises the network efficiency and supports good performance at higher network loads than self-clocked transport protocols. This effect is easy to demonstrate[26] and has potential to reduce network costs for all large content providers. As a consequence many content providers are testing or have already deployed BBR.

BBRv2 patches are published under dual GPLv2/BSD license[27]. An earlier version is present in the Linux 4.19 LTS train[4] People interested in serious experimentation or bug reporting must follow the conversation on the public e-mail list[28] and use the most recent BBR patches. BBR over UDP is part of QUIC in chromium (BSD license)[29], and saves the effort of configuring and building kernels. For FAQs and the most up-to-date information about releases see the documentation site[22]. Netflix is working on porting BBR to FreeBSD. BBR is very much a work in progress and the team is actively seeking contributions from the community.

4 OPENING A NEW ERA...

These are new research questions, following from BBR.

There is still a lot of work to do before BBR is completely "finished." Although Google has good data on BBR performance in its own serving environment, this data may not be representative of the Internet at large. In particular, paths from Google and other large content providers all have similar structures: the serving end of the path is typically subject to load balancing and admission control, and is only rarely congested. The bottleneck is nearly always near the user and has large buffers. The path in between, through the ISP, is likely to be short. In this environment, BBR quickly discovers the available capacity at the user's bottleneck and can keep it full without creating large queues. Google tests paths fitting into this pattern at scale, so it is no surprise that the results for the latest BBR are consistently better than with CUBIC (faster and/or lower queue occupancy)[30].

However this does not prove that BBR is safe and stable in all parts of the Internet. Although available in Linux today, it is still very much a work in progress, and off by default. Content providers and researchers who have sufficient knowledge and interest can turn it on and monitor its impact on their own services and networks. As more site engineers and researchers try it out, the community will gain confidence that there are no lurking surprises.

The academic user community has empirically demonstrated that BBRv1 starves CUBIC under some conditions[31, 32]. BBRv2 improves coexistence with CUBIC by spacing out the BW probe experiments to loosely match CUBIC's declining control frequency at high data rates and long RTTs. Note that there is a tradeoff here: making BBR more CUBIC compatible also means that it is subject to some part of CUBIC's scale limitations. Making BBR probe less frequently reduces BBR's ability to quickly fill freed capacity on high speed paths. There is a tussle between BBR's rate agility and extent to which BBR avoids starving CUBIC. There are several additional tussles including: the excess queue needed to probe for bandwidth; and the maximum steady-state packet loss that BBR can tolerate or cause.

Note that balancing tussles requires heuristic design compromises that make BBR less optimal under myopic views of its performance. In the case of CUBIC compatibility, one extreme position might be "crush CUBIC and move on"; while strict backwards compatibility might require that "BBR not harm CUBIC under any circumstance". Neither of these positions is healthy for the Internet. A better approach is to strike a balance between these extremes that can decay over time to follow CUBIC's declining share of high-speed traffic.

Since balancing tussles has the potential to cause winners and losers in other parts of the Internet, the compromises should be informed by public conversations in Internet standards and research communities. These conversations take place on the BBR public e-mail list[28] and in the Internet Congestion Control Research Group (ICCRG) of the Internet Research Task Force (IRTF). Standardization is still a ways off.

There are also open opportunities to add more heuristics to BBR to support additional congestion signals. For example in networks where ACKs authentically reflect packet delivery times, delay gradient[33, 34] and chirping[35] are known to be able to provide early and more accurate estimates of the path capacity. We believe that many window based congestion control algorithms could be recast into BBR's rate-based framework. Other examples include DCTCP[36],[5] and Remy[37]. Furthermore, to the extent that these algorithms can be adapted to BBR, BBR has the potential to match or outperform any congestion control algorithm that has ever been built on Jacobson88, limited only by the design compromises needed to balance tussles.

It is our belief that convincing the community that BBR is safe enough to unconditionally deploy in all environments (i.e. is a candidate default congestion control) requires either adding algorithms to use all significant congestion signals or demonstrating why these signals are fully redundant and never provide additional information.

5 ... AND CLOSING AN OLD ERA

These are old research questions that we thought we understood, but maybe not. In general we picked on the oldest paper in each area. Subsequent derived work likely suffers a similar fate.

[3] As long as ECN marking rate and the loss rate stay reasonable and there is headroom in the buffers, BBR can't cause too much backpressure on legacy protocols.

[4] The LTS version does not perform well over WiFi and other environments that aggregate or thin ACKs, but it can be turned on in Debian and many other Linux distros: **sudo sysctl -w net.ipv4.tcp_congestion_control=bbr ;**. All of the versions in the mainline kernel are known to give anomalous results in some environments.

[5] BBRv2 includes a reimplementation of DCTCP. The ECN community is currently engaged in an active tussle over ECN semantics and algorithms. BBR is likely to evolve to match the emerging consensus.

BBR is a radical change from all past congestion control algorithms. It is so different that it is difficult to tell what parts of the existing knowledge base may apply to BBR.

Needless to say, the old TCP Models[1, 4] do not apply to BBR at all, because BBR is not AIMD. Furthermore, all papers or standards built on these models are also suspect. The results may still be correct, but at the very least, the underlying assumptions in the research need to be revisited.

The traditional TCP self clock clashes with channel arbitration algorithms present in nearly all mobile and shared media, such as WiFi, LTE, and DOCSIS. This phenomenon was first observed in the Ethernet capture effect[40] and has heavily influenced many channel arbitration designs since then. The essence of the problem is that data on the forward path blocks the ACKs on the return path which are needed to preserve the self clock. The natural consequence is that you observe alternating bursts of data and ACKs. In many (poor) designs you see one burst of data per round trip time and the bottleneck is idle while the data and ACKs traverse the rest of the path. In better designs the channel reverses direction more frequently and the shared bottleneck is kept busy by overlapping it with data and ACKs traversing the rest of the path.

BBR changes the details: it keeps sending data for a while, even without ACKs, which might make the problem better (data keeps flowing) or worse, if more newly arriving data keeps the channel stuck in the forward direction for longer. In principle all channel arbitration designs and their measurement studies might need to be revisited with BBR. However, many of these problems were observed in serving content to Google's users, and were in fact used to tune BBR's design. In particular, the BW probe phase deliberately creates a temporary queue that sufficiently resembles CUBIC's behavior in order to trigger scheduling heuristics present in some channel arbitration algorithms. These heuristics use queue backlogs to trigger high bandwidth modes.

It would be better if the LTE scheduling algorithms could be co-engineered with BBR, because as it stands today BBR induces temporary queues that potentially jitter all other flows for all Internet users everywhere it is deployed. Although temporary queues are necessary to probe *max_BW*, their dimensions (duration and/or queue depth) might be reduced if LTE used some other signal to detect bulk flows.

Another area open for new research is to consider how BBR might affect future network design at the highest levels. For example, the widespread use of CDNs (Content Delivery Networks) has divided up paths at the transport layer resulting in shorter round trip times. There has been a symbiotic relationship between CDNs and the TCP described by the Macroscopic Model because it predicts that shorter RTTs improve performance. BBR may no longer have this limitation and could shift designs at the highest level to emphasize capacity more and packet loss and RTT less when building networks.

All observations about packet trains and bursts in the Internet need to be revisited[41]. It was an explicit goal of the pacing to better interleave traffic (to avoid back-to-back data packets). From queue occupancy statistics mixed traffic from multiple BBR sources appears to be non-stationary Gaussian (no self correlation at the time scales smaller than queue drain times) and thus simple queue occupancy models apply.

Self clocking in TCP leads to interdependence between packet timings on a timescale of round trips rather than individual packet transmissions. This amplifies minute effects into larger scale effects and evidence of this can be seen in a variety of studies done in the 1990s[42–45]. In a world where much traffic becomes paced instead of self-clocked, at least one source of inherent burstiness will fundamentally change. This suggests a need for new traffic studies to understand what has changed, and what remains the same. A companion editorial in this issue[46], by Mark Crovella, examines the mindset needed to conduct good research into Internet traffic.

Measurenet Lab is rolling out a new platform[38] that will support an improved version of NDT over BBR[39]. Early results suggest Internet performance measurements that are more stable and better calibrated than other measurement tools.

For everybody who has worked on congestion control, reflect on how deprecating Jacobson88 might affect your results. And don't forget all textbooks and coursework as well.

6 CONCLUSION

BBR is a break from the past, unconstrained by many of the assumptions and principles in Jacobson88. It already out performs CUBIC over a large portion of the Internet, generally without creating queues of the sort needed by earlier congestion control algorithms. We see the transition from window based to rate-based congestion control to be inevitable, due to its potential to more efficiently use precious network buffers and the intrinsic deployment incentives for anybody serving content at large scales. We believe that BBR's core algorithms form a sound framework for an evolving suite of heuristic measurement strategies that will eventually recoup most that we have learned from Jacobson88. As BBR matures, it offers the potential to scale better and use substantially less queue buffer space than prior algorithms.

We were fortunate to be able to contribute to the understanding of TCP dynamics at a time when the Internet was just beginning the transition from an academic and research tool to becoming in many ways the lifeblood of modern society. As a community, our congestion control knowledge base, mental models and intuition have been built on the principles espoused in Jacobson88 more than three decades ago. In reflecting back for this 50th anniversary of SIGCOMM, we see that the era of TCP dynamics built upon self-clocked, window-based congestion control is coming to a close.

Looking ahead, who can say what the next 30 years might bring? It isn't unreasonable to think that the Internet could be approaching 1 trillion connected devices. At the outskirts, commercial space flight may mean a much vaster Internet than we have today. Closer to home, IoT will bring more and more connected devices into our homes and personal spaces, requiring careful attention to privacy and security. Some applications such as transportation and medical devices hold lives in their hands. In recent years we've seen a significant public awakening to many of the risks associated with an ever more connected world. Transport layer encryption, largely via TLS/SSL, is becoming the norm. Some applications are raising the standard to end-to-end application security by design (including techniques such as "encryption at rest"[48] and "end-to-end encryption"[49, 50] which are designed to protect security even if the"service they are using performs unsatisfactorily"[51].

Jacobson88 was a direct response to congestion collapse in the early Internet. Today, an equivalent event at scale would be enormously disruptive. 30 years from now, it could be disastrous. Safety and security have become increasingly important in all aspects of networking, and need to be a bedrock principle in research going forward.

The coming era will require re-examination of much of what we have learned about Internet traffic dynamics. Some observations will stand, particularly ideas anchored in the underlying statistics and mathematics of packet switched networks. Others need to be swept away with Jacobson88.

In any case, we will have three decades of past knowledge to revisit, review and relearn.

ACKNOWLEDGMENTS

We'd like to thank Jeff Semke and Teunis Ott for their invaluable contributions to the development of the TCP Macroscopic Model.

We'd also like to thank all of the BBR authors and testers: Neal Cardwell, Yuchung Cheng, Van Jacobson and a cast of many who developed and tested BBR.

Thank you to the internal reviewers, David Wetherall, and Neal Cardwell for their insightful comments, and to thank Huapeng Zhou and Jiafei Wen for their insights on BBR usage within Facebook and WhatsApp.

Special thanks to Eric Dumazet for his work on TCP_fq[23] and to Ahmed Saeed and Nandita Dukkipati for their work on Carousel[24], which are required infrastructure for deploying BBR on an industrial scale server. Through their effort, Linux now supports lockless pacing that scales to millions of flows.

REFERENCES

[1] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. 1997. The macroscopic behavior of the TCP congestion avoidance algorithm. SIGCOMM Comput. Commun. Rev. 27, 3 (July 1997), 67-82. DOI: https://doi.org/10.1145/263932.264023

[2] Van Jacobson and Mike Karels. 1988. Congestion Avoidance and Control. Proc. of ACM SIGCOMM '88, Vol 18 No. 4, 314-329. DOI: https://doi.org/10.1145/52324.52356

[3] Raj Jain, K.K. Ramakrishnan, and Dah-Ming Chiu. 1987. Congestion avoidance in computer networks with a connectionless network layer. Digital Equipment Corporation Technical Report DEC-TR-506.

[4] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. 1998. Modeling TCP Throughput: A Simple Model and Its Empirical Validation. Proc. of ACM SIGCOMM '98, 303-314. DOI: https://doi.org/10.1145/285237.285291

[5] E. Blanton and M. Allman. 2009. TCP congestion control. RFC5681. RFC Editor. DOI: https://doi.org/10.17487/RFC5681

[6] Kun Tan, Jingmin Song, Qian Zhang, and Murari Sridharan. 2005. A Compound TCP Approach for High-speed and Long Distance Networks. Microsoft Research Technical Report MSR-TR-2005-86.

[7] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. SIGOPS Oper. Syst. Rev. 42, 5 (July 2008), 64-74. DOI: http://doi.org/10.1145/1400097.1400105

[8] Arie Van Praag. 1994. Introduction to the Hippi Specifications. Retrieved from http://hsi.web.cern.ch/HSI/hippi/spec/introduc.htm

[9] D. Katz. 1989. A Proposed Standard for the Transmission of IP Datagrams over FDDI Networks. RFC1103. RFC Editor. DOI: https://doi.org/10.17487/RFC1103

[10] J. Jamison, R. Nicklas, G. Miller, K. Thompson, R. Wilder, L. Cunningham, C. Song. 1998. "vBNS: not your father's Internet" IEEE Spectrum, 35, 7 (July 1998), 38-46. DOI: https://doi.org/10.1109/6.694354

[11] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow. 1996. TCP Selective Acknowledgment Options. RFC2018. RFC Editor. DOI: https://doi.org/10.17487/RFC2018

[12] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. Queue 14, 5, Pages 50 (October 2016). DOI: https://doi.org/10.1145/3012426.3022184

[13] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR: Congestion-Based Congestion Control. Commun. ACM 60, 2 (January 2017), 58-66. DOI: https://doi.org/10.1145/3009824

[14] T. P. Morgan. 2016. BROADCOM STRIKES 100G ETHERNET HARDER WITH TOMAHAWK-II. Retrieved from https://www.nextplatform.com/2016/10/31/broadcom-strikes-100g-ethernet-harder-tomahawk-ii/

[15] Curtis Villamizar and Cheng Song. 1994. High Performance TCP in ANSNet. SIGCOMM Comput. Commun. Rev. 24, 5 (October 1994), 45-60. DOI: https://doi.org/10.1145/205511.205520

[16] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. 2004. Sizing router buffers. SIGCOMM Comput. Commun. Rev. 34, 4 (August 2004), 281-292. DOI: https://doi.org/10.1145/1030194.1015499

[17] A. Vishwanath, V. Sivaraman, and M. Thottan. 2009 Perspectives on Router Buffer Sizing: Recent Results and Open Problems. SIGCOMM Comput. Commun. Rev. 39, 2 (March 2009), 34-39. DOI: https://doi.acm.org/10.1145/1517480.1517487

[18] N. McKeown and C. Diot. 2019. Workshop on Buffer Sizing (Dec. 2-3, 2019). Announcement retrieved from https://buffer-workshop.stanford.edu/, 2019.

[19] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. 2019. Sizing router buffers (Redux). SIGCOMM Comput. Commun. Rev. 49, 5 (October 2019).

[20] Jim Gettys and Kathleen Nichols. 2012. Bufferbloat: dark buffers in the internet. Commun. ACM 55, 1 (January 2012), 57-65. DOI: https://doi.org/10.1145/2063176.2063196

[21] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. 1994. TCP Vegas: new techniques for congestion detection and avoidance. Proc. of ACM SIGCOMM '94, 24-35. DOI: https://doi.org/10.1145/190314.190317

[22] google/bbr. 2019. GitHub repository, retrieved https://github.com/google/bbr

[23] Eric Dumazet, Yuchung Cheng. 2013. TSO, fair queuing, pacing: three's a charm. Presentation to IETF/TCPM. Retrieved from https://www.ietf.org/proceedings/88/slides/slides-88-tcpm-9.pdf

[24] A. Saeed, N. Dukkipati, V. Valancius, V. Lam, C. Contavalli and A. Vahdat. 2017, Carousel: Scalable traffic shaping at end hosts. Proc. of ACM SIGCOMM '17, 404-417. DOI: https://doi.org/10.1145/3098822.3098852

[25] Neal Cardwell, et. al. 2019. BBR v2: A Model-based Congestion Control. ICCRG Working Group, IETF 104, Prague (Mar 2019). Retrieved from https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00

[26] N. Cardwell and S. Yeganeh. 2019. TCP BBR Quick-Start: Building and Running TCP BBR on Google Compute Engine. Retrieved from https://github.com/google/bbr/blob/master/Documentation/bbr-quick-start.md

[27] N. Cardwell. 2019. TCP BBR v2 Alpha/Preview Release. Retrieved from https://github.com/google/bbr/blob/v2alpha/README.md

[28] BBR Development. 2019. Google Group. Retrieved from https://groups.google.com/forum/#!forum/bbr-dev Email address: bbr-dev@googlegroups.com

[29] QUIC/BBR sources for chromium. 2019. Chromium source code repository. Retrieved from https://cs.chromium.org/chromium/src/net/third_party/quiche/src/quic/core/congestion_control/

[30] Neal Cardwell, et. al. 2019. BBR v2: A Model-based Congestion Control, IETF 105 Update. ICCRG Working Group, IETF 105, Montreal (July 2019). Retrieved from https://datatracker.ietf.org/meeting/105/materials/slides-105-iccrg-bbr-v2-a-model-based-congestion-control-00

[31] H. Haile, P. Hurtig, K. Grinnemo, A. Brunstrom, E. Atxutegi, F. Liberal, and A. Arvidsson. 2018. Impact of TCP BBR on CUBIC Traffic: A Mixed Workload Evaluation. Proc. of 2018 30th International Teletraffic Congress (ITC 30). DOI: https://doi.org/10.1109/ITC30.2018.00040

[32] Mario Hock, Roland Bless, and Martina Zitterbart. 2017. Experimental Evaluation of BBR Congestion Control Proc. of IEEE 25th Int'l. Conf. Network Protocols, vol. 17, pp. 1-10. DOI: https://doi.org/10.1109/ICNP.2017.8117540

[33] David A. Hayes, Grenville Armitage. 2011. Revisiting TCP congestion control using delay gradients. Proc. of the 10th international IFIP TC 6 conference on Networking - Volume Part II (NETWORKING'11), Vol. Part II., 328-341.

[34] Radhika Mittal, et. al., 2015. TIMELY: RTT-based Congestion Control for the Datacenter. SIGCOMM Comput. Commun. Rev. 45, 4 (August 2015), 537-550. DOI: https://doi.org/10.1145/2829988.2787510

[35] M. Kuhlewind, B. Briscoe. 2010. Chirping for Congestion Control-Implementation Feasibility. Proc. of Int'l Wkshp on Protocols for Future, Large-scale & Diverse Network Transports (PFLDNeT'10).

[36] Mohammad Alizadeh, et al. 2010. Data Center TCP (DCTCP). Proc. of ACM SIGCOMM '10, 63-74. DOI: https://doi.org/10.1145/1851275.1851192

[37] Keith Winstein and Hari Balakrishnan. 2013. TCP ex Machina: Computer-Generated Congestion Control Proc. of ACM SIGCOMM '13, 123-134. DOI: https://doi.org/10.1145/2486001.2486020

[38] Chris Ritzo. 2018. Modernizing the M-Lab Platform. Retrieved from https://www.measurementlab.net/blog/modernizing-mlab/

[39] Measurement Lab. 2019. NDT (Network Diagnostic Tool). Retrieved from https://www.measurementlab.net/tests/ndt/

[40] K. K. Ramakrishnan and H. Yang. 1994. The Ethernet capture effect: analysis and solution. Proc. of 19th Conference on Local Computer Networks. DOI: https://doi.org/10.1109/LCN.1994.386597

[41] Hao Jiang and Constantinos Dovrolis. 2005. Why is the internet traffic bursty in short time scales?. Proc. of ACM SIGMETRICS '05, 241-252. DOI: https://doi.org/10.1145/1064212.1064240
[42] S. Floyd and V. Jacobson. 1992. On Traffic Phase Effects in Packet-Switched Gateways. Internetworking: Research and Experience Vol. 3, No. 3 (September 1992), p.115-156.
[43] Lixia Zhang, Scott Shenker, and Daivd D. Clark. 1991. Observations on the dynamics of a congestion control algorithm: the effects of two-way traffic. Proc. of ACM SIGCOMM '91, 133-147. DOI: https://doi.org/10.1145/115992.116006
[44] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. 1997. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. IEEE/ACM Trans. Netw. 5, 1 (February 1997), 71-86. DOI: https://doi.org/10.1109/90.554723
[45] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. 1994 On the self-similar nature of Ethernet traffic (extended version). IEEE/ACM Trans. Netw. 2, 1 (February 1994), 1-15. DOI: https://doi.org/10.1109/90.282603
[46] Mark Crovella. 2019. The Skillful Interrogation of the Internet SIGCOMM Comput. Commun. Rev. 49, 5 (October 2019).
[47] Hao Jiang and Constantinos Dovrolis. 2005. Why is the Internet Traffic Bursty in Short Time Scales?. Proc. of ACM SIGMETRICS '05, 241-252. DOI: https://doi.org/10.1145/1064212.1064240
[48] Apple, Inc. 2018. iOS Security. Retrieved from https://www.apple.com/ca/ business/resources/docs/iOS_Security_Guide.pdf
[49] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, D. Stebila. 2017 A Formal Security Analysis of the Signal Messaging Protocol Retrieved from https://eprint.iacr.org/2016/1013.pdf
[50] WhatsApp, Inc. 2017. WhatsApp Encryption Overview. Retrieved from https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf
[51] E. Omara, B. Beurdouche, E. Rescorla, S. Inguva, A. Kwon, A. Duric. 2019 The Messaging Layer Security (MLS) Architecture. Retrieved from https://www.ietf.org/id/draft-ietf-mls-architecture-02.txt