EPJ Web of Conferences **214**, 08026 (2019) https://doi.org/10.1051/epjconf/201921408026
CHEP 2018

Integration and Evaluation of QUIC and TCP-BBR in long-haul Science Data Transfers

Raul H. C. Lopes[1,3,*], *Virginia N. L.* Franqueira[2,**], and *Duncan* Rand[1,***]

[1]Jisc, Lumen House, Library Avenue, Harwell Campus, Didcot, Oxfordshire OX11 0SG, United Kingdom
[2]College of Engineering and Technology, University of Derby, Derby, DE22 1GB, UK
[3]College of Engineering, Design and Physical Sciences, Brunel University London, Uxbridge, UB8 3PH, UK

Abstract.
Two recent and promising additions to the internet protocols are TCP-BBR and QUIC. BBR defines a congestion policy that promises a better control in TCP bottlenecks on long haul transfers and can also be used in the QUIC protocol. TCP-BBR is implemented in the Linux kernels above 4.9. It has been shown, however, to demand careful fine tuning in the interaction, for example, with the Linux Fair Queue. QUIC, on the other hand, replaces HTTP and TLS with a protocol on the top of UDP and thin layer to serve HTTP. It has been reported to account today for 7% of Google's traffic. It has not been used in server-to-server transfers even if its creators see that as a real possibility.
Our work evaluates the applicability and tuning of TCP-BBR and QUIC for data science transfers. We describe the deployment and performance evaluation of TCP-BBR and comparison with CUBIC and H-TCP in transfers through the TEIN link to Singaren (Singapore). Also described is the deployment and initial evaluation of a QUIC server.
We argue that QUIC might be a perfect match in security and connectivity to base services that are today performed by the Xroot redirectors.

1 Introduction

For many years the Internet was running on 5 protocols: IP for routing, TCP to reliably transfer packets, SSL to authenticate and DNS to map names to addresses. UDP was the forgotten protocol. And that was good enough in the 1980s and 1990s when bandwidths were limited to kilobits and megabits per second.

The introduction of gigabit pipes brought the weight of TCP to the forefront. A naive deployment of a CentOS 7 Linux test bed on a 100 Gbps using the best available Mellanox switches will not give more than 20 Gbps speed. Extreme tuning and special drivers with a kernel-by-pass will lead the venerable and light weight *iperf3* to around 90 Gbps. Conclusion:

*e-mail: rlopes@cern.ch
**e-mail: v.franqueira@derby.ac.uk
***e-mail: duncan.rand@jisc.ac.uk

tuning TCP is hard. And security in TCP is an afterthought. Security was not considered in the scope of experiments leading to this paper[1].

In this paper, we examine some pitfalls in the design of TCP that make achieving maximum usage of the available bandwidth hard and security a demanding task. We survey recent additions in congestion control that look into improving TCP performance, and specifically deploy and test TCP-BBR. We examine UDP limits and investigate the QUIC protocol. QUIC has been presented by its creators as a tunnelling protocol aiming at reducing the latency present in TCP, while providing security by design. We conjecture whether QUIC might be an ideal protocol for science data transfers and, more specifically, for the WLCG.

2 The TCP protocol congestion control

The TCP protocol has built-in design choices that constrain any work to improve its performance [1]:

- TCP has a mandatory slow start in the form of a 3-way handshake;
- Another 3-way handshake is added if safety is brought in through TLS.
- Limitations to performance are also present in its *ACK* pacing congestion control.
- TCP's big quality in the form of guarantees of packet delivery impose limitations in the of head-of-line blocking.
- TCP is unsafe by design.
- TCP is ungraceful in the presence of link errors leading to slow recovery and unfairness in apportion of blame among streams sharing a same connection.

In this section we study how recent congestion control algorithms try to improve the *ACK* pacing limitations of TCP. QUIC, examined in section 3, is investigated to try to understand and overcome some of the other problems identified in the TCP protocol.

2.1 TCP limitations and congestion control

TCP has been described as an *ACK* pacing protocol, term borrowed from Geoff Huston [2], also described in the TCP-BBR seminal paper by Cardwell et al. [3] (also in [4]).

The TCP congestion control uses a congestion window variable (*cwnd*) to limit the amount of data it can send before receiving any *ACK* reply. The value of *cwnd* start at a maximum segment size value is increased more or less faster depending on the algorithm used. The value of *cwnd* can be increased until a *rwnd* value defined by the receiver or until congestion is detected either by receipt of a duplicate *ACK* or a timeout. How the *cwnd* is decreased from that point is also determined by the congestion control.

Recent developments in congestion control for TCP have concentrated in decreasing the time to achieve maximum usage of the link available and keeping that maximum usage stable. Those targets conflict with the fact that if a sender exceeds in packet sending, packets will be dropped and TCP will have to retransmit, and even stop transmission to flush buffers. The demands on the protocol become starker when we take into account that TCP has to act in the present based on old history.

The resulting speeding in a transfer is strongly influenced by the bandwidth available, end-to-end latency and efficiency of protocol and the receiver running it. TCP tries to minimise packet loss and re-ordering, and maximise usage of bandwidth [5]. In order to achieve

[1] At the time of writing Jisc has test beds in Slough and London to test transfers in realm of hundred of gigabits per second.

that, a TCP sender uses the *ACK* sequences it receives to build a model of the round trip time, efficiency of its receiver and buffer length available in this receiver. The processing of *ACK* packets is the only signal that TCP has to decide whether it can increase the rate of sending or it must suspend it to let the receiver recover.

The search for more efficient usage of bandwidth with reduced latency has lead to a flurry of research in congestion control. Dordal [6] describes 12 *recent* additions to the family of congestion control algorithms available, and BBR is not even described among them.

A standard CentOS install, running on kernel 3, ships with three congestion algorithms: Reno, H-TCP, and CUBIC. If a more recent kernel is installed, 4.10 through 4.19, BBR becomes available.

The first of these congestion control algorithms was Reno [6]. It doubles the packet sending rate until it finds packet loss. Then, it decreases the sending rate by 50% over the following *round trip time* (*RTT*). It has been shown Reno achieving at most 3 Mbps on a 10 Gbps bandwidth in the presence of a 100ms *RTT* and 1% loss.

The algorithm H-TCP has its first description in [7]. It targets a faster growth of the *cwnd*. Growth is determined by the time elapsed since the last loss, typically by starting an acceleration of *cwnd* growth after some fixed time threshold since the last loss. H-TCP behaves as starting from point zero after each loss and, consequently, will not immediately enter its top speed mode, even when the *cwnd* is large, which makes it resemble and behave fairly to TCP Reno.

CUBIC, described in [8], tries to improve efficiency when the product of bandwidth times delay is large, which the standard in intercontinental WLCG transfers, as in Europe to USA or Europe to Eastern Asia. CUBIC uses a cubic function to try to achieve fast windown expansion. It then slows the windpow increase when the *cwnd* approaches the previous network ceiling.

The algorithms up to CUBIC tend to increase more or less quickly the *cwnd* value until they find packet loss, and then decrease until achieving buffer drained state. The congestion control policy defined by BBR takes the different approach of controlling the sending rate by measuring the *RTT*. At any state, if the *RTT* is equal to the *RTT* in the previous state, BBR assumes that there is available bandwidth to increase *cwnd*. If the probed *RTT* increases, BBR assumes that it is reaching the onset of queueing. It does not use packet loss as a deterministic indication of saturation.

2.2 TCP-BBR, CUBIC and H-TCP on intercontinental data transfer tests

We have been conducting tests trying to improve the utilization of the TEIN link for transfers between London and Singapore. We have run tests on a local test bed to probe TCP for the capacity to fill an available link of 100 Gbps. Tuning of the TCP stack and use of kernel-bypass from Mellanox got us above the 90 Gbps rate, but not to 95 Gbps.

We have also used a setup with two servers at Brunel University London and Singaren, Singapore. The servers at Brunel run on Ubuntu and CentOS 7, with kernel 4.15 and 4.17 respectively. The server at Singaren runs on CentOS 7 with a kernel 3.10. A plot of rates obtained between the Ubuntu server with iperf3 is shown in Figure 1. The parallel plots show results respectivelly for 1 and 32 parallel streams. Tests conducted with a CentOS 7, kernel 4.14, server at Brunel did not differ significantly.

For the sake of comparison, the same tests were repeated between Brunel Ubuntu server and a ESnet DTN at CERN. Plots are shown in Figure 2. TCP-BBR shows overall a better performance. TCP-BBR, however, can be too efficient: concurrent transfers with TCP-BBR and CUBIC have shown that the former can take over the link and *crowd out* the former,

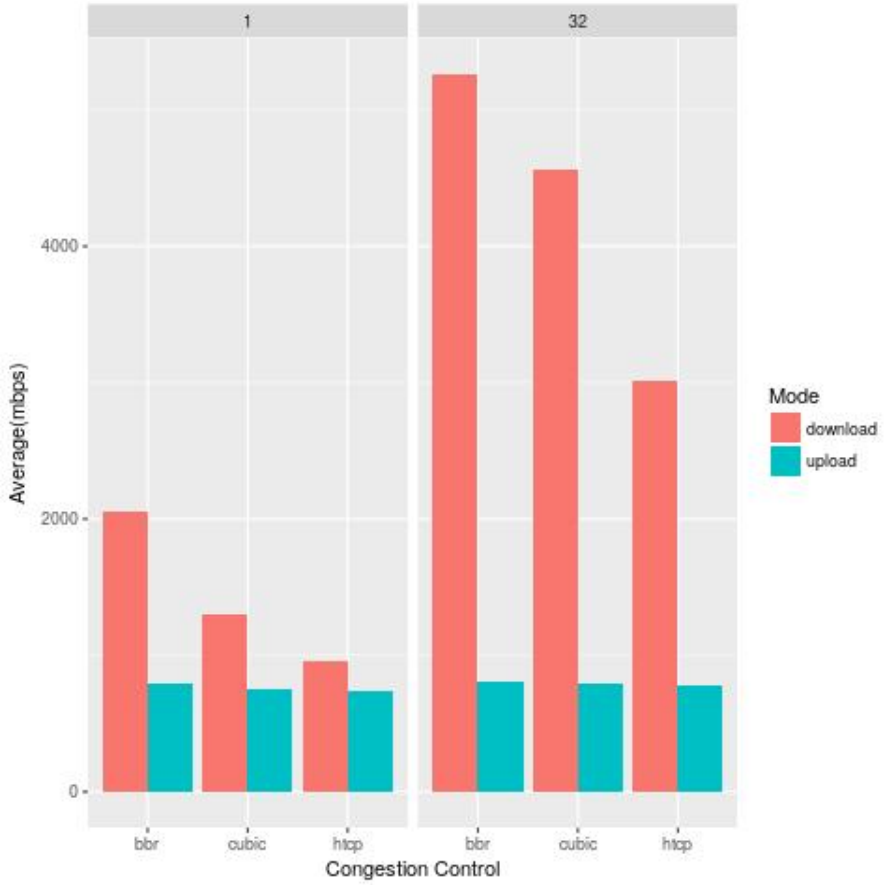

Figure 1. Brunel-Singaren upload and download iperf3 tests

.

which is also described by Huston in [2]. That, therefore raises doubts about TCP-BBR quality of service.

3 The QUIC protocol

Intercontinental data transfers typically face a round trip of 90 ms when connecting Europe to US or more than 180 ms to connect London to Australia via TEIN. Unless we could use neutrino beams, that situation cannot improve much due the limitations in light speed. In such conditions a 6-way handshake that must be performed for each connection establishment or after loss resumption is always going to be a serious obstacle to ramp up to top speed. The difficulties pile up when one has to add long timeouts for loss detection and the weight of encryption added to a design that did not contemplate security in first place.

The group that initially created the QUIC protocol [9, 10] presents as their initial intent to have a protocol that offered low latency and security from its design. The first document published introduces a protocol with a 1-*RTT* connection establishment and 0-*RTT* on resumption. Efficient loss detection and true concurrent streams contribute to reduce latency.

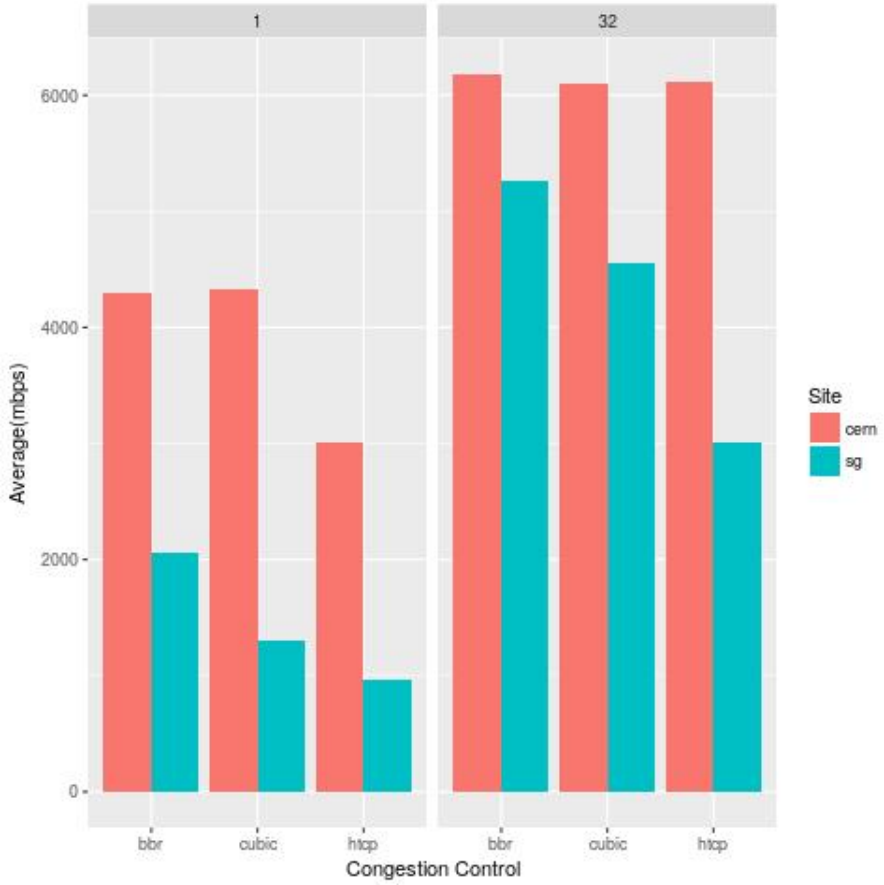

Figure 2. Brunel-ESnet upload and download iperf3 tests

In addition, encryption of all metadata and transfers offer the safety that might attract communities that are frequently skeptical of using the WLCG transfer and storage services.

3.1 QUIC requirements

QUIC's most importiant feature is that, in opposition to TCP, it was designed as a secure by default protocol. This is reflected in the connection handshake whose design also contributes to a lower latency on connection. In this section, we discuss some of design decissions that make QUIC distinctive and that might make an essentially faster and safer option for data science projects and the WLCG in particular.

HTTPS establishes an initial connection through a 3-way TCP handshake followed by a TLS handshake. That means two layers involved in a connection that will demand at least 3 round trips, which might mean close to a third or even half of a second just in connection establishment or in resumption after error. More time is wasted when the case is considered of two WLCG sites transferring two sets of 5 GB data from Chicago to London: in the absence of errors, over a 100 Gbps link, one third of the time demanded may be spent in

the connection round trips. This can also be easily tested with a transfer from London to CERN of a 300 Mbytes file over davs or https, when one third of time will be spent in the handshaking and connection establishment. Protocols like *root*, or *gsiftp* will demand more, with *srm* being the slowest of them all, both in connection establishment and transfer.

QUIC establishes a first connection between a new client and a server using a 1-*RTT* connection. Future connections, as in the case above, may be started on a 0-*RTT*.

A QUIC client initiates a connection to a server by sending a *hello message* that will force a reject. The reject, however, will carry information that can be cached by the client to authenticate, to encrypt future transfers, and to use a 0-*RTT* handshake for future transfers with the same server.

QUIC exchanges are almost all encrypted and authenticated. Only early handshake and reset packets are not encrypted. The unencrypted parts of a QUIC packet are essentially those elements that are necessary for routing or description of the packet, like connection identification and length of packet. However, protection against tampering is achieved by adding the unencrypted parts of the handshake to derivation of the connection's keys, as shown in [10, 11].

HTTP/2 gains efficiency over previous HTTP versions by using several HTTP requests on the same TCP connection. That itself can be a gain for the multi stream transfers frequently used in the data science. TCP, however, is a serial protocol that guarantees ordered delivery of the transferred packets. As a result, data loss in one transfer will block all multiplexed streams of that same connections.

QUIC introduces true concurrency when different HTTP streams are mapped to different transport streams, even when they are sharing the same connection and, as in the case of HTTP/2, avoiding additional handshakes. This reduces a TCP problem identified as head-of-line blocking, as shown in [12].

TCP efficiency can be hindered by retransmission ambiguity. TCP sequence numbers are used to add reliability and to define the order in which bytes are delivered. In the case of retransmitted packets, sequence numbers are repeated for both data and *ACK* transmission. TCP must then use lengthy timeouts that add to latency and bandwidth under utilization. QUIC packets, on the other hand, each carry its own unique number, and that includes distinct numbers for retransmission packets.

QUIC acknowledgment encodes the delay between the respective packet being received and *ACK* being sent, and enables a precise evaluation of the round trip time that can be used in the congestion control. Also, thanks to larger *ACK* blocks, QUIC can keep more bytes in transit in the presence of loss and re-ordering.

The protocol does not rely on any specific congestion control policy. Google has reported deploying gQUIC using CUBIC [12], but there are also reports of tests with BBR [3].

QUIC connections are uniquely identified by a *Connection ID*, which is independent of IP address and connection port. A client can migrate across different networks and still maintain its connection to a server. This feature raises interesting possibilities. Researchers using their laptops in a CMS *Anywhere-Any-Data-Any-Time* project can dream of migrating across WiFi connection without losing their transfer or proxy session. And data transfers may find more graceful failover options than what is presently offered by the Xroot re-directors.

3.2 Early evaluation and test bed

Microsoft has recently announced a forthcoming QUIC implementation for Windows and Azure. Cloudflare, see [11], has a service that can be used to test iQUIC implementations, and there are a dozen of open source implementations of the iQUIC standard, documented in [11].

Google seems to have the only large scale deployment of QUIC. The company reported in 2017 on an experiment that showed a reduction of 8% in latencies for users connecting to Google services using QUIC [10]. Also, the company claims [12] a reduction in video rebuffer ranging from 15% for mobile users to 18% for desktop users.

It is early days to fully evaluate large scale deployments of iQUIC. The standard is still changing, running at time of writing in draft 14, while TCP is reaping the benefits of long time improvements and research in the Linux kernel. Recently, a kernel-by-pass has been made available in the Linux stack [13] and similar efforts have been open-sourced by the BBC [14] and Intel [15].

We are building a test bed in a Jisc data center in London that makes available two servers running iQUIC, the IETF standardized QUIC, for tests: *edtn-slough-10g.ja.net.* Running a kernel 4.19, it makes available iperf3, iperf, and iQUIC services that can be used to compare iQUIC and TCP.

4 Conclusion

The final target of our research is to achieve better usage of available bandwidth mainly in the presence of large *RTT* and packet loss.

We have worked so far on the study of the limitations of TCP on transfers over multiple hundred gigabits per second. We have studied, and presented some initial results in section 2.2, comparing the performance of TCP-BBR against CUBIC and H-TCP.

We are in the process of deploying a test bed for experimental evaluation of protocols, which includes services based on QUIC.

This paper shows an initial evaluation of the use of the TCP-BBR congestion control algorithm to try to conquer better available bandwidth even in the presence of long *RTT* and errors. Our next target would be to further compare TCP-BBR and QUIC deployments.

5 Acknowledgments

Lopes, and Rand are members of the GridPP collaboration and wish to acknowledge support from Jisc/UK and funding from the Science and Technology Facilities Council, UK.

References

[1] L. Peterson, B. Davie, *Computer Networks: a systems approach* (Elsevier, 2012)
[2] G. Huston, *TCP and BBR*, accessed: 2018-11-30, `https://ripe76.ripe.net/presentations/10-2018-05-15-bbr.pdf`
[3] N. Cardwell, Y. Cheng, C.S. Gunn, S.H. Yeganeh, V. Jacobson, Queue **14**, 50:20 (2016)
[4] N. Cardwell, Y. Cheng, C.S. Gunn, S.H. Yeganeh, V. Jacobson, *BBR: Congestion-based congestion control*, accessed: 2018-12-02, `https://blog.acolyer.org/2017/03/31/bbr-congestion-based-congestion-control/`
[5] I. Grigorik, *High Performance Browsing Network* (O'Reilly, 2013)
[6] P.L. Dordal, *An introduction to computer networkks*, accessed: 2018-12-01, `http://intronetworks.cs.luc.edu/1/html/index.html`
[7] D. Leith, R.N. Shorten, Y. Lee, Tech. rep. (2005)
[8] S. Ha, I. Rhee, L. Xu, ACM SIGOPS Operating Systems Review - Research and developments in the Linux kernel **42** (2008)

[9] J. Roskind, *Quic: Design document and specification rationale*, accessed: 2018-12-01, https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/mobilebasic

[10] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar et al., *The QUIC Transport Protocol: Design and Internet-Scale Deployment*, in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (ACM, New York, NY, USA, 2017), SIGCOMM '17, pp. 183–196, ISBN 978-1-4503-4653-5, http://doi.acm.org/10.1145/3098822.3098842

[11] *Quic: A udp-based multiplexed and secure transport*, accessed: 2018-12-02, https://datatracker.ietf.org/doc/draft-ietf-quic-transport/

[12] A.M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, A. Mislove, *Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols*, in *Proceedings of the 2017 Internet Measurement Conference* (ACM, New York, NY, USA, 2017), IMC '17, pp. 290–303, ISBN 978-1-4503-5118-8, http://doi.acm.org/10.1145/3131365.3131368

[13] Linux Foundation, *XDP - eXpress Data Path*, accessed: 2018-12-02, https://prototype-kernel.readthedocs.io/en/latest/networking/XDP/

[14] A. Bonney, *High speed networking: Open sourcing our kernel bypass work*, accessed: 2018-11-30, https://www.bbc.co.uk/rd/blog/2018-04-high-speed-networking-open-source-kernel-bypass

[15] Linux Foundation, *DPDK: Data Plane Developer Kit*, accessed: 2018-11-30, https://www.dpdk.org/