

Evaluating BBRv2 on the Dropbox Edge Network

Alexey Ivanov

Dropbox, Inc
Mountain View, CA
SaveTheRbtz@GMail.com

Abstract

Nowadays, loss-based TCP congestion controls in general and CUBIC specifically became the *de facto* standard for the Internet. BBR congestion control challenges the loss-based approach by modeling the network based on estimated bandwidth and round-trip time. At Dropbox, we've been using BBRv1 since 2017 and are accustomed to its pros and cons. BBRv2 introduces a set of improvements to network modeling (explicit loss targets and inflight limits) and fairness (differential probing and headroom for new flows.) In this paper, we go over experimental data gathered on the Dropbox Edge Network. We compare BBRv2 to BBRv1 and CUBIC showing that BBRv2 is a definite improvement over both of them. We also show that BBRv2 experimental results match its theoretical design principles.

Keywords

BBR, BBRv2, CUBIC, TCP, Congestion Control, AQM.

Introduction

Since the Bottleneck Bandwidth and Round-trip propagation time (BBR) congestion control paper[5] was released it became production-ready and was added to Linux, FreeBSD, and Chrome (as a part of QUIC.) Back then, Dropbox evaluated[15] BBRv1 congestion control on our edge network and it showed promising results compared to CUBIC:

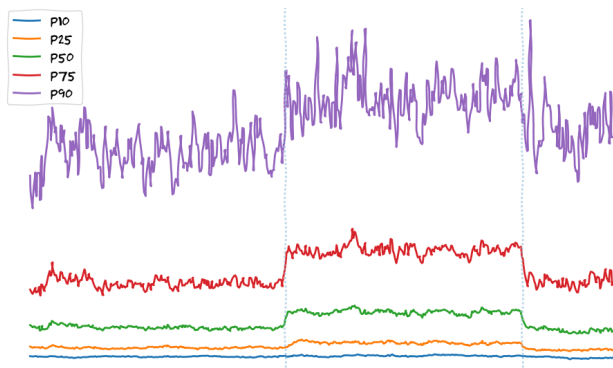


Figure 1: Dropbox desktop clients download goodput.

After BBRv1 has been fully deployed to Dropbox Edge Network we started to identify its corner cases and short-

comings. Some of them were eventually fixed, for example, BBRv1 being measurably slower for Wi-Fi users. Other trade-offs were quite conceptual, for example BBRv1s unfairness towards loss-based congestion controls[21], RTT-unfairness between BBRv1 flows[9], and its disregard for packet loss. Dropbox Edge Network **observed packet loss rates up to 6% on hosts using BBRv1, compared to around 0.5% on hosts using CUBIC with pacing.**

BBRv1 shortcomings

BBR developers were aware of these problems and actively worked on solutions. Following issues were identified[12]:

- Low throughput for Reno/CUBIC flows sharing a bottleneck with bulk BBR flows.
- Loss-agnostic; high loss if $bottleneck < 1.5 * BDP$.
- ECN-agnostic.
- Low throughput for paths with high degrees of aggregation (e.g. Wi-Fi.)
- Throughput variation due to low cwnd in `PROBE_RTT`.

BBR version 2

BBRv2 aims to solve some of the major drawbacks of the first version. Here is the list of BBR design principles (**bold** means its new in BBRv2[13]. See Table 1.):

- **Leave headroom: leave space for entering flows to grab.**
- **React quickly: using loss/ECN, adapt to delivery process now to maintain flow balance.**
- Dont overreact: dont do a multiplicative decrease on every round trip with loss/ECN.
- **Probe differentially: probe on a time scale to allow co-existence with Reno/CUBIC.**
- Probe robustly: try to probe beyond estimated max bw, max volume before we cut estimation.
- **Avoid overshooting: start probing at an inflight measured to be tolerable.**
- **Grow scalably: start probing at 1 extra packet; grow exponentially to use free capacity.**

	Cubic	BBRv1	BBRv2
Model parameters for the state machine	N/A	Throughput, RTT	Throughput, RTT, max aggregation, max inflight
Loss	Reduce cwnd by 30% on window by any loss	N/A	Explicit loss rate target
ECN	RFC3168 (Classic ECN)	N/A	DCTCP-inspired ECN (See “Explicit Congestion Notification” section.)
Startup	Slow-start until RTT rises (Hystart) or any loss	Slow-start until throughput plateaus	Slow-start until throughput plateaus or ECN or Loss rate ζ target

Table 1: Whats new in BBRv2: a summary

Test constraints

Non-mobile ISPs

The test was focusing on the Dropbox Desktop Client traffic and therefore mostly excludes mobile ISPs.

Bulk traffic focus

This experiment was aimed at high-throughput workloads. All TCP connections and nginx logs mentioned in the paper were filtered by having at least 1Mb of data transferred.

Localized test

This experiment was performed in a single point of presence (PoP) in Tokyo, Japan. Therefore, it carries over some of the biases common to traffic from that geographic area, including internet speeds, operating systems, client devices, etc.

Real user traffic

This is a real-world experiment with all of its pros and cons, including ISPs with heavy over-subscription, broken embedded TCP/IP implementations, etc¹.

Test setup

Following combinations of kernels and congestion control algorithms were tested:

- 5.3 kernel, `cubic`
- 5.3 kernel, `bbr`
- 5.3 kernel, `bbr2`

All of the servers are using a combination of `mq` and `sch_fq` qdiscs with default settings. Kernel is based on `Ubuntu-hwe-edge-5.3.0-18.19.18.04.2` with all patches from the `v2alpha-2019-11-17` tag.

Graphs were generated from either connection-level information from `ss -neit` sampling, machine-level stats from `/proc`, or server-side application-level metrics from web-server logs.

¹For testing congestion controls in a lab environment github.com/google/transperf can be used. It allows testing TCP performance over a variety of emulated network scenarios, including RTT, bottleneck bandwidth, and policed rate that can change over time.

Caveats

Keeping the kernel up-to-date

Newer kernels usually bring quite a bit of improvement² to all subsystems including the TCP/IP stack. For example, if we compare 4.15 performance to 5.3, we can see the latter having around 15% higher goodput:

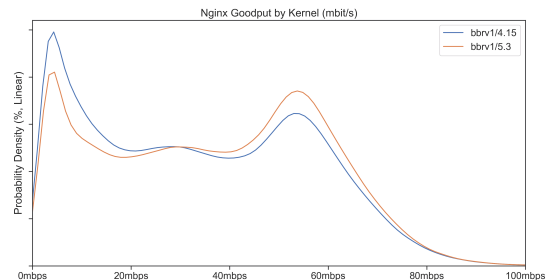


Figure 2: 4.15 vs 5.3 kernels server-side file upload goodput.

Most likely candidates for this improvement are:

- “tcp_bbr: adapt cwnd based on ack aggregation estimation,” fixing Wi-Fi performance.
- “tcp: switch to Early Departure Time model,” fixing the RTT jitter observed by TCP when used with pacing (See “Beyond As Fast As Possible” section.)

Keeping userspace up-to-date

Having recent versions of userspace is quite important if you are using kernels that are newer compared to what your OS was bundled with. This is especially true for packages like `ethtool` and `iproute2`.

In our configuration we’ve used Ubuntu 16.04 Xenial with a fairly recent 5.3.0 kernel along with `iproute2-5.4.0`. Here is an example of using the new version of `ss` (with the new output fields in **bold**):

²Recent Linux kernels also include mitigations for the newly discovered CPU vulnerabilities. We highly discourage disabling them (especially on the edge!) so be prepared to also take a CPU usage hit.

Listing 1: iproute2 5.4.0

```
$ ss -tie
ts sack bbr rto:220 rtt:16.139/10.041 ato:40
mss:1448 pmtu:1500 rcvmss:1269 advmss:1428
cwnd:106 ssthresh:52 bytes_sent:9070462
bytes_retrans:3375 bytes_acked:9067087
bytes_received:5775 segs_out:6327 segs_in:551
  data_segs_out:6315 data_segs_in:12 bbr: (bw
:99.5Mbps, mrtt:1.912, pacing_gain:1, cwnd_gain
:2) send 76.1Mbps lastsnd:9896 lastrcv:10944
lastack:9864 pacing_rate 98.5Mbps
delivery_rate 27.9Mbps delivered:6316 busy
:3020ms rwnd_limited:2072ms (68.6%) retrans
:0/5 dsack_dups:5 rcv_rtt:16.125 rcv_space
:14400 rcv_ssthresh:65535 minrtt:1.907
```

As you can see, the new `ss` version has all the new goodies from the kernels `struct tcp_info`, plus the internal BBR state from the `struct tcp_bbr_info`. This adds lots of metrics that can be used even in day-to-day TCP performance troubleshooting. Including very useful insufficient sender buffer and insufficient receiver window/buffer stats from the “tcp: sender chronographs instrumentation” patchset.

Experimental results

Packet loss

Most notably, switching from BBRv1 to BBRv2 results in an enormous drop in retransmits:

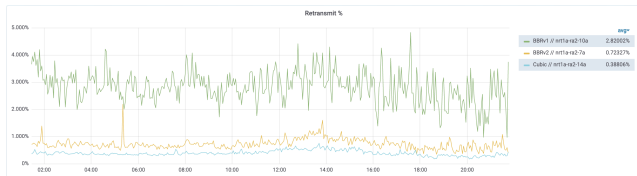


Figure 3: Retransmits, %. CUBIC, BBRv1, and BBRv2.

`RetransSegs` on these boxes is still higher than on ones using CUBIC but given that BBR was designed to ignore non-congestion induced packet loss, we would assume that things work as intended.

Looking deeper at the `ss` stats we can confirm this: BBRv2 packet loss is multiple times lower than BBRv1 (Figure 4.) though still higher than CUBIC (Figure 5.).

The deeper inspection also shows that BBRv2 has connections with $> 60\%$ packet loss. These types of outliers are present neither on BBRv1 nor CUBIC machines. Looking at some of these connections closer does not reveal any obvious patterns: connections with absurdly large packet loss come from different OSes (based on Timestamp/ECN support,) connections types (based on MSS,) and locations (based on RTT.)

Aside from these outliers, BBRv2 retransmissions are lower across all RTTs, when compared to BBRv1.

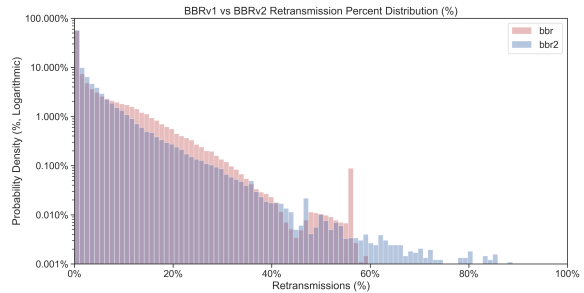


Figure 4: Retransmits, %, PDF. BBRv1 vs BBRv2.

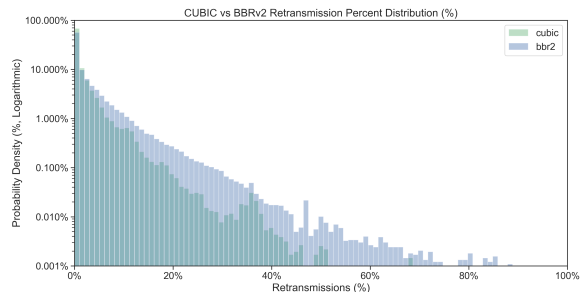


Figure 5: Retransmits, %. PDF. CUBIC vs BBRv2.

Throughput

On the throughput side, we looked at the nginx file upload goodput metric³ (Figure 7.). For lower percentiles of connection speeds, BBRv2 performance is closer to CUBIC. For higher ones, it starts getting closer to BBRv1. This is likely due to BBRv2 being fairer to other TCP connections on the bottleneck since the slower a connection is the more likely it is being congested (instead of being app limited.)

Connection-level stats confirm that BBRv2 has lower bandwidth than BBRv1 (Figure 8.) but still higher than CUBIC (Figure 9.).

So, does that mean that BBRv2 is slower? Yes, it does, at least to some extent. So, what are we getting in return? Based on connection stats, quite a lot. We've already mentioned lower packet loss (and therefore higher “goodput”) but there is more.

Packets in-flight

We've observed way fewer unacked packets which is a good proxy for bytes in-flight. BBRv2 looks way better than in BBRv1 (Figure 10.) and even slightly better than CUBIC (Figure 11.).

Plotting RTT vs in-flight heatmap shows that in the BBRv1 case amount of data in-flight tends to be dependent on the RTT. This is fixed in BBRv2 (Figure 15.).

³From the Traffic Team's perspective, one of our SLIs for edge performance is end-to-end client-reported download speed. For this test, we've been using server-side file upload speed as the closest proxy for it.

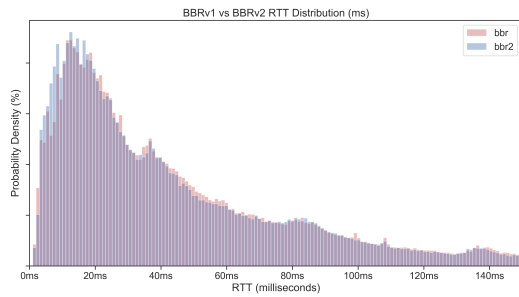


Figure 6: Round Trip Time. BBRv1 vs BBRv2.

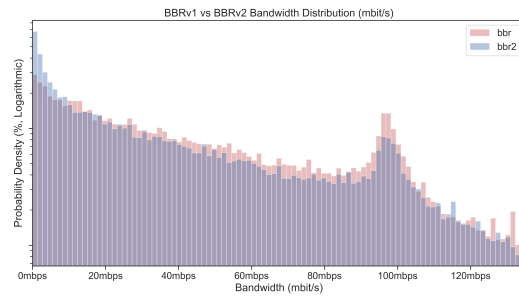


Figure 8: Bandwidth distribution. BBRv1 vs BBRv2.

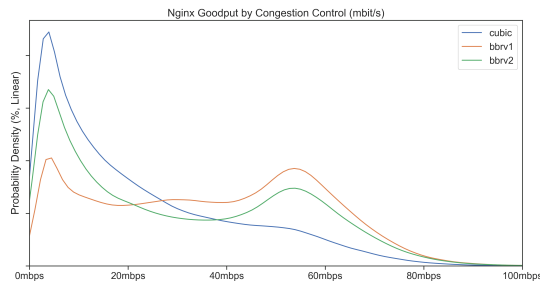


Figure 7: File upload goodput from nginx point of view.

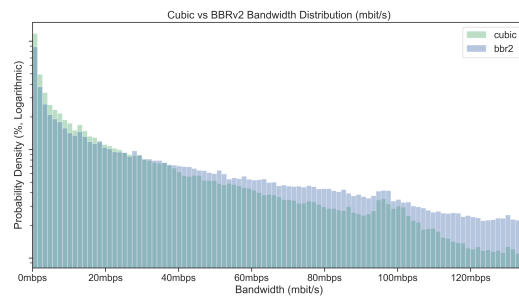


Figure 9: Bandwidth distribution. CUBIC vs BBRv2

As BBR co-author, Neal Cardwell, explains it:

“In all of the cases I’ve seen with unfairness due to differing *min_rtt* values, the dominant factor is simply that with BBRv1 each flow has a *cwnd* that is basically $2 * bw * min_rtt$, which tends to try to maintain $1 * bw * min_rtt$ in the bottleneck queue, which quite directly means that flows with higher *min_rtt* values maintain more packets in the bottleneck queue and therefore get a higher fraction of the bottleneck bandwidth. The most direct way I’m aware of to improve RTT fairness in the BBR framework is to get rid of that excess queue, or ensure that the amount of queue is independent of a flow’s *min_rtt* estimate.”

Receive Window Limited connections

We similarly observed that BBRv2 connections spend way less time being receive window limited than both BBRv1 (Figure 12.) and CUBIC⁴(Figure 13.).

Round Trip Time

Based on connection stats BBRv2 also has a lower RTT than BBRv1 (Figure 6.). This can be explained by a more graceful PROBE_RTT phase.

Correlations between Min RTT and Bandwidth

If we construct *bbr_mrtt* vs *bbr_bw* heatmap then vertical bands represent a network distance from user to our

⁴CUBIC was using *sch_fq* and pacing in this test too.

Tokyo PoP⁵ while horizontal bands represent common Internet speeds (Figure 16.)

What’s interesting here is the exponentially decaying relationship between MinRTT and bandwidth for both BBRv1 and BBRv2. This means that there are some cases where bandwidth is artificially limited by RTT. Since this behavior is the same between BBRv1 and BBRv2 we did not dig too much into it. It’s likely though that the bandwidth is being artificially limited by the users receive window.

CPU Usage

Previously, BBRv1 updated the whole model on each ACK received, which is quite a lot of work given that our average server receives millions of them every second. BBRv2 has an even more sophisticated network model but it also adds a fast path for ACK processing⁶ that skips model updates for the application-limited case. This, in theory, should greatly reduce CPU usage on common workloads.

In our tests, though, we did not see any measurable difference in CPU usage between BBRv1 and BBRv2. This is likely due to BBRv2 having quite a lot of debug code enabled.

⁵The 130ms RTT band represents cross-Pacific traffic and can be attributed to GSLB failure to properly route users to the closest PoP. This since have been fixed by RUM DNS[19][8].

⁶Along with ACK fast-path other optimizations to the BBR were introduced, aimed at both CPU usage improvements and goodput increase e.g. improved TSO auto-sizing, faster ACKs, and reduced standing queuing at the bottleneck with many competing BBR flows[14].

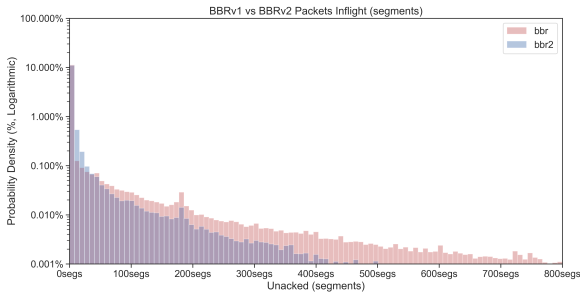


Figure 10: Unacked packets distribution. BBRv1 vs BBRv2.

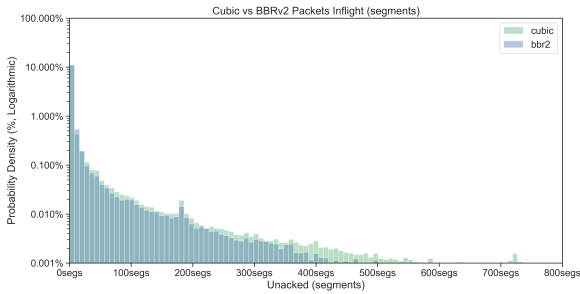


Figure 11: Unacked packets distribution. CUBIC vs BBRv2.

Conclusions

In our testing, BBRv2 showed the following properties:

- Bandwidth is comparable to CUBIC for users with lower percentiles of Internet speeds.
- Bandwidth is comparable to BBRv1 for users with higher percentiles of Internet speeds.
- Packet loss is 4 times lower compared to BBRv1⁷; still 2 times higher than CUBIC.
- Data in-flight is 3 times lower compared to BBRv1; slightly lower than CUBIC.
- RTTs are lower compared to BBRv1; still higher than CUBIC.
- Higher RTT-fairness compared to BBRv1.

Overall, BBRv2 is a great improvement over the BBRv1 and indeed seems way closer to being a **drop-in replacement for Reno/CUBIC** in cases where one needs higher bandwidth with lower buffer bloat. Adding the experimental ECN support to that and we can even see BBRv2 being used as a **drop-in replacement for Data Center TCP (DCTCP)**.

⁷Minus the 0.0001% of the outliers with a > 60% packet loss.

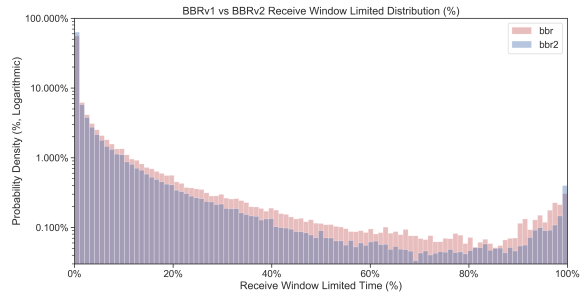


Figure 12: Receive Window Limited. BBRv1 vs BBRv2.

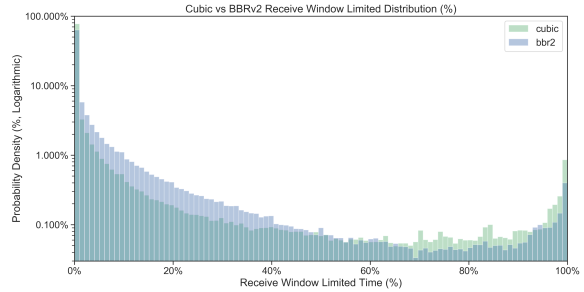


Figure 13: Receive Window Limited. CUBIC vs BBRv2.

Appendices

Explicit Congestion Notification

ECN is a mechanism for the network bottleneck to proactively notify the sender to slow down before it runs out of buffers and starts tail dropping packets. Currently, though, ECN on the Internet is mostly deployed in a so-called passive mode. Based on Apples data 74% most popular websites passively support ECN[20]. In our Tokyo PoP, **we observe 3.68% of connections being negotiated with ECN, 88% of which have the ecnseen flag.**

One downside of the Classic ECN[6] is that it's too prescriptive about the explicit congestion signal. Some RFCs, like the Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback[17], call out the low granularity of classic ECN that is only able to feed back one congestion signal per RTT. Also for a good reason: DCTCP (and BBRv2⁸) implementation of ECN greatly benefits from its increased accuracy[1].

Another RFC, namely Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation[3], tries to fix it by relaxing this requirement. That way implementations are free to choose behavior outside of the one specified by Classic ECN.

Talking about ECN its hard to not mention that there is also a congestion-notification conflict going over a single code

⁸Pay special attention to the CPU usage if you are testing BBR with ECN enabled since it may render GRO/GSO/TSO unusable for high packet loss scenarios.

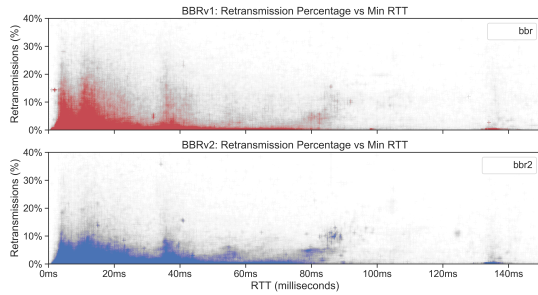


Figure 14: Retransmits vs RTT heatmap. BBRv1 vs BBRv2.

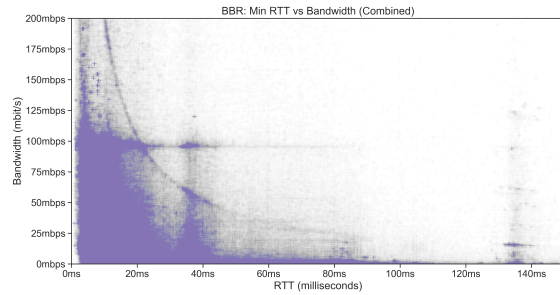


Figure 16: bbr_mrtt vs bbr_bw heatmap.

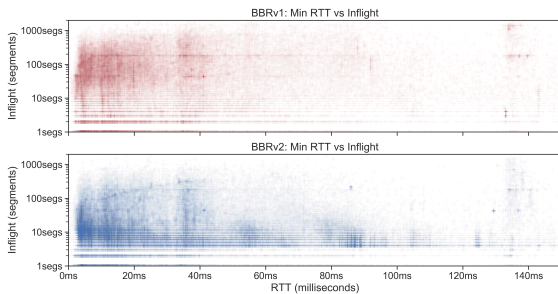


Figure 15: Unacked vs RTT heatmap. BBRv1 vs BBRv2.

point (a half a bit) of space in the IP header between the Low Latency, Low Loss, Scalable Throughput (L4S)[4] proposal and the bufferbloat folks behind the The Some Congestion Experienced ECN Codepoint (SCE)[18] draft.

As Jonathan Corbet summarizes it:

“These two proposals are clearly incompatible with each other; each places its own interpretation on the ECT (1) value and would be confused by the other. The SCE side argues that its use of that value is fully compatible with existing deployments, while the L4S proposal turns it over to private use by suitably anointed protocols that are not compatible with existing congestion-control algorithms. L4S proponents argue that the dual-queue architecture is necessary to achieve their latency objectives; SCE seems more focused on fixing the endpoints.”

Time will show which, if any, draft is approved by IETF, in the meantime, we can all help the Internet by deploying AQMs[2](e.g. fq-codel[10], cake[11]) to the network bottlenecks under our control.

Beyond As Fast As Possible

Evolving from AFAP Teaching NICs about time

There is a great talk by Van Jacobson about the evolution of computer networks and that sending as fast as possible is not an optimal strategy in today's Internet and even inside a datacenter[16].

This talk is a great summary of the reasons why one might consider using pacing on the network layer and a delay-based congestion control algorithm.

Fair Queue scheduler

All our Edge boxes are running Fair Queue qdisc. Our main goal is not the fair queuing itself but the pacing introduced by `sch_fq`⁹.

One can use `bpfttrace qdisc-fq.bt`¹⁰ to measure the time difference between packets being enqueued into the qdisc and dequeued from it, and hence the effect of pacing on the network.

Listing 2: `qdisc-fq.bt` output on a live system.

```
# bpfttrace qdisc-fq.bt
@us:
[0]          237486 |          |
[1]          8712205 |@@@@     |
[2, 4)      21855350 |@@@@@@@@@@@@@@ |
[4, 8)      4378933 |@@       |
[8, 16)     372762 |         |
[16, 32)    178145 |         |
[32, 64)    279016 |         |
[64, 128)   603899 |         |
[128, 256)  1115705 |         |
[256, 512)  2303138 |@        |
[512, 1K)   2702993 |@@       |
[1K, 2K)    11999127 |@@@@@@@@@ |
[2K, 4K)    5432353 |@@@      |
[4K, 8K)    1823173 |@        |
[8K, 16K)   778955 |         |
[16K, 32K)  385202 |         |
[32K, 64K)  146435 |         |
[64K, 128K) 31369 |         |
[128K, 256K) 2967 |         |
[256K, 512K) 271 |         |
```

In our tests, deploying FQ across the Dropbox internal network essentially eliminated the premium queue frame discards on shallow-buffered switches.

⁹Earlier fq implementations did add some jitter to TCPs RTT estimations which can be problematic inside the data center since it will likely inflate p99s of RPC requests. This was solved in “tcp: switch to Early Departure Time model.” and should be available since Linux 4.20.

¹⁰`qdisc-fq.bt` is a part of supplementary material to the “BPF Performance Tools: Linux System and Application Observability” book by Brendan Gregg[7].

References

- [1] Alizadeh, M.; Greenberg, A.; Maltz, D. A.; Padhye, J.; Patel, P.; Prabhakar, B.; Sengupta, S.; and Sridharan, M. 2010. Data center TCP (DCTCP). *ACM SIGCOMM Computer Communication Review* 40(4):63.
- [2] Baker, F., and Fairhurst, G. 2015. IETF Recommendations Regarding Active Queue Management. RFC 7567.
- [3] Black, D. L. 2018. Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation. RFC 8311.
- [4] Briscoe, B.; Schepper, K. D.; Bagnulo, M.; and White, G. 2019. Low Latency and Low Loss and Scalable Throughput (L4S) Internet Service: Architecture. Internet-Draft draft-ietf-tsvwg-l4s-arch-04, Internet Engineering Task Force. Work in Progress.
- [5] Cardwell, N.; Cheng, Y.; Gunn, C. S.; Yeganeh, S. H.; and Jacobson, V. 2017. BBR. *Communications of the ACM* 60(2):58–66.
- [6] Floyd, S.; Ramakrishnan, D. K. K.; and Black, D. L. 2001. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168.
- [7] Gregg, B. 2019. *BPF Performance Tools (Addison-Wesley Professional Computing Series)*. Addison-Wesley Professional.
- [8] Guba, O., and Ivanov, A. 2018. Dropbox traffic infrastructure: Edge network. Available at <https://blogs.dropbox.com/tech/2018/10/dropbox-traffic-infrastructure-edge-network/>.
- [9] Hock, M.; Bless, R.; and Zitterbart, M. 2017. Experimental evaluation of BBR congestion control. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE.
- [10] Hiland-Jrgensen, T.; McKenney, P.; Tht, D.; Gettys, J.; and Dumazet, E. 2018. The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm. RFC 8290.
- [11] Hiland-Jrgensen, T.; Tht, D.; and Morton, J. 2018. Piece of CAKE: A comprehensive queue management solution for home gateways. In *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE.
- [12] 2019a. *IETF ICCRG 104. BBR v2: A Model-based Congestion Control*.
- [13] 2019b. *IETF ICCRG 105. BBR v2: A Model-based Congestion Control*.
- [14] 2019c. *IETF ICCRG 106. BBR v2: A Model-based Congestion Control. Performance Optimizations*.
- [15] Ivanov, A. 2017. Optimizing web servers for high throughput and low latency. Available at <https://blogs.dropbox.com/tech/2017/09/optimizing-web-servers-for-high-throughput-and-low-latency/#congestion-control>.
- [16] Jacobson, V. 2018. Evolving from AFAP: Teaching NICs about time.
- [17] Khlewind, M.; Scheffenegger, R.; and Briscoe, B. 2015. Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback. RFC 7560.
- [18] Morton, J., and Grimes, R. 2019. The Some Congestion Experienced ECN Codepoint. Internet-Draft draft-morton-tsvwg-sce-01, Internet Engineering Task Force. Work in Progress.
- [19] Shirokov, N. 2020. Intelligent DNS based load balancing at Dropbox. Available at <https://blogs.dropbox.com/tech/2020/01/intelligent-dns-based-load-balancing-at-dropbox/>.
- [20] Stuart Cheshire and David Schinazi and and Christoph Paasch. 2017. Advances in networking. In *2017 WWDC The Apple Worldwide Developers Conference*. Apple.
- [21] Turkovic; Belma; Kuipers; A., F.; Uhlig; and Steve. 2019. Fifty Shades of Congestion Control: A Performance and Interactions Evaluation. *arXiv e-prints* arXiv:1903.03852.