Comparing BBR and CUBIC Congestion Controls

Rune Johan Borgli
University of Oslo
runejb@ifi.uio.no

Joakim Misund
University of Oslo
joakimmi@ifi.uio.no

1 INTRODUCTION

The goal of this technical report is to compare and evaluate the performance of CUBIC and BBR congestion controls. We will look at the following metrics: *Fairness*, *Packet loss rate*, *Throughput*, *Oscillations in throughput* and *Delay in round- trip-time*. The presented results can hopefully help NopBox choose their future congestion control. NopBox provides globe-spanning cloud backup and file-sharing.

NopBox wants their congestion control to have high throughput and be fair to traffic that it shares the network with. High throughput makes their product better. Fairness to competing traffic pleases their users who uses a variety of applications.

2 SYSTEM CONFIGURATION

The testbed consists of two virtual machines from Amazon, one in North Virginia and one in Ireland. They both run Ubuntu 16.04 with kernel version 4.9.7. The two machines can communicate using their public IP-addresses. The round-trip-time between them varies from 60 to 100ms, and the available bandwidth varies from 50Mbit/s to 100Mbit/s. We define the Ireland VM to be the receiver, and the North Virginia VM to be the sender. We create a virtual queue at the receiver side to be able to have control over the bottleneck. The virtual queue is implemented using an virtual interface, packet redirection and qdiscs. Packets arriving at the inbound interface, eth0, is redirected to a virtual interface, vb0. On the egress side of ifb0 we attached a hierarchical token bucket filter with a default class with 10 megabits per second. On this default class we appended a *pfifo queue* with the desired queue size. It is fair to assume that our introduced queue is the bottleneck because the observed capacity between the two virtual machines is much greater than that of the queue, but we will discuss this as an error source later on. We attach a FQ-qdisc at the sender side in experiments with one or more BBR flows.

We try to keep the experimental environment as close to reality as possible. Therefore we try to avoid changing too many parameters. We use the standard parameters in Linux for both CUBIC and BBR. Hybrid slow start only affects slow start, and does not make CUBIC more disruptive to existing flows. The beta value is 0.717. BBR has no parameters. We have not disabled *tso* and *gso*, which are HW optimisations for TCP. `tcp_no_metrics_saved` is enabled so that all flows start with the same inital state.

3 METHODOLOGY

We created a centralized Python script that runs on our own PCs. It takes care of running experiments and collecting data. It uses *tmux* on the sender and receiver for executng commands and starting flows. Using tmux enables us to be disconnected from the servers while the experiments are running. The advantage of this is that the scripts control traffic is not affecting the tests. When the experiments are done the tmux windows are killed, and the data downloaded locally.

We collect data using *tcpdump* on the sender and the receiver. Loss rate data is collected by downloading the output from *tc*.

For analysis we used a dedicated Python script which uses *tshark*, *tcpprobe*, and a selfmade transport-layer throughput calculation program implemented in C.

A different Python script takes care of plotting the data produced by the analyzer. The plotting- script uses *matplotlib's pyplot*. R is used for statistical plots.

We used *netcat* as workload generator. Initially we used *iperf3*, but we encountered some strange results and had to change. Netcat sends a continuous stream of zero-bytes.

We used tshark to generate the round-trip-time values that can be seen in figure 8. Tcpprobe was used to get the average transport-layer throughput used in fairness index calculations based on Jain's fairness index function. Throughput over time as seen in figure 1 is generated by a self-made program that takes a receiver-side *pcap* file. The step-time can be specified – we used 500ms.

4 ERROR SOURCES AND LIMITATIONS

Our testbed consists of two virtual machines from Amazon. There is one in North Virginia and one in Ireland. They communicate through the Internet, an environment we have no control over. In this section we will highlight some potential error sources and assumptions that we have made. We will also admit to mistakes and false assumptions we have discovered through the work.

Bottleneck queue size and packet size: We have chosen to specify the queue size in number of packets rather than number of bytes. We assumed that all packets from the sender to the receiver were of the same size. This proved to be false. The majority of the packets carried 2896 bytes of payload, but some packets carried only 1448 bytes. A consequence is that our queue size in bytes can not be accurately determined.

Limit on bottleneck size: We assumed that the receiver could handle arbitrary queue lengths. Some tests showed strange results with large queue sizes. Results involving large queue sizes can be affected by this potential limitation.

Variance in round-trip-time: Since we have no control over the Internet and its characteristics, the round- trip-time (RTT) changed from experiment to experiment. Therefore we measured the RTT again before each batch of experiments. We assumed that the RTT would remain fairly stable during the experiments. The RTT can be affected by cross-traffic along the path from the sender to the receiver.

Packet loss rate: The loss rate in the Internet is out of our control. It depends on the amount of cross traffic and capacity along the route. We measured that the path could handle 50-100 megabits per second. Since we limited our bottleneck to 10 megabits per second,

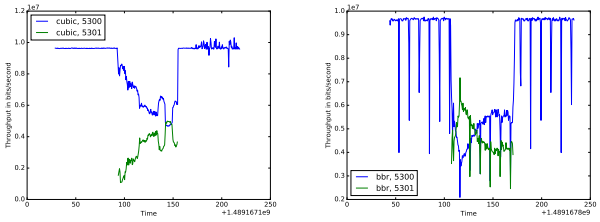

Figure 1: Shows how the throughput converges when a new flow joins an existing one. The queue size is set to 105 packets.

our traffic should not cause congestion in other hops on its own. Because cross-traffic can not be controlled, our experiments can contain variation due to the packet loss at hops along the path other than the created bottleneck.

Workload from the virtual machines: We used greedy flows as workloads for all experiments. The virtual machines that we used are subject to events such as context switches and migrations. This makes it impossible for us to guarantee that the machines are constantly sending data. Our workload could have been affected by activities related to virtual machine management.

Our queue is the bottleneck: The bottleneck is the hop along the path with the least available bandwidth. We assumed that out virtual queue was the bottleneck, but this was not guaranteed. However this was a fair assumption because the measured bandwidth between the sender and receiver was much larger than 10 megabits per second.

Effect of other hops/queues: To get a better understanding of the test-environment we ran a *tracepath* from the sender to the receiver. It gave us a total of 17 hops. Any of these hops can have variable cross-traffic.

5 RESULTS

In this section we will present our results from the conducted experiments. Each subsection will present and analyze one metric.

5.1 Fairness

We decided to look at fairness between flows in three different scenarios:

- All flows running CUBIC.
- All flows running BBR.
- A mix of flows running BBR and CUBIC.

We measured fairness using Jain's fairness index. It is defined as $f(x_1, x_2, ..., x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} (x_i)^2}$, where x_i is the acheived bandwidth of flow i and n is the number of flows. It gives a value between 0 and 1. Higher values indicates better fairness. We used the average transport-layer throughput as the acheived bandwidth.

In addition we analysed throughput graphs to identify how the algorithms behave separately and together. We will look at how fast they converge and how they share the capacity.

5.1.1 Getting up to speed. Figure 1 shows how CUBIC and BBR converges towards their fair share when a new flow joins a stable existing flow. The plots show the throughput measured at the receiver. The first flow is allowed to stabilize before the second flow joins.

For CUBIC it takes around 50 seconds before the flows get equal share of the throughput. For BBR it takes only a matter of seconds. It seems that BBR is much more aggressive than CUBIC is in the starting phase. BBR is much more disruptive to the existing flow. Which of the two approaches is most fair? CUBIC uses a long time to get to its fair share. This means that the joining flow gets less capacity than what it should for a long time. BBR on the other hand quickly reaches its fair share. One can say that BBR is more fair than CUBIC in the startup phase, because new flows get to their fair share more quickly.

We have not looked at loss rate in this experiment. A more disruptive startup phase usually means more packet loss. It would be interesting to see how the loss rate is affected by BBRs aggressive startup phase.

5.1.2 Fairness between flows running the same protocol. In figure 2 we have plotted Jain's fairnes index for four different combinations of congestion controls with varying queue sizes. In all the experiments the flows ran for 5 minutes, which allowed them to stabilize and run in steady state for enough time to provide statistical viable throughput data.

The blue line shows index values for experiments where two flows run CUBIC. The green line shows index values for experiments where two flows run BBR. CUBIC has good fairness for all queue sizes. BBRs fairness varies more as the queue size increases.

5.1.3 Fairness between flows running different protocols . In figure 2 the cyan line shows Jain's fairness index for experiments with one flow running CUBIC and one flow running BBR. The fairness rises as the queue size grows toward a certain value, and then it decreases as the queue size grows further. There is an "optimal" point for fairness between one CUBIC and BBR flow. The red line shows the fairness index for 2 flows running CUBIC and 2 flows running BBR. We can see that high fairness occurs at higher queue size.

Let us have a look at this "optimal" point. The majority of the packets in the experiment has a size of around 2900 bytes, which means that the queue size in bytes is roughly 55 packets * 2900 * 8 bits/packet = 1276000 bits. The RTT for the experiment was around 83ms, thus a BDP of 830000 bits. This gives us that the queue size is roughly 1276000 bits / 830000 bits = 1.537. The "optimal" point is just above 1.5 times the BDP.

Figure 3 shows the throughput graphs from the most extreme queue size values for one CUBIC flow and one BBR flow from figure 2. BBR is unfair to CUBIC when the queue size is low, but when the queue size is large CUBIC is unfair to BBR.

In figure 4 we have plotted the throughput of each flow in two experiments with different numbers of CUBIC flows and one BBR flow. They were run back-to-back and we assume that the conditions were the same. In the plot to the left we can see that the two flows are fairly fair to each other. The fairness index is 0.994. In the plot to the right we have increased the number of CUBIC flows. The BBR flow gets much more throughput than each of the CUBIC flows. It looks like the BBR flow gets roughly the same throughput in both plots. If that is the case the CUBIC flows only compete with each other. Our theory is that BBR has an operating range in the bottleneck queue where CUBIC can not operate. CUBIC flows effectivly use what is left of the queue after BBR has claimed its

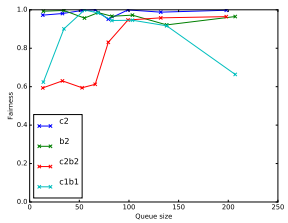

Figure 2: Jain's fairness index for varying queue sizes and competing congestion controls. Each flow is denoted by either 'c' for CUBIC or 'b' for BBR and a number saying how many flows of that type was used. The CUBIC flow is started before the BBR flow in each experiment. Each experiment ran for 5 minutes.

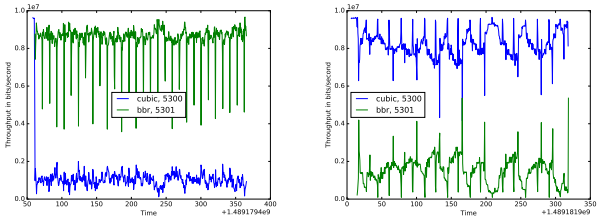

Figure 3: Plot of throughput for c1b1 with queue sizes 13 and 207 from figure 2. BBR overruns CUBIC when the queue is small, but CUBIC returns the favor when the queue is large.

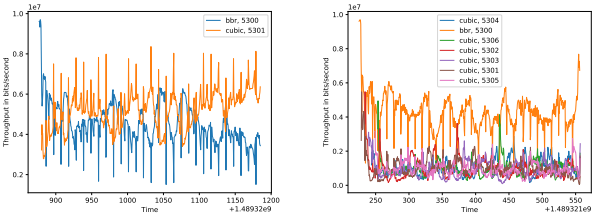

Figure 4: Shows how BBR overruns CUBIC as the number of CUBIC flows increases. Queue size is set to 52 packets. Fairness index is 0.994 for the first plot and 0.606 for the second plot.

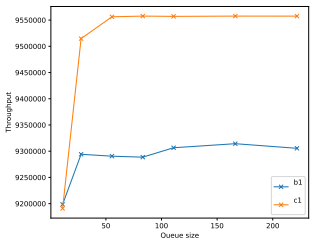

Figure 5: Transport-layer througput graph of single flow CUBIC and BBR with varying queue sizes.

part. New CUBIC flows results in larger reduction in throughput for existing CUBIC flows than for the existing BBR flow.

5.2 Throughput

In this section we will look at how CUBIC and BRR differs in terms of throughput.

5.2.1 Acheived Throughput. Figure 5 shows the measured transport-layer throughput with varying queue sizes. We can see that a single CUBIC flow acheives higher throughput than a single BBR flow. We have also looked at increased number of flows. The total througput increases with the number of flows until the bottleneck capacity.

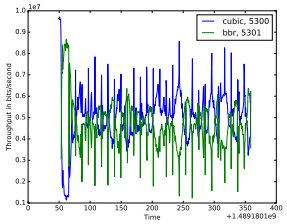

Figure 6: Plot of throughput for c1b1 with the fairest queue size from figure 2.

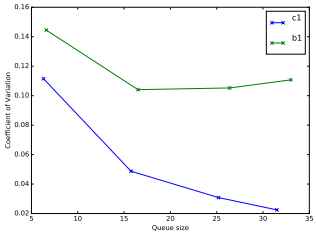

Figure 7: COV values for transport-layer throughput with single flow CUBIC and BBR with varying queue sizes.

5.2.2 Oscillation in Throughput. Oscillation in throughput is how throughput varies over time. We have looked at the Coefficient of Variation which is defined as the standard deviation divided by the mean.

Figure 7 shows the Coefficent of Variation of the throughput measured for one CUBIC flow and one BBR flow with varying queue sizes. We can see that BBR oscillates more than CUBIC for all the queue sizes. We think that the drainage phases of BBR causes this difference. We could have removed those samplings as insignificant outliers, but we think it would be incorrect. The drainage of the queue is part of BBR and should therefore be included in the calculation.

The plot on the left of figure 4 shows the throughput of one CUBIC flow and one BBR flow. They both get their fair share. We can however see that the throughput oscillates quite a lot. In figure 6 we have plotted the throughput of two flows. One running CUBIC, and the other running BBR. The queue size is at the "optimal" point. This figure clearly shows that when BBR and CUBIC have fair share their throughput can oscillate substantially.

5.3 Effect on round-trip-time

In figure 8 we have plotted the RTTs of two flows with different queue sizes running alone from North Virginia to Ireland. In the plot to the right the queue size is 63 packets, while on the left the queue size is 189. The RTTs are calculated by tshark using the sender-side tcpdump capture files. The outliers are probably due to packet loss, but they are not important for showing the tendency.

In the plot on the right the RTTs of the two flows are very similar. BBR and CUBIC introduces about the same delay to the base RTT. In the plot to the left the flow running CUBIC is far above the flow running BBR. There is a about a 200ms difference in RTT. The BBR flows RTT remains very stable with some drops due to the drainage of the queue. The CUBIC flows RTT follows it characteristic CUBIC function.

Figure 8 shows that CUBIC introduces more delay than BBR does when the queue size increases. CUBIC relies on loss as a signal. Therefore it has to cause the queue to drop packets by increasing

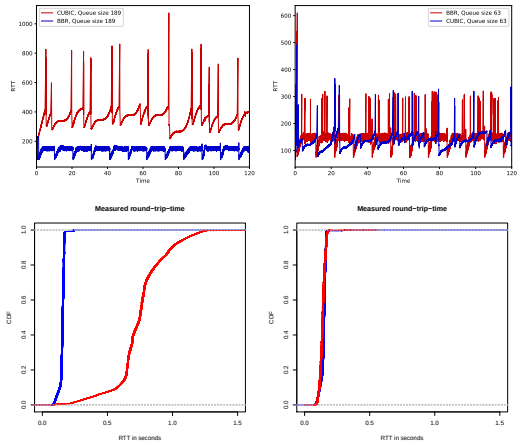

Figure 8: Measured RTTs for two flows from North virginia to Ireland with two different queue sizes. Base RTT is 76 milliseconds. The lower two plots are the CDFs of the RTTs.

its size. BBR on the other hand, does not rely on loss as a signal. It primarily uses time-measurements as a signal of congestion. This allows BBR to operate without filling the bottleneck queue.

From the CDF plots of the RRTs we can see that half of the RTTs for CUBIC is below 750ms, and half of the RTTs for BBR is below 180ms.

5.4 Packet Loss Rate

BBR does not use loss as the only signal for congestion. We wanted to see how much loss BBR caused compared to CUBIC.

In figure 9 we have plotted the total loss rate, recorded at the bottleneck, with varying bottleneck queue sizes. The blue line labeled b1 is BBR, and the orange line labeled c1 is CUBIC. The flows ran alone for five minutes, which gave them enough time to run for some time in steady state. We include loss in the startup phase because the two algorithms differ in this phase.

As we can clearly see the loss rate for BBR is much higher than that for CUBIC when the bottleneck queue size is lower than 100 packets. Once the queue size reaches a certain limit the loss rate for BBR goes down to zero. BBR avoids loss if the queue is large enough and there is no other traffic. Despite the high packet loss rate the average transport-layer throughput remains high as shown in figure 5. This is a plot of the average transport-layer throughput for the same tests as in figure 9.

The loss rate of the CUBIC flows increases with the queue size. This was a bit surprising because we thought that increased queue size would increase the time between congestion events which in turn would decrease the loss rate. We think it is the aggressive ramp up by the CUBIC function that causes this loss rate increase. The time between congestion events increases, but so does the severity of the events as well.

6 CONCLUSION

The goal of the technical report was to compare and evaluate the performance of CUBIC and BBR congestion controls, with emphasis on the following metrics: *Fairness*, *Packet loss rate*, *Throughput*,

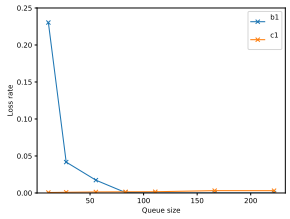

Figure 9: Total loss rate for single flow CUBIC and BBR with varying queue sizes.

Oscillations in throughput. We decided to include *delay in round-trip-time* because it shows an important property of the two congestion controls. Throughput and fairness is the two most important metrics for NopBox. High throughput allows NopBox's cloud backup and file-sharing services to be fast. NopBox traffic should be fair to other traffic because its users uses other applications aswell.

In section 5.1 we presented the obtained fairness results. We found that flows running BBR are fair to other flows running BBR and that flows running CUBIC are fair to other flows running CUBIC. When flows run different congestion controls the fairness depends on the queue size and the number of flows. We found that the queue size has to be roughly 1.5 times the bandwidth-delay-product for one BBR flow to be fair to one CUBIC flow. This is not the true with increased number of flows. We also saw that one BBR flow gets increasingly unfair as the number of CUBIC flows increases when the queue size is 1.5 times the bandwidth-delay-product.

The throughput, discussed in section 5.2, differs only slightly. CUBIC acheives higher average throughput than BBR and has less oscillation in throughput. When we tested one CUBIC flow against one BBR flow we observed very much oscillation in throughput.

One of the benefits of using BBR is that it does not have to introduce loss as discussed in section 5.4. When the queue size comes to a certain length BBR can operate without causing loss. This also reduces the introduced delay as discussed in section 5.3. These two benefits requires the queue size to be very large, which is bad for CUBIC which will experience much delay as discussed in section 5.3.

So what should NopBox do? Considering acheived throughput alone there is no need to change from CUBIC to BBR. BBR gets faster up to its fair share when the queue size is large, which is important for short flows. In NopBox's case this benefit should not matter. Considering fairness, BBRs interaction with CUBIC depends on the number of CUBIC flows and the bottleneck queue size. The number of CUBIC flows can be many and we have seen that only one BBR flow can make their throughput plunge. The users of NopBox are probably fairly distributed around the Internet. They have different paths, bottleneck queue sizes, and RTTs. NopBox can not control these variables.

Based on the findings in this technical report we would advice NopBox to stay with CUBIC until BBR fixes its fairness issues when competing with CUBIC.