Exploring the BBRv2 Congestion Control Algorithm for use on Data Transfer Nodes

Brian Tierney, Eli Dart, Ezra Kissel
Lawrence Berkeley National Laboratory
Berkeley, CA, USA
Email: {bltierney,dart,kissel}@es.net

Eashan Adhikarla
Lehigh University
Bethlehem, PA, USA
Email: eaa418@lehigh.edu

Abstract—**It is well known that loss-based TCP congestion control algorithms are problematic for high-speed high-latency flows that are common in Big Science. In 2016 Google released a new congestion control algorithm called 'BBR' (Bottleneck Bandwidth and Round-trip time) that uses a model-based approach, and the design has since been refined in an alpha release of BBRv2. In this paper, we describe and perform a set of experiments that assess the suitability of BBRv2 for use on Data Transfer Nodes (DTNs). The experiments were run on both production R&E networks as well as within a controlled testbed environment. Our analysis of the results show that BBRv2 improves upon BBRv1 for common Big Science transfer scenarios and is a promising option in high-speed short-queue networking environments.**

I. Introduction

A Science DMZ [1] is a portion of a network built at or near a campus local network perimeter that is designed such that the equipment, configuration, and security policies are optimized for high-performance workflows and large data sets. High-performance servers, called Data Transfer Nodes (DTNs) are connected directly to the Science DMZ. The DTNs handle all data transfers, and need to be tuned for maximum network throughput. DTNs typically run parallel flow data transfer tools to overcome the limitation of loss-based congestion control algorithms.

In this paper we evaluate the use of Google's TCP congestion control algorithm, known as BBR [2], [3] (Bottleneck Bandwidth and Round-trip time), for use on high-performance DTNs. The results shown in this paper are from tests run on the ESnet Testbed, described below, and on production networks using a variety of public perfSONAR [4] hosts. We show that BBR does much better than CUBIC on lossy paths, and the higher the loss rate and round-trip time (RTT), the more BBR outperforms CUBIC. We confirm that BBR prefers smaller switch buffers, and CUBIC prefers larger buffers. We show that the BBRv1 retransmit rate is unacceptably high with parallel flows, and thus BBRv1 should not be used with parallel data transfer applications.[1] Finally, we confirm the results from previous papers on BBR that are primarily based on evaluations using Mininet [5] environments.

[1]To avoid confusion, we use the following nomenclature in this paper: *BBRv1* is BBR version 1, *BBRv2* is BBR version 2, and *BBR* is used when the concept applies to both BBRv1 and BBRv2.

II. Overview and Goals

The main goal of this work is to evaluate whether or not the BBR congestion control algorithm is appropriate for use on DTNs in a Science DMZ environment. The DTN network characteristics we are interested in include:

- High-speed hosts (often 40G or 100G), sending to slower hosts (often 10G)
- A large number of parallel flows (up to 16)
- High-latency paths, often 100ms RTT or higher

Most of the previous papers comparing BBR to CUBIC, described in the related work section below, are focused on single flow results, and paths where the sender and receiver are the same speed. Most of these papers also use Mininet rather than real networks.

Historically, the R&E community has deployed network devices with deep buffers in a Science DMZ. This is to help CUBIC avoid packet loss. A rule of thumb was to try for twice as much buffering as the BDP (Bandwidth Delay Product) of the largest data transfers. The problem with this approach is that it does not scale well to network speeds of 100G, 400G, and beyond. BBR was designed in part to eliminate the dependence on large buffers, and one of our goals is to understand if BBR eliminates the need for deep buffers in the DTN use case.

The contributions of this paper include verifying that results from previous papers using results collected from Mininet are valid on real networks, and are valid for large numbers of parallel flows and mismatched host speeds as well. We show that not only are previous results valid, but in some cases the DTN use case shows an even greater advantage for BBRv2, as parallel flows and speed mismatch lead to greater packet loss, which BBR handles much better than CUBIC.

BBRv2 Background

TCP Congestion Control Algorithms fall into 2 general categories: loss-based (e.g.: Reno[6] and CUBIC[7]), where the sender slows down if loss is detected, and delay-based (e.g.: Vegas[8] and Fast[9], where the sender slows down if additional delay is detected. The Internet has largely used loss-based congestion control algorithms, which assume that packet loss is equivalent to congestion.

However, equating packet loss with congestion is problematic. In particular, when TCP's instantaneous rate is higher than its average rate (which is TCP's normal bursty behavior), a loss-based congestion control algorithm can dramatically reduce TCP's performance because of microburst congestion. This is more likely if the path contains routers or switches with shallow buffers. Using routers and switches with deep buffers can improve the performance of loss-based congestion control, but deep-buffered devices cost more and increase the risk of buffer bloat.

The BBR congestion control algorithm takes a different approach, and does not assume that packet loss signals congestion. BBR builds a model of the network path in order to avoid and respond to actual congestion. BBR uses pacing to set the sending rate to the estimated bottleneck bandwidth. The pacing technique spaces out or paces packets at the sender node, spreading them over time. This approach is a departure from the traditional loss-based algorithms, where the sending rate is established by the size of the congestion window, and the sender node may send packets in bursts, up to the maximum rate of the sender's interface. Thus, traditional algorithms rely on routers to perform buffering to absorb packet bursts.

While BBRv1 has shown dramatic improvements over CUBIC, several papers [10], [11] have reported issues with BBRv1 such as CUBIC unfairness and high retransmission rates. In 2019 a new alpha version of BBR, version 2 (BBRv2) [12] was released, which strives to address these limitations. BBRv2 uses two estimates to establish the sending rate of a connection: the bottleneck bandwidth and the RTT of the connection, and adapts bandwidth probing for better coexistence with Reno/CUBIC. BBRv2 also incorporates Explicit Congestion Notification (ECN) and better estimates the packet loss rate to establish the sending rate.

III. Related Work

Since BBR was released in 2016, a number of papers have been published analyzing how BBR behaves in a number of environments.

Scholz et al. [11] perform an analysis using Mininet of BBR inter-flow unfairness and inter-protocol fairness when competing with TCP CUBIC flows, and find that in most cases, BBR and CUBIC do not share bandwidth fairly. They also confirm that the bottleneck buffer size is crucial for the fairness between competing BBR and CUBIC flows. They show that with a buffer size of up to 1.5 times bandwidth delay product (BDP), BBR causes constant packet loss. CUBIC interprets this as a congestion signal and reduces its sending rate. With a buffer size of 3 times the BDP, both types of flows shared the path equally, and further increasing buffer size, CUBIC steadily claims more of the bandwidth. They state that this is due to the fact that CUBIC fills up the ever-growing buffers.

Cao et al. [10] use Mininet to show that BBRv1 performs much better than CUBIC in a shallow buffer environment, but has a much higher (200x) retransmit rate. Other papers showing similar results include [13] and [14], which used a DPDK-based software switch. The paper with goals most similar to this paper is [15], which analyzes BBRv1 with parallel flows, and shows that BBR is much better than CUBIC with parallel flows as well as single flows.

More recent work has also begun exploring BBRv2. Kfoury et al. [16] also used Mininet to show that BBRv2 is indeed an improvement over BBRv1, with fewer retransmissions and greater fairness with CUBIC. They also show that pacing improves fairness, and that drop tail queues should be avoided. Zhang confirms similar BBRv2 improvements using ns3 [17]. Song et al. [18] have a newly published paper based on Mininet results showing similar behavior, and also found that when identical BBRv2 flows enter a bottleneck link with large buffers at different times, their achievable transfer rates do not converge.

To the best of our knowledge, this is the first paper that addresses the practical use of BBR/BBRv2 on a Data Transfer Node (DTN). It is also one of the few papers to include results from a real-world, production network.

IV. Testing Tools and Environment

The results in this paper include data collected on the ESnet testbed [19], and data collected from an ESnet perfSONAR[20] host in Boston, MA, USA. We used perfSONAR's scheduler to avoid test collisions. As BBRv2 only requires sender side modifications, we made use of our perfSONAR host with BBRv2 support to run tests to public perfSONAR hosts all around the world.

We modified *iperf3* [21] to run odd numbered flows using the default congestion control for the host, and even numbered flows using the congestion control specified using the iperf3 *"-C"* flag. This allows us to run parallel flows of both cubic and BBRv2 within the same iperf3 test. We experimented with giving some iperf3 flows a one second head start, to see if performance varied with new BBRv2 flows encounter existing cubic flows, but found no performance difference. The default iperf3 behaviour is thus used for all results.

All sending hosts were running the Ubuntu Operating System with a BBRv2 patched version 5.10.0 kernel. Host tuning followed guidelines from *http://fasterdata.es.net*, including the recommendation that hosts be configured with 9000 byte MTUs. We use TCP buffer autotuning for all tests, and the maximum TCP buffer autotuning buffer is set to 512MB. We also set *net.core.default_qdisc* to *fq*, as recommended for BBRv1 and BBRv2.

As perfSONAR does not support the version of Ubuntu used on our testing hosts, we run the perfSONAR 'Testpoint' docker container [22]. We modified this container to run our customized version of iperf3, and updated versions of *ss* and *systat*. This container is available on Docker Hub [23].

Many DTNs run Globus GridFTP [24] and are configured to use 16 TCP flows in their default configurations for data transfers [25]. Another popular data transfer tool for Science DMZs is *FDT* [26], which also uses parallel flows. To represent this common scenario, our parallel TCP testing made use of 16 parallel flows. For single flow tests we use the *'fq'* packet scheduler to pace all flows to 9.9 Gbps, and for 16 flow tests we shape the flows to 2.4 Gbps. Our INDIS 2016 talk

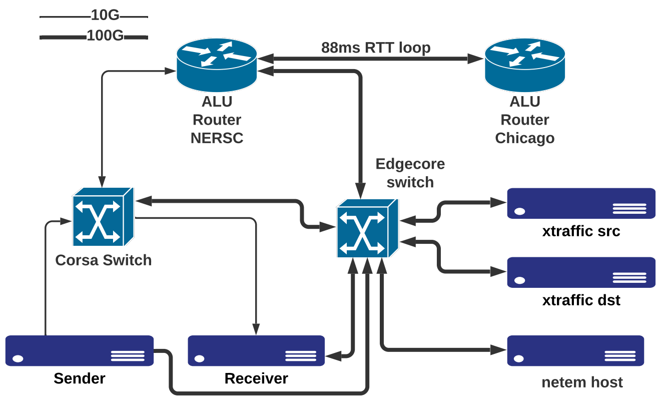


Fig. 1: ESnet Testbed

[27] presented pacing test results and justified the recommend default pacing rate of 2.4 Gbps. As the source and/or sink of many scientific data transfers are still 10G hosts, and DTNs will often be servicing multiple DTN transfer requests at one time, the chances that any single flow would ever get above 2.4 Gbps is low, and this pacing value is likely to improve overall throughput in most circumstances.

Our test host on a production network, *bost-dtn.es.net*, was connected to ESnet at 40G. Since packet bursts over 10G will lead to packet loss at the receive host, we pace single flow tests to 9.9 Gbps. We note that any parallel flow tests will exceed 10G given a 40G physical interface unless the sum of all flows is appropriately paced to match a 10G destination. We chose to run five minute tests on the testbed, which is long enough to determine flow stability and to observe flow dynamics in the plots. For the tests using public perfSONAR hosts, we were only able to run one minute tests, as that the is default test time limit in perfSONAR.

In order to collect data for this paper and ensure consistent test execution, we developed a testing harness in Python. It supports a number of features, including: (a) Enabling native instrumentation through *tcpdump* (including filters) and/or *ss* for each test, (b) Execution of arbitrary commands before and after a test, (e.g. to set MTU size or TCP parameters), (c) Invocation of external scripts to control settings such as network emulation parameters, (d) Parameter sweeps over a range of values, for example emulated latencies and queue sizes and various BBRv2 parameter settings, and (e) Configuration of the *fq* pacing rate for each test. Finally, the harness also supports exporting a subset of test metadata and instrumentation results to an external measurement archive stack (ELK). A sample configuration file for the test harness is shown in Listing 1 below. The test harness implementation is open source and publicly available at *https://github.com/esnet/testing-harness*.

ESnet Testbed Configuration

Figure 1 shows the components of the ESnet Testbed used for the experiments described in this paper. All testbed source and destination hosts used in this work had 24 cores of type Intel Xeon CPU E5-2643 v3 in a dual-socket configuration while the netem host had dual-socket AMD EPYC 7451 24-core processors. Our testing made use of 10G NICs for the sender and receiver nodes and 100G NICs were used for the netem and cross-traffic nodes. A Corsa model DP2400 switch is used for buffer size experiments, and has a default buffer size of 100MB. The Edgecore model AS9716-32d switch is used to interconnect all the hosts and 10G switch, and has a maximum shared buffer size of 64MB.

One of the useful features of the ESnet testbed is that it includes both a real, high latency path, and a host configured to add latency using *netem* [28]. This allowed us to compare real vs emulated networks, and confirm that the results agree. All results in this paper use *netem* configured on a single port of the netem host NIC, which impacts egress traffic on the full duplex link. Our *netem* configuration set a buffer limit value of 10000 (packets) for each test. Looking at results from the ESnet perfSONAR dashboard [29], we commonly see a range of packet loss on various paths ranging from 0 to 0.2%, with occasional path with up to 2% loss. For this paper, we chose to run tests at 0.1%, 0.01%, and 0.001% loss.

We always ran 10 iterations of each test and computed the mean and variation for each result. When plotting flows, we select a representative example from the 10 iterations, so that fine-grained flow behavior is visible in the plot. To calculate observed loss rates, the test harness uses the Unix tool *ss* [30] to count the number of data segments and retransmitted segments for each flow, sampled at 0.5 second intervals.

We confirmed that our *netem* path accurately represents a real path by setting the netem latency to 44ms (88ms RTT), to match the latency of the ESnet testbed loop. We ran a set of tests over both the netem path and the real path. Results in both cases were essentially identical, which provides a high degree of confidence that the *netem* environment does not introduce any unexpected flow behavior in the testbed.

V. Results

All the figures in this paper have the same components. Time is on the x-axis, and throughput and retransmits are both on the Y-axis. Mean throughput in the plot is the mean for just this plot, not the full set of 10 tests. Counts of packet retransmissions and data segments shown in the plot are collected from the output of the *ss* tool run at 0.5 second intervals. Plots using *netem* will have the *netem* settings in the lower right corner. Plots labeled *non-overlapped* at the top are from separate CUBIC and BBR tests, overlayed on the same plot. Plots labeled *overlapped* are where CUBIC and BBR flows are running at the same time.

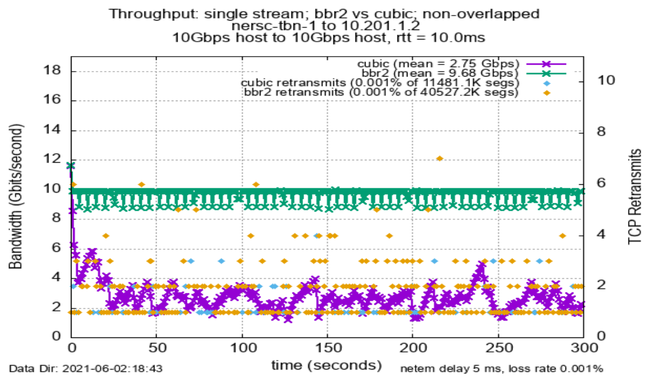
A. Comparison and competition between CUBIC and BBRv2

The goal of this experiment is to compare CUBIC with BBRv2 throughput in the presence of varying amounts of packet loss and latency configured in our testbed environment, and to determine how much the presence of BBRv2 flows reduce the throughput of CUBIC flows.

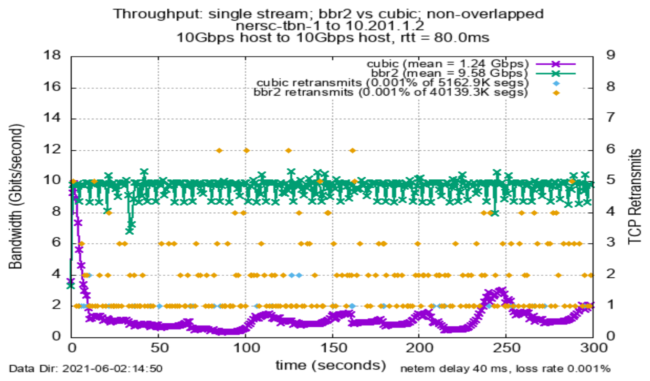
```
type = perfSONAR
enabled = true
iterations = 10
src = localhost
dst = 10.201.1.2
src-cmd = pscheduler task --format json throughput --congestion=bbr2 --ip-version 4 --parallel 16
↪  --duration PT5M --dest {dst}
# run this before each test command on the source host
pre-src-cmd = /usr/sbin/sysctl -w net.ipv4.tcp_congestion_control=bbr2
post-src-cmd = /usr/sbin/sysctl -w net.ipv4.tcp_congestion_control=cubic
instrument = true
tcpdump-filt = -s 128 -i ens2np0 "host {dst} and port 5201"
netem-loss = 0.001
lat-sweep = 2,5,10,20,30,50
pacing = 2.4gbit
```

Listing 1: A test harness sample configuration entry for a perfSONAR throughput sweep.

Figure 2 shows single CUBIC and BBRv2 flows with *netem* loss set to 0.001% packet loss. The upper plot has a path round trip time (RTT) of 10ms, and the lower plot has a 80ms RTT. As expected, BBRv2 throughput is close to line rate in the presence of loss, while CUBIC throughput drops considerably, even with this very low packet loss rate.



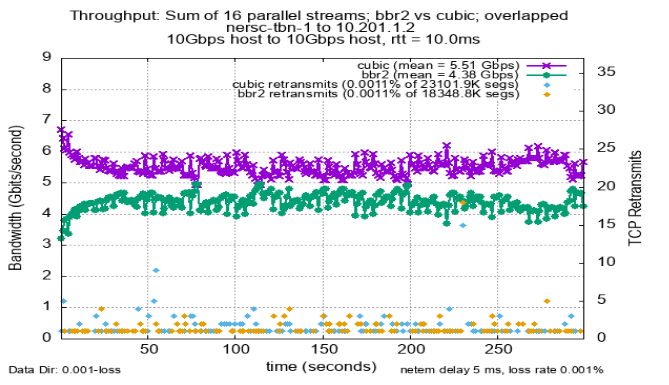
(a) 10ms RTT, BBRv2 and CUBIC



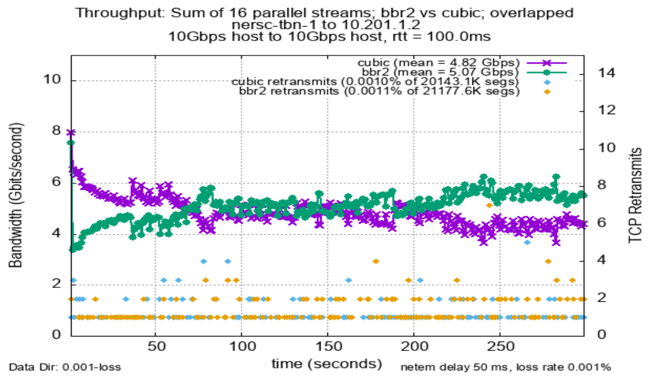
(b) 80ms RTT, BBRv2 and CUBIC

Fig. 2: single flow results: BBRv2 does much better than CUBIC on paths with 0.001% packet loss

Next we look at results for 16 parallel flows, with the same loss rate of 0.001%, shown in Figure 3. Note that with 16 flows and this relatively low loss rate, CUBIC and BBRv2 have very similar throughput, even with high latency.



(a) 10ms RTT, BBRv2 and CUBIC, 16 flows, loss = 0.001%



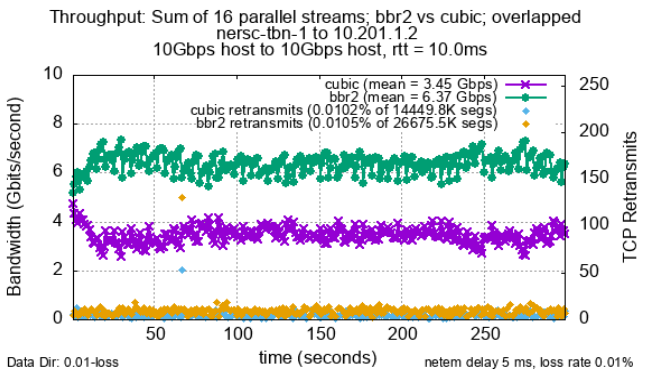
(b) 100ms RTT, BBRv2 and CUBIC, 16 flows, loss = 0.001%

Fig. 3: 16 flow results: with only a small amount of loss BBRv2 and CUBIC throughput are similar.

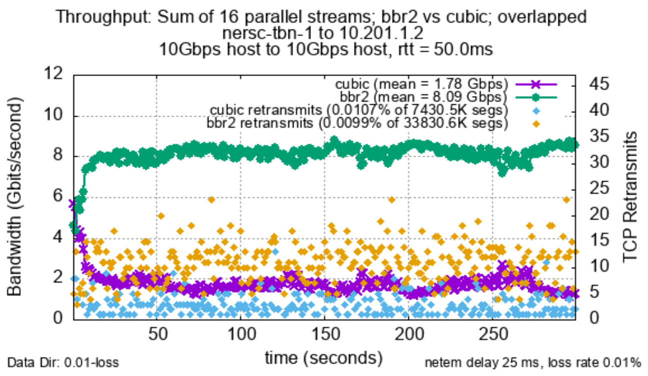
We next compare competing CUBIC and BBRv2 with higher packet loss rates of 0.01% loss (Figure 4), and a much higher packet loss rate of 0.1% (Figure 5). As expected, we see that at with higher packet loss rates BBRv2 is a clear winner, and the BBRv2 advantage increases as the RTT increases.

100G to 10G Testing

As the ESnet testbed hosts have both 100G and 10G NICs, we want to see how BBRv2 performs in the scenario of a 100G DTN sending to a 10G client over the 88ms testbed loop.

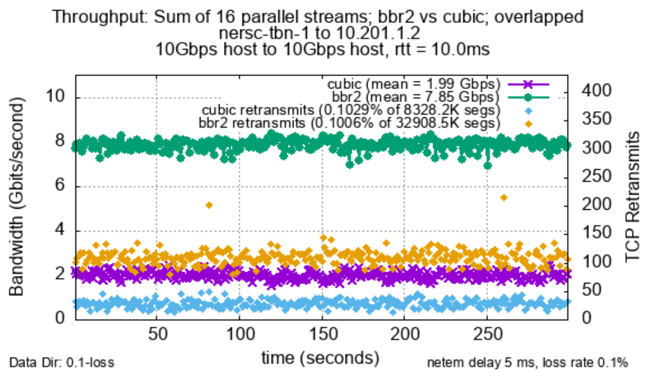


(a) 10ms RTT, BBRv2 and CUBIC, 16 flows, loss = 0.01%

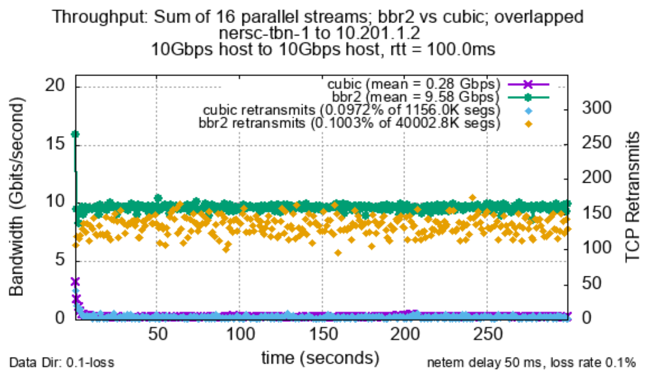


(b) 100ms RTT, BBRv2 and CUBIC, 16 flows, loss = 0.01%

Fig. 4: 16 flow results: BBRv2 does much better than CUBIC with 0.01% loss.



(a) 10ms RTT, BBRv2 and CUBIC, 16 flows, loss = 0.1%



(b) 100ms RTT, BBRv2 and CUBIC, 16 flows, loss = 0.1%

Fig. 5: 16 flow results: BBRv2 does even better yet with 0.1% loss.

Figure 6a shows consecutive CUBIC and BBRv2 16 flow tests. Note that the retransmit rate of BBRv2 is roughly 20X that of CUBIC, but both methods have no trouble approaching 10 Gbps. Figure 6b shows overlapping CUBIC and BBRv2 flows. Here BBRv2 is 20X faster than CUBIC, and actually has a lower rate of retransmits.

B. Buffer Size Testing

This experiment aims to compare the impact of small buffered switches on BBRv2 compared to CUBIC. The Corsa DP2400 switch in our testbed provided 10G connectivity between the source and destination hosts and the router providing the 88ms loop service. Applied to the three 10G Corsa ports (Fig. 1), we created a queue profile that allowed us to change the size of the available buffer on each. For each buffer size test, we ran a background 1 Gbps UDP flow between two other hosts on the testbed across the same path to create congestion. UDP was used to ensure a constant rate of background traffic.

Results setting the port buffer size to 64MB, and a much smaller 12MB are shown in Figure 7. As predicted by the Mininet results in papers [10] and [16], BBRv2 does much better with small buffers, and CUBIC does better with large buffers.

We also ran tests with buffer sizes of 8MB, 16MB, and 32MB, similar to a range of lower cost network devices on the market [31]. As the buffer size decreases, BBRv2 throughput increases, and CUBIC throughput decreases, with BBRv2 and CUBIC having similar throughput with 32MB buffers. Results are summarized in Table I.

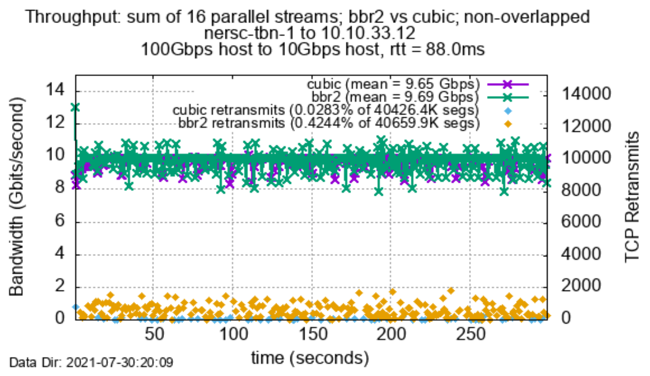
We also ran tests where all 16 flows were CUBIC, and all 16 flows were BBRv2. For these tests buffer size had no impact, and achievable aggregate throughput was always 8.85 Gbps, due to the 1 Gbps background UDP traffic.

TABLE I: PORT BUFFER SIZE TEST RESULTS (10G)

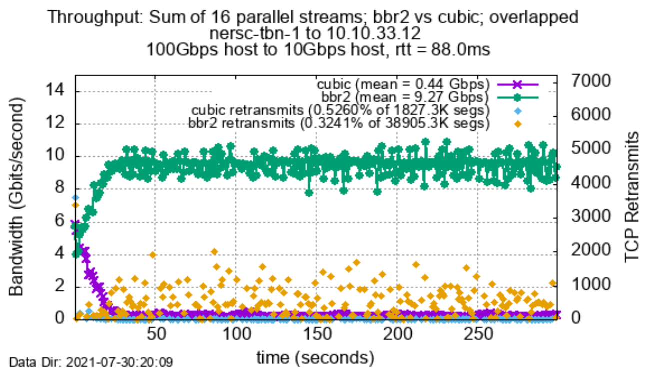
Buffer Size	CUBIC throughput	BBRv2 throughput
8 MB	0.4 Gbps	8.3 Gbps
12 MB	0.9 Gbps	8.0 Gbps
16 MB	1.8 Gbps	6.9 Gbps
32 MB	4.5 Gbps	4.3 Gbps
64 MB	4.6 Gbps	4.2 Gbps

C. Real World Testing

We next compare results obtained on the ESnet testbed with what we see on the Internet using data collected from tests to public perfSONAR hosts to confirm that the performance patterns we see on the testbed agree with those seen on the R&E networks. Most perfSONAR hosts are configured to allow only 1-minute tests. However, ESnet hosts allow for longer tests from another ESnet host, and we were able to

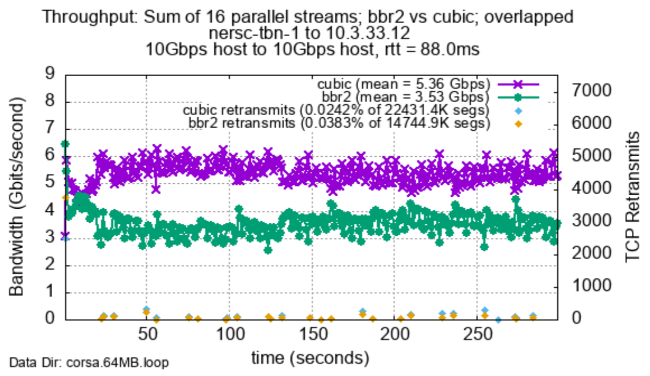


(a) BBRv2 and CUBIC results, consecutive test runs

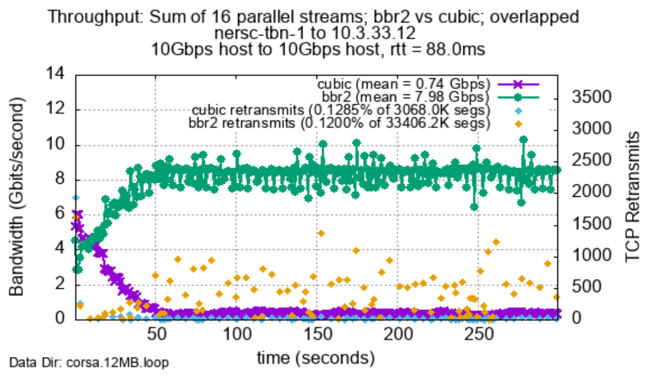


(b) BBRv2 and CUBIC results, overlapping flows

Fig. 6: 100G sender to 10G receiver



(a) 64MB switch buffer size (deep buffers)



(b) 12MB switch buffer size (shallow buffers)

Fig. 7: Switch buffer size test results: BBRv2 does much better with small buffers

contact colleagues at the University of Cambridge in the UK and have them temporarily allow us to run 5-minute tests to their perfSONAR host.

We first ran a set of tests between the 40G ESnet test host *bost-dtn.es.net* and ten different ESnet perfSONAR hosts across the network with RTTs ranging from 1ms to 87ms. In all cases, the network path was a 40G sender across 100G backbone to 10G receive hosts. A 40G host was chosen as many large R&E data centers have recently deployed 40G and 100G DTNs. We ran two sets of tests: one where the sender paced all of the 16 flows to 620 Mbps each, for a total of less than 10G, and the other where the sender was paced to 2.4 Gbps per flow, or 38.4 Gbps total for 16 flows, which will oversubscribe the 10G receive host.

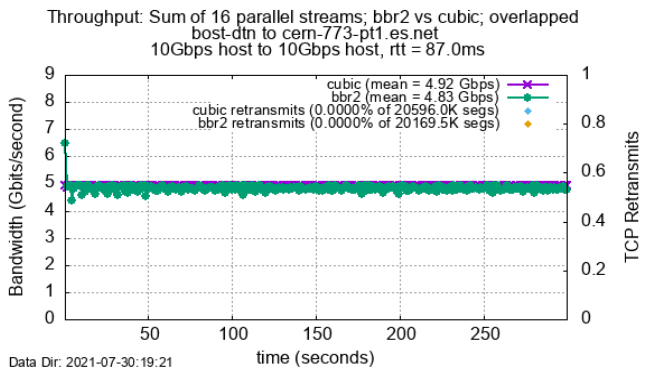
Sample results for both a 10G sender and a 40G sender over ESnet paths are shown in Figure 8. Note that there is no packet loss with a 10G sender, but 0.05% loss with a 40G sender. Also, BBRv2 is almost four times faster than CUBIC with the 40G sender, but exactly the same as CUBIC with the 10G sender due to the lack of packet loss.

In all test cases, if our sending host sent at a rate less than 10 Gbps there was close to zero packet loss. In tests of both 10G and 40G senders to a 10G receiver, CUBIC and BBRv2 achieved full line rate when run in isolation; however, with simultaneous CUBIC and BBRv2 we see some interesting loss patterns with a 40G sender.

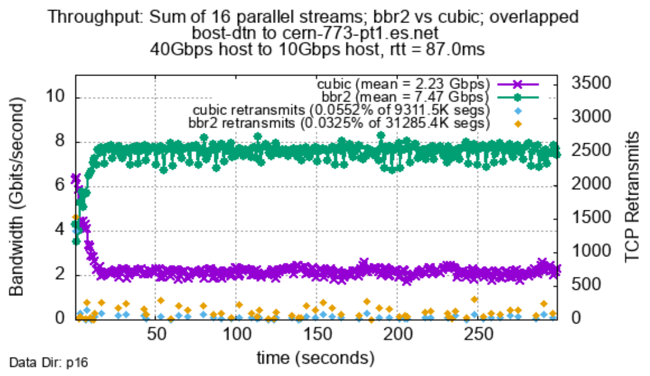
One surprising result was for a short RTT path, 16 flows, and a sending host faster than the receiving host, CUBIC performs roughly four times better than BBRv2, as shown in Figure 9. Based on our buffer size results described above, we expected CUBIC to be faster on this short, deep buffer path. The 40G to 10G speed mismatch appears to amplify this difference. Note that despite the speed mismatch between sender and receiver, packet loss rates are quite low. We also ran 4, 8 and 12 flow tests, and found that BBRv2 and CUBIC throughput is very similar up to 8 flows, but CUBIC is faster with 12 and 16 flows on this path.

Another pattern we often see for slightly longer RTTs is that CUBIC and BBRv2 start out sharing the link equally, but over time CUBIC gets a larger and larger share of the pipe, as shown in Figure 10. We believe this is due to BBRv2 probing and fairness mechanisms interacting with parallel flows in short-RTT, deep buffer environments, allowing loss-based algorithms to gain an advantage up to some RTT threshold.

For paths with an RTT greater than 30ms, we see about 10 times more retransmits than on shorter paths. Figure 11 depicts an example of longer retransmits. This is likely due to the fact that as the path gets longer, the bursts of packets coming from the 40G sender become larger. These packet bursts can be seen in this (and other) plots as throughput spikes of greater than



(a) 10G sender (620 Mbps pacing/flow)



(b) 40G Sender (2.4 Gbps pacing/flow)

Fig. 8: Comparison of 10G to 10G vs. 40G to 10G

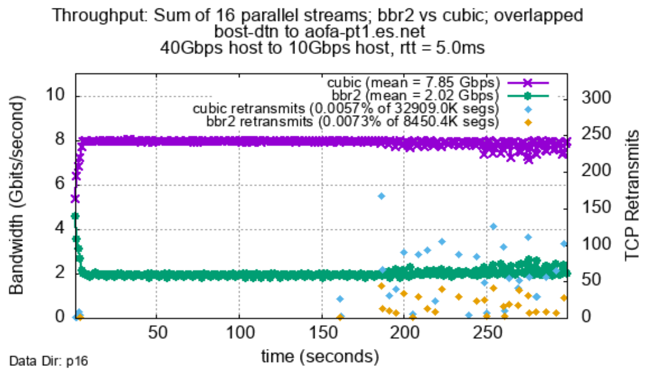


Fig. 9: 5ms RTT, low packet loss, CUBIC is considerably faster

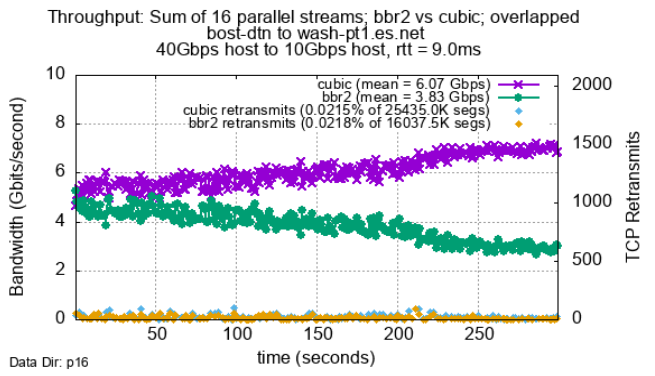


Fig. 10: 9ms RTT, CUBIC and BBRv2 start out similar, then CUBIC gains a larger share of the pipe over time

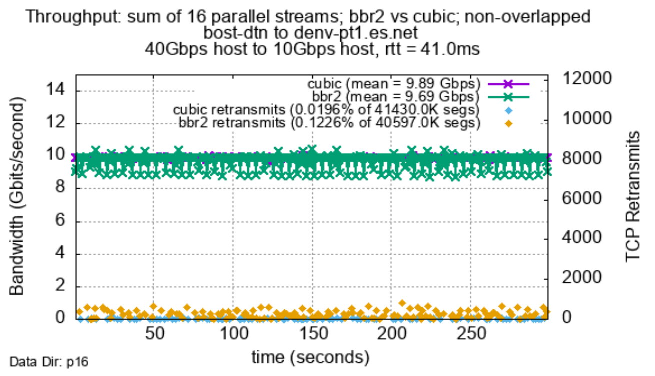


Fig. 11: 41ms RTT, similar performance, but BBRv2 has roughly 6 times more retransmits

10 Gbps, as measured by iperf3 sender. Note that in this plot, the CUBIC and BBRv2 tests are separate, not overlapped as in the other plots.

We conducted several experiments to compare 8 flows (4 CUBIC, 4 BBRv2) versus 16 flows (8 CUBIC, 8 BBRv2). Results are shown in Figure 12. Note that CUBIC achieves more than twice as much bandwidth with twice as many flows.

We next look at some known lossy paths, where even single flow tests show packet loss. Sample results are shown in Figure 13. In both Figures 13a and 13b we see it take quite a while for flow performance to stabilize. CUBIC and BBRv2 throughput start out similar, but over time BBRv2 gets faster and faster as loss events cause CUBIC to back off. These plots are considerably different from what we see on the testbed, where CUBIC and BBRv2 stabilize quite quickly. This is likely due to packet loss being more bursty on production networks.

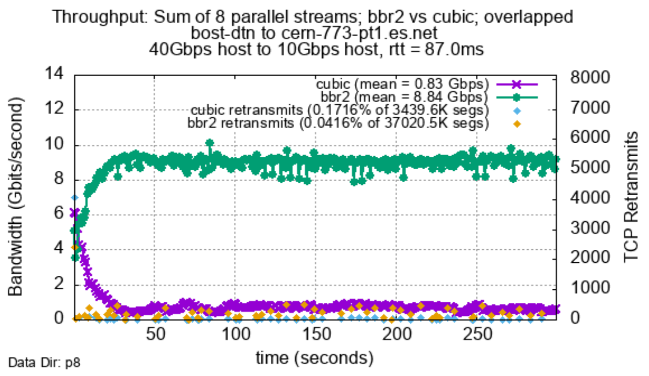
Comparison of Testbed Results to Real World Results

We see very similar patterns in the testbed results and results from the production R&E network paths, despite the variance in the results caused by background traffic on production paths. In both cases, for 16 flows over high latency paths with packet loss around 0.01%, BBRv2 is around 4 times faster (Figures 4b and 8b).

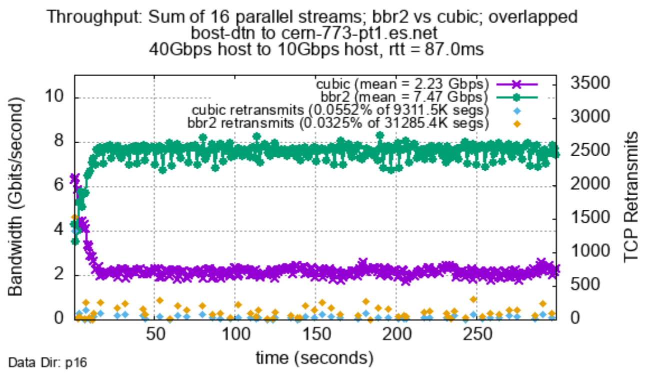
The only case where results are consistently different between testbed results and ESnet results is for paths with short RRT and low packet loss (Figure 4a compared to Figure 9). On the testbed we see BBRv2 is about 15% faster, while over ESnet paths, we see CUBIC is 4 times faster. This is likely due to the fact that the routers on ESnet path have much larger buffers.

D. BBRv1 vs BBRv2

As mentioned in the related work section above, other papers that have already compared BBRv2 to BBRv1, and con-

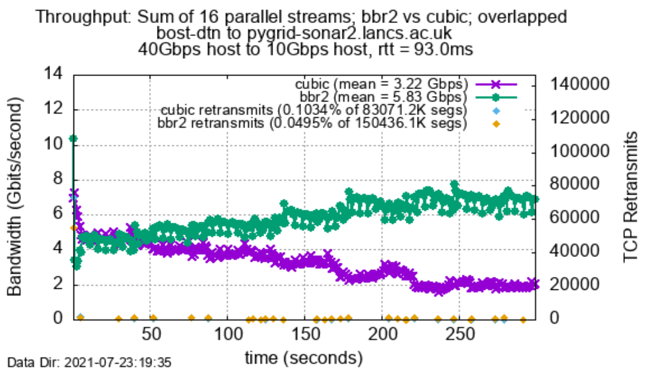


(a) 8 flows

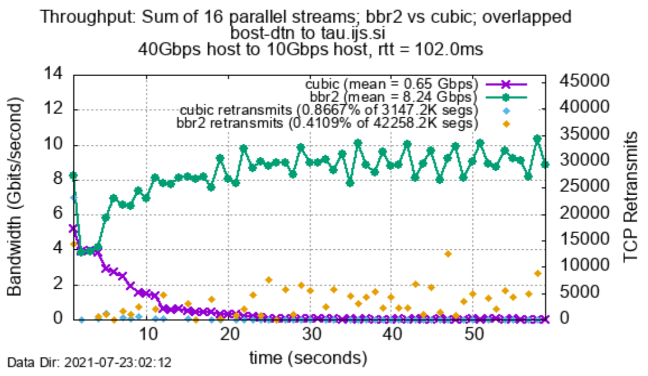


(b) 16 flows

Fig. 12: CUBIC benefits from additional flows, BBRv2 does not



(a) large RTT, lossy path, example 1, 5min test



(b) large RTT, lossy path, example 2, 60sec test

Fig. 13: BBRv2 advantage over CUBIC increases over time

firmed that BBRv2 is a significant improvement. In particular, BBRv1 has known issues with a large number of retransmits. We confirmed this both on the testbed and on ESnet.

Figure 14 shows that BBRv2 outperforms BBRv1 on the 40G Boston to 10G Sacramento path, and results for all other test paths were similar. Both CUBIC and BBRv2 had far fewer retransmits, and CUBIC throughput also increased by more than a factor of 10. For the BBRv1 test, the retransmit rate for this configuration was over 11%, an unacceptably high number, regardless of throughput. BBRv1 also caused a very large number of retransmits for CUBIC, while BBRv2 did not.

Results were even more dramatic for 16 flow tests on the testbed, using a 100G sender over the 88ms loop to a 10G receiver. BBRv1 caused over 12% of the packets to be retransmitted, and only allowed the 8 CUBIC flows a total of 40 Mbps.

E. BBRv2 Parameter Tuning

The goal of this experiment was to see if the default BBRv2 settings seem optimal for the Science DMZ use case. There are a total of 50 BBRv2 parameters that can be modified by editing the defaults set in */sys/module/tcp_BBRv2/parameters*. Our test harness supports the ability to modify these BBRv2 variables. After reading through the code, we determined that the following parameters might have an impact in our environment:

- min_rtt_win_sec (default 10 sec) : When min_rtt estimate expires, we enter PROBE_RTT mode. We tested values of 1,2,5,10,20.
- probe_rtt_mode_ms (default = 200ms). How long to probe for RTT. We tested values of 10,25,50,100,200,500.
- loss_thresh (default = 5%), Estimate bandwidth probing has gone too far if loss rate exceeds this level. We tested values of 1,2,3,4,5.
- pacing gain: a vector of values controlling the BBRv2 state machine. We tested a range of values for the first two vector elements.

We ran a set of tests with 4 parallel flows (2 BBRv2 and 2 CUBIC), and it turned out that the default setting for all of these parameters gave the best performance. One sample plot is shown in Figure 15, and a full set of plots is available in the data archive for this paper [32]. In the future we plan to run tests with 16 flows, with mismatched host speeds, and test across additional BBRv2 parameters as well.

F. Test Result Consistency and Protocol Stability

All of the plots presented in the paper are representative from the set of 10 tests we ran for each experiment. To confirm that these plots are in fact representative, we did a simple statistical analysis of the results. Tables II and III show the

TABLE II: COMPARING MEAN (M) & COEF. OF VARIANCE (C.V) FOR ESNET TESTBED.

Test		RTT < 30ms				RTT ⩾ 30ms			
		BBRv2		CUBIC		BBRv2		CUBIC	
		Mean	*C.V.*	*Mean*	*C.V.*	*Mean*	*C.V.*	*Mean*	*C.V.*
No loss	bbrv2/cubic - p1	9.6533	0.0030	9.8799	0.0024	9.4749	0.0080	9.8435	0.0019
	bbrv2/cubic - p16	9.7891	0.0064	9.8827	0.0007	9.8044	0.0039	9.8348	0.0029
	both - p16	3.1188	0.1834	6.7642	0.0849	3.3604	0.0627	6.4739	0.0334
0.001% loss	bbrv2/cubic - p1	9.6545	0.0021	3.3341	0.4694	9.4834	0.0073	1.2988	0.1541
	bbrv2/cubic - p16	9.7918	0.0061	9.8819	0.0008	9.7838	0.0041	9.7794	0.0071
	both - p16	4.2258	0.1360	5.6566	0.1026	4.9394	0.0390	4.8894	0.0435
0.01% loss	bbrv2/cubic - p1	2.3477	0.0017	1.0500	0.5585	2.3041	0.0018	0.2454	0.0722
	bbrv2/cubic - p16	9.7586	0.0053	9.0397	0.1325	9.8131	0.0017	3.9534	0.0205
	both - p16	6.1650	0.1954	3.6777	0.3352	8.0112	0.0068	1.7950	0.0276
0.1% loss	bbrv2/cubic - p1	8.8108	0.0788	0.3308	0.5180	8.7230	0.0746	0.0472	0.2533
	bbrv2/cubic - p16	9.7969	0.0037	5.1883	0.5058	9.7824	0.0038	0.7438	0.2552
	both - p16	7.5959	0.1542	2.2361	0.5284	9.4057	0.0068	0.3652	0.2545
100G-to-10G	bbrv2/cubic - p16	-	-	-	-	9.6275	0.0004	9.4377	0.0344
	both - p16	-	-	-	-	9.2094	0.0028	0.4254	0.0473

TABLE III: COMPARING M & C.V, BOST-DTN to ESNET & NON-ESNET HOSTS.

Test			RTT < 30ms				RTT ⩾ 30ms			
			BBRv2		CUBIC		BBRv2		CUBIC	
			Mean	*C.V.*	*Mean*	*C.V.*	*Mean*	*C.V.*	*Mean*	*C.V.*
10G-to-10G	**ESNET**	both - p16	4.7750	0.0726	5.0057	0.1122	4.7733	0.0055	4.8860	0.0043
	NON-ESNET	both - p16	4.2526	0.0742	4.6333	0.0309	3.9346	0.2188	3.8361	0.2972
40G-to-10G	**ESNET**	both - p8	4.5768	0.2991	5.2852	0.2399	8.3485	0.0899	1.2883	0.6450
		both - p16	4.3490	0.2291	5.1557	0.1906	6.9421	0.1222	2.4023	0.3816
	NON-ESNET	both - p8	-	-	-	-	8.2697	0.0626	2.9697	0.2500
		both - p16	-	-	-	-	8.1870	0.1512	1.9163	0.6094

mean and the coefficient of variation, or CV, which is defined as the ratio of the standard deviation to the mean. The higher the coefficient of variation, the greater the level of dispersion around the mean. In general, distributions with CV less than one are considered low-variance, while those with CV greater than one are considered high-variance [33].

Table II shows the mean and CV from the testbed 10G *netem* experiments. In the table headings, M = mean and CV = coefficient of variation. P1 are single flow tests, and P16 are 16 flow tests. CUBIC throughput is highly dependant on RTT, but due to space constraints we just divide the results into two buckets: RTT $<$ 30ms, and RTT $\geq$ 30ms. One can see that the CV for testbed results is quite low in all cases.

Table II includes the mean and CV from tests between the 40G ESnet host and a set of ESnet perfSONAR hosts, and Table III shows the mean and CV from tests between the 40G ESnet host and a set of non ESnet hosts, chosen mostly from paths with known packet loss. Latency is again in just two buckets: RTT $<$ 30ms, and RTT $\geq$ 30ms These tables show that the CV from *bost-dtn.es.net* to other ESnet hosts is moderately low CV, and tests to other hosts have a CV that is quite a bit higher, but still considerably less than one. This allows us to look at the details of individual tests and expect them to be representative.

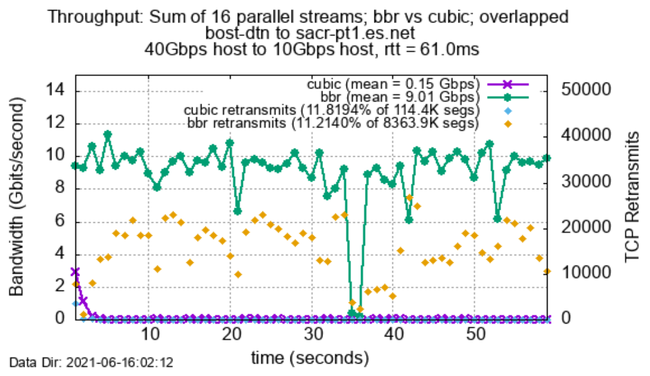
The statistical analysis also shows that BBRv2 is much more stable than CUBIC. On observing the average CV for RTT less than 30ms across all the loss rates, we see average CV of BBRv2 is 0.0647, which is approximately four times more stable than CUBIC, which has an average CV of 0.2699. Furthermore, for RTT greater than equal to 30ms, the average CV of BBRv2 is 0.0184, which is approximately five times more stable than CUBIC, which has an average CV of 0.0939. Therefore, BBRv2 has overall more stability than CUBIC across a wide range of loss rates.

VI. SUMMARY AND FUTURE WORK

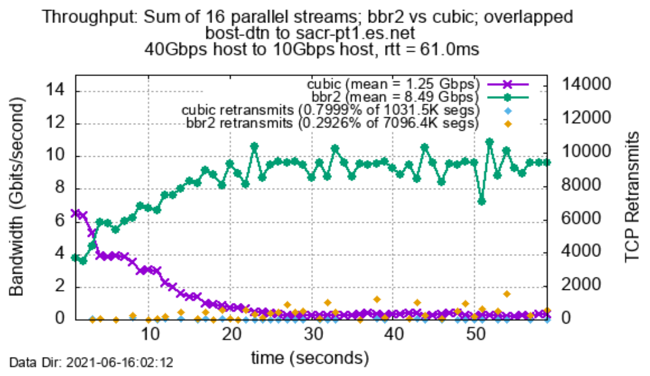
This paper confirms that results from previous papers on BBR which use data from Mininet are indeed applicable in a Science DMZ environment with DTNs running large numbers of parallel flows.

The key takeaways from the results presented in this paper are:

- BBR does much better than CUBIC on lossy paths, and the higher the loss rate and RTT, the more BBR wins out.
- Faster hosts sending parallel flows to slower hosts leads to packet loss, and BBR does much better than CUBIC in this situation.
- The BBRv1 retransmit rate is unacceptably high with parallel flows, and thus BBRv1 should not be used with parallel data transfer applications.
- BBR prefers smaller switch buffers, and CUBIC prefers larger buffers. As network speed increases, larger and



(a) BBRv1 and CUBIC, clean path, 61ms RTT



(b) BBRv2 and CUBIC, clean path, 61ms RTT

Fig. 14: Dramatic difference in retransmit rate between BBRv1 and BBRv2

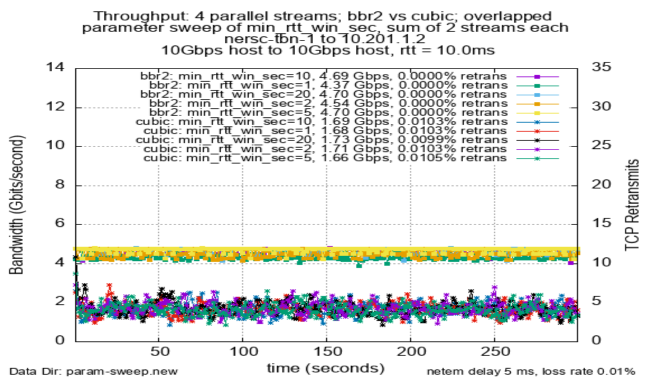


Fig. 15: testing various setting for min_rtt_win_sec

larger buffers are impractical, and lead to unstable CUBIC behaviour, so BBR will be a better choice in the future.

- Long BBRv2 flows do eventually push aside CUBIC flows on long latency high packet loss paths.

A common DTN use case is where the RTT is typically 50-100ms and parallel flows are used - here BBRv2 is a big win over CUBIC. High Speed (40G and 100G) DTN's are often used to provide data to 10G hosts. This speed mismatch leads to additional packet loss, which BBR handles much better than CUBIC.

Since BBR is not a loss-based congestion control algorithm, it will tend to push aside loss-based congestion control such as CUBIC on long lived flows. Data transfer tools that use parallel flows partially mitigate this issue. We did find one situation where parallel flows CUBIC was considerably faster than BBRv2: that being a large number of flows on a short, deep buffered path and a fast host sending to a slow host. Overall we feel that the advantages of BBRv2 greatly outweigh concerns about BBRv2 negatively impacting CUBIC.

Some additional testing we would like to do includes using other representative shallow buffer network devices in larger topologies and exploring protocol behavior in additional speed mismatched scenarios on the testbed. We plan do additional testing to determine the optimal number of parallel BBRv2 flows, and we plan to test additional BBRv2 parameters for the DTN use case. We are also looking for additional perfSONAR hosts that will allow us to run five minute tests.

Currently running BBRv2 requires patching the kernel, something most production DTN administrators are unwilling to do. We hope the Linux kernel team will accept BBRv2 into the mainline kernel soon, as BBRv2 will be a big win for Science DMZs, and the data-intensive science projects which use them.

All data collected for this paper are available at *https://downloads.es.net/INDIS-2021/*. This includes output from *iperf3* and *ss*, as well at the gnuplot [34] files used to generate the plots in this paper.

VII. Acknowledgments

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research (ASCR), of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

This manuscript has been authored by an author at Lawrence Berkeley National Laboratory under Contract No. DE-AC02-05CH11231 with the U.S. Department of Energy. The U.S. Government retains, and the publisher, by accepting the article for publication, acknowledges, that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. Government purposes.

In addition, see the content disclaimer, below.[2]

[2]This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

References

[1] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski, "The science dmz: A network design pattern for data-intensive science," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2503210.2503245

[2] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time," *Queue*, vol. 14, no. 5, p. 20–53, Oct. 2016. [Online]. Available: https://doi.org/10.1145/3012426.3022184

[3] N. Cardwell, Y. Cheng, S. H. Yeganeh, and V. Jacobson, "BBR Congestion Control," Working Draft, IETF Secretariat, Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-00, 2017. [Online]. Available: http://www.ietf.org/internet-drafts/draft-cardwell-iccrg-bbr-congestion-control-00.txt

[4] "PerSONAR Nodes Worldwide." [Online]. Available: http://stats.es.net/ServicesDirectory/

[5] "Mininet." [Online]. Available: http://mininet.org/

[6] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm," Internet Requests for Comments, RFC Editor, RFC 6582, April 2012. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6582.txt

[7] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, p. 64–74, Jul. 2008. [Online]. Available: https://doi.org/10.1145/1400097.1400105

[8] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," in *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, ser. SIGCOMM '94. New York, NY, USA: Association for Computing Machinery, 1994, p. 24–35. [Online]. Available: https://doi.org/10.1145/190314.190317

[9] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, Architecture, Algorithms, Performance," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1246–1259, 2006.

[10] Y. Cao, A. Jain, K. Sharma, A. Balasubramanian, and A. Gandhi, "When to use and when not to use bbr: An empirical analysis and evaluation study," in *IMC '19: Proceedings of the Internet Measurement Conference*, 10 2019, pp. 130–136.

[11] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a Deeper Understanding of TCP BBR Congestion Control," in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, 05 2018, pp. 1–9.

[12] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, V. Vasiliev, P. Jha, Y. Seung, M. Mathis, and V. Jacobson, "BBRv2: a model-based congestion control," Presentation in the Internet Congestion Control Research Group (ICCRG) at IETF 105 Update, Montreal, Canada, July, 2019. [Online]. Available: https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00

[13] B. Jaeger, D. Scholz, D. Raumer, F. Geyer, and G. Carle, "Reproducible measurements of tcp bbr congestion control," *Computer Communications*, vol. 144, 05 2019.

[14] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of BBR congestion control," in *2017 IEEE 25th International Conference on Network Protocols*, 10 2017, pp. 1–10.

[15] J. Crichigno, Z. Csibi, E. Bou-Harb, and N. Ghani, "Impact of segment size and parallel streams on tcp bbr," *2018 41st International Conference on Telecommunications and Signal Processing (TSP)*, pp. 1–5, 2018.

[16] E. Kfoury, J. Gomez, J. Crichigno, and E. Bou-Harb, "An emulation-based evaluation of tcp bbrv2 alpha for wired broadband," *Computer Communications*, vol. 161, 07 2020.

[17] S. Zhang, "An evaluation of bbr and its variants," *ArXiv*, vol. abs/1909.03673, 2019.

[18] Y.-J. Song, G.-H. Kim, I. Mahmud, W.-K. Seo, and Y.-Z. Cho, "Understanding of bbrv2: Evaluation and comparison with bbrv1 congestion control algorithm," *IEEE Access*, vol. PP, pp. 1–1, 02 2021.

[19] "ESnet Testbed." [Online]. Available: https://www.es.net/network-r-and-d/experimental-network-testbeds/100g-sdn-testbed/

[20] B. Tierney, J. Boote, E. Boyd, A. Brown, M. Grigoriev, J. Metzger, M. Swany, M. Zekauskas, and J. Zurawski, "perfSONAR: Instantiating a global network measurement framework"," in *Proceedings of the SOSP Workshop on Real Overlays and Distributed Systems*, 2009.

[21] "iperf3." [Online]. Available: http://software.es.net/iperf/

[22] "perfSONAR Docker images." [Online]. Available: https://docs.perfsonar.net/install_docker.html

[23] "perfSONAR Docker image for BBR testing." [Online]. Available: https://hub.docker.com/r/dtnaas/perfsonar-testpoint

[24] I. Foster, "Globus Online: Accelerating and Democratizing Science through Cloud-Based Services," *IEEE Internet Computing*, vol. 15, no. 3, p. 70–73, May 2011. [Online]. Available: https://doi.org/10.1109/MIC.2011.64

[25] "Globus network configuration." [Online]. Available: https://docs.globus.org/globus-connect-server/v4/#setting_endpoint_network_use_options

[26] "Fast Data Transfer (FDT)." [Online]. Available: https://github.com/fast-data-transfer/fdt

[27] B. Tierney, "Recent Linux TCP Updates, and how to tune your 100G host," Presented at INDIS 2016. [Online]. Available: https://scinet.supercomputing.org/community/documents/36/sc16-techtalk01-100G-Tuning-INDIS.tierney.pdf

[28] S. Hemminger, "Network emulation with netem," *Linux Conf Au*, 05 2005.

[29] "ESnet's perfSONAR Dashboard." [Online]. Available: http://ps-dashboard.es.net/maddash-webui/

[30] "ss tool." [Online]. Available: https://linux.die.net/man/8/ss

[31] J. Warner, "Network device buffer size listing." [Online]. Available: https://people.ucsc.edu/~warner/buffer.html

[32] "ESnet Data Repository." [Online]. Available: https://downloads.es.net/INDIS-2021

[33] "Coefficient of Variation." [Online]. Available: https://en.wikipedia.org/wiki/Coefficient_of_variation

[34] "gnuplot." [Online]. Available: http://www.gnuplot.info/