ESnet
ENERGY SCIENCES NETWORK

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

Eli Dart
Energy Sciences Network (ESnet)
Lawrence Berkeley National Laboratory

TNC22
Trieste, Italy
16 June, 2022

U.S. DEPARTMENT OF ENERGY
Office of Science

BERKELEY LAB

This talk contains (and builds upon) the work of many people

- Brian Tierney, Ezra Kissel, Eli Dart, ESnet
- Eashan Adhikarla, Lehigh University
- Matt Mathis, Google
- Many others at Google (BBRv2 development team and others)
- Many people in the R&E community who have done network performance, network design, and related work over many years
- ….and many others
- Thanks to all of you!

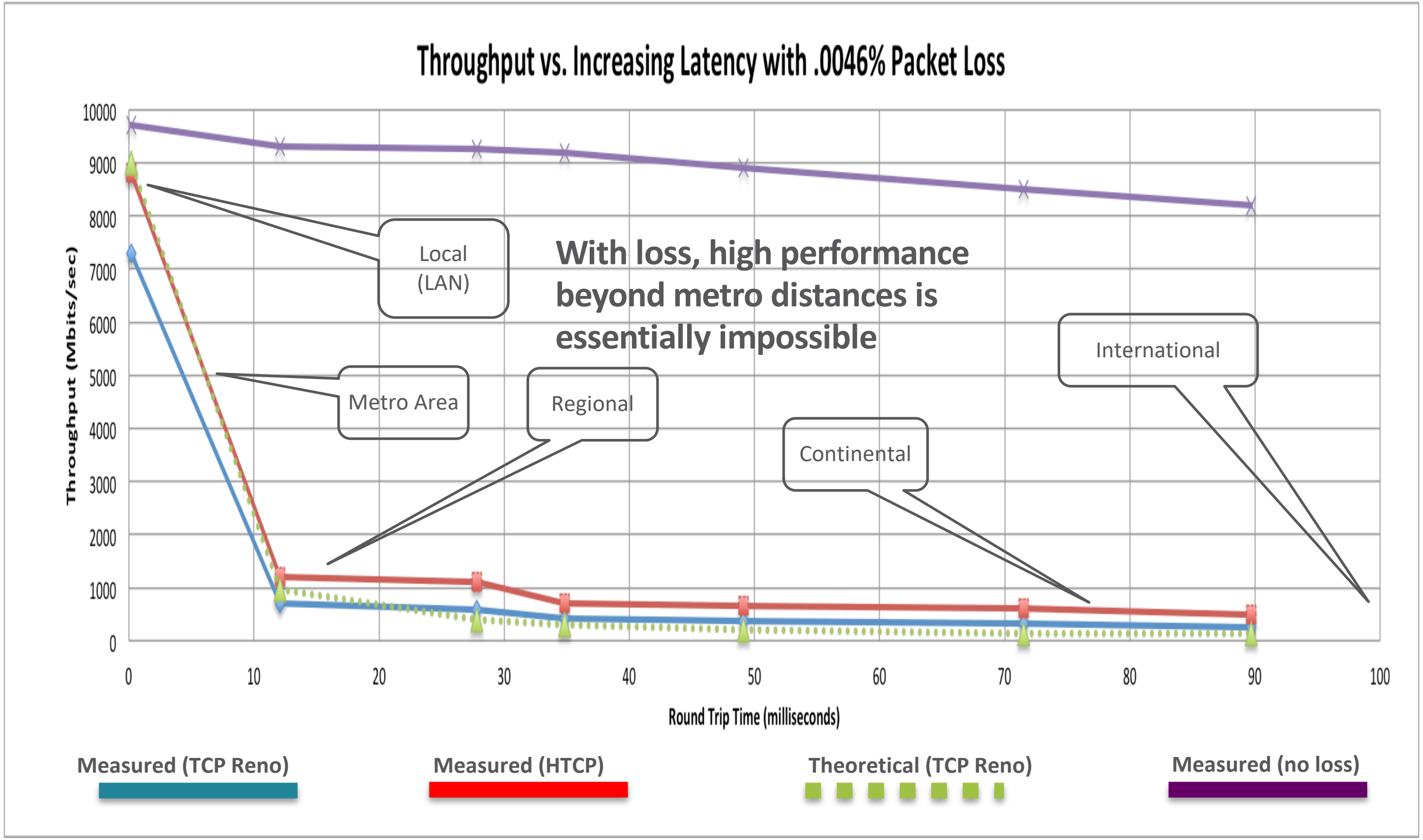
TCP Congestion Control 40 year History

- 1981 Base specification [RFC 793]
- 1986: TCP Reno (First appeared in BSD4.3)
- 1988 Van Jacobson's landmark TCP paper
- 1996: “Mathis Equation” paper defining relationship between loss and bandwidth
- 1997: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery (RFC2001)
- 1999: New Reno (RFC 2582)
- 2004: Cubic TCP released
- 2005: Fast TCP and Hamilton TCP (H-TCP) released
- 2006: Cubic becomes the default in Linux
- 2013: ESnet’s TCP slide motivation for a Science DMZ (next slide)
- 2013: FQ traffic shaper added to Linux
- 2016: BBRv1 (Bottleneck Bandwidth and Round-trip propagation time)
- 2019: BBRv2

See Matt Mathis’s talk from March 2020 for excellent summary of TCP congestion control history

– https://www.es.net/science-engagement/ci-engineering-lunch-and-learn-series

A small amount of packet loss makes a huge difference in TCP performance: BBR addresses this



ESnet

TCP Congestion Control

- Congestion Control Algorithms fall into 2 general categories:
  - Loss-based. (e.g.: Reno and Cubic)
    - Sender slows down if loss is detected
  - Delay-based (e.g.: Vegas and Fast)
    - Sender slows down if additional delay is detected
- The Internet has largely used loss-based congestion control algorithms
  - assumes that packet loss is equivalent to congestion
- But packet loss **is not** equivalent to congestion.
  - Congestion: network path has more data in flight than the bandwidth-delay product (BDP) of the path.
- Loss-based CC is increasing problematic due to:
  - Shallow buffers: in shallow buffers, packet loss happens before congestion
  - Deep buffers: at bottleneck links with deep buffers, congestion happens before packet loss.
- The BBR congestion control algorithm takes a different approach
  - *Does not assume that packet loss = congestion,*
  - BBR builds a model of the network path in order to avoid and respond to actual congestion.

ESnet

BBR TCP (slide from Matt Mathis presentation, March 2020)

- BBR: new first principles for Congestion Control
  - BBR builds an explicit model of the network
    - Estimate max_BW and min_RTT
- The BBR core algorithm:
  - By default pace at a previously measured Max_BW
    - Transmit based on a clock, not ACKs
  - Vary the pacing rate to measure model parameters
    - increase to observe new max rates
    - decrease to observe the min RTT
    - gather other signals such as ECN (bbr2)
- BBR's "personality" is determined by the heuristics used to vary the rates and perform the measurements
  - These heuristics are completely unspecified by the core algorithm
  - Relatively easy to extend or adapt
  - Many different heuristics algorithms can work together

ESnet

BBRv2 TCP

- Addresses the following BBRv1 issues
  - Low throughput for Reno/CUBIC flows sharing a bottleneck with bulk BBR flows
  - High packet loss rates if bottleneck queue < 1.5*BDP
  - Low throughput for paths with high degrees of aggregation (e.g. wifi)
  - Throughput variation due to low cwnd in PROBE_RTT
  - Adapts bandwidth probing for better coexistence with Reno/CUBIC

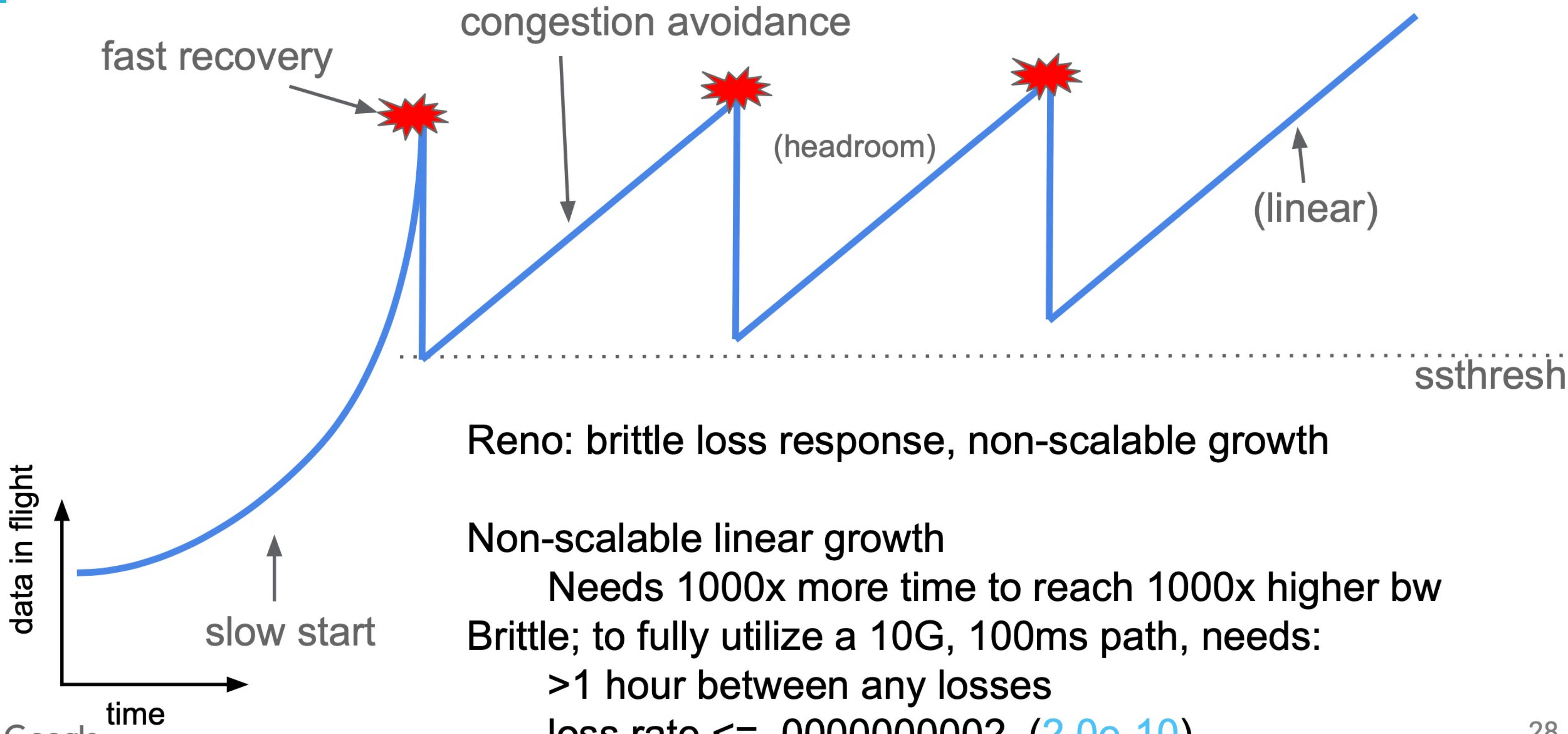
- https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00

- BBRv2 is currently being used on a small percentage of global YouTube traffic, and deployed as default TCP congestion control for internal Google traffic

ESnet

Reno

Slide from Google presentation at IETF 104



Reno: brittle loss response, non-scalable growth

Non-scalable linear growth

- Needs 1000x more time to reach 1000x higher bw

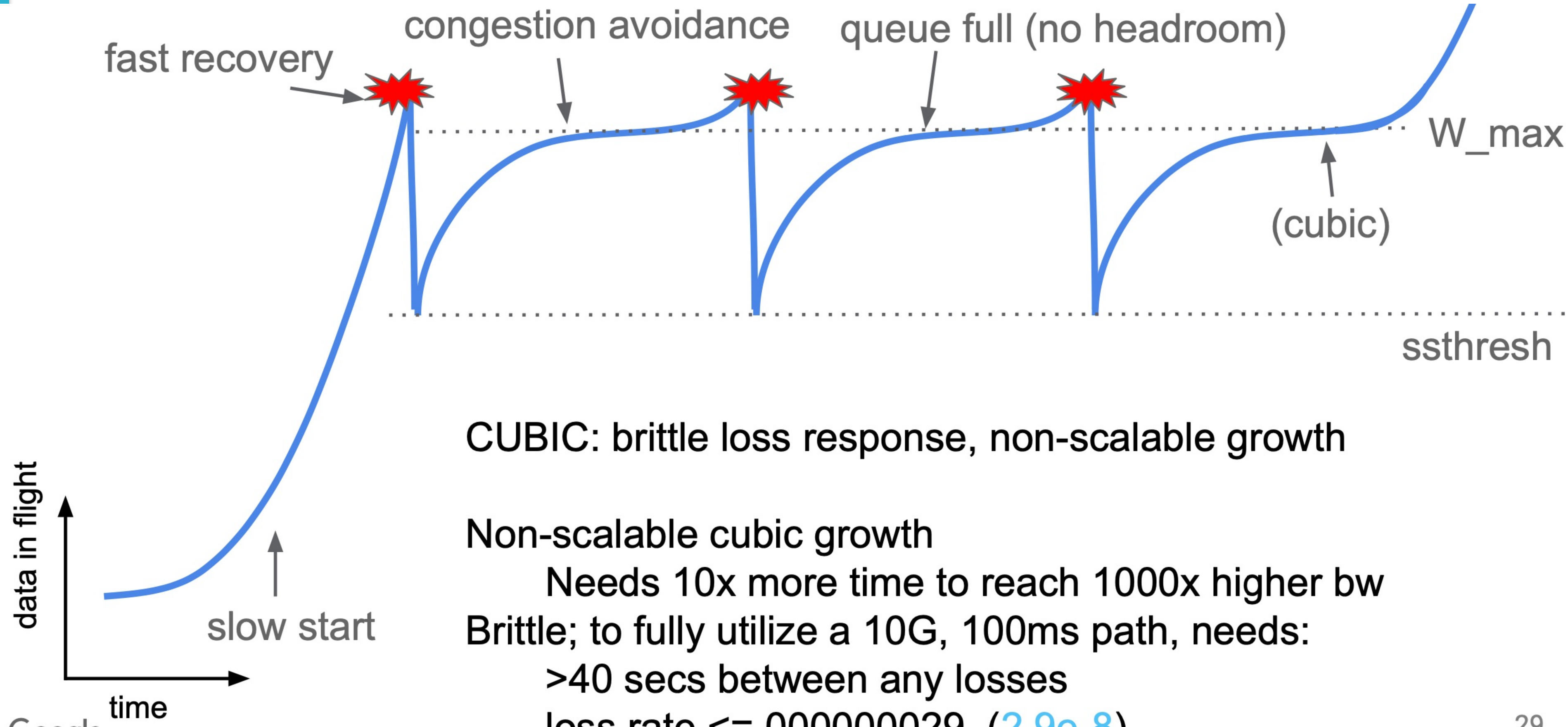
Brittle; to fully utilize a 10G, 100ms path, needs:

- >1 hour between any losses
- loss rate <= .0000000002 ([2.0e-10](2.0e-10))

Google

Cubic

Slide from Google presentation at IETF 104



CUBIC: brittle loss response, non-scalable growth

Non-scalable cubic growth

    Needs 10x more time to reach 1000x higher bw

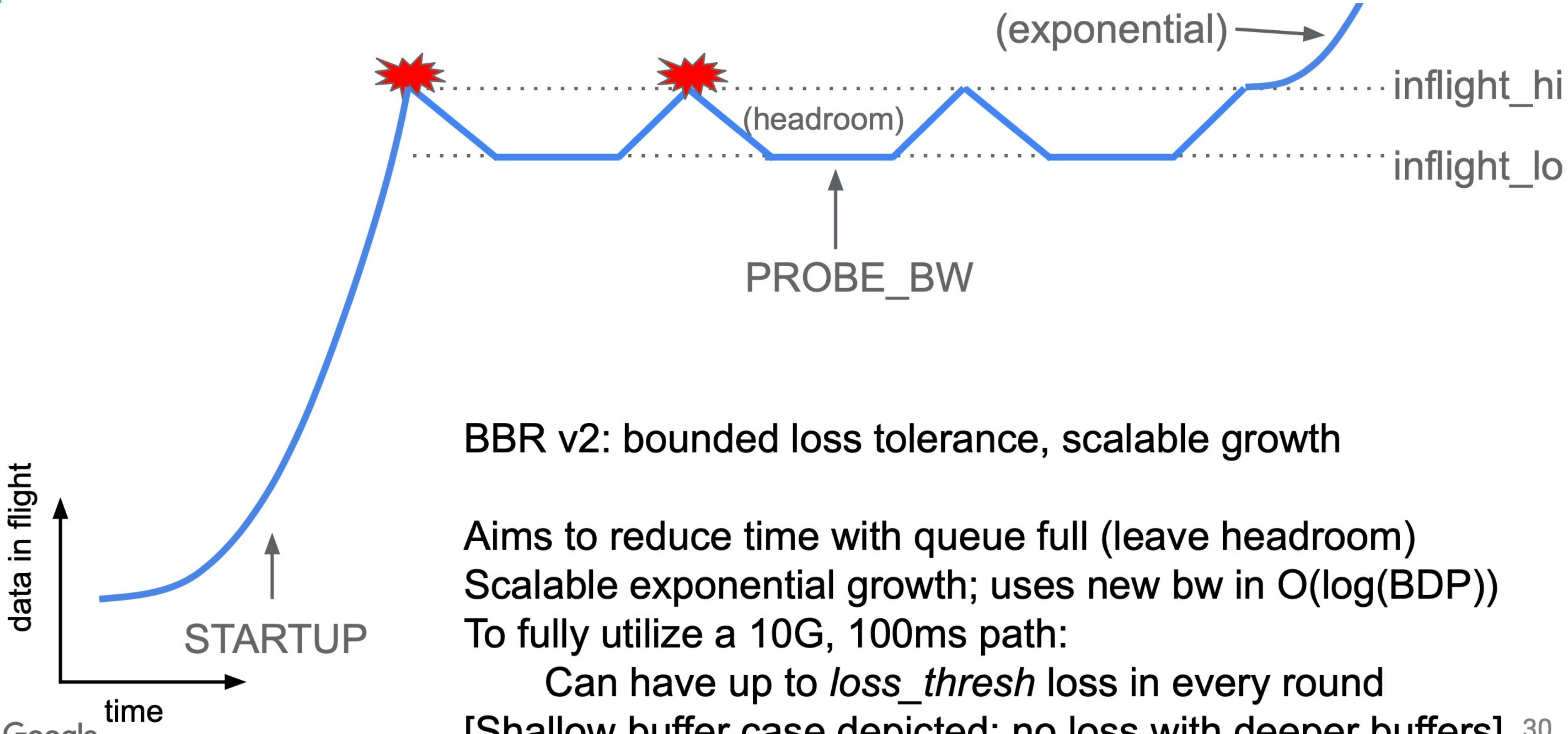
Brittle; to fully utilize a 10G, 100ms path, needs:

    >40 secs between any losses

    loss rate <=.000000029 (2.9e-8)

BBRv2

Slide from Google presentation at IETF 104



BBR v2: bounded loss tolerance, scalable growth

Aims to reduce time with queue full (leave headroom)
Scalable exponential growth; uses new bw in O(log(BDP))
To fully utilize a 10G, 100ms path:
    Can have up to *loss_thresh* loss in every round
[Shallow buffer case depicted; no loss with deeper buffers]

ESnet's BBRv2 Evaluation Project

Evaluate BBRv2 for large science data transfers

- 40G / 100G hosts ("Data Transfer Nodes")
- Data transfer tools that use parallel flows (e.g.: GridFTP)
- Focus is on R&E (research and education) networks, not commodity internet
  - Very different use case than Google/YouTube requirements
- Share results with protocol dev community and gather feedback
- Anticipate future small-buffer, high-BDP networks and wider adoption

Key question: will BBRv2 enable scientific applications to perform well in the absence of deep switch and router buffers?

ESnet

https://fasterdata.es.net/assets/Uploads/INDIS-2021-bbr2.final.pdf

BBRv2 has some assumptions 'baked in'

Comment in the BBRv2 source code:

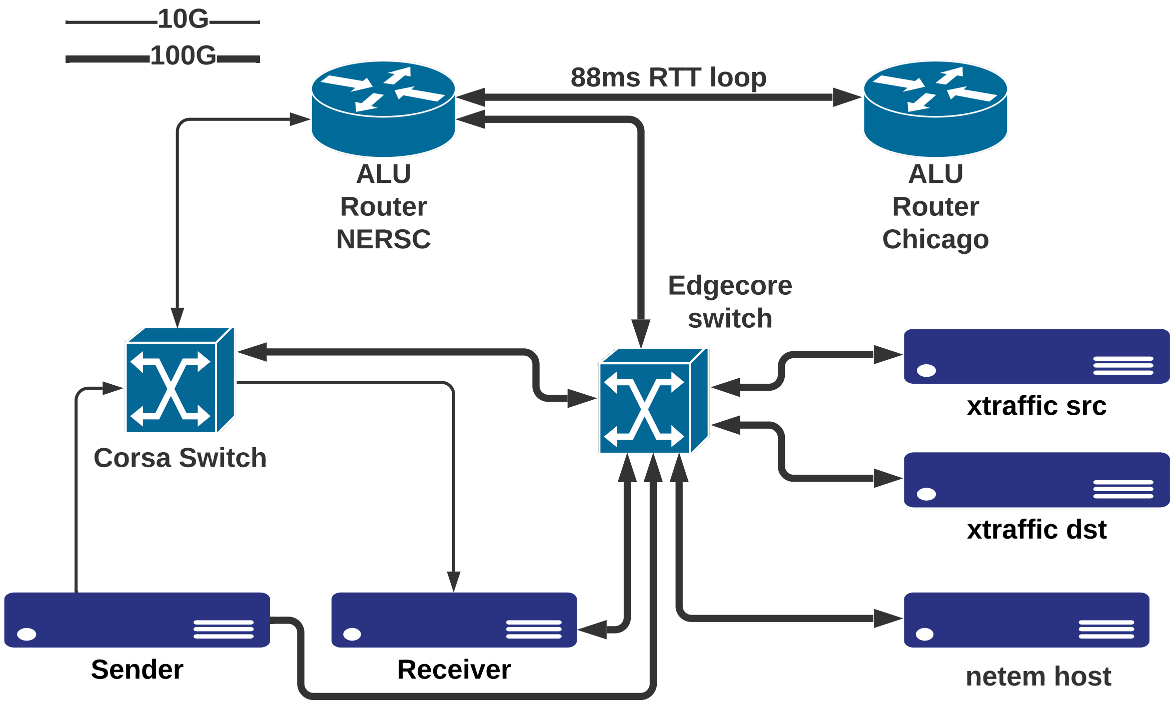
```
/*
 * We bound the Reno-coexistence inter-bw-probe time to be 62-63 round trips.
 * This is calculated to allow fairness with a 25Mbps, 30ms Reno flow,
 * (eg 4K video to a broadband user):
 *   BDP = 25Mbps * .030sec /(1514bytes) = 61.9 packets
 */
```

- Our use case is quite different
  - Incoming DTN transfers to a ScienceDMZ will be a mix of BBR and CUBIC while BBR catches on
- Does BBRv2 work well for the DTN use case? How well does it coexist with CUBIC flows?

ESnet

Testing Methodo

- Run Tests in a controlled environment
  - ESnet Testbed
- Run Tests over the Internet:
  - Using perfSONAR



ESnet Testbed Configuration

ESnet

‘Real world’ Testing

Source Node:

- 40G host directly connected to ESnet backbone
- Ubuntu 20, 5.10.0 kernel with bbr2 patches
- perfSONAR *Testpoint* Docker container
  - https://docs.perfsonar.net/install_options.html
  - perfSONAR only allows 1 throughput test to be run at a time

Destination Nodes:

- There are roughly 2000 registered perfSONAR hosts worldwide
  - most of which allow testing from ESnet
  - many of which allow testing from anywhere
  - most restrict testing to 1 minute, but ESnet allows longer tests from other ESnet hosts.
- Tests are running on production networks, with no control over competing traffic
- We selected a variety of test hosts of various RTTs and various loss characteristics

ESnet

Test Harness

- Python program to facilitate running tests and collecting instrumentation data.
- Sample config file entry:

```
[pscheduler_bbr2_p16]
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}
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
tcpdump = 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
```

ESnet

Raw Data

Our test harness has the ability to collect the following:

- iperf3 JSON output (as reported by pscheduler tool)
- ss (socket stats)
- tcpdump / tcptrace
- mpstat (CPU load)

The data used to generate these plots is available at:

- https://downloads.es.net/INDIS-2021/

ESnet

Testing / Plotting Methodology and Terminology

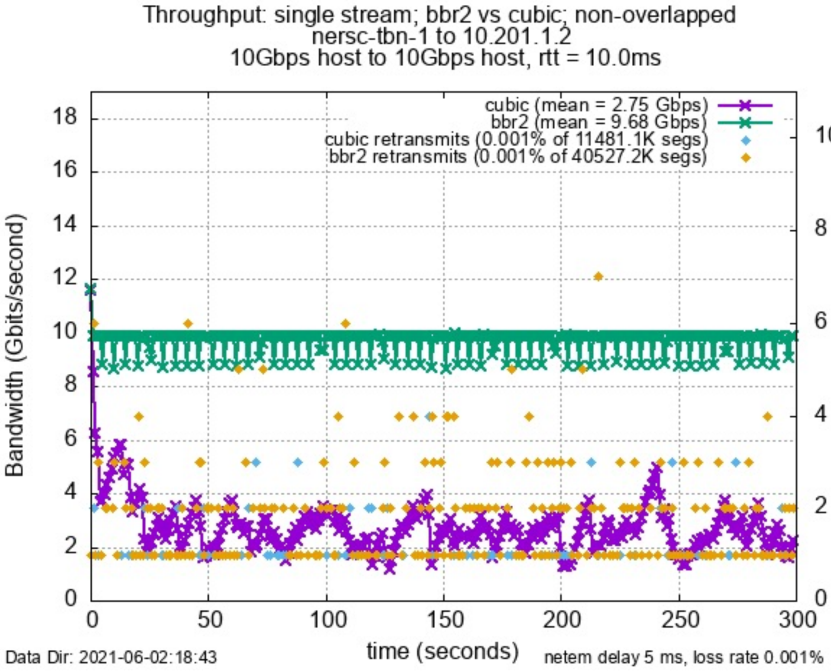
- Parallel Flow tests all use 16 flows
  - This is a common default for Globus and other DTN tools
- “non-overlapped” means a 16 flow CUBIC test, followed by a 16 flow BBRv2 test
- “overlapped” means 8 CUBIC flows and 8 BBRv2 flows, all at the same time
- Netem-based results have netem setting in the lower right of the plot

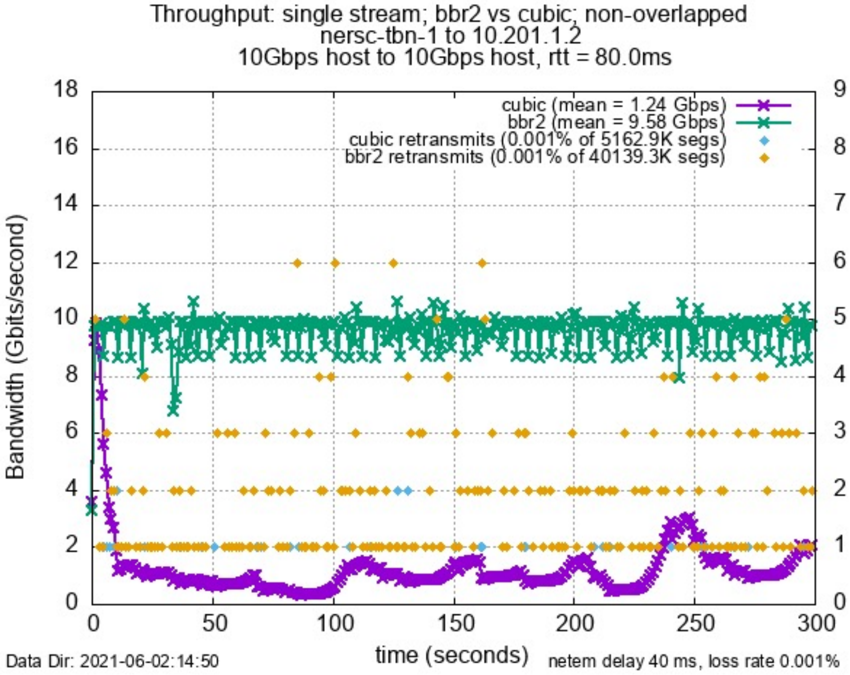
ESnet

Test Variability

- We ran 10 runs of each experiment configuration, and computed the coefficient of variation (CV) of each
    - CV 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.
- The CV for all experiments was < 1 (i.e.: reasonable)
- BBRv2 results were 4-5 times more stable than CUBIC based on the CV
- See the paper for more details

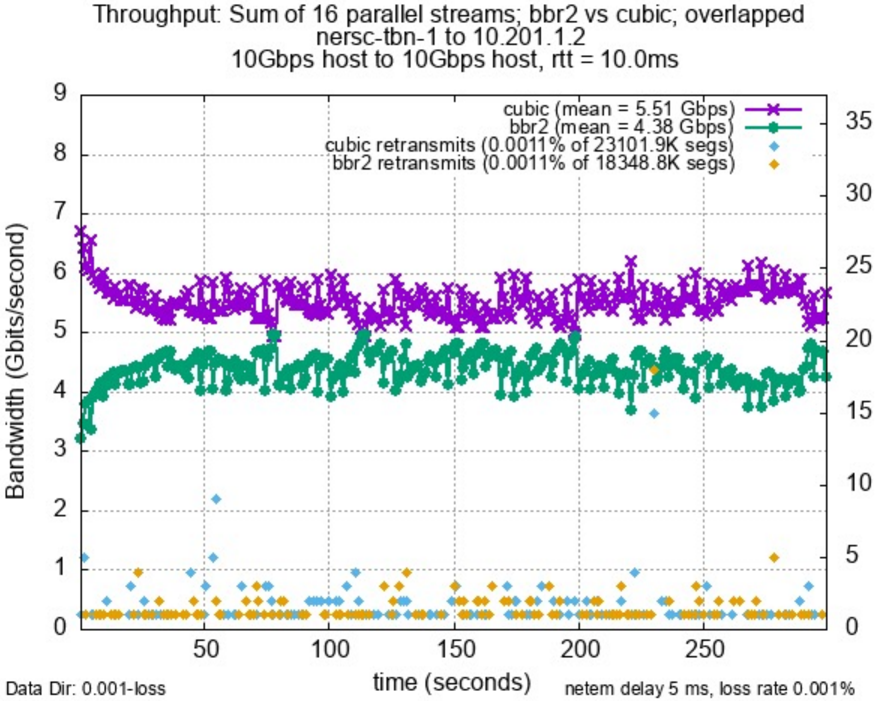
Single flow results: BBRv2 vs CUBIC, 0.001%

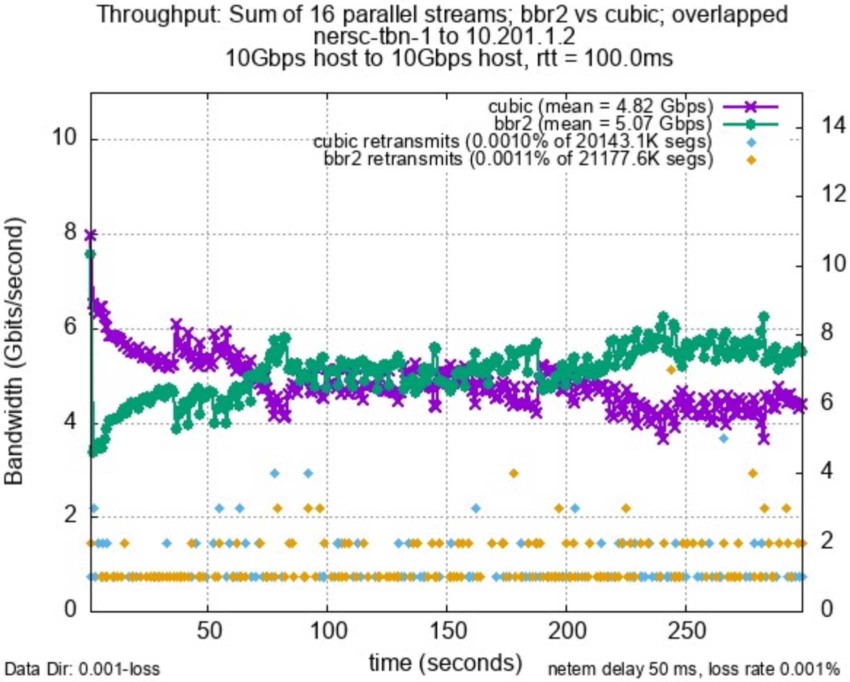




- For single flows, BBRv2 does much better than CUBIC on paths, even with low (0.001%) packet loss
- BBRv2 advantage increases with longer RTT

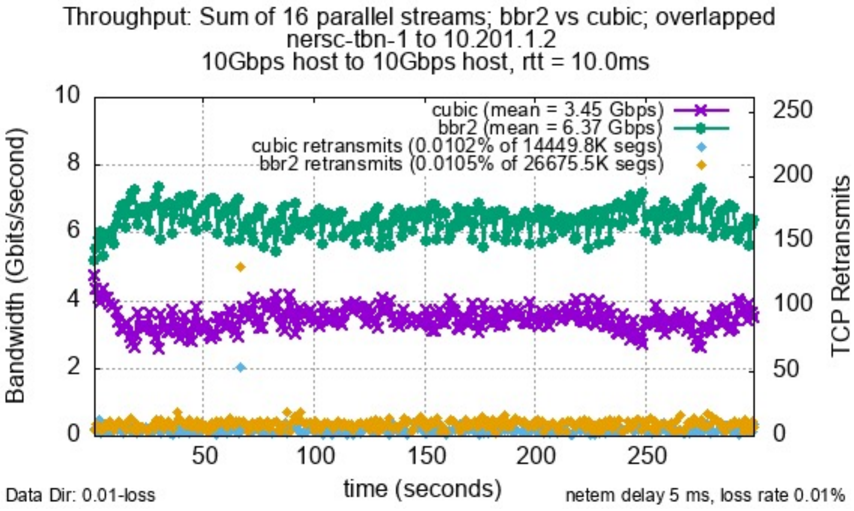
16 flow results: BBRv2 vs CUBIC, 0.001% packet

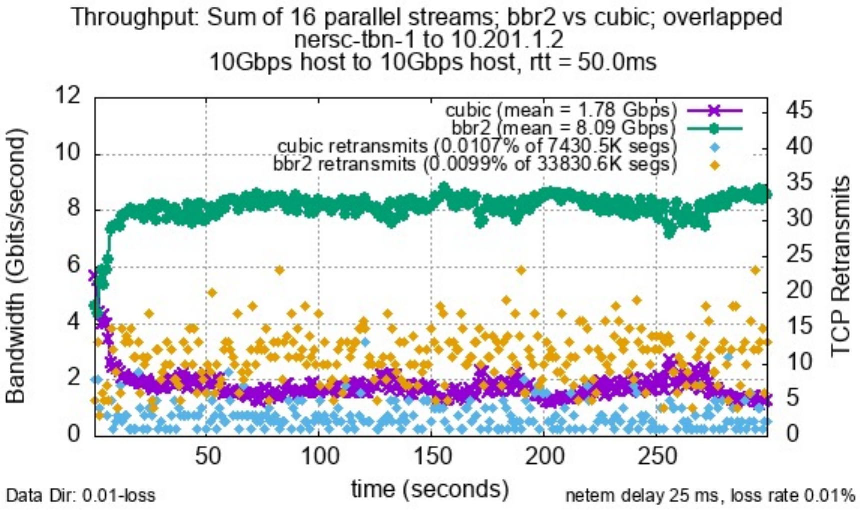




- Parallel CUBIC flows compensate for BBRv2's advantage for low packet loss rates.
- BBRv2 and CUBIC throughputs are similar.

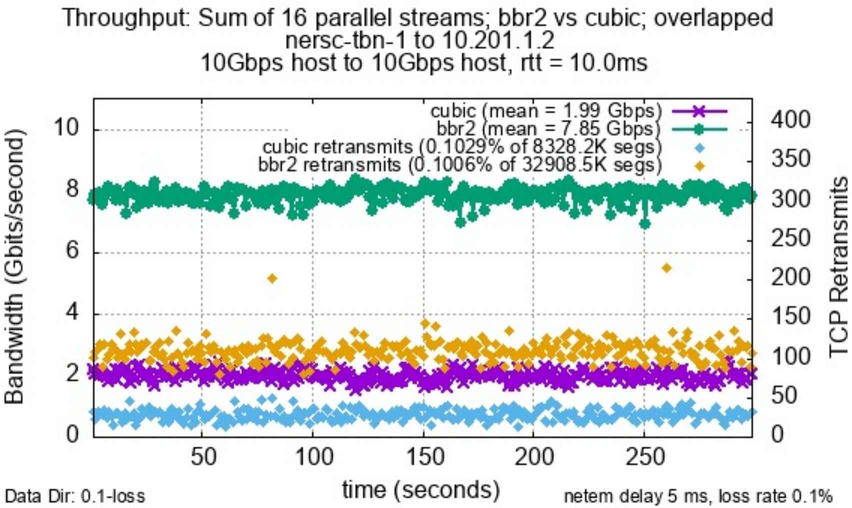
16 flow results: BBRv2 vs CUBIC, 0.01% packet loss

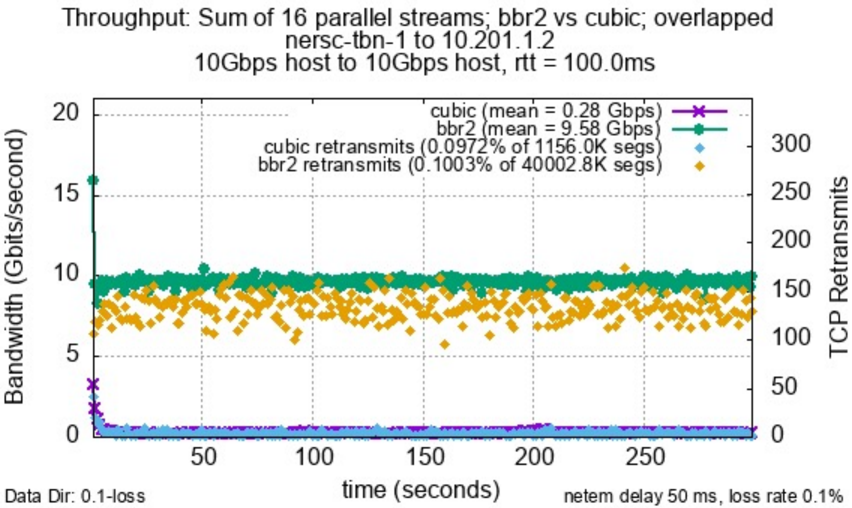




- With additional packet loss (0.01%) parallel BBRv2 starts to do much better than CUBIC, especially on long paths

16 flow results: BBRv2 vs CUBIC, 0.1% packet loss

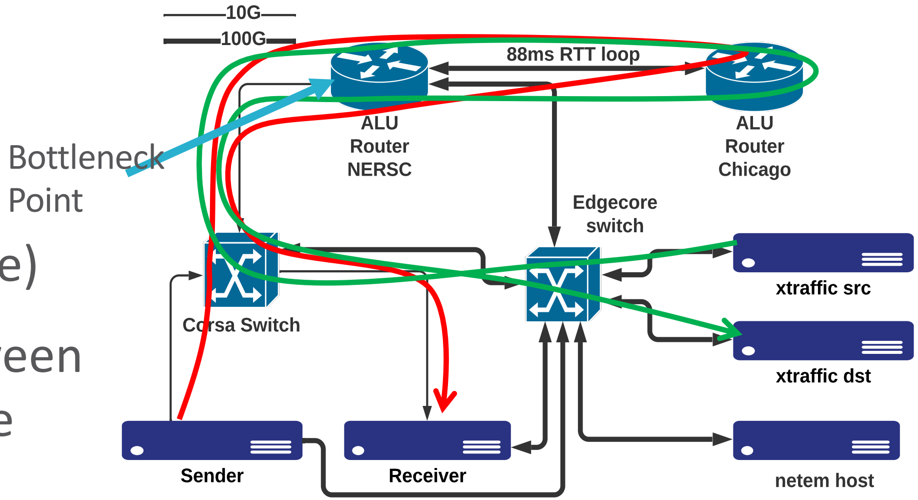




- BBRv2 does even better yet with 0.1% loss.
- 4x on 10ms path, and more than 30x faster on a 100ms path

Buffer Size results

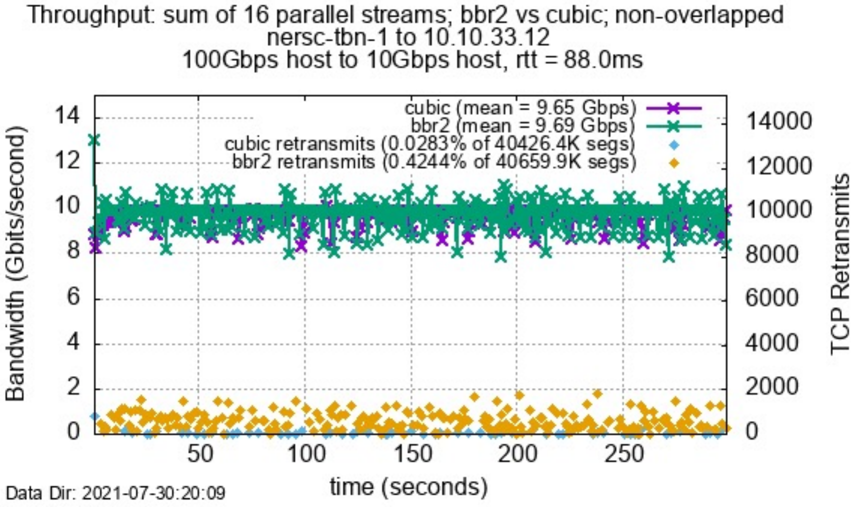
- TCP over 10G 88ms loop path (red line)
- Background 1 Gbps UDP stream between testbed hosts xtraffic src/dst to create congestion (green line)

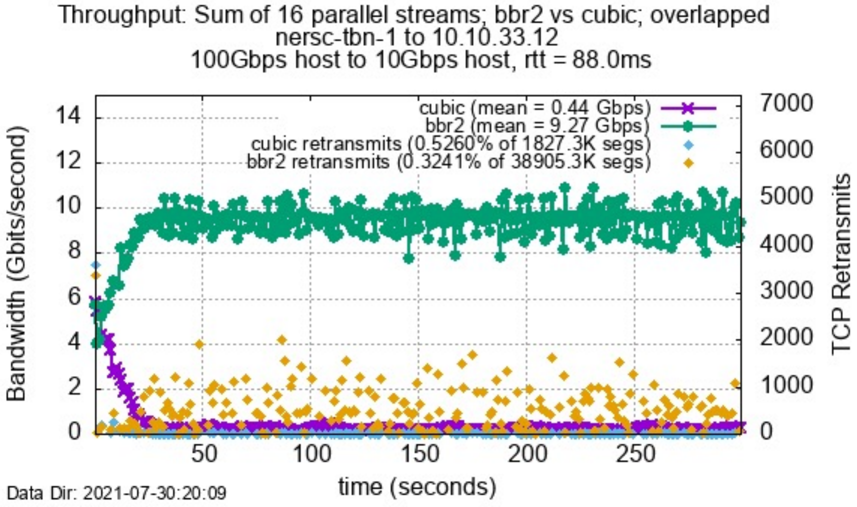


Buffer Size	CUBIC throughput	BBRv2 throughput
8 MB	0.4 Gbps	8.3 Gbps
12 MB	0.9 Gbps	8.0 Gbps
16 MB	1.8 Gbps	6.9 Gbps
32 MB	4.5 Gbps	4.3 Gbps
64 MB	4.6 Gbps	4.2 Gbps

ESnet

16 flow results: Testbed, 100G sender to 10G receiver





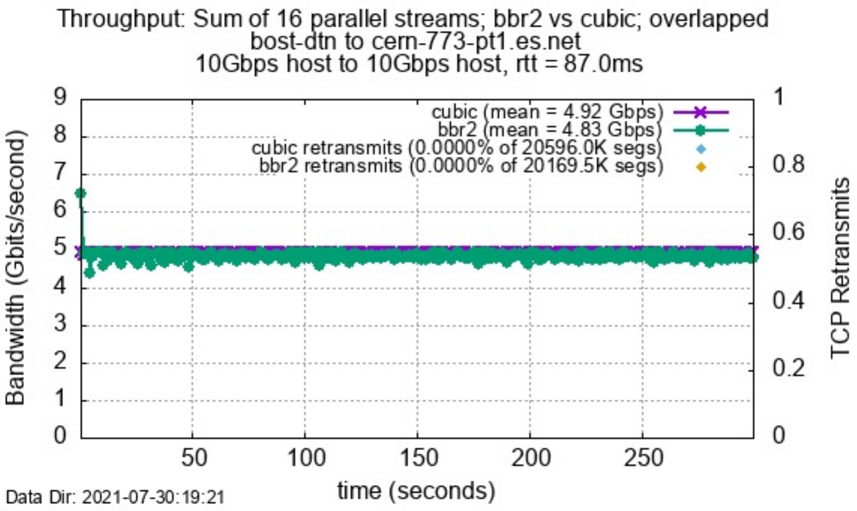
- BBRv2 and CUBIC both do well on a clean path, but BBRv2 retransmit rate is consistently about 20x higher than CUBIC

- With overlapped flows, BBRv2 steps on CUBIC flows, is 20 times faster, and has fewer retransmits.

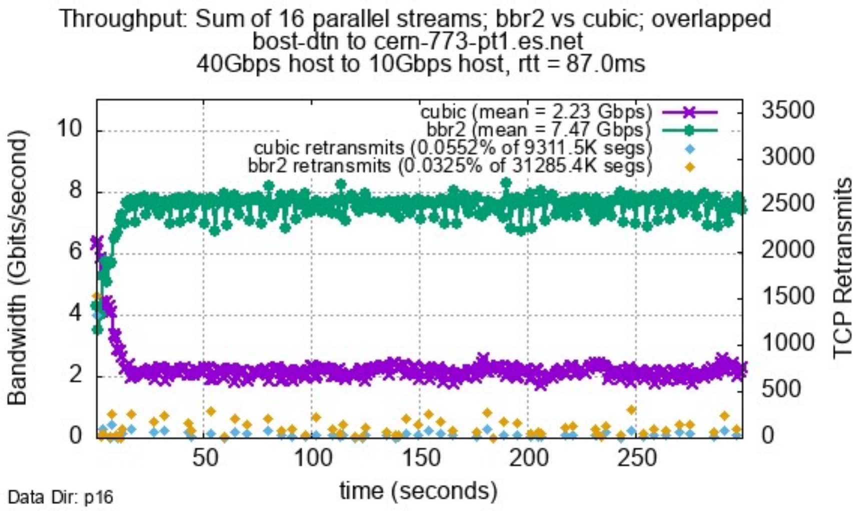
Open Question: why is BBRv2 retransmit rate so high in non-overlapped case?

ESnet

16 flow results: ESnet results, 40G to 10G



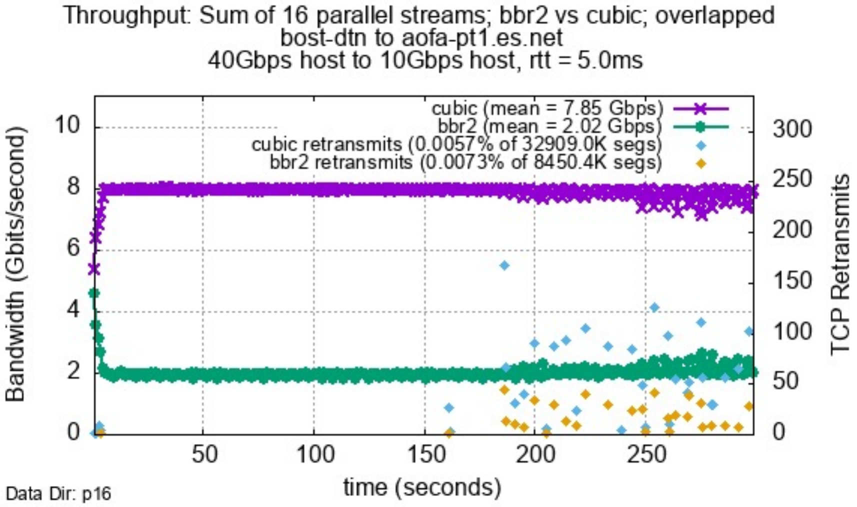
10G sender (620 Mbps pacing/flow, 9.9G total)



40G sender (2.4 Gbps pacing/flow, 38.4G total)

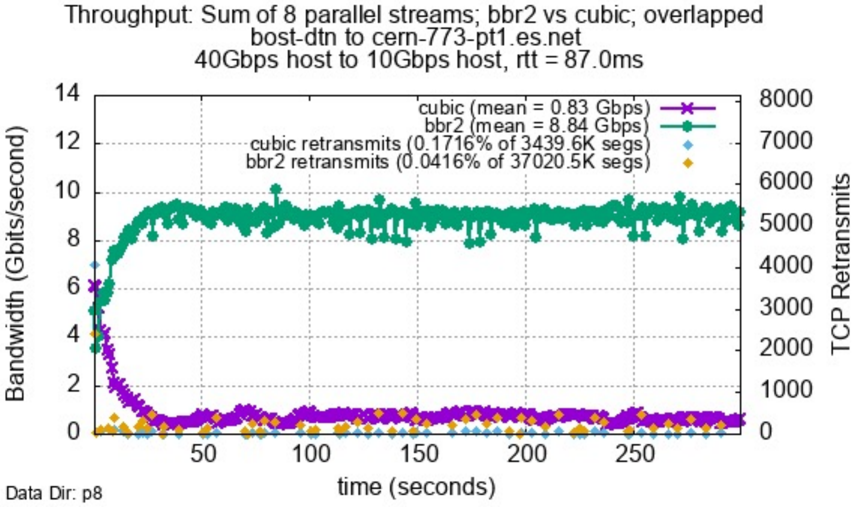
- No speed mismatch = No packet loss = CUBIC and BBRv2 are equivalent
- But BBRv2 does much better when sender is faster than receiver

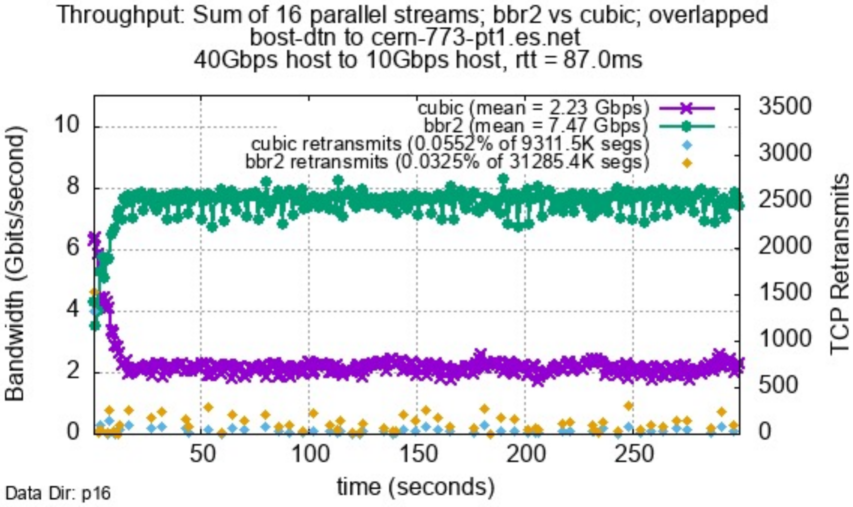
But, Sometimes CUBIC is faster



- Overlapped CUBIC and BBR2 flows
- 5ms RTT, low packet loss
- CUBIC is considerably faster
- Note: very deep buffers on this path

How many parallel flows?

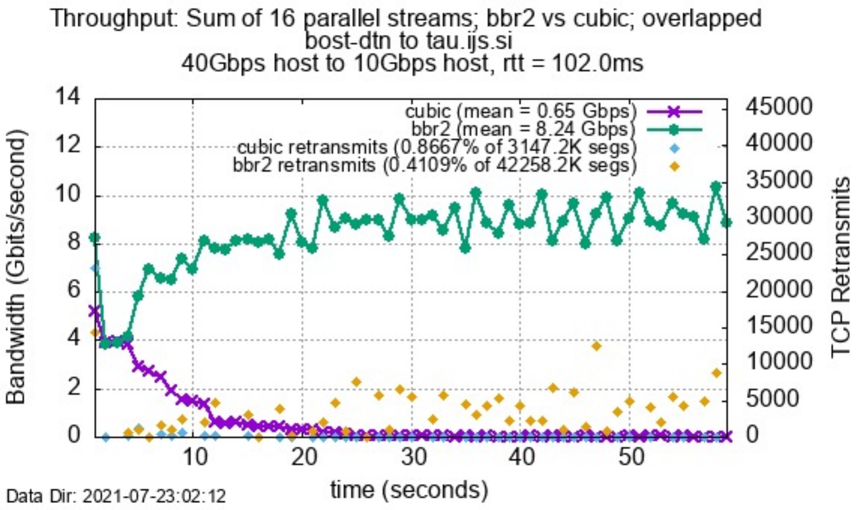


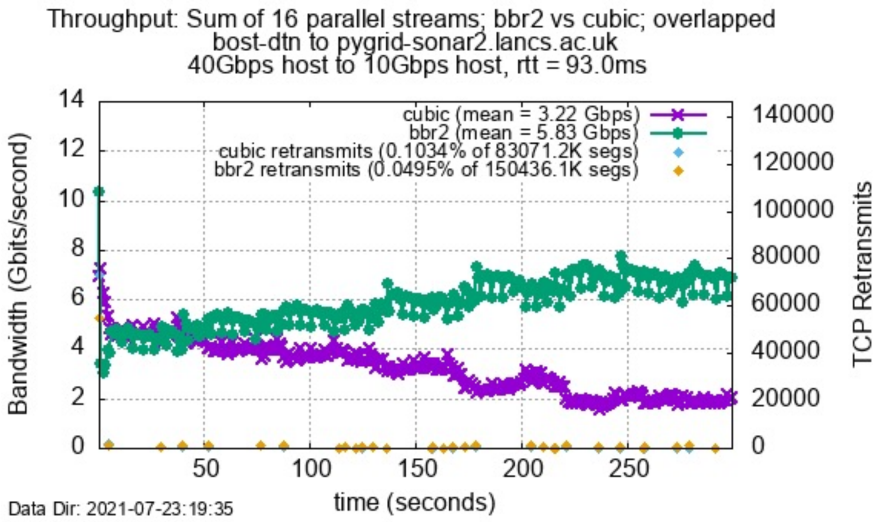


- CUBIC benefits from additional flows, BBRv2 does not
- Initial testing shows that maximum BBRv2 throughput is achieved with 2-4 flows; more testing needed

ESnet

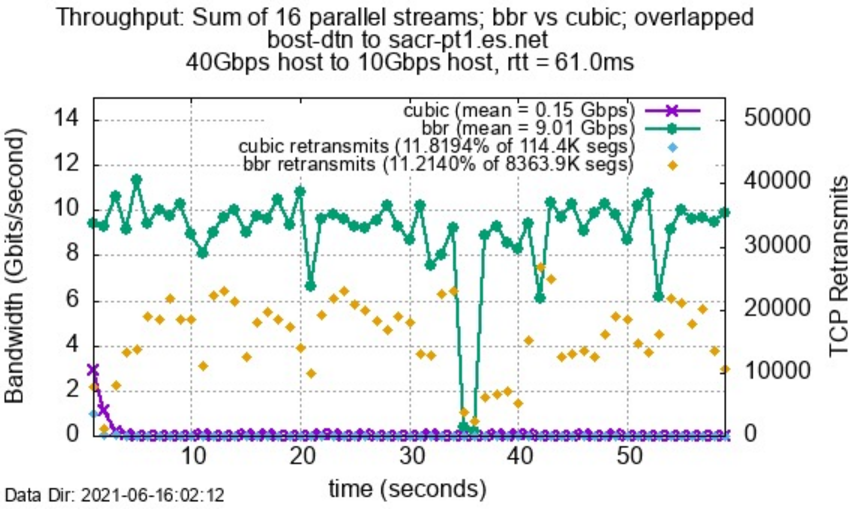
BBRv2 gains greater share of the pipe over time

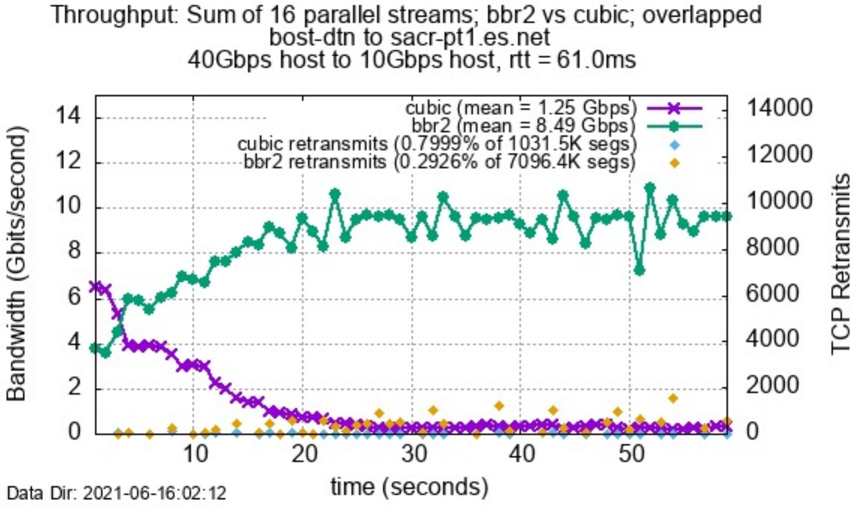




- Sometimes this happens in the 1st 20 seconds of the flow, and sometimes not until much later.

BBRv2 vs BBRv1





BBRv1 has way more retransmits and is way more unfriendly to CUBIC

- CUBIC only gets 0.15Gbps, vs 1.25Gbps with BBRv2
- Retransmits > 11% for BBRv1, and < 1% for BBRv2

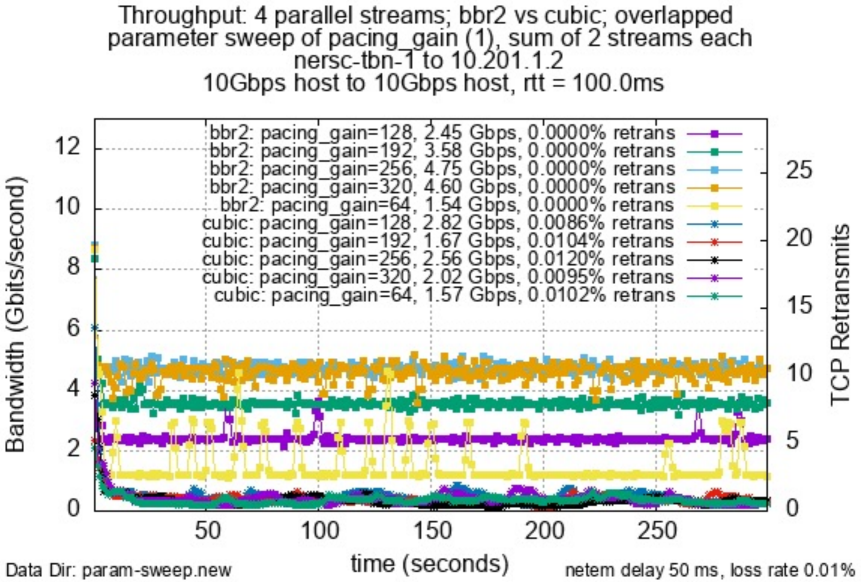
ESnet

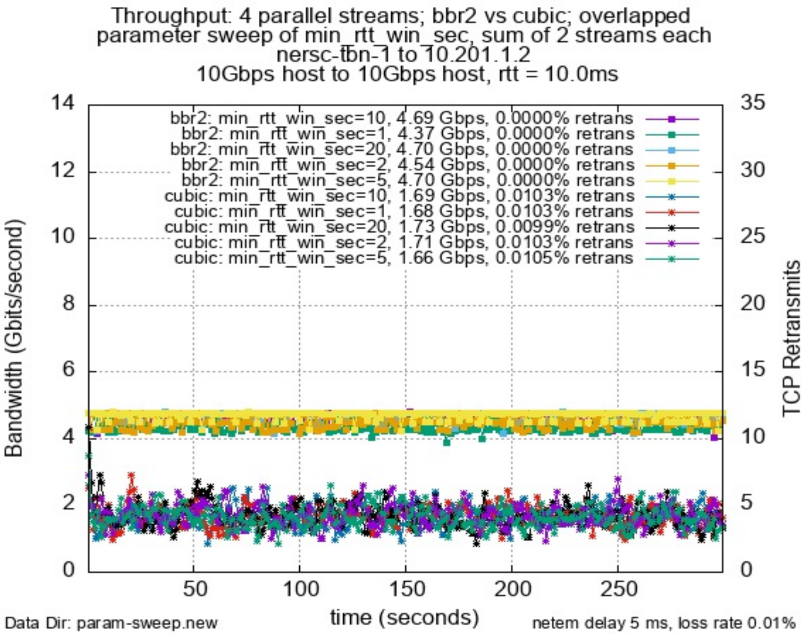
BBRv2 Tuning Parameters

- Lots of tuning knobs (/sys/module/tcp_bbr2/parameters/)

```
ack_epoch_acked_reset_thresh  bw_probe_rand_us    extra_acked_gain
inflight_headroom  probe_rtt_cwnd_gain bw_probe_reno_gain
extra_acked_in_startup  full_bw_cnt      loss_thresh    probe_rtt_mode_ms
usage_based_cwnd bw_probe_base_us        cwnd_gain
ecn_factor         extra_acked_max_us    full_bw_thresh  min_rtt_win_sec
probe_rtt_win_ms bw_probe_max_rounds            cwnd_min_target
drain_gain         ecn_max_rtt_us    extra_acked_win_rtts
full_ecn_cnt    min_tso_rate    refill_add_inc
bw_probe_pif_gain          cwnd_tso_bduget   drain_to_target
ecn_reprobe_gain  fast_ack_mode    full_loss_cnt   pacing_gain
startup_cwnd_gain   bw_probe_rand_rounds          cwnd_warn_val
ecn_alpha_gain      ecn_thresh          fast_path
high_gain        precise_ece_ack      tso_rtt_shift
```

Parameter Sweep Results





- Our test harness supports testing a range of BBRv2 parameters
- Results to date show that default settings appear optimal
- Much more testing is needed

ESnet

Fairness to CUBIC

- Under some circumstances, BBRv2 is unfair to CUBIC
  - High-BDP paths with packet loss (e.g. from shallow buffer switch or congestion)
  - Speed mismatch (e.g. 100G host to 10G host)
- In theory, it is useful to study fairness, because it helps us understand protocols
- In practice, CUBIC requires very expensive engineering to be competitive with BBRv2
  - Very low packet loss requires deep buffers, significant human effort – especially for high-BDP environments (e.g. science/DTN workloads)
  - How should we account for the difference in cost to achieve “fairness?”
- Practical deployment concerns are likely to favor the adoption of BBRv2 and the phase-out of CUBIC over time

ESnet

Next Steps

- 100G Testing
  - Are there any surprises at 100G?
- More buffer testing with other small buffered devices
- More BBRv2 parameter sweep testing
  - Especially at 100G

Key Takeaways

- BBR (both v1 and v2) does much better than CUBIC on lossy paths
  - The higher the loss rate and RTT, the more BBR wins out.
- Faster hosts sending parallel flows to slower hosts leads to packet loss
  - BBR does much better than CUBIC in this situation.
- The BBRv1 retransmit rate is unacceptably high with parallel flows, and is very unfair to CUBIC
  - BBRv1 should not be used with parallel data transfer applications.
- BBR prefers smaller switch buffers, and CUBIC prefers larger buffers.
  - As network interface speed increases, larger and larger buffers are impractical (and thus more expensive)
  - Therefore BBR will be a better choice in the future.

ESnet

Run your own tests

- Install BBR kernel patch: https://github.com/google/bbr/blob/v2alpha/README.md

- Customized Docker container for running your own perfSONAR testpoint on a bbr2 enabled host:
  - https://hub.docker.com/r/dtnaas/perfsonar-testpoint

- Test harness source code:
  - https://github.com/esnet/testing-harness

ESnet

For more information

- BBRv2:
  – https://groups.google.com/g/bbr-dev
  – Links to all of Google's BBR papers and talks can be found there.
- Relevant pages on FasterData:
  – https://fasterdata.es.net/science-dmz/DTN/tuning/
  – https://fasterdata.es.net/network-tuning/packet-pacing/
- 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 files used to generate the plots in this paper.

ESnet

ESnet
ENERGY SCIENCES NETWORK

Thanks!

Eli Dart (dart@es.net)

Energy Sciences Network (ESnet)

Lawrence Berkeley National Laboratory

http://fasterdata.es.net/

http://my.es.net/

http://www.es.net/

U.S. DEPARTMENT OF ENERGY Office of Science

BERKELEY LAB

Variance

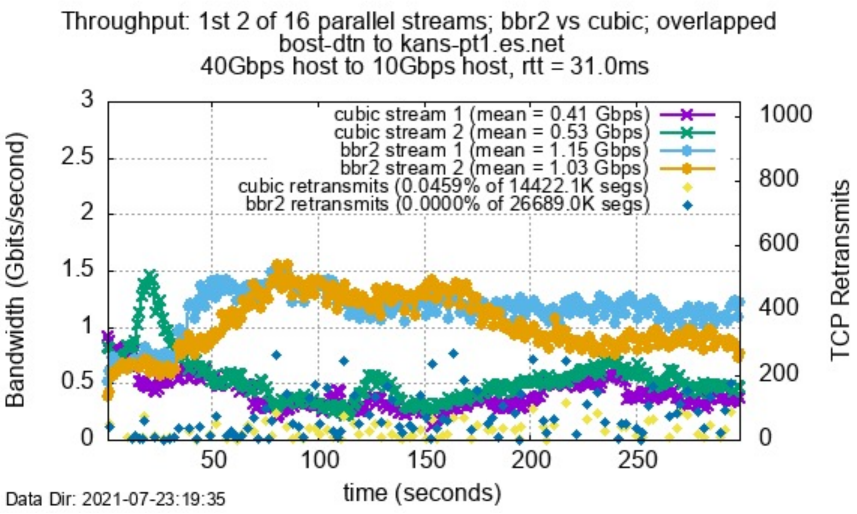
TABLE II: COMPARING MEAN (M) & COEF. OF VARIANCE (C.V) FOR ESNET TESTBED.

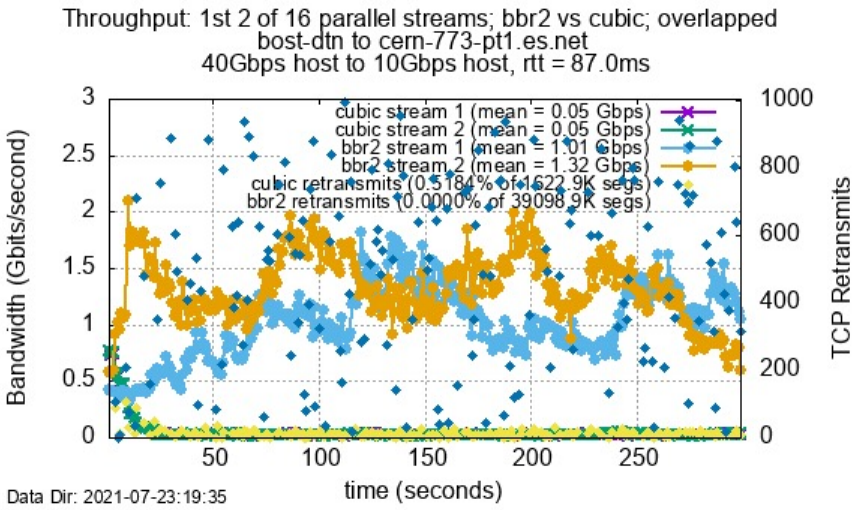
Test		RTT < 30ms				RTT ⩾ 30ms			
		BBRv2		CUBIC		BBRv2		CUBIC	
		Mean	*C.V.*	*Mean*	*C.V.*	*Mean*	*C.V.*	*Mean*	*C.V.*
No loss	bbrv2/cubic - p1	9.6533	0.0030	9.8799	0.0024	9.4749	0.0080	9.8435	0.0019
	bbrv2/cubic - p16	9.7891	0.0064	9.8827	0.0007	9.8044	0.0039	9.8348	0.0029
	both - p16	3.1188	0.1834	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

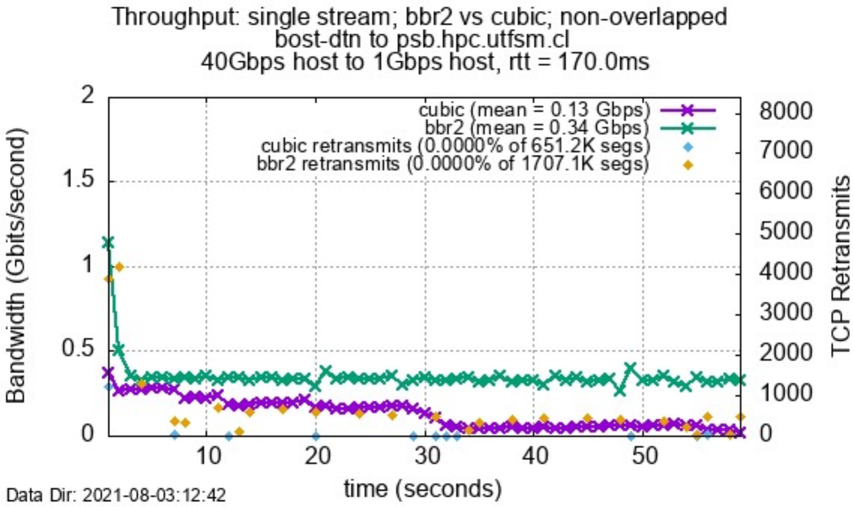
Parallel Stream Behavior

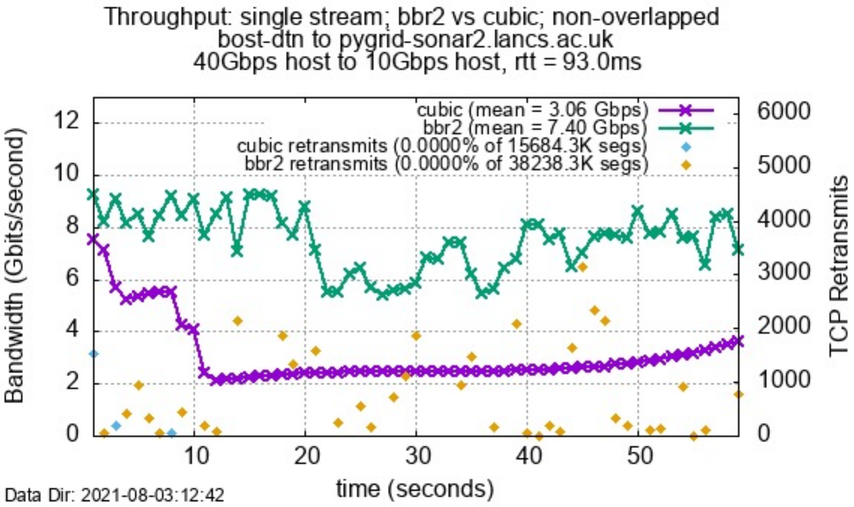




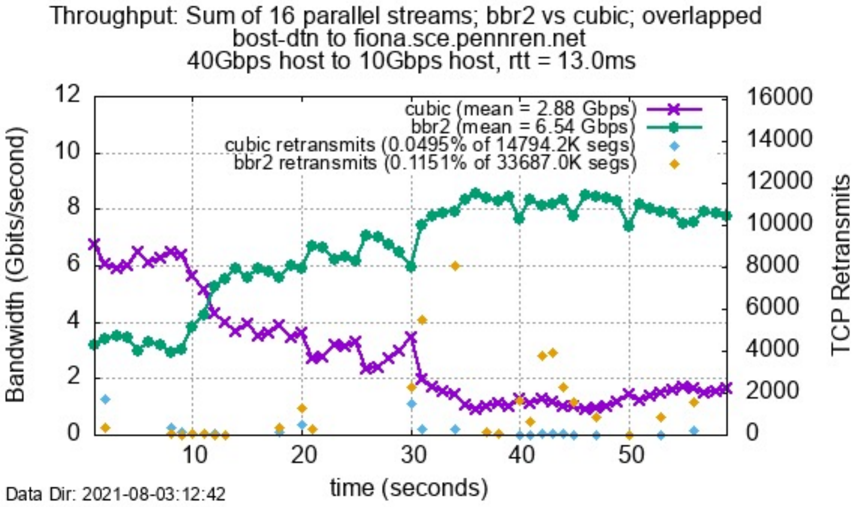
- BBRv2 performance not very stable in this environment

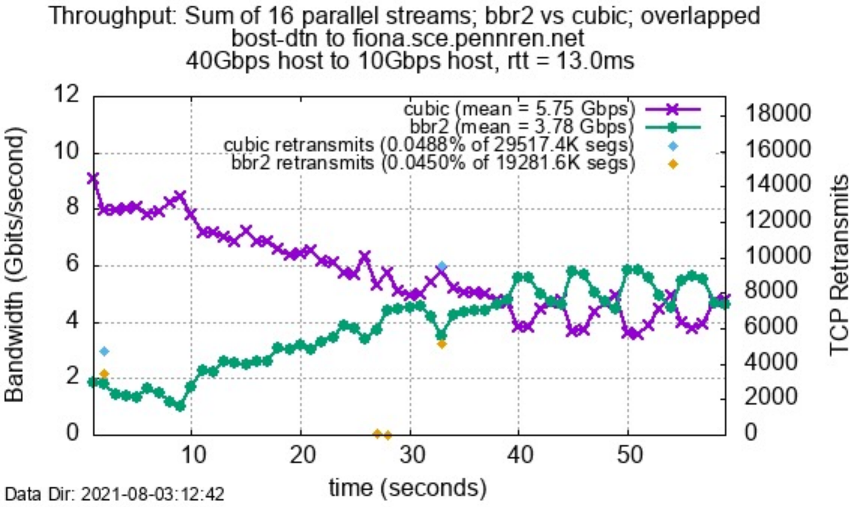
More Single Flow Examples



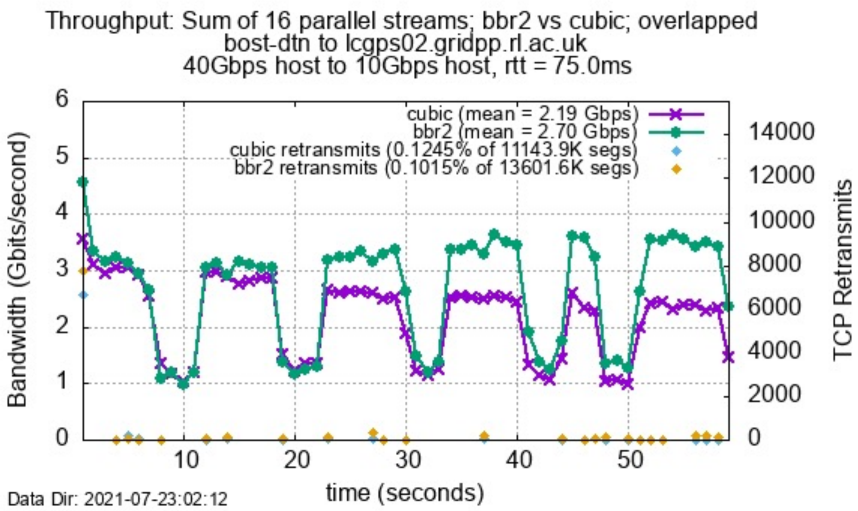


More 16-flow parallel examples: Some paths are odd..





More 16-flow parallel examples: Some paths are odd..



More 16-flow parallel examples