

# TCP and BBR

Geoff Huston  
APNIC

# Computer Networking is all about moving data

- The way in which data movement is controlled is a key characteristic of the network architecture
- The Internet protocol passed all controls to the end systems, and treated the network as a passive switching environment
- But that was many years ago, and since then we have seen the deployment of active middleware in networks and control traffic flows
- All this is changing again as we see a new generation of flow control algorithms being adopted in the Internet
- Lets look at one of the more interesting initiatives: Bottleneck Bandwidth and Round-Trip Time (BBR)

# Let's talk about speed

- How fast can we push a single session to move data through the network?
- Session speed is the result of a combination of:
  - available transmission speeds,
  - end-to-end latency and
  - protocol efficiency
- All of these factors are critical

# The Evolution of Speed

1980's

- TCP rates of Kilobits per second

1990's

- TCP rates of Megabits per second

2000's

- TCP rates of Gigabits per second

2010's

- TCP rates of Gigabits per second

# Today

- Optical transmission speeds are approaching Terrabit capacity
- But peak TCP session speeds are not keeping up
- What's going on?

# TCP

- The Transmission Control Protocol is an end-to-end protocol that creates a reliable stream protocol from the underlying IP datagram device
- TCP operates as an adaptive rate control protocol that attempts to operate fairly and efficiently

# TCP Design Objectives

To maintain an average flow which is **Efficient** and **Fair**

- **Efficient:**

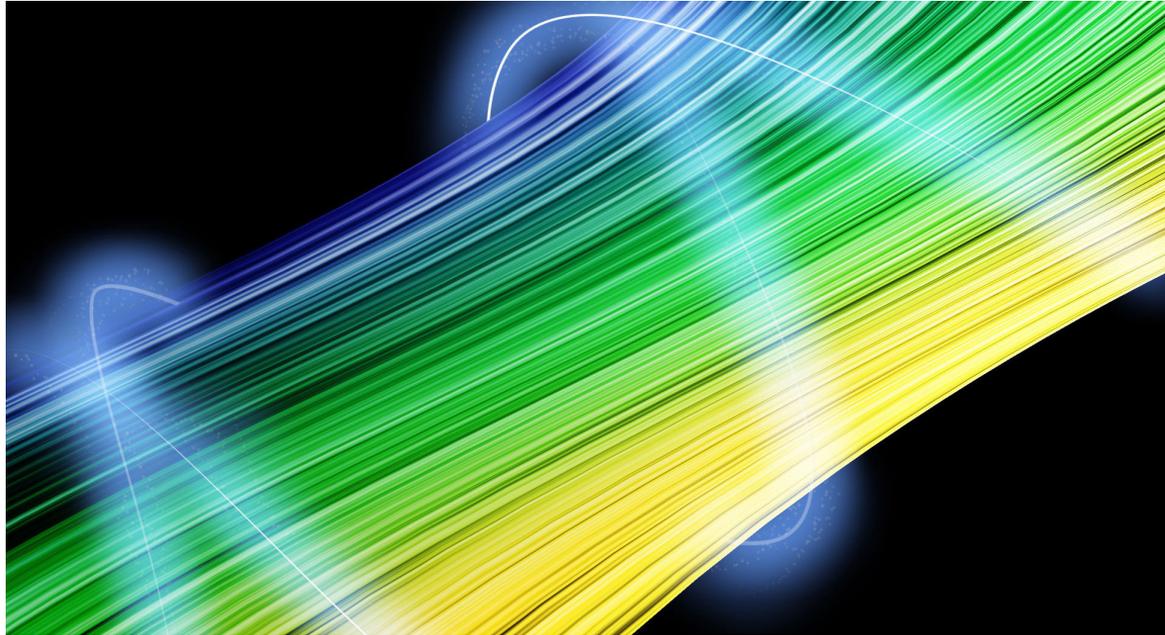
- Minimise packet loss
- Minimise packet re-ordering
- Do not leave unused path bandwidth on the table!

- **Fair:**

- Do not crowd out other TCP sessions
- Over time, take an average  $1/N$  of the path capacity when there are  $N$  other TCP sessions sharing the same path

# It's a Flow Control process

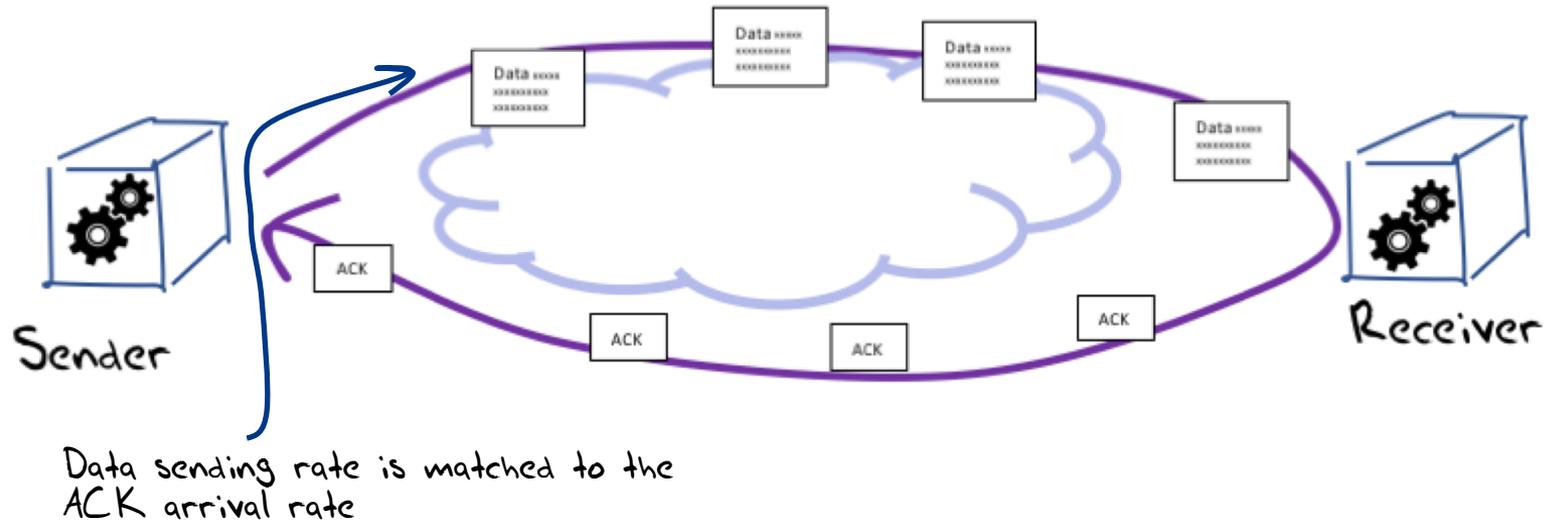
- Think of this as a multi-flow fluid dynamics problem
- Each flow has to gently exert pressure on the other flows to signal them to provide a fair share of the network, and be responsive to the pressure from all other flows



**How can we achieve this?**

# TCP Control

TCP is an *ACK Pacing* protocol



# TCP Control

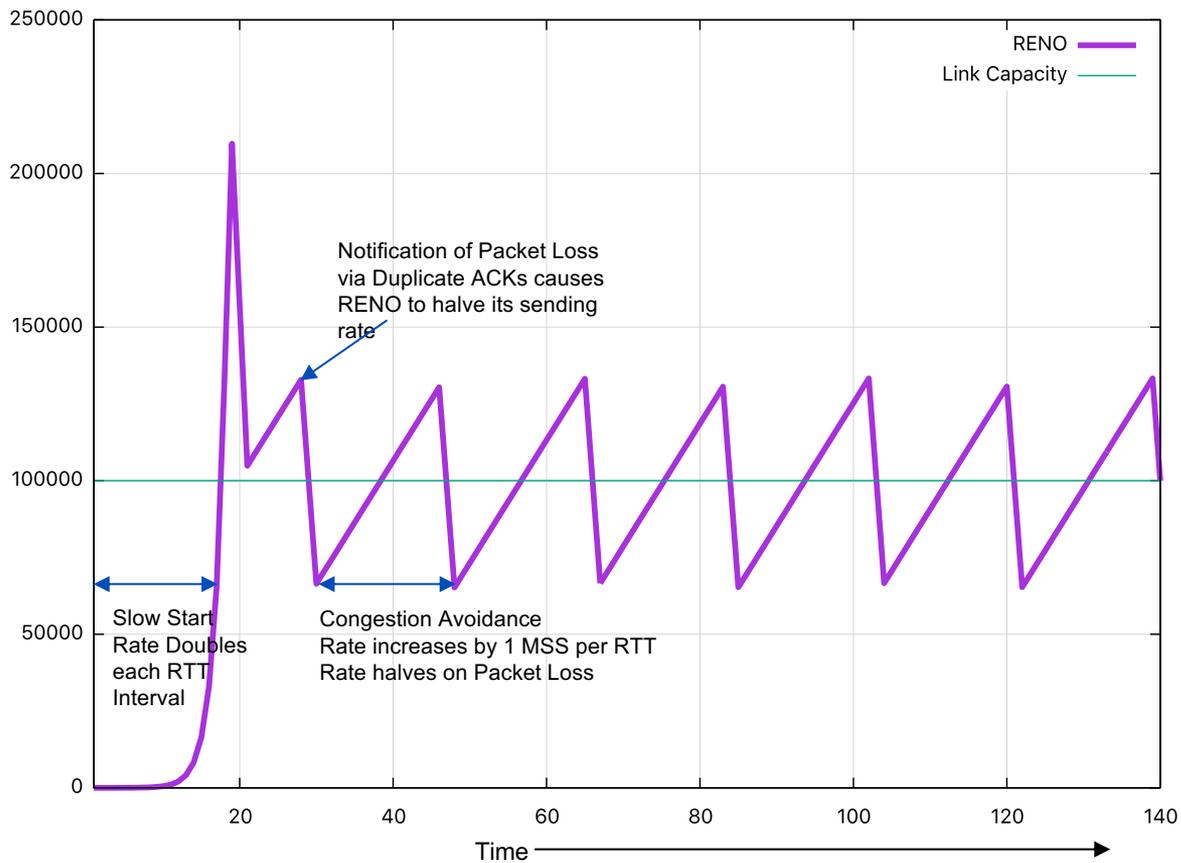
ACK pacing protocols relate to a **past** network state, not necessarily the **current** network state

- The ACK signal shows the rate of data that left the network at the receiver that occurred at  $\frac{1}{2}$  RTT back in time
- So if there is data loss, the ACK signal of that loss is already  $\frac{1}{2}$  RTT old!
  - **So TCP should react quickly to ‘bad’ news**
- If there is no data loss, that is also old news
  - **So TCP should react conservatively to ‘good’ news**

# "Classic TCP" - TCP Reno

- Additive Increase Multiplicative Decrease (AIMD)
  - While there is no packet loss, increase the sending rate by One Segment (MSS) each RTT interval
  - If there is packet loss decrease the sending rate by 50% over the next RTT Interval
- Start Up
  - Each RTT interval, double the sending rate
  - We call this “slow start” – probably because its anything but slow!!!

# Idealised TCP Reno



# Reno is too slow

- TCP Reno tries to oscillate between sending rates  $R$  and  $2 \times R$  that span the actual link capacity
- It increases its sending rate slowly so it's really lousy when trying to run at very high speed over long delay networks
- It over-corrects on loss and leaves available path capacity idle
  - 10Gbps rates over 100ms RTT demands a packet loss rate of less than 0.000003%
  - An average 1% loss rate over a 100ms RTT can't operate faster than 3Mbps

# Faster than Reno?

- Could we make TCP faster and more efficient by changing the way in which the sending rate is inflated?

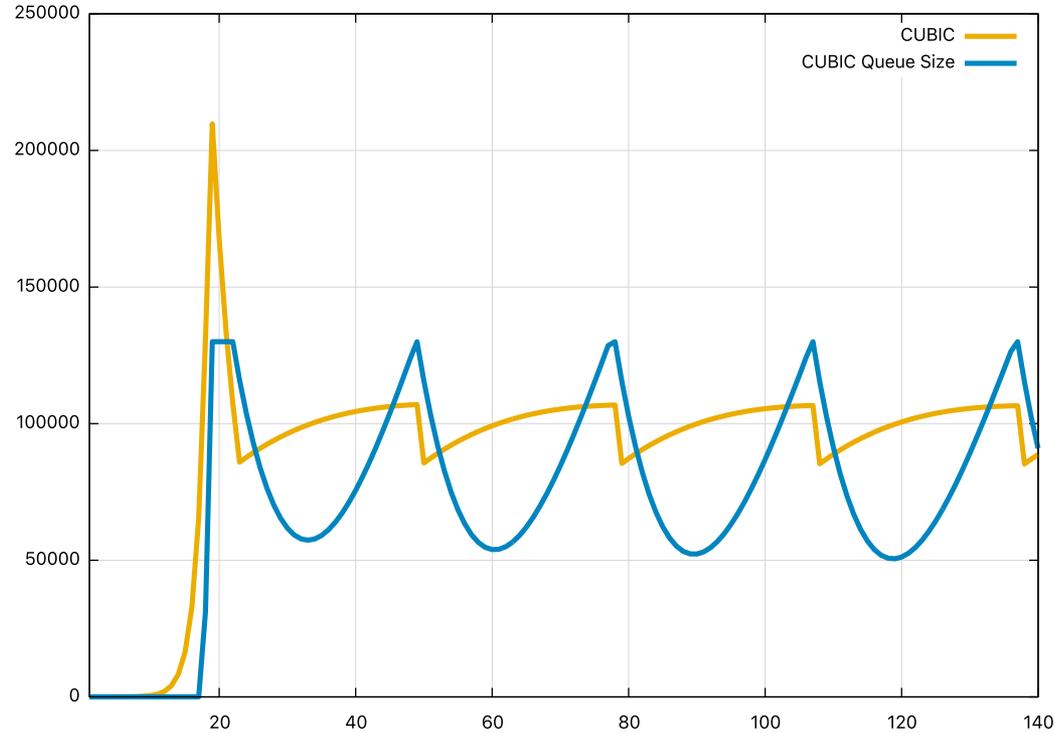
# Refinements to TCP

- There have been many efforts to alter TCP's flow control algorithm to improve on RENO
- In a loss-based control system the essential parameters are the manner of rate increase and the manner of loss-based decrease
  - For example:
    - MultTCP behaves as if it were  $N$  simultaneous TCP sessions: i.e. increase by  $N$  segments each RTT and rate drop by  $1/N$  upon packet loss
- What about varying the manner of rate increase?

# CUBIC

- CUBIC is designed to be useful for high speed sessions while still being 'fair' to other sessions and also efficient even at lower speeds
- Rather than probe in a linear manner for the sending rate that triggers packet loss, CUBIC uses a non-linear (cubic) search algorithm

# CUBIC and Queue formation



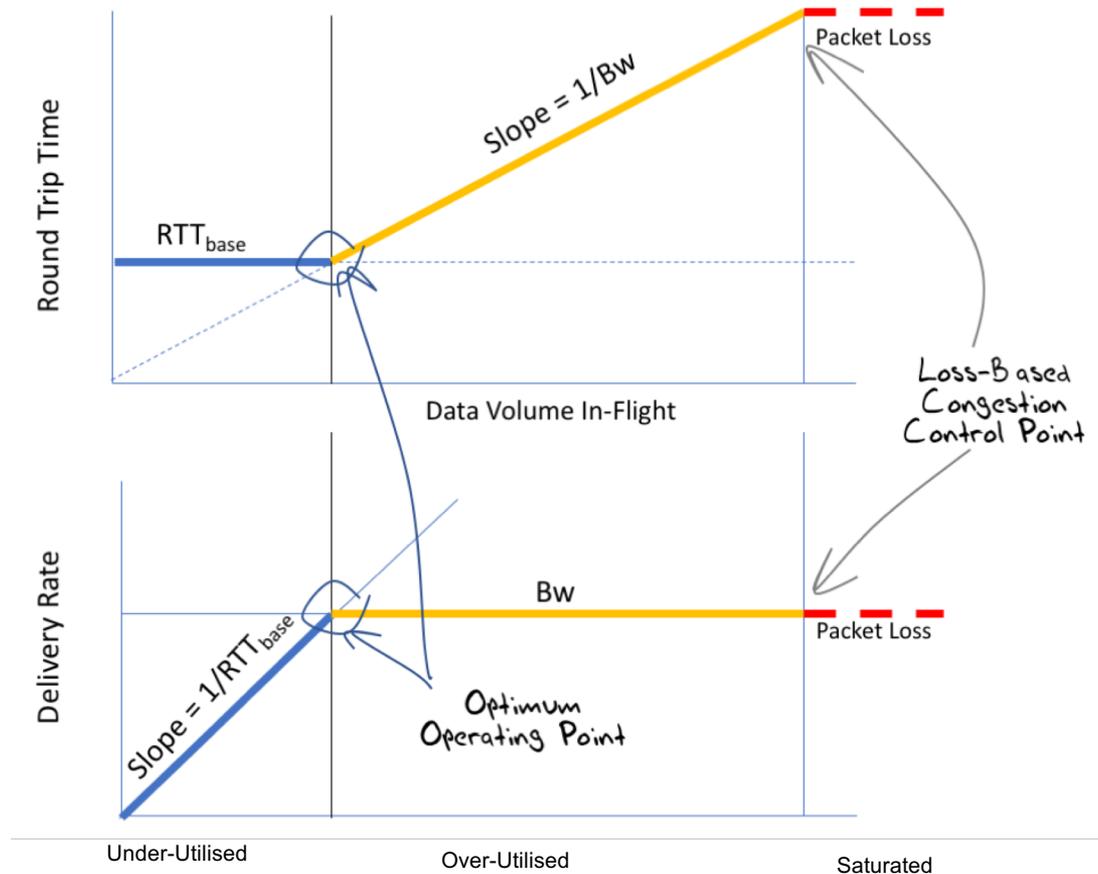
# **CUBIC is today's state of the art**

- Can react quickly to available capacity in the network
- Tends to sit for extended periods in the phase of queue formation
- Can react efficiently to long fat pipes and rapidly scale up the sending rate
- Operates in a manner that tends to exacerbate 'buffer bloat' conditions

# Can we do better?

- Lets look at the model of the network once more
- There are three 'states' of flow management in this network:
  - Under-Utilised – where the flow rate is below the link capacity and no queues form
  - Over-Utilised – where the flow rate is greater than the link capacity and queues form
  - Saturated – where the queue is filled and packet loss occurs
- Loss-based control systems probe upward to the saturation point, and back off to what they guess is the under-utilised state in order to let the queues drain
- But the optimal operational point for the flow is at the state change from Under to Over utilised

# RTT and Delivery Rate with Queuing



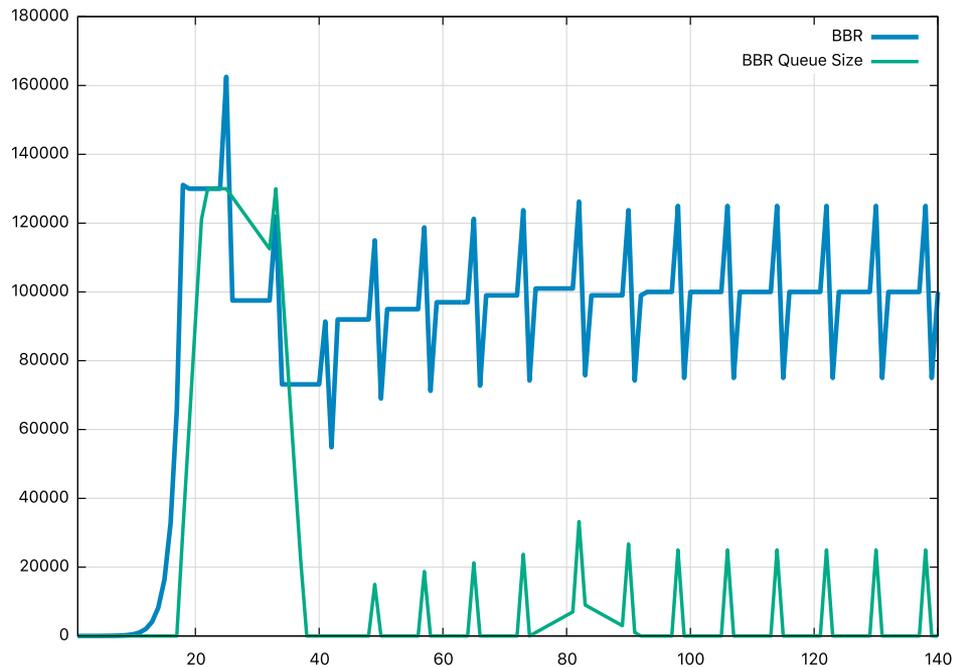
# How to detect the **onset** of queuing?

- By carefully measuring the Round Trip Time!

# BBR Design Principles

- Probe the path capacity only intermittently
- Probe the path capacity by increasing the sending rate for a short interval and then drop the rate to drain the queue:
  - If the RTT of the probe equals the RTT of the previous state then there is available path bandwidth that could be utilised
  - If the RTT of the probe rises then the path is likely to be at the onset of queuing and no further path bandwidth is available
- Do not alter the path bandwidth estimate in response to packet loss
- Pace the sending packets to avoid the need for network buffer rate adaptation

# Idealised BBR profile



sending rate

network queues

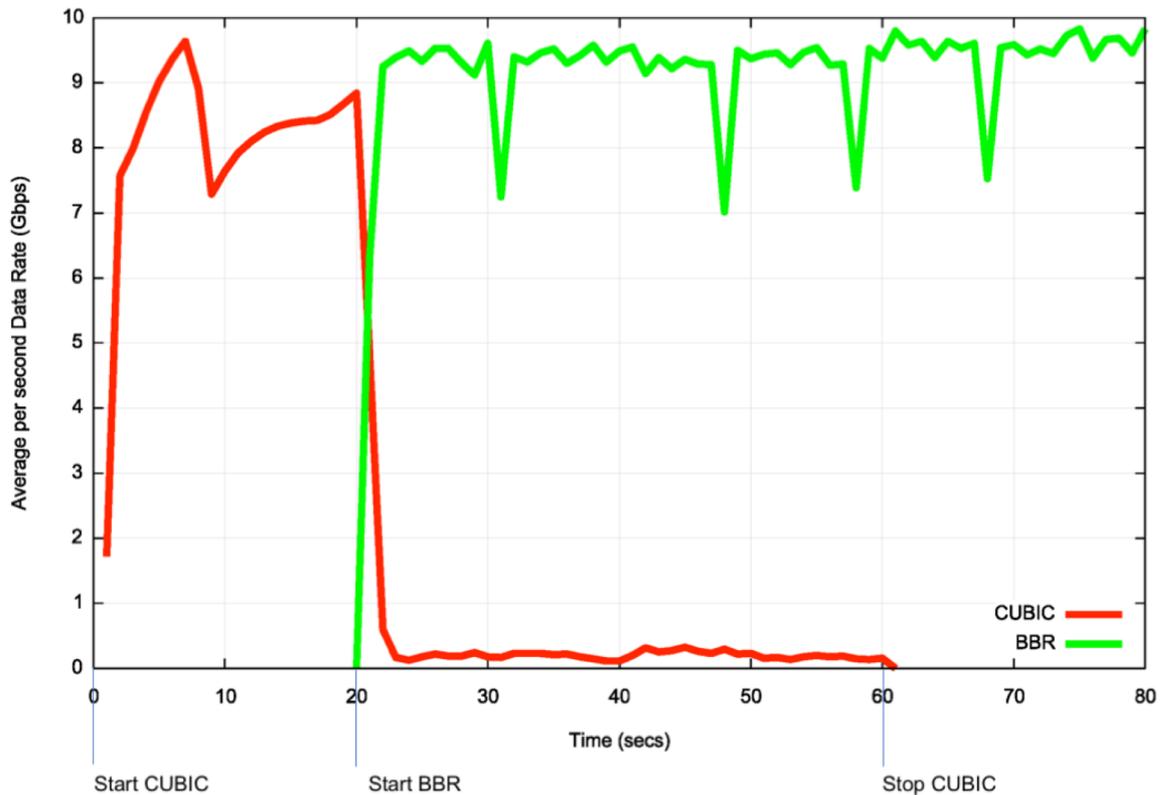
# BBR Politeness?

- BBR will probably not constantly pull back when simultaneous loss-based protocols exert pressure on the path's queues
- BBR tries to make minimal demands on the queue size, and does not rely on a large dynamic range of queue occupancy during a flow

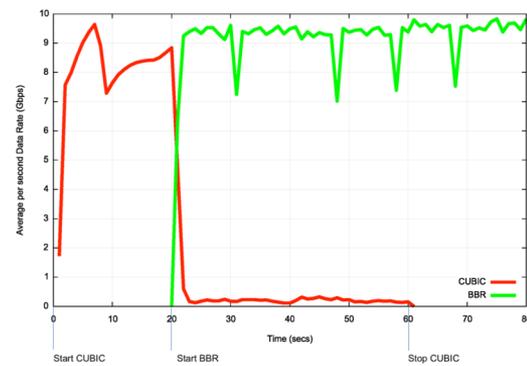
# From Theory to Practice

- Lets use BBR in the wild
- I'm using iperf3 on Linux platforms (Linode)
  - The platforms are dedicated to these tests
- It's the Internet
  - The networks paths vary between tests
  - The cross traffic is highly variable
  - No measurement is repeatable to a fine level of detail

# Cubic vs BBR over a 12ms RTT 10G circuit



# Wow!



- That was BRUTAL!
- As soon as BBR started up it collided with CUBIC, and BBR startup placed pressure on CUBIC such that CUBIC's congestion window was reduced close to zero
- At this stage CUBIC's efforts to restart its congestion window appear to collide with BBR's congestion control model, so CUBIC remains suppressed
  - The inference is that BBR appears to be operating in steady state with a relatively full network queue in order to crowd out CUBIC

# BBR vs Cubic - second attempt

```
gih@wally:~$ traceroute testbed-de.rand.apnic.net
traceroute to testbed-de.rand.apnic.net (172.104.147.241), 30 hops max, 60 byte packets
 1 202.158.221.221 (202.158.221.221) 0.412 ms 0.410 ms 0.401 ms
 2 et-0-3-0.pe1.rsby.nsw.aarnet.net.au (113.197.15.10) 4.290 ms 4.294 ms 4.312 ms
 3 113.197.15.157 (113.197.15.157) 4.265 ms 4.273 ms 4.332 ms
 4 xe-0-2-4.bdr1.a.sjc.aarnet.net.au (202.158.194.162) 150.825 ms 150.828 ms 150.823 ms
 5 208.185.52.77.available.above.net (208.185.52.77) 150.930 ms 150.931 ms 150.926 ms
 6 ae12.cr1.sjc2.us.zip.zayo.com (64.125.25.21) 151.439 ms 151.165 ms 151.186 ms
 7 ae16.mpr3.sjc7.us.zip.zayo.com (64.125.31.13) 155.216 ms 151.696 ms 151.594 ms
 8 ix-ae-5-0.tcore1.SQN-San-Jose.as6453.net (63.243.205.9) 151.945 ms 151.966 ms 151.96
 9 if-ae-12-2.tcore1.NTO-New-York.as6453.net (63.243.128.28) 304.046 ms 304.074 ms 304.
10 if-ae-7-2.tcore1.NOV-New-York.as6453.net (63.243.128.26) 292.651 ms 292.585 ms 292.7
11 if-ae-2-2.tcore2.NOV-New-York.as6453.net (216.6.90.22) 294.608 ms if-ae-32-2.tcore2.LD
12 if-ae-15-2.tcore2.L78-London.as6453.net (80.231.131.117) 296.843 ms 296.812 ms if-ae-
13 if-ae-14-2.tcore2.AV2-Amsterdam.as6453.net (80.231.131.161) 301.732 ms 301.484 ms 29
14 if-ae-2-2.tcore1.AV2-Amsterdam.as6453.net (195.219.194.5) 293.261 ms if-ae-3-2.tcore1.
15 if-ae-11-2.tcore1.PVU-Paris.as6453.net (80.231.153.49) 308.666 ms if-ae-6-2.tcore1.FNM
16 if-ae-9-2.tcore2.FNM-Frankfurt.as6453.net (195.219.87.13) 307.218 ms if-ae-2-2.thar1.F
17 195.219.61.30 (195.219.61.30) 287.745 ms if-ae-14-3.thar1.F2C-Frankfurt.as6453.net (19
18 139.162.129.2 (139.162.129.2) 289.321 ms 139.162.129.11 (139.162.129.11) 294.369 ms 1
19 li1663-241.members.linode.com (172.104.147.241) 303.581 ms 303.558 ms 139.162.129.15
```

Same two endpoints, same network path across the public Internet

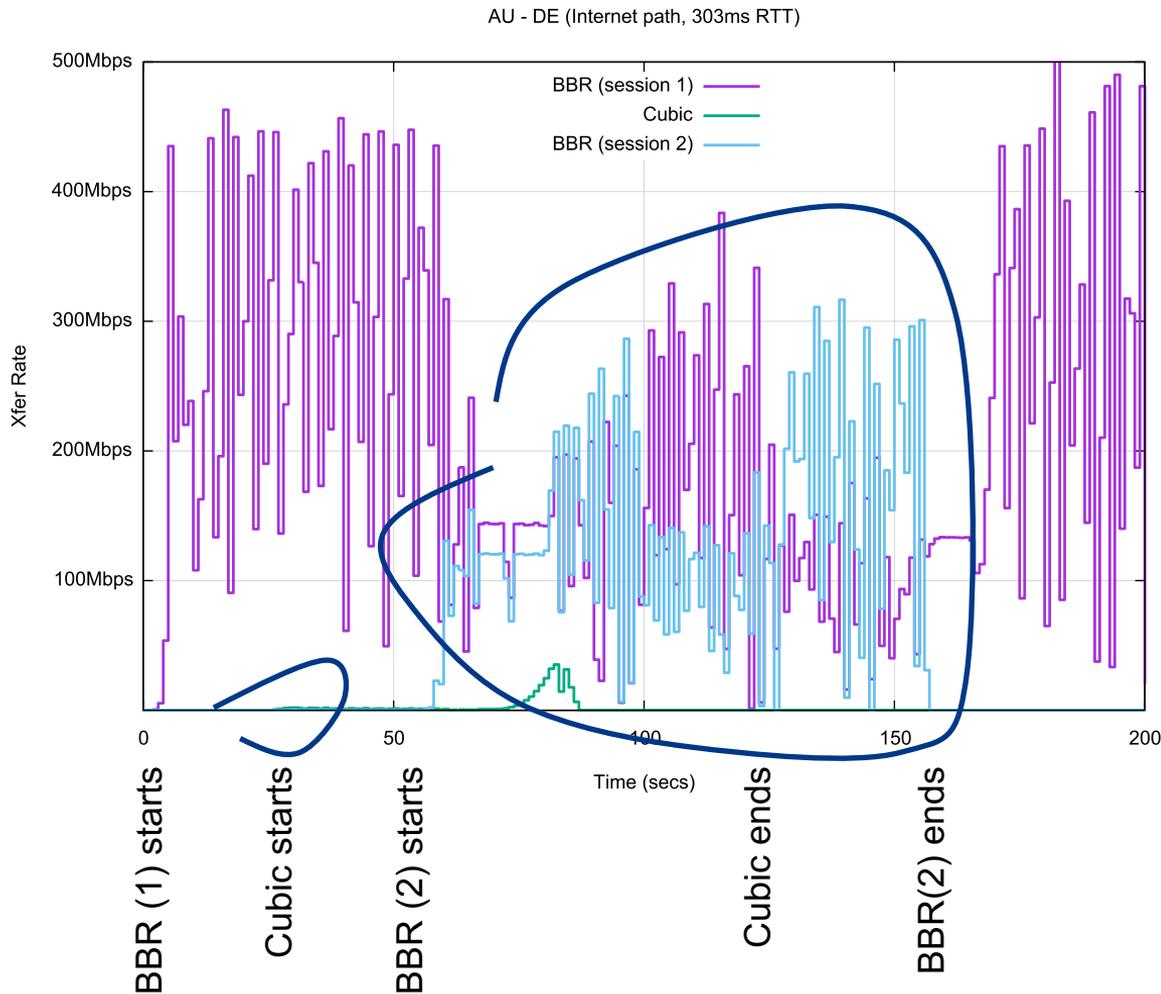
Using a long delay path AU to Germany via the US

# BBR vs Cubic

The Internet is capable of offering a 400Mbps capacity path on demand!

In this case BBR is apparently operating with filled queues, and this crowds out CUBIC

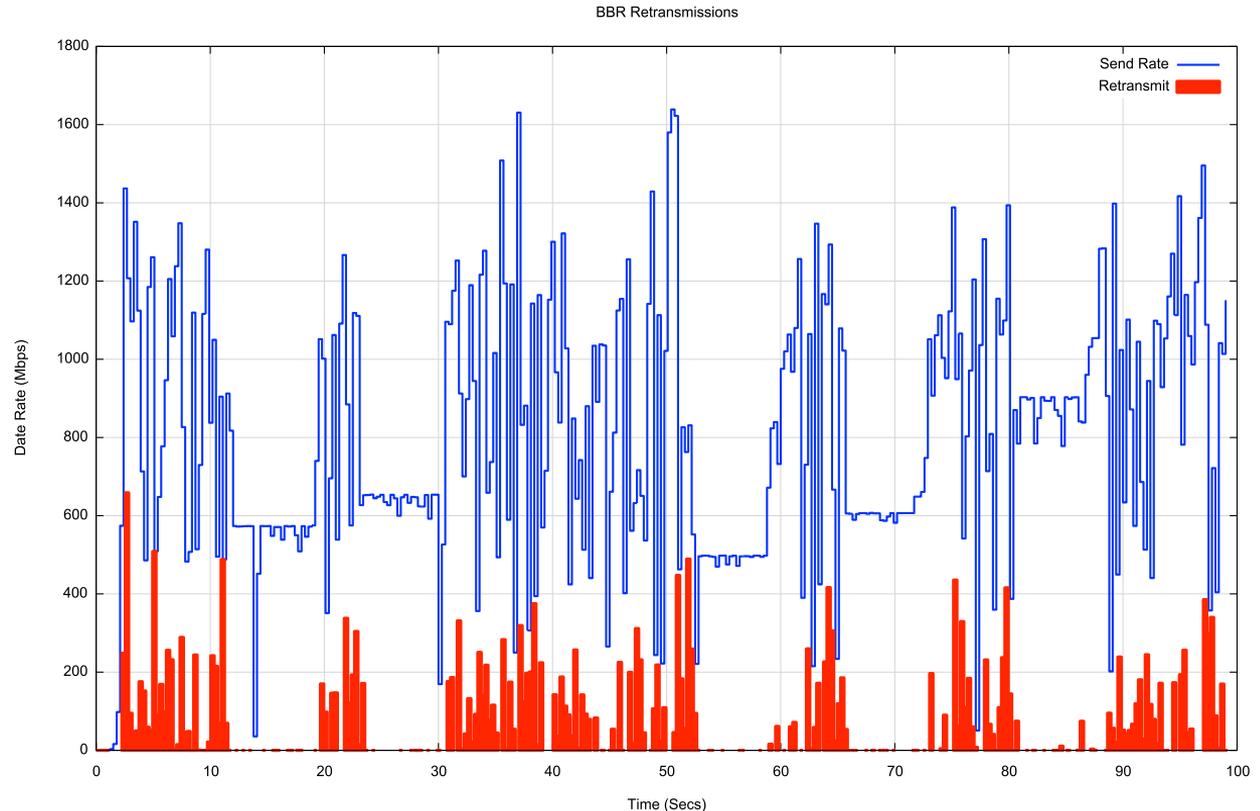
BBR does not compete well with itself, and the two sessions oscillate in getting the majority share of available path capacity



# BBR and Loss Recovery

Packet loss causes retransmission that appears to occur in addition to the stable link capacity model used by BBR.

Once loss is reduced, BBR maintains a more consistent sending model



# So what can we say about BBR?

It's "interesting" in so many ways:

- It's a move away from the more common loss-based flow control protocols
- It looks like it will operate very efficiently in a high-speed small-buffer world
  - High speed small buffer chips are way cheaper, but loss-based TCP reacts really badly to small buffers by capping its flow rate
- It will operate efficiently over ECMP paths, as it is relatively impervious to packet re-ordering
- It also looks as if it will operate efficiently in rate policed environments
- Unlike AIMD systems, it will scale from Kbps to Gbps over long delay paths very efficiently
- It resists the conventional network-based traffic control mechanisms

# Why use BBR?

- Because it achieves
- Its incredibly efficient
- It makes minimal demands on network buffer capacity
- It's fast!

# Why **not** use BBR?

- Because it **over achieves!**
- The classic question for many Internet technologies is scaling
  - “what if everyone does it?”
  - BBR is not a scalable approach
  - It works so well while it is used by just a few users, some of the time
  - But when it is active, BBR has the ability to slaughter concurrent loss-based flows
  - Which sends all the wrong signals to the TCP ecosystem
    - The loss-based flows convert to BBR to compete on equal terms
    - The network is then a BBR vs BBR environment, which is unstable

# Is this BBR experiment a failure?

Is it just too 'greedy' and too 'insensitive' to other flows to be allowed out on the Internet to play?

- Many networks have been provisioned as a response to the aggregate behaviours of loss-based TCP congestion control
- BBR changes all those assumptions, and could potentially push many networks into sustained instability
- We cannot use the conventional network control mechanisms to regulate BBR flows
  - Selective packet drop just won't create back pressure on the flow

# Where now?

## BBR 2.0

- Alter BBR's 'sensitivity' to loss rates, so that it does not persist with an internal bandwidth delay product (BDP) that exceeds the uncongested BDP  
This measure would moderate BBR 1.0's ability to operate for extended periods with very high loss levels
- Improve the dynamic sharing fairness by moderating the Bandwidth Delay Product by using an estimated 'fair' proportion of the path BDP
- Accommodate the signal distortion caused by ACK stretching middleware
- Place an upper bound on the volume of in-flight data
- Alter the +/- 25% probe factors dynamically (i.e. allow this to be less than 25% overload)

# What about QUIC?

- Google has also announced QUIC as well as BBR
- QUIC essentially “hides” the TCP control settings from the underlying network using encryption
- Network middleware and control measures are not very effective when the transport control parameters are deliberately obscured

# The new Network Architecture

- We are seeing a shift away from network-level active middleware back towards edge-centric control in the Internet
- QUIC and BBR are instances of a recent push back from the network-level QoS bandwidth control mechanisms, and result in greater levels of autonomous control being passed back to the end hosts

# What is all this telling us?

- The Internet still contains a large set of important unsolved problems
- Moving large data sets over high speed networks requires a different approach to what we are doing today
- BBR seems to be a big step in a useful direction for very high speed terrabit networking
- But it still calls for more research and more testing at scale
- And that's where research networks can help!

That's it!



Questions?