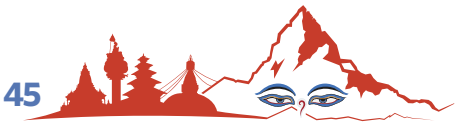
TCP and BBR

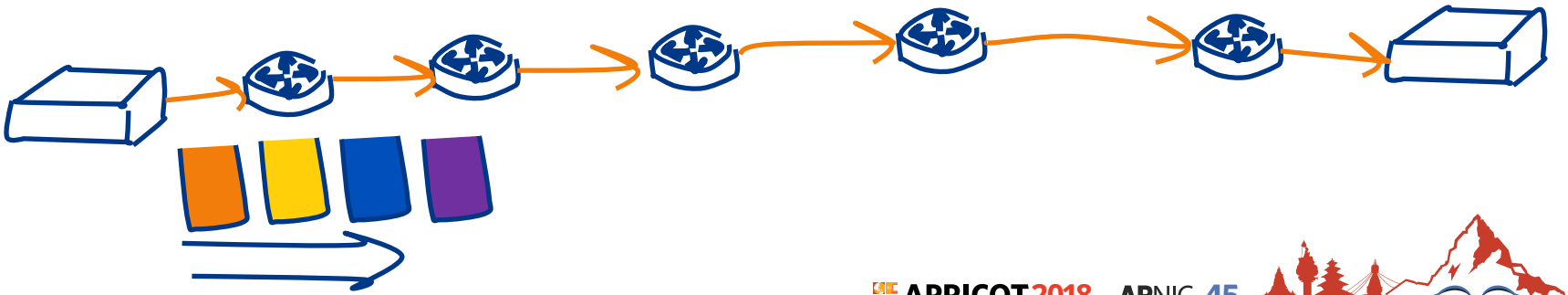
Geoff Huston
APNIC



The IP Architecture

At its heart IP is a datagram network architecture

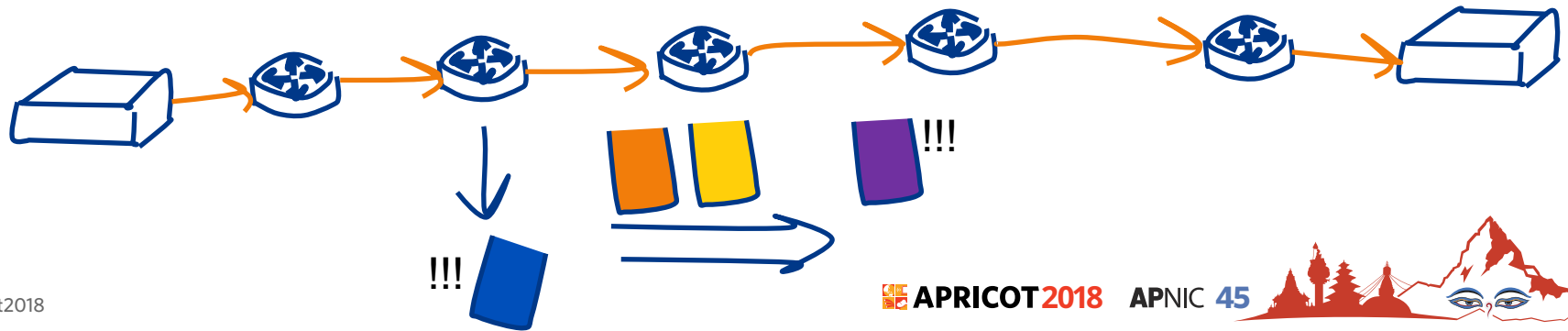
- Individual IP packets may be lost, re-ordered, re-timed and even fragmented



The IP Architecture

At its heart IP is a datagram network architecture

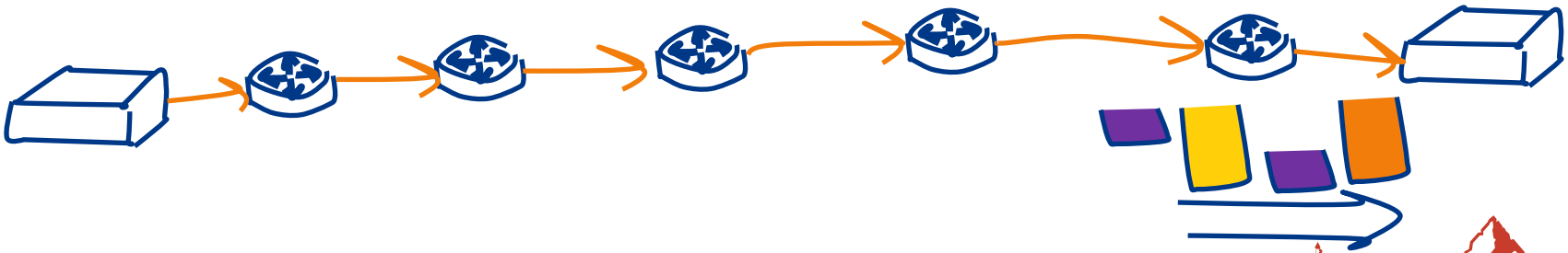
– Individual IP packets may be lost, re-ordered, re-timed and even fragmented



The IP Architecture

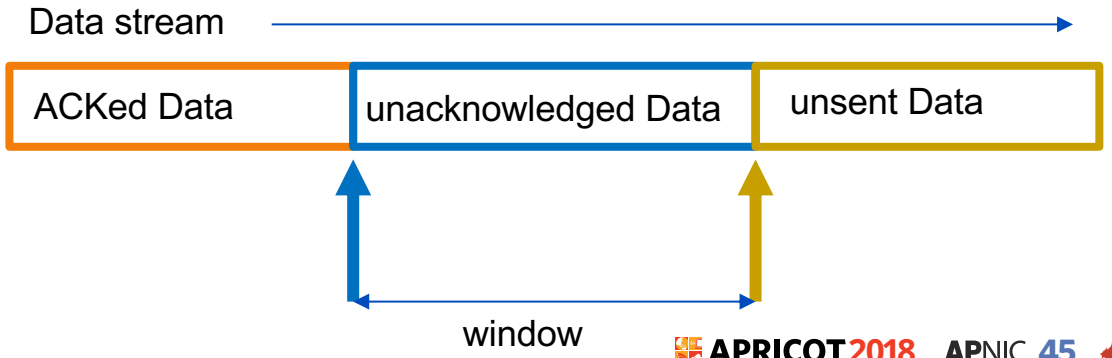
At its heart IP is a datagram network architecture

- Individual IP packets may be lost, re-ordered, re-timed and even fragmented



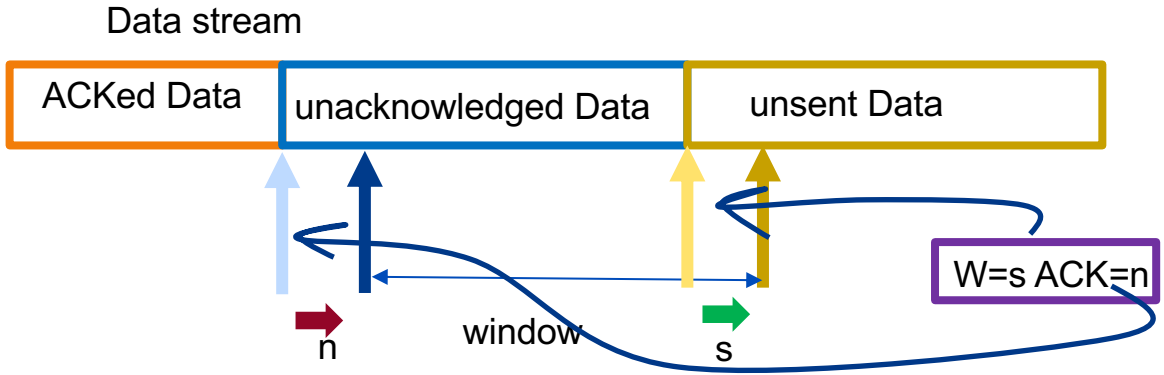
TCP

- The Transmission Control Protocol is an END-TO-END protocol that creates a reliable stream protocol from the underlying IP datagram device
- TCP uses a **sliding window** flow control protocol to manage the data flow

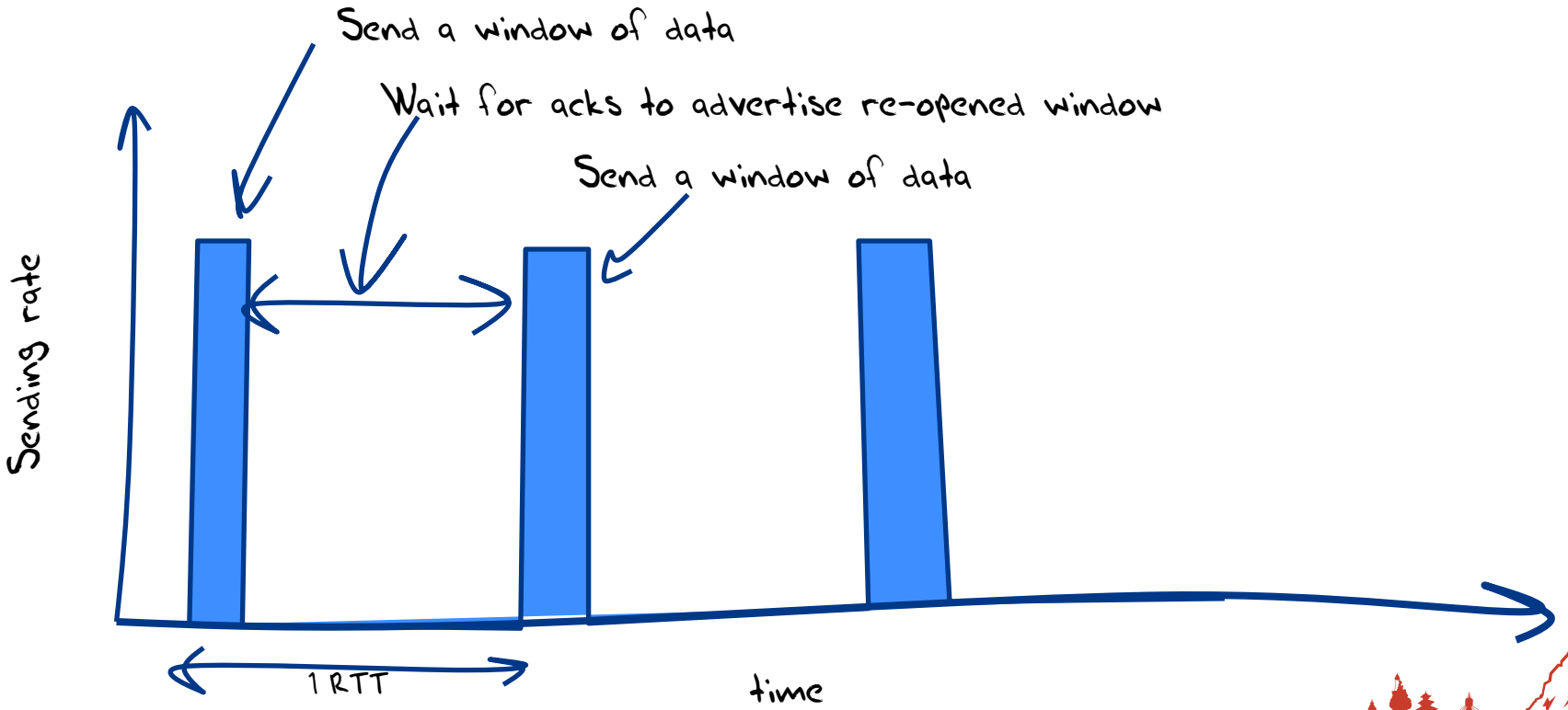


TCP Sliding Window

- Each ACK advertises the sequence number of the last “good” received byte and the available buffer size in the receiver
- The sender can send up to a window size of data before pausing for a window update

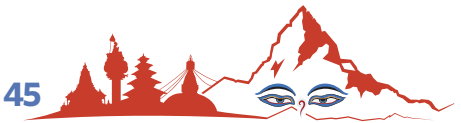


Sliding Window protocols tend to be "bursty"



Flow Control

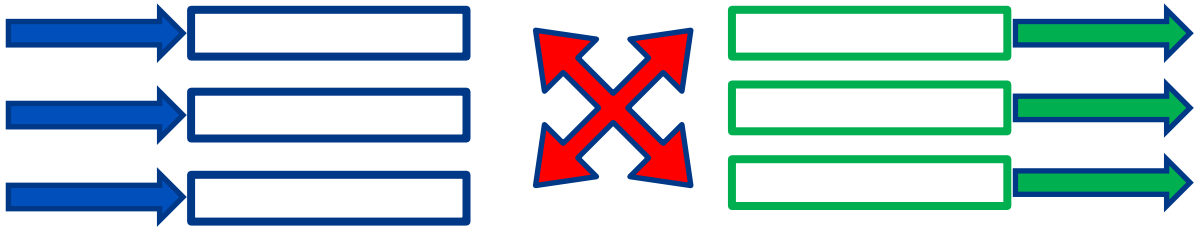
- While TCP could send up to one window of data into the network as fast as it can, it is possible/likely that this would flood the network and generate packet loss
- The question is then how to regulate TCP's sending behaviour so that it sends as much data as it can while avoiding network packet loss
- So lets look at networks…



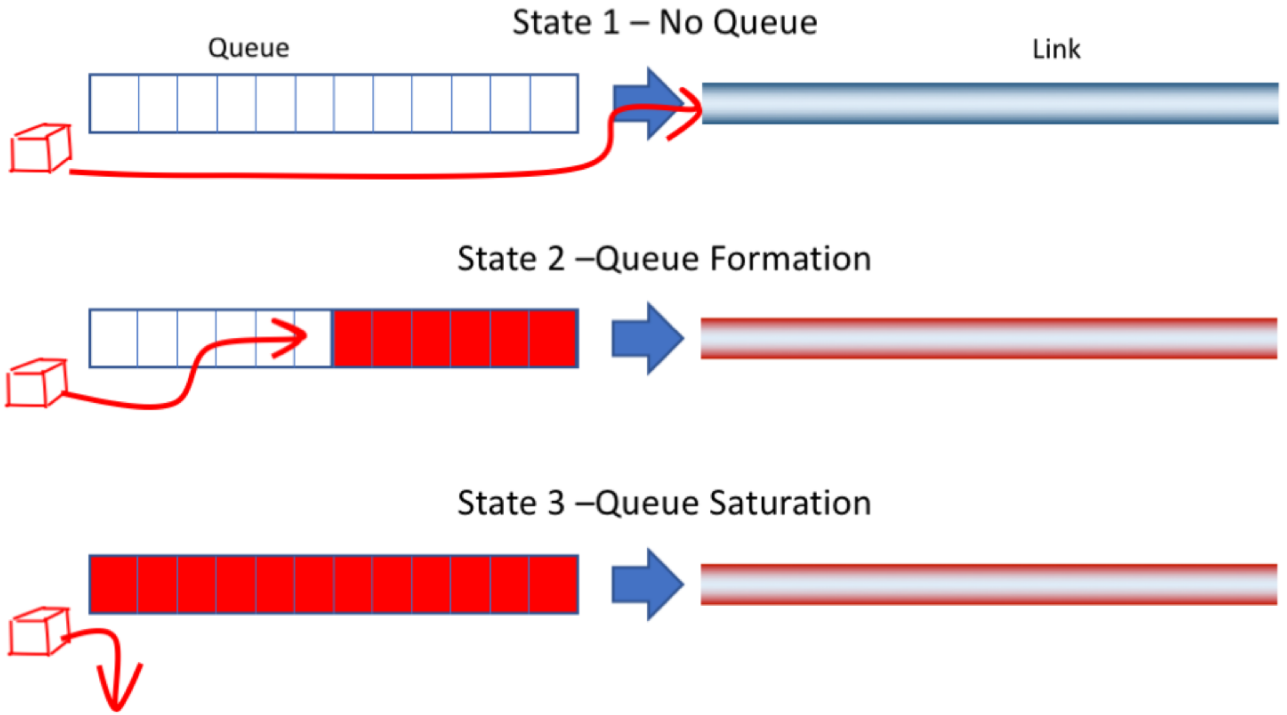
Network Considerations

Networks routers are constructed of **links** and **routers**

- **Links** are a constant delay lossless pipe
- **Routers** are a buffered switch
  - Buffers within the router constrain the total amount of data that can be held in the network



Queue Formation in Network Buffers



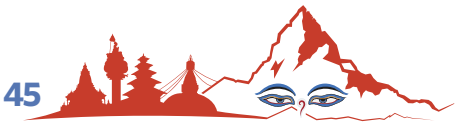
Buffer Considerations

Buffers play the role of **multiplexing adaptors**

- If two packets arrive at the same instant, one packet is queued in a buffer while the other is being services

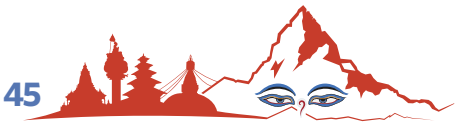
Buffers also play the role of **rate adaptation**

- But only from fast to slow, and only in a limited role!



Buffer Considerations

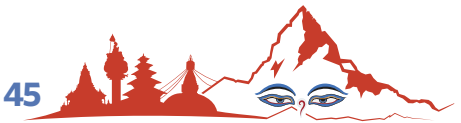
- When the queue fills then incoming packets are dropped – so larger buffers are better to reduce the incidence of queue drop!
- When packets spend time in the queue it adds to the additional time this packets spends in transit (this delay variation is called *jitter*) – too much imposed jitter is bad, so smaller queues are better!
- **A reasonable rule of thumb for IP routers is to use a buffer size equal to the *delay bandwidth product* of the next hop link**



TCP Design Objectives

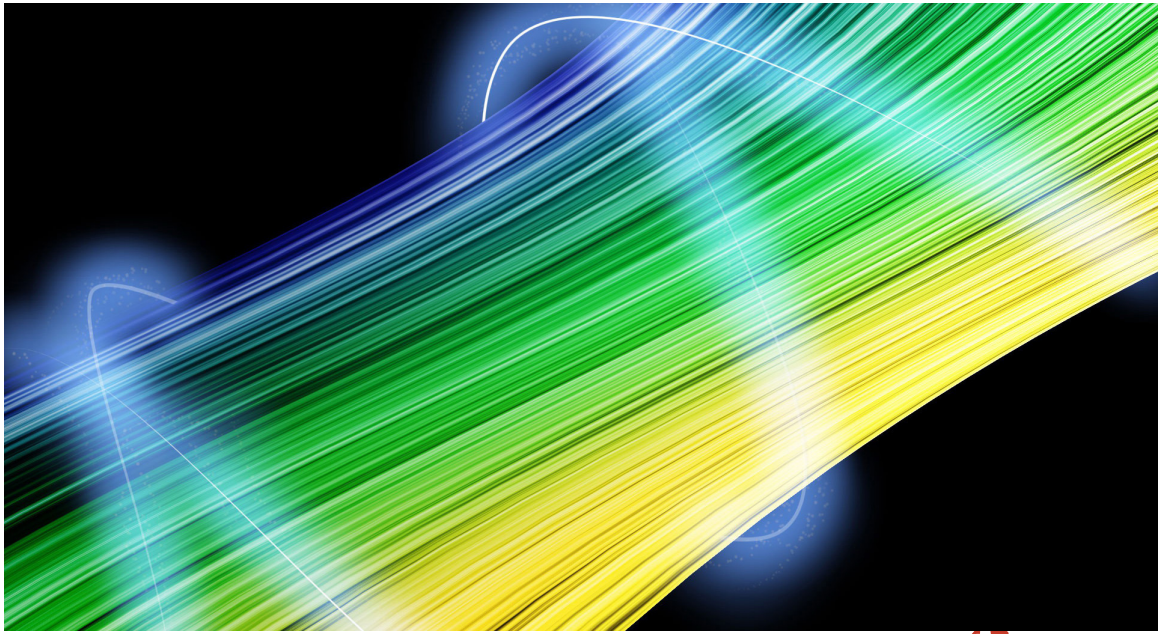
To maintain an average flow which is **Efficient** and **Fair**

- **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

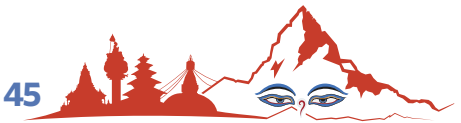


It's a Flow Control process

- 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

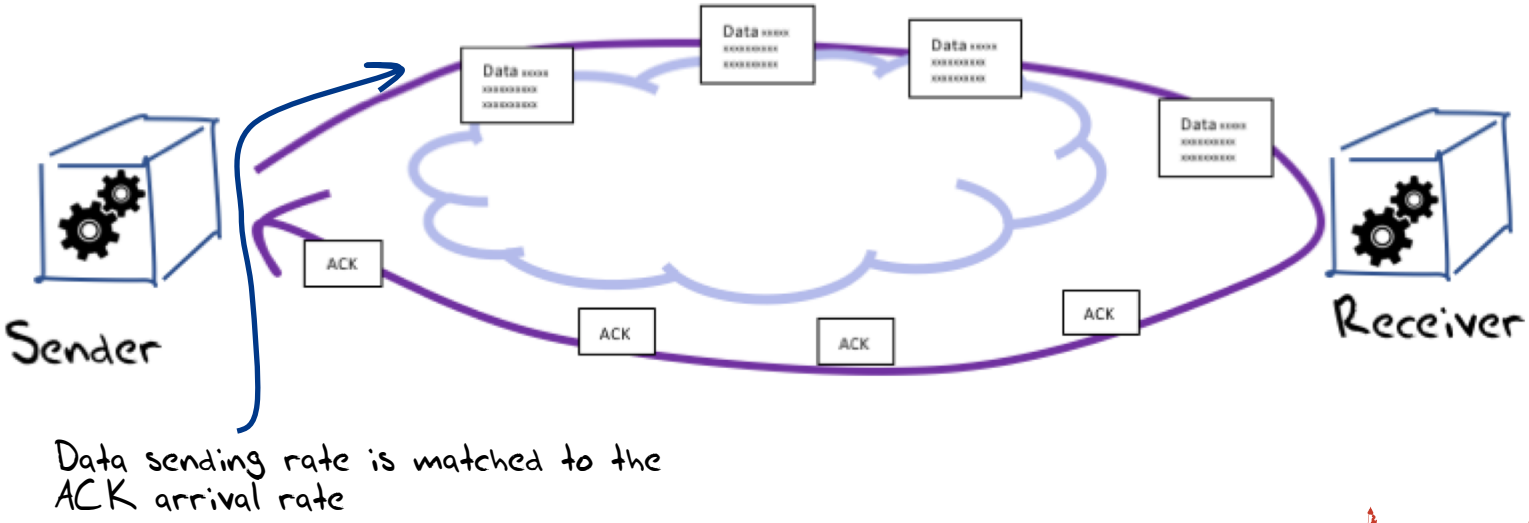


How can we achieve this?



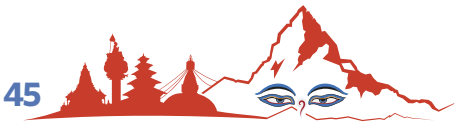
A few more observations about TCP

- TCP is an ***ACK Pacing*** protocol



A few more observations about TCP

- TCP is an ***ACK Pacing*** protocol
  - Each received ACK tells the sender how many bytes of data were received by the remote receiver – which is the same as the number of bytes that *left* the network
  - If a sender paces its data to ensure that the same number of bytes *enters* the network, then it will maintain a steady rate of network "pressure" assuming a constant capacity network path
  - If the sender sends more data into the network than is spanned by the ACK then it increases its data rate and its network pressure
  - Similarly if it sends less than the ACK span then it decrease its sending rate and its network pressure

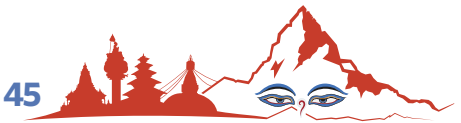


A few more observations about TCP

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 ½ RTT back in time

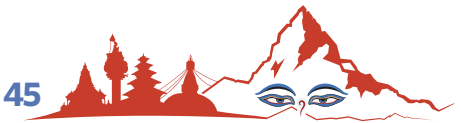
- So if there is data loss, the ACK signal of that loss is already ½ RTT old!
  - **So TCP should react quickly to 'bad' news**
- If there is no data loss, that is also old news
  - **So TCP should react conservatively to 'good' news**



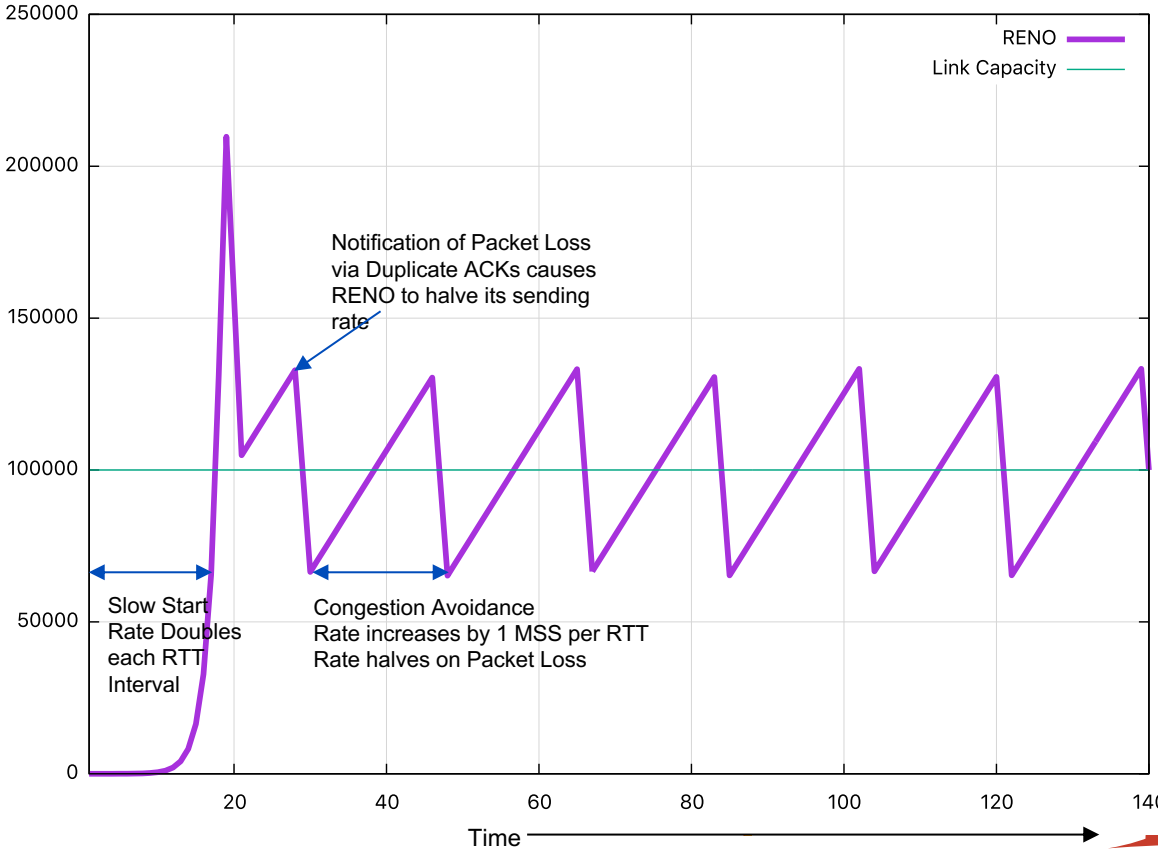
"Classic TCP" - TCP Reno

- 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% each RTT Interval

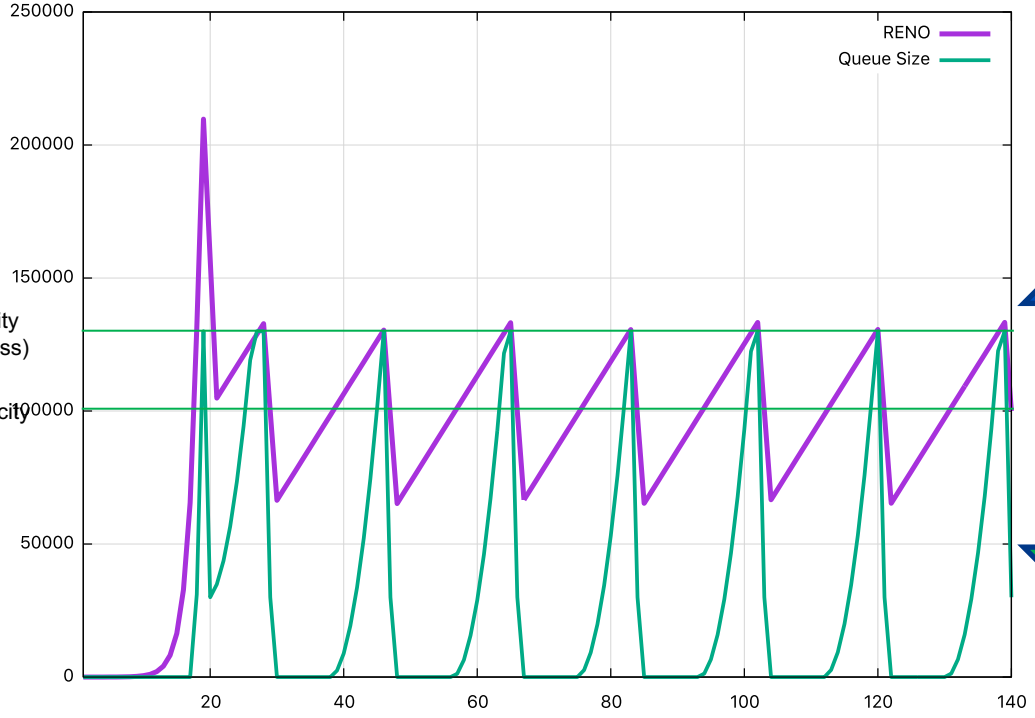
- Start Up
  - Each RTT interval, double the sending rate
  - We call this “slow start” – probably because its anything but slow!!!



Idealised TCP Reno



TCP RENO and Idealized Queue Behaviour



Total Queue Capacity
(Onset of Packet Loss)

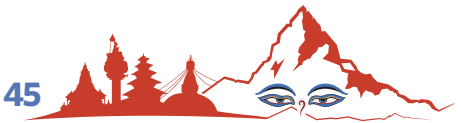
Link Capacity Capacity
(Onset of Queuing)

Network Buffers Fill

Network Buffers Drain

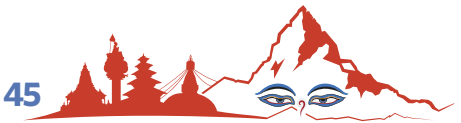
Reno is "coarse"

- TCP Reno tries to oscillate between sending rates *R* and *2 x R* that span the 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%
  - A more common average 1% loss rate over a 100ms RTT maxes AIMD to 3Mbps



Reno is too "coarse"

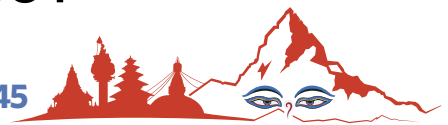
- Could we make TCP faster and more efficient by changing the way in which the sending rate is inflated?



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:

    MulTCP behaves as it 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

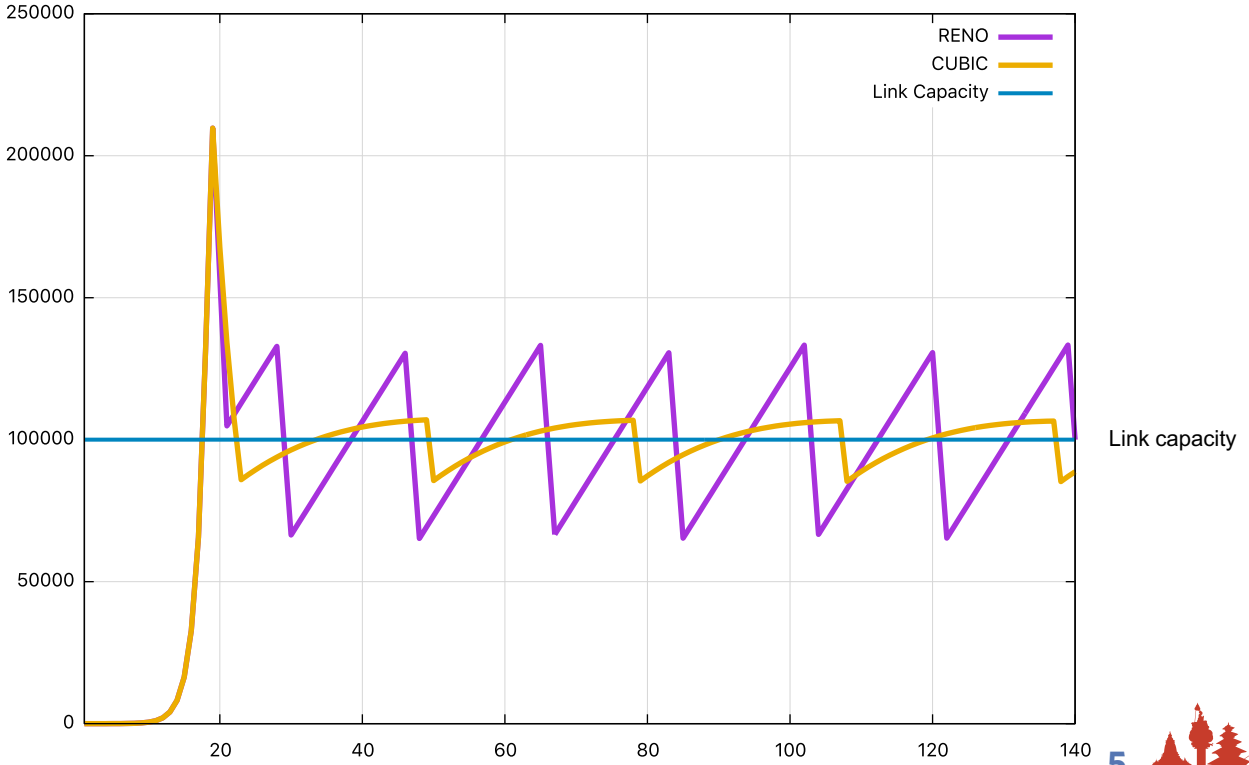
- CUBIC is designed to be useful for high speed sessions while still being ‘fair’ to other sessions and also efficient even at lower speeds
- Rather than probe in a linear manner for the sending rate that triggers packet loss, CUBIC uses a non-linear (cubic) search algorithm

$$W_{cubic} = C(t-K)^3 + W_{\max}$$

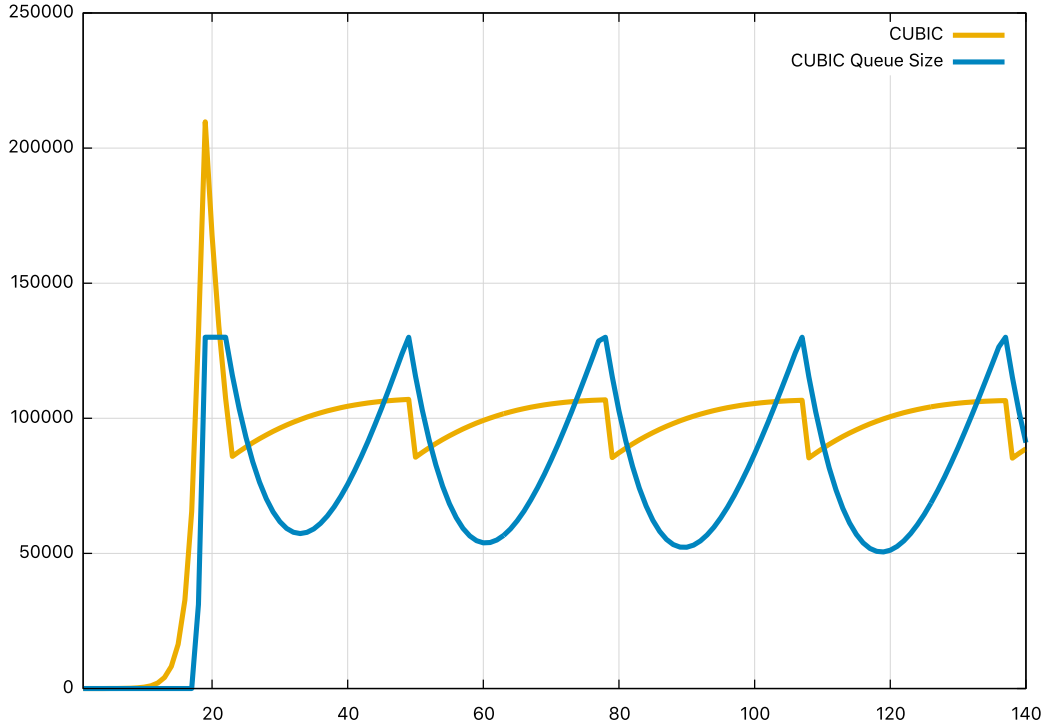
$$K = \sqrt[3]{W_{\max}\beta/C}$$

C is a scaling factor
ß is the window deflation factor,
t is the time since the most recent window deflation
W_{max} is the window size prior to the window deflation

Idealized CUBIC operation

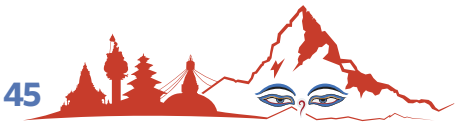


CUBIC and Queue formation



CUBIC

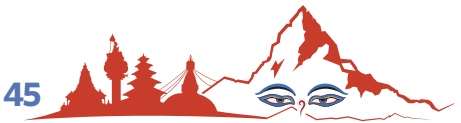
- Can react quickly to available capacity in the network
- Tends to sit for extended periods in the phase of queue formation
- Can react efficiently to long fat pipes and rapidly scale up the sending rate
- Can operate in a manner that tends to exacerbate ‘buffer bloat’ conditions



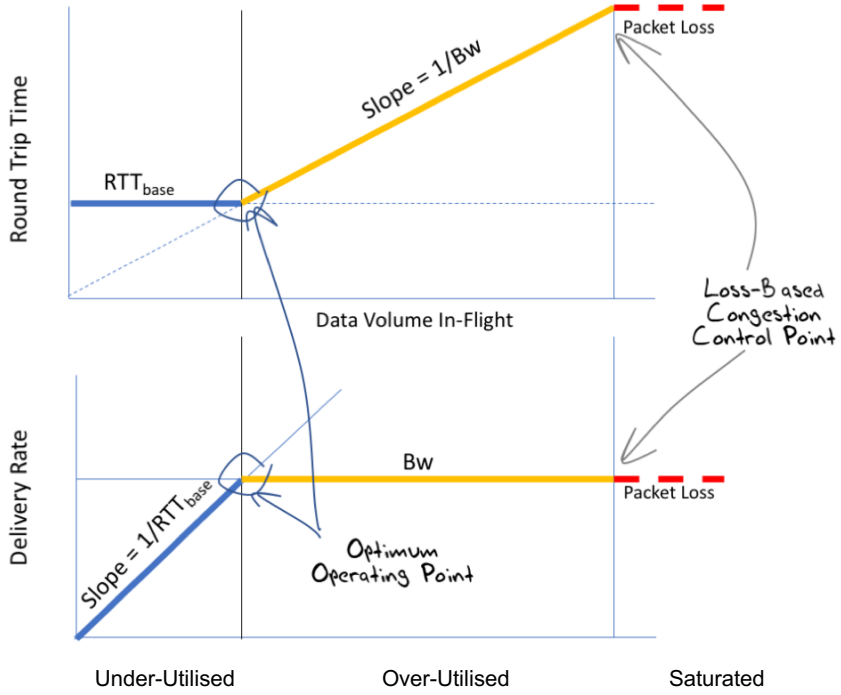
Can we do better?

- 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 that 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 the let the queues drain
- But the optimal operational point for the flow is at the state change from Under to Over utilised

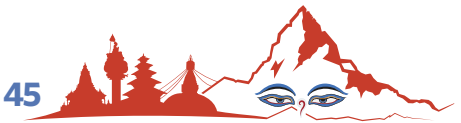


RTT and Delivery Rate with Queuing



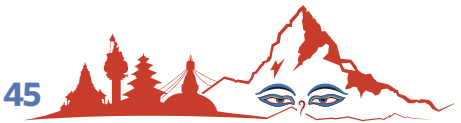
How to detect the onset of queuing?

- By carefully measuring the Round Trip Time!



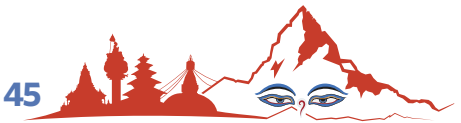
BBR Design Principles

- Probe the path capacity intermittently
- Probe the path capacity by increasing the sending rate for a short interval:
  - 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

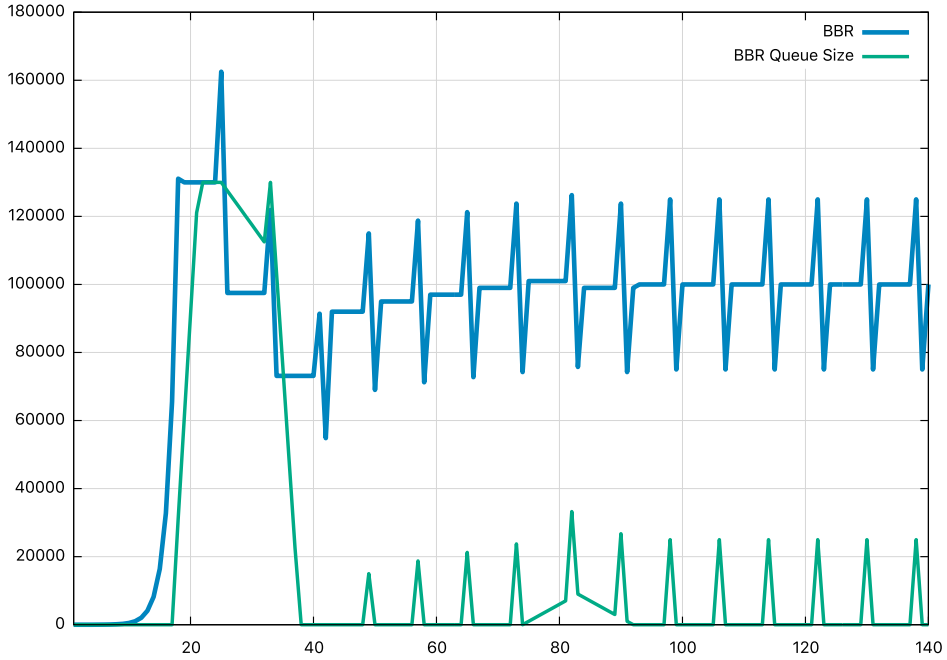


BBR

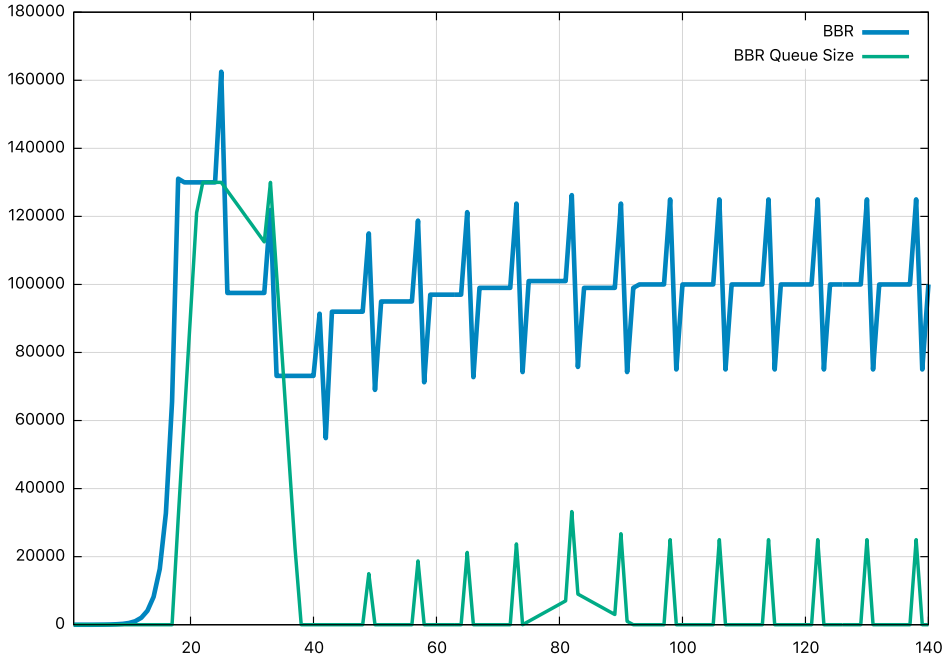
- Maintain a long term stable estimate of path RTT and a shorter updated estimate of the bottleneck capacity of the path
- Probe these estimates regularly, but not continuously
  - Maintain the sending rate at this estimated bottleneck capacity rate for 6 RTT intervals
  - For the next RTT, raise the sending rate by 25% and sample the RTT
  - If the RTT increased across this probe, then for the next RTT drop the sending rate by 25%
  - Otherwise assume this is the new bottleneck bandwidth



Idealised BBR profile



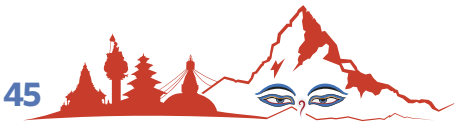
Idealised BBR profile



i'm not sure i have the queue size profile right in this simulation

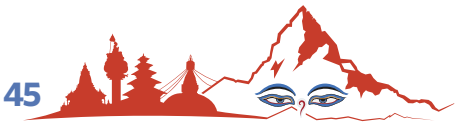
BBR Politeness?

- 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

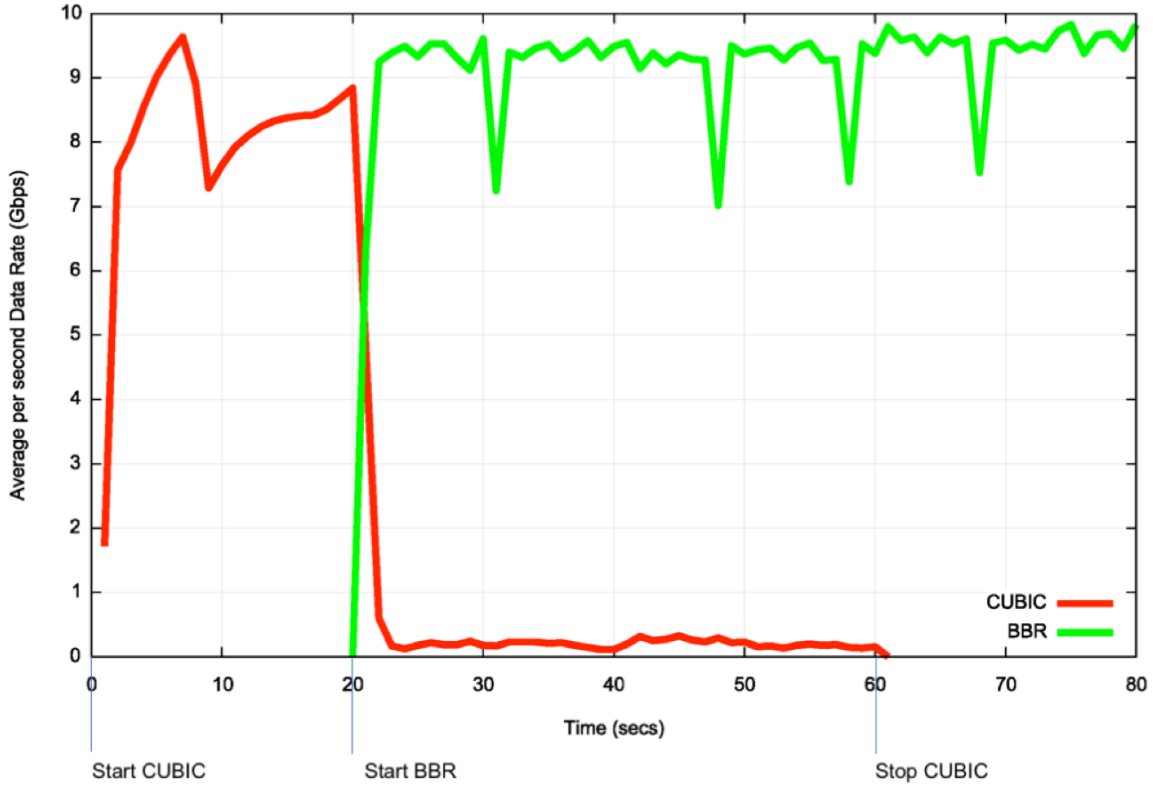


From Theory to Practice

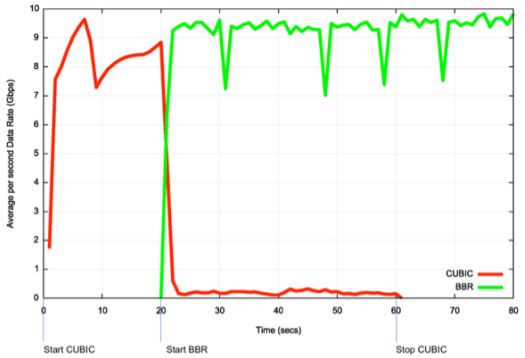
- Lets use BBR in the wild
- I'm using iperf3 on Linux platforms (Linode)
  - The platforms are dedicated to these tests
- It's the Internet
  - The networks paths vary between tests
  - The cross traffic is highly variable
  - No measurement is repeatable to a fine level of detail
- These are long pipes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



Cubic vs BBR over a 12ms RTT 10G circuit



Wow!



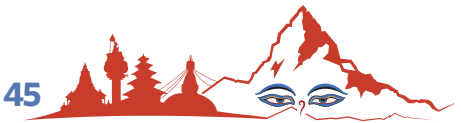
- That was BRUTAL!
- As soon as BBR started up it collided with CUBIC, and BBR startup placed pressure on CUBIC such that CUBIC's congestion window was reduced close to zero
- At this stage CUBIC's efforts to restart its congestion window appear to collide with BBR's congestion control model, so CUBIC 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

Same two endpoints, same network path across the public Internet

Using a long delay path AU to Germany via the US

```
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.N0V-New-York.as6453.net (63.243.128.26)  292.651 ms  292.585 ms  292.7
11  if-ae-2-2.tcore2.N0V-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
```

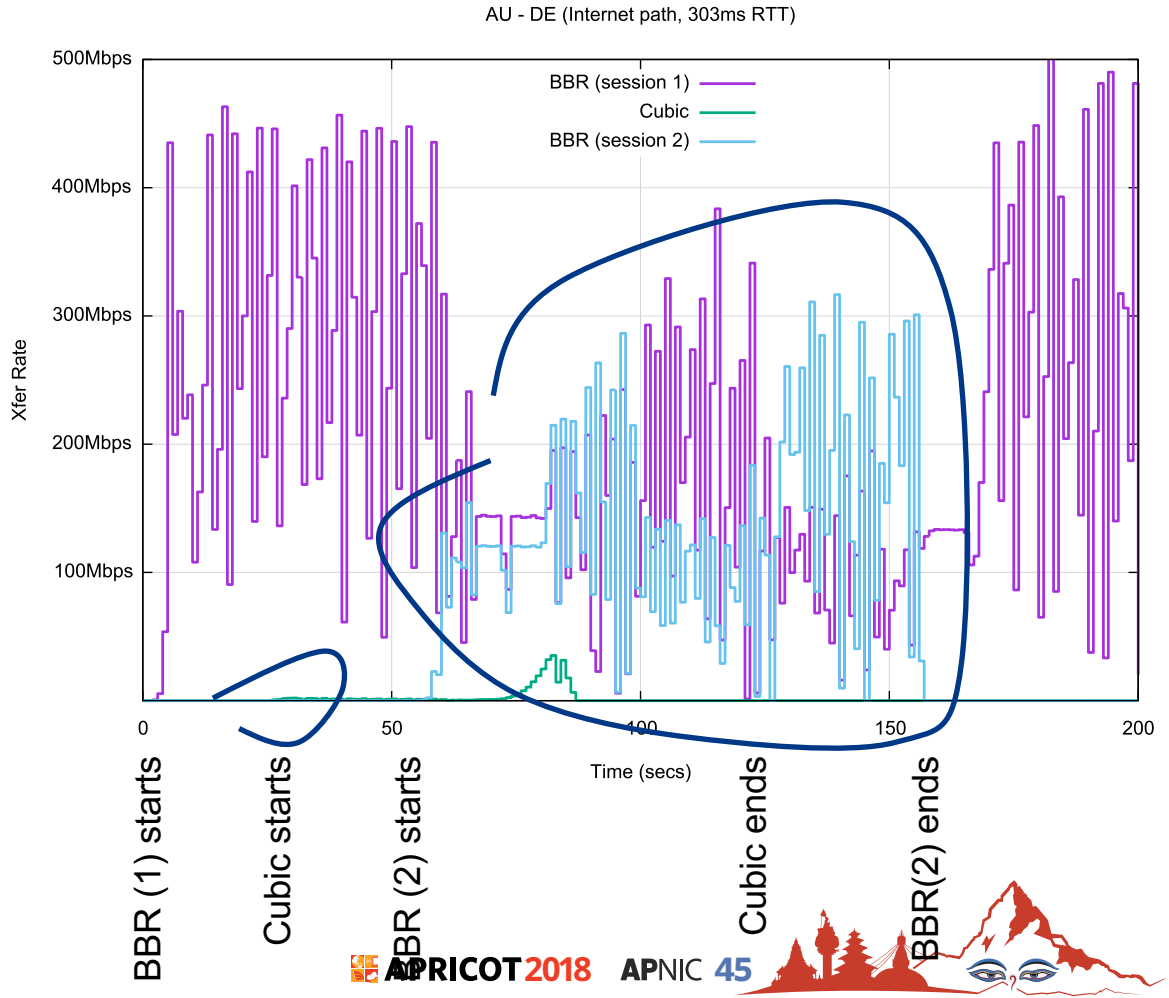



BBR vs Cubic

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



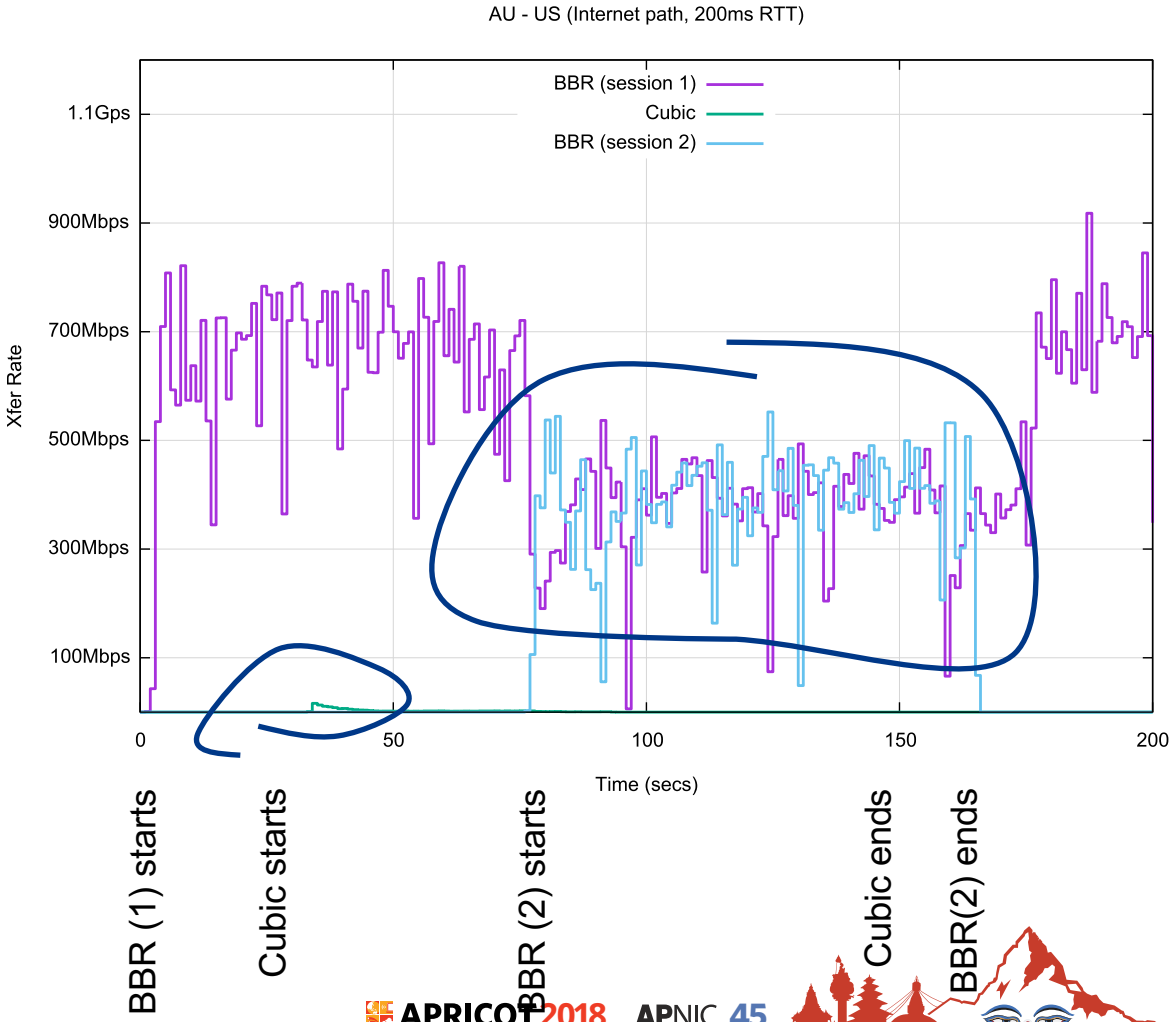
BBR vs Cubic

Using a shorter (200ms) creates an entirely different profile

Its possible to drive the network harder, and NNR is pulling some 700Mbps in capacity

Cubic is still completely crowded out of the picture!

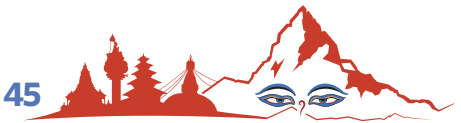
Interestingly in this case two BBR sessions share the capacity very effectively



So what can we say about BBR?

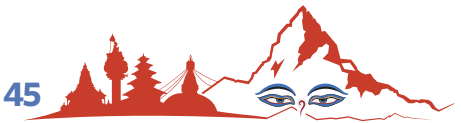
It's "interesting" in so many ways:

- It's a move away from the more common loss-based flow control protocols
- It looks likeit 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 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



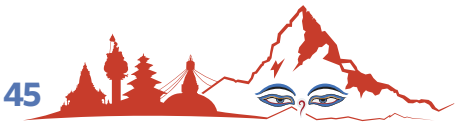
Why use BBR?

- Because it achieves!



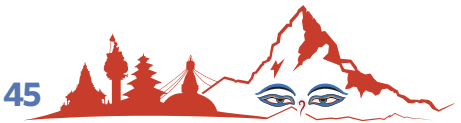
Why not use BBR?

- 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 for the user 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 highly unstable
    - And we all loose!



While we are bad-mouthing BBR…

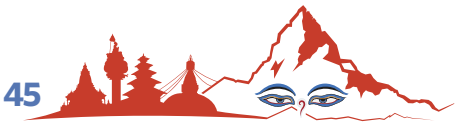
- Its really unfair in the (loss-based) Internet environment
  - Its like driving a rocket-propelled bulldozer down the freeway!
- Its unstable, in that it appears to react quite radically to small variations in the path characteristics
- It spends 75% of its time flying “blind” with respect to its sending rate
- BBR’s RTT estimates are too susceptible to admit the onset of queueing, and maintains a (too) high sending rate in the face of large scale congestion loss.



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 wont create back pressure on the flow



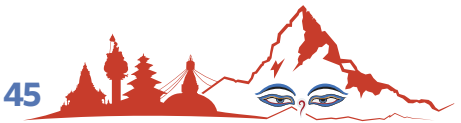
Where now?

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 BDP by using an estimated 'fair' proportion of the path BDP
- Alter the +/- 25% probe factors dynamically (i.e. allow this to be less than 25% overload)



That's it!

Questions?