Precept 4

Main topic:

- Assignment 2: TCP Congestion Control and Bufferbloat

Breakout rooms:

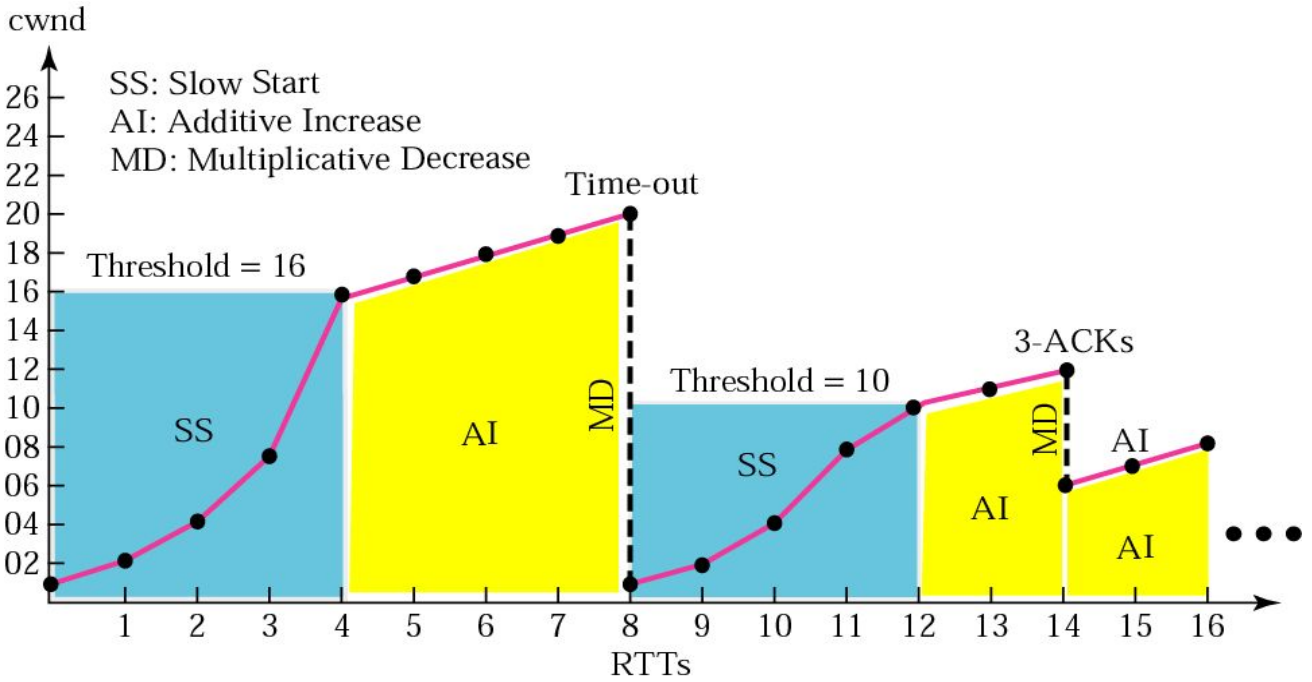
– The effect of parallel TCP connections on the congested network?
– The effect of UDP datagram at a very high data-rate?
– Optional: how can lower layer provide better throughput without modifying TCP?

TCP Congestion Window Size

- cwnd - the TCP congestion window size parameter
  - maintained by the sender
  - determines how much traffic can be outstanding (sent but not acknowledged) at any time.
- There are many algorithms for cwnd
  - Goal: maximizing the connection's throughput while preventing congestion.
  - Tahoe, Reno, New Reno, SACK, CUBIC

Congestion detection

- Time-out - lost packets (e.g. buffer overflow at routers)
- 3-ACKs - long delays (e.g. queueing in router buffers)
  - less severe since 1 segment is missing but 3 other segments have been received



Source: TCP/IP Protocol Suite 4th

Bandwidth Utilization

- How much time is needed increase cwnd of a 10Gbps from half utilization to full utilization?
  - 1500-byte PDU
  - 100 ms RTT
- Full utilization cwnd = 10Gbps/1500byte ~=83333
- Half utilization cwnd = 83333/2 = 41666.5

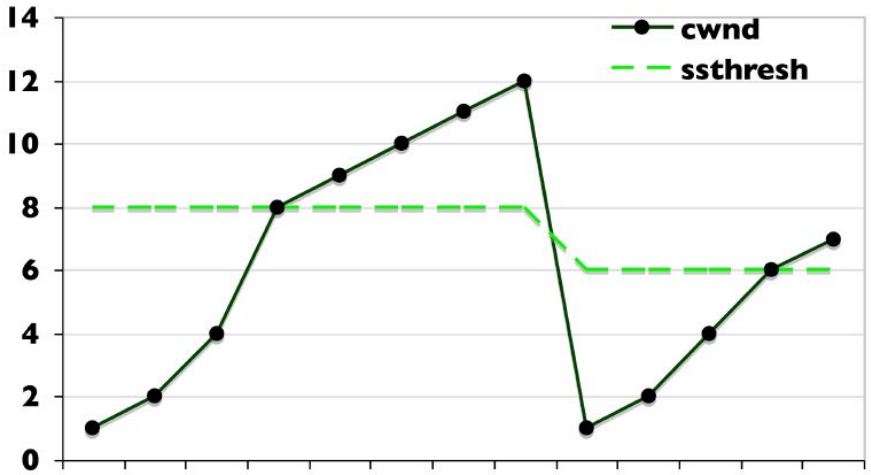
If cwnd is increased by 1 for each RTT

⇒ 41667 RTT is needed to fully utilized the link

⇒ 41667 RTT * 100ms(RTT time) = 69.44minutes

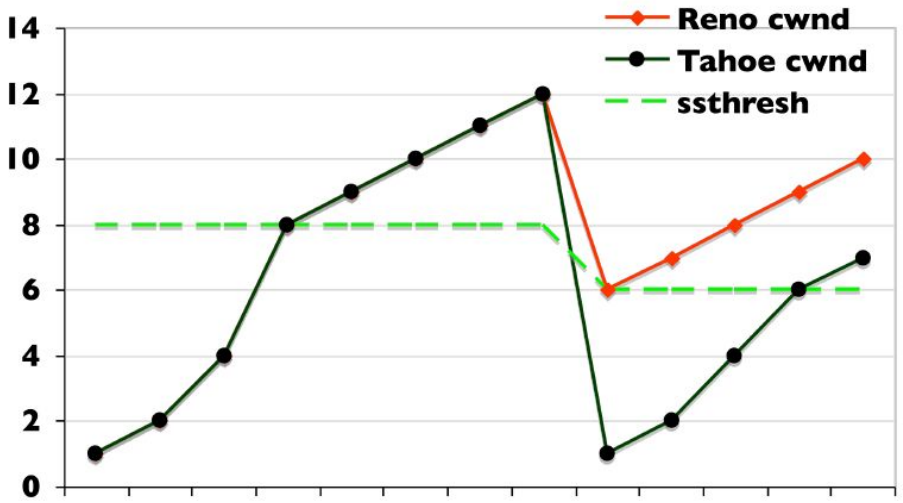
Tahoe TCP

- **Per RTT (per window):**
  if(cwnd < ssthresh)
  cwnd *= 2
  else
  cwnd += 1
- **timeout/3rd dup ack:**
  - Retransmit all unacked.
  - ssthresh = cwnd/2
  - cwnd = 1
- Packets still getting through in dup ack – no need to reset the cwnd!



Reno TCP

- **Per RTT (per window): :**
  If(cwnd < ssthresh)
  cwnd *= 2
  else
  cwnd += 1
- **timeout:**
  Retransmit 1st unacked
  ssthresh = cwnd/2
  cwnd = 1
- **3rd dup ack:**
  Retransmit 1st unacked
  ssthresh = cwnd/2
  cwnd = ssthresh + 3
  – Fast Recovery: the pipe is still almost full – no need to restart

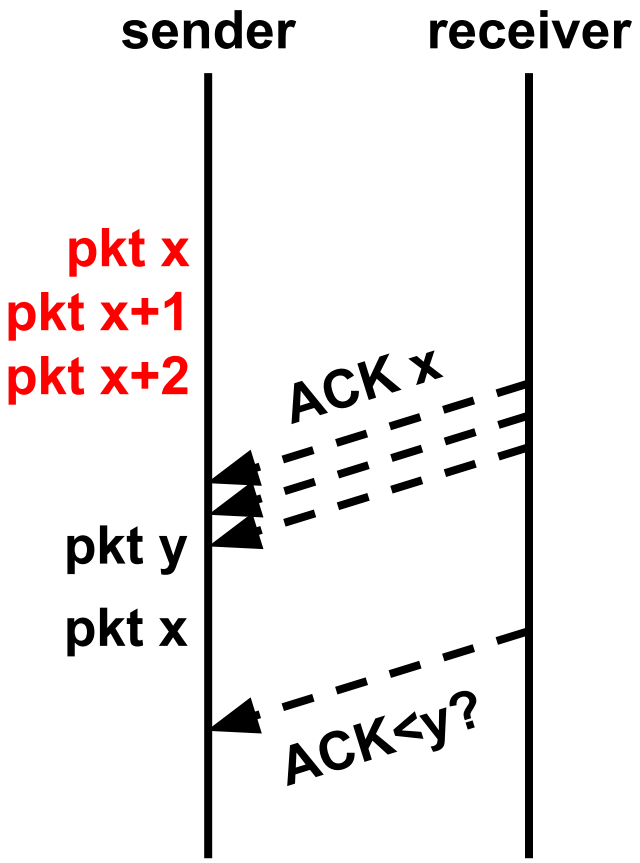


Problem with Reno

- Multiple packet losses within a window of data
  - Terminates recovery prematurely
  - Deflates cwnd to ssthresh
  - Detection of second loss relies on another fast retransmission
    - But with much less incoming dup ACKs
    - Much less new data packets begin sending out
    - Lose self-clocking

New Reno TCP

- Idea: use partial ACKs to stay in fast recovery and fix more lost segments
- **3rd dup ack:**
  Retransmit 1st unacked
  ssthresh = cwnd/2
  cwnd = cwnd/2 + 3
- **subsequent dup ack:**
  cwnd++
- **"complete" ack:**
  cwnd = ssthresh
- **"partial" ack:**
  retransmit
  cwnd = ssthresh

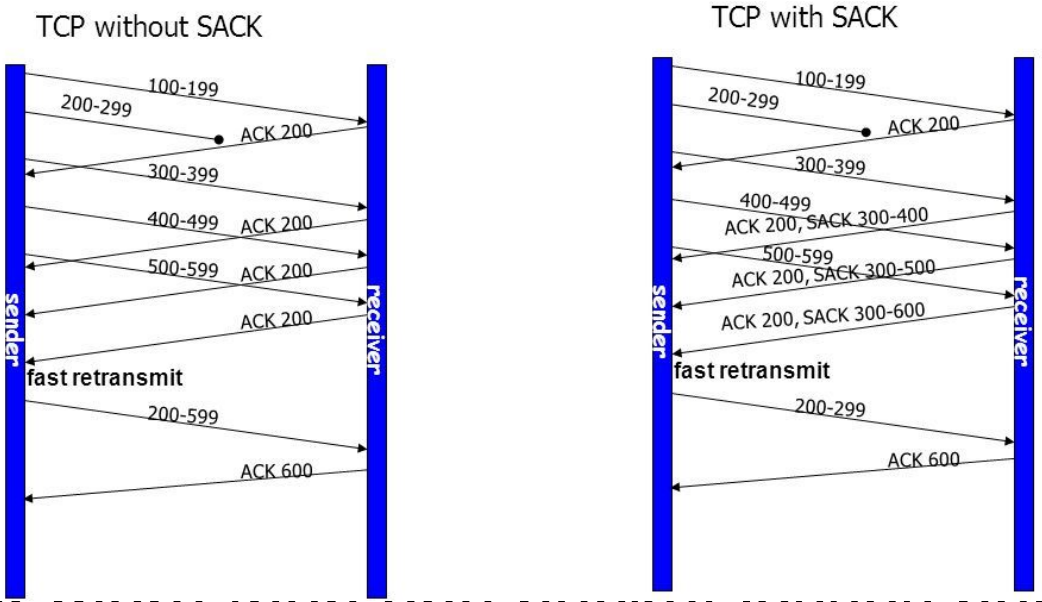


Problem with New Reno

- TCP uses cumulative ACKs
  - Receiver identifies the last byte of data successfully received
  - Out of order segments are not ACKed
  - Receiver sends duplicate ACKs
- TCP forces the TCP sender
  - To wait an RTT to find out a segment was lost
  - To unnecessarily retransmit data that has been correctly received
- □ reduced overall throughput

SACK TCP

- Selective Ack (SACK) + Selective Retransmission Policy
  - Receiver informs sender about all segments that are successfully received
  - Sender fast retransmits only the missing data segments

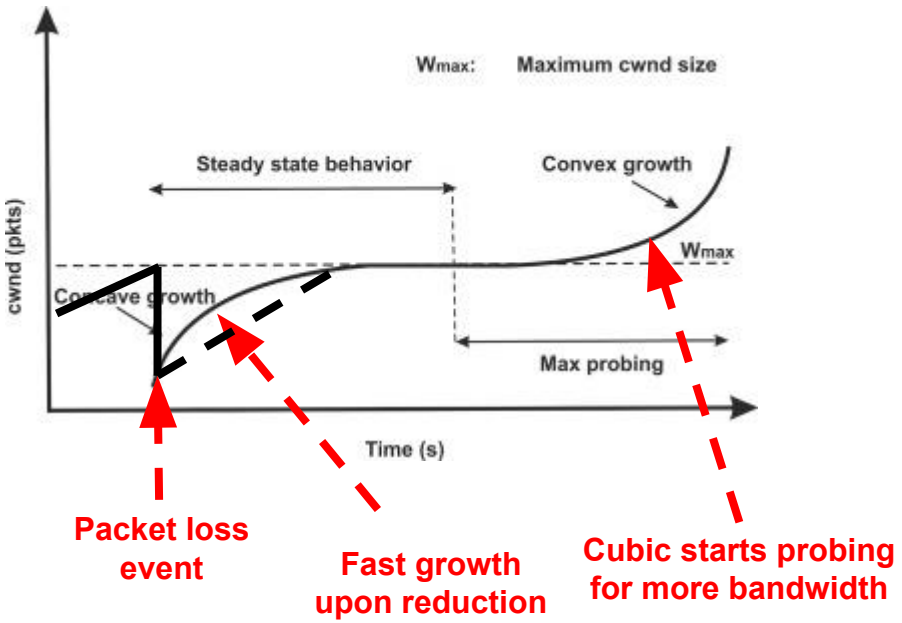


- Can recover more than one packet losses per RTT since sender now knows which packets are dropped

CUBIC TCP

- Transition from the linear window growth function to cubic
- Good bandwidth utilization

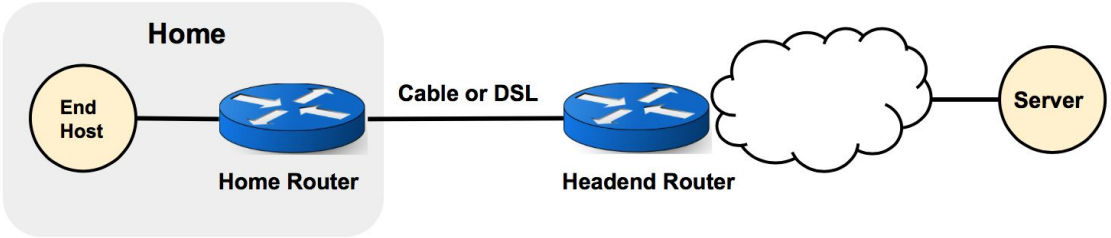
- W_{max}: sending rate at which congestion loss was detected
- Starts at the concave profile and moves to a convex profile
- Center flat region gives TCP time to stabilize
- $cwnd = C(T - K)^3 + W_{max}$
  where $k = \left(\frac{W_{max}\beta}{C}\right)^{1/3}$



w_{max}: Window size just before the last reduction
β: Multiplicative decrease factor
T: Time elapsed since the last window reduction
C: A Scaling constant
cwnd: The congestion window at the current time

Bufferbloat

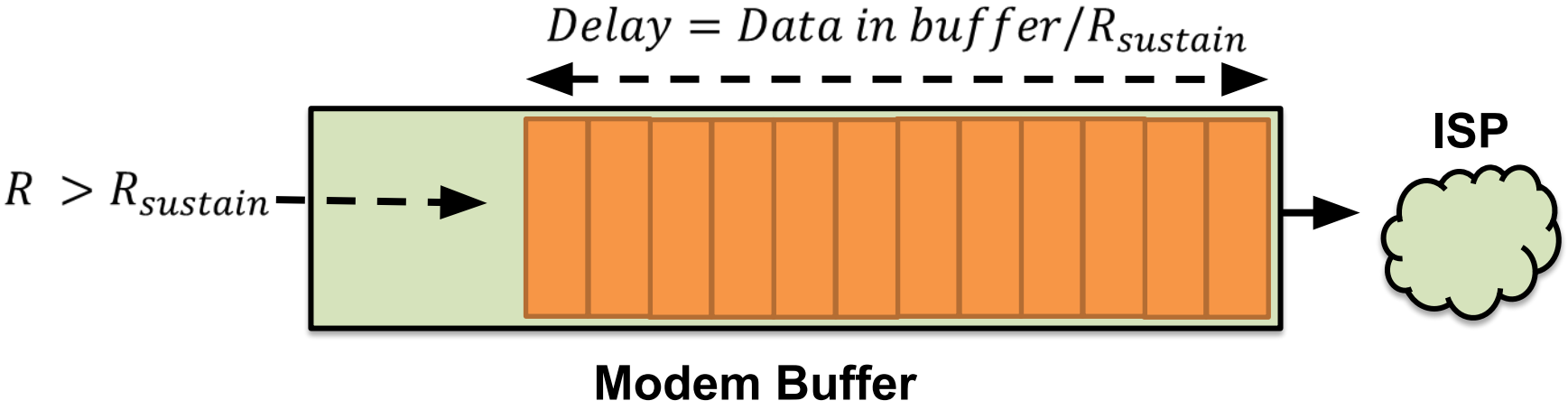
- A switching device is configured to use excessively large buffers to avoid losing packets
  - high latency and jitter.
- This can happen even in a typical home network



- Here, the end host in the home network is connected to the home router.
- The home router is then connected, via cable or DSL, to a headend router run by the Internet service provider (ISP).

Bufferbloat

- TCP sender sends until they see lost packet
  - but if the buffer is large, the senders is unable to see the lost packets until this buffer has already filled up
- TCP sender sends at increasingly faster rates until they see a lost, then the sending rate is already much larger than the network's capacity.
- Buffer experiences the increasing delay



Questions for Thought

- How can parallel TCP connections help congested network?
  - Can they hurt the performance instead?
- What happens if one of clients sends UDP datagram at a very high data-rate?
- How can lower layer provide better throughput without modifying TCP?

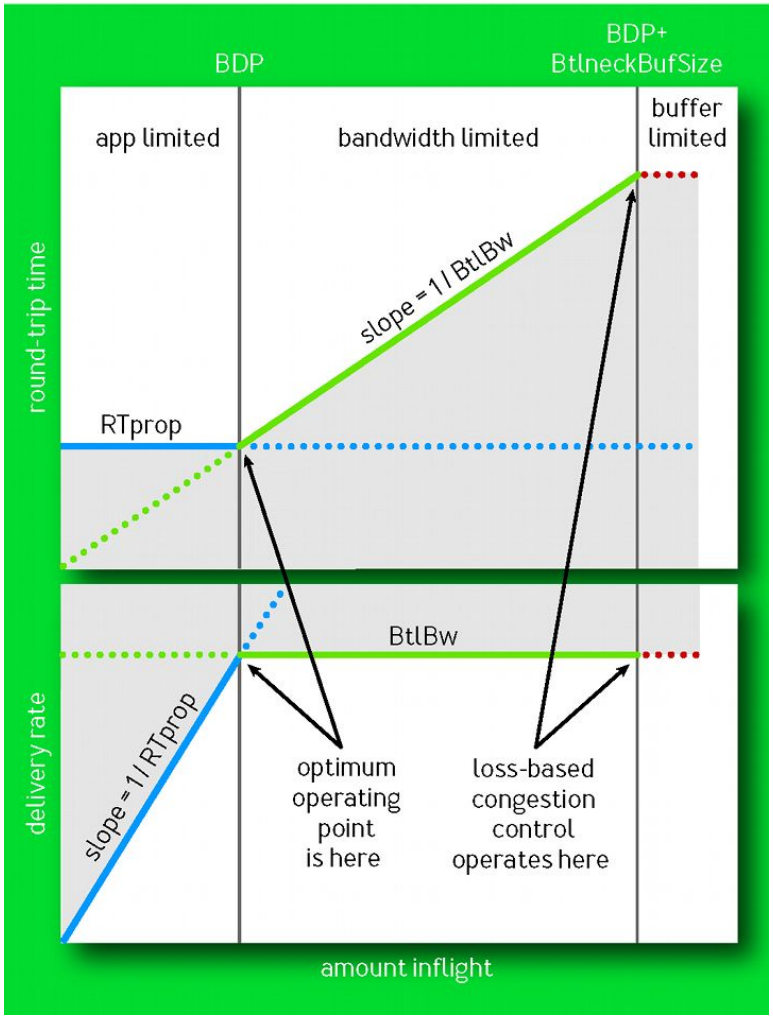
Probing for Throughput: TCP BBR

BBR: TCP Congestion Control Protocol introduced by Google

BBR does not use packet loss to determine congestion

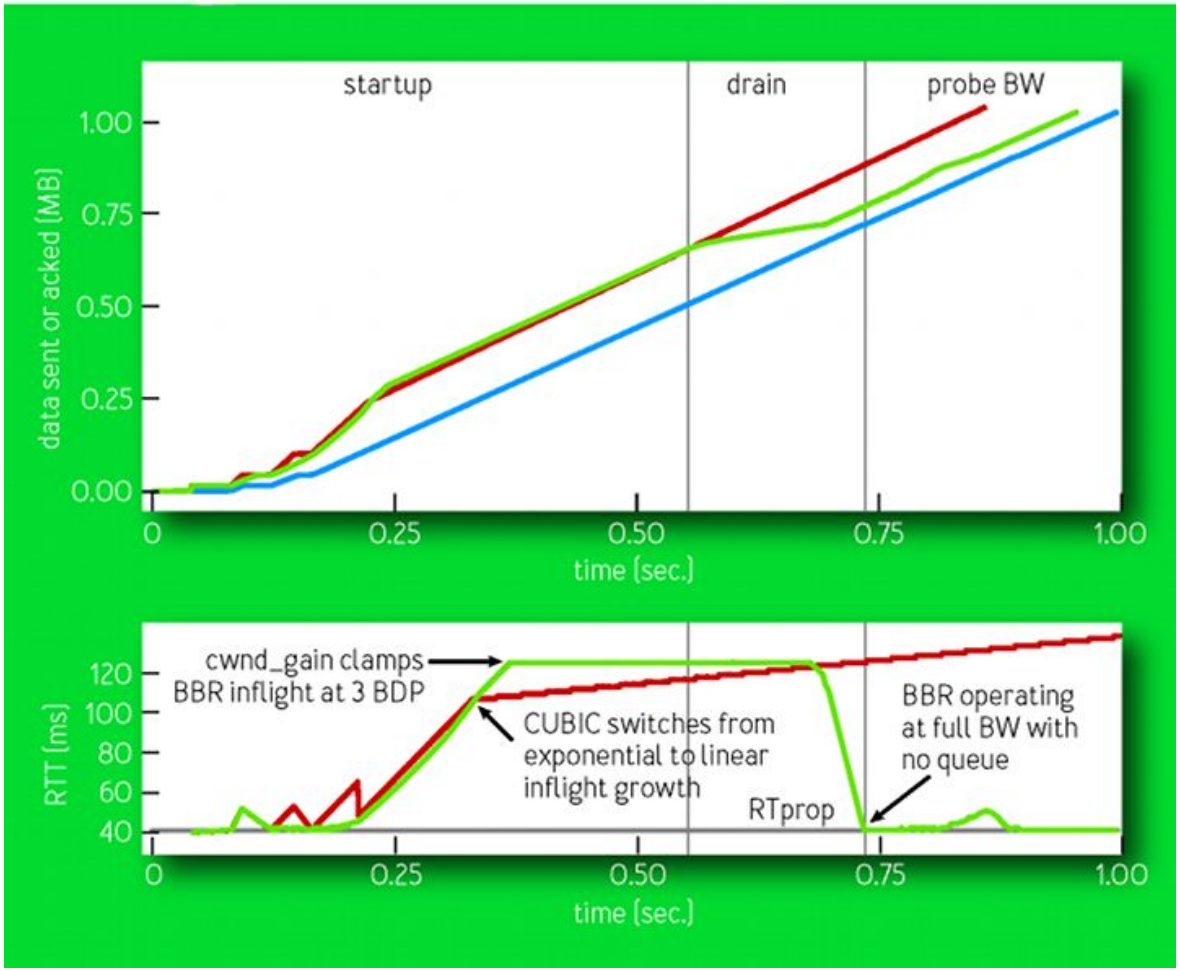
Estimates bottleneck bandwidth by measuring packet delivery rate.

FIGURE 1: DELIVERY RATE AND ROUND-TRIP TIME VS. INFLIGHT



Probing for Throughput: TCP BBR

FIGURE 4: FIRST SECOND OF A 10-MBPS, 40-MS BBR FLOW



Probing for Throughput: TCP BBR

FIGURE 5: FIRST 8 SECONDS OF 10-MBPS, 40-MS CUBIC AND BBR FLOWS

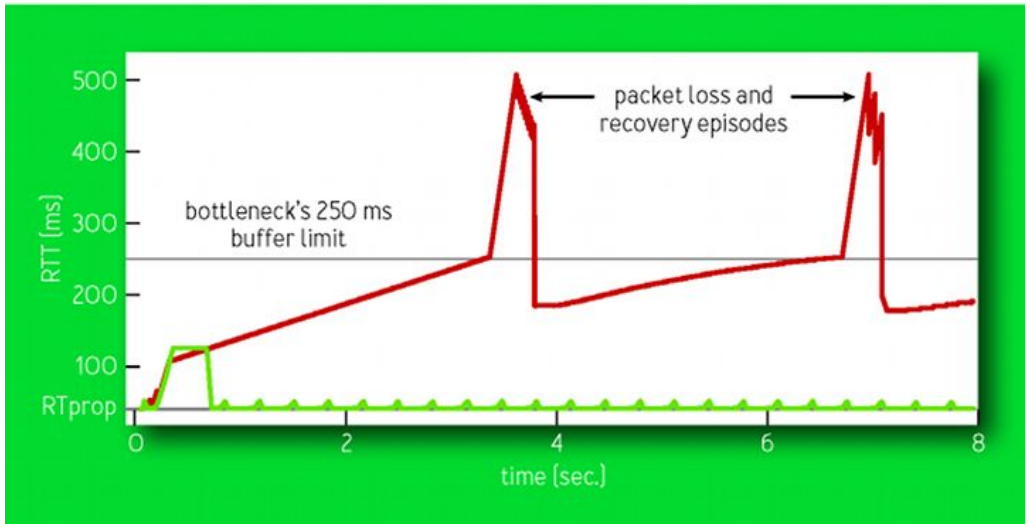


FIGURE 6: THROUGHPUTS OF 5 BBR FLOWS SHARING A BOTTLENECK