BBR: Congestion-Based Congestion Control

Cardwell et al., 2016 ACMQueue

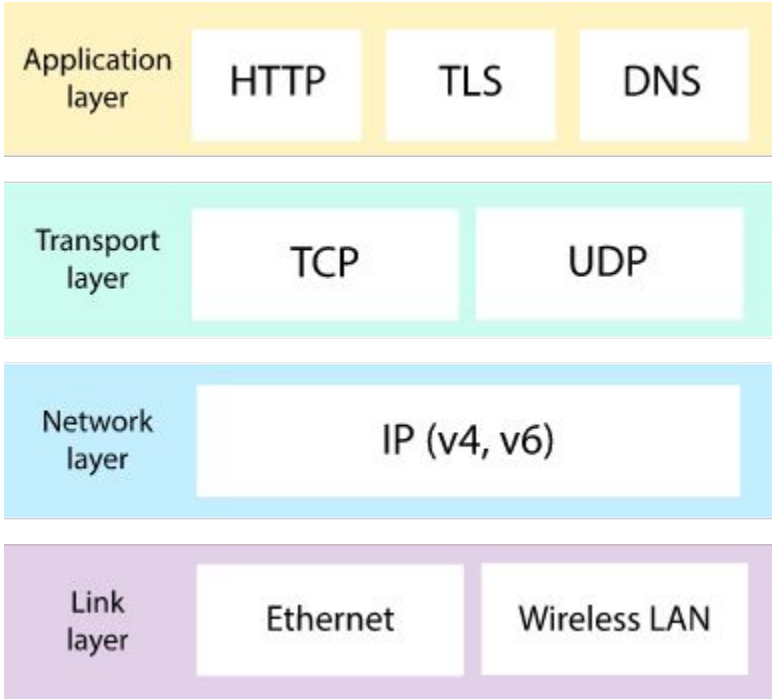
2/17/2023

Allison Chen, Julian Ost, Christina Shatford

Agenda

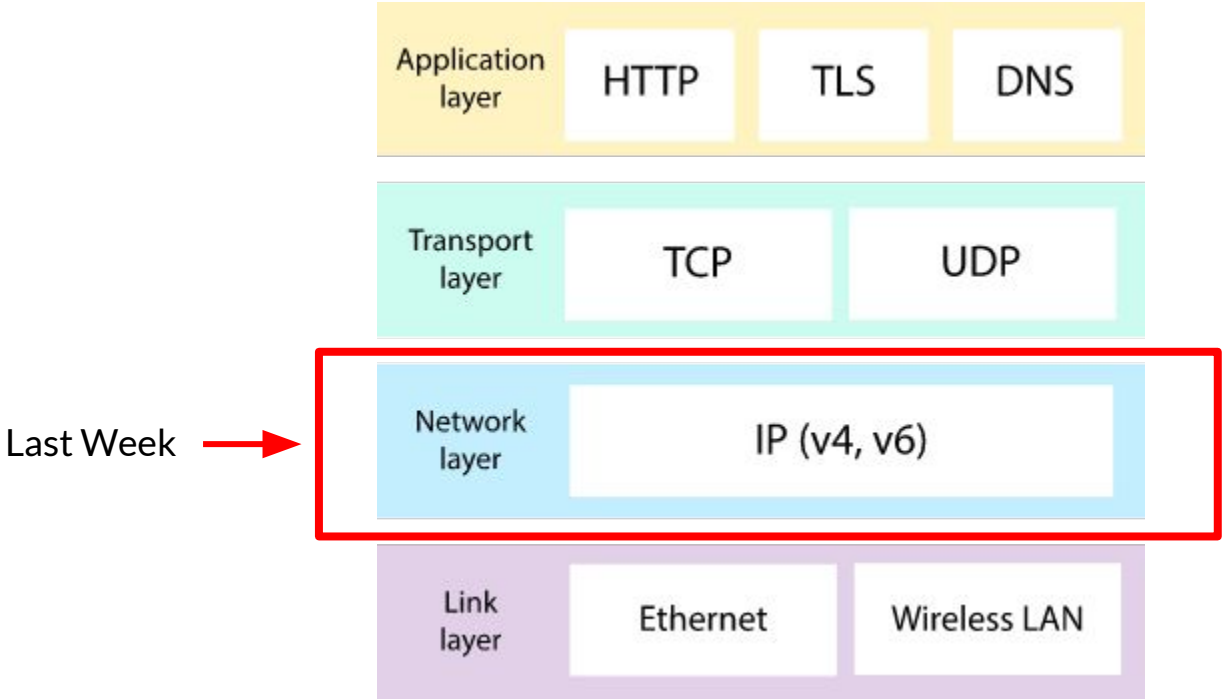
- Background
  - Congestion & Bottlenecks
  - Loss Based Congestion Control
  - Important Properties of the Link
- Algorithm
- Experiments
- Follow Up Work & Improvements

Where in the Stack Are We?



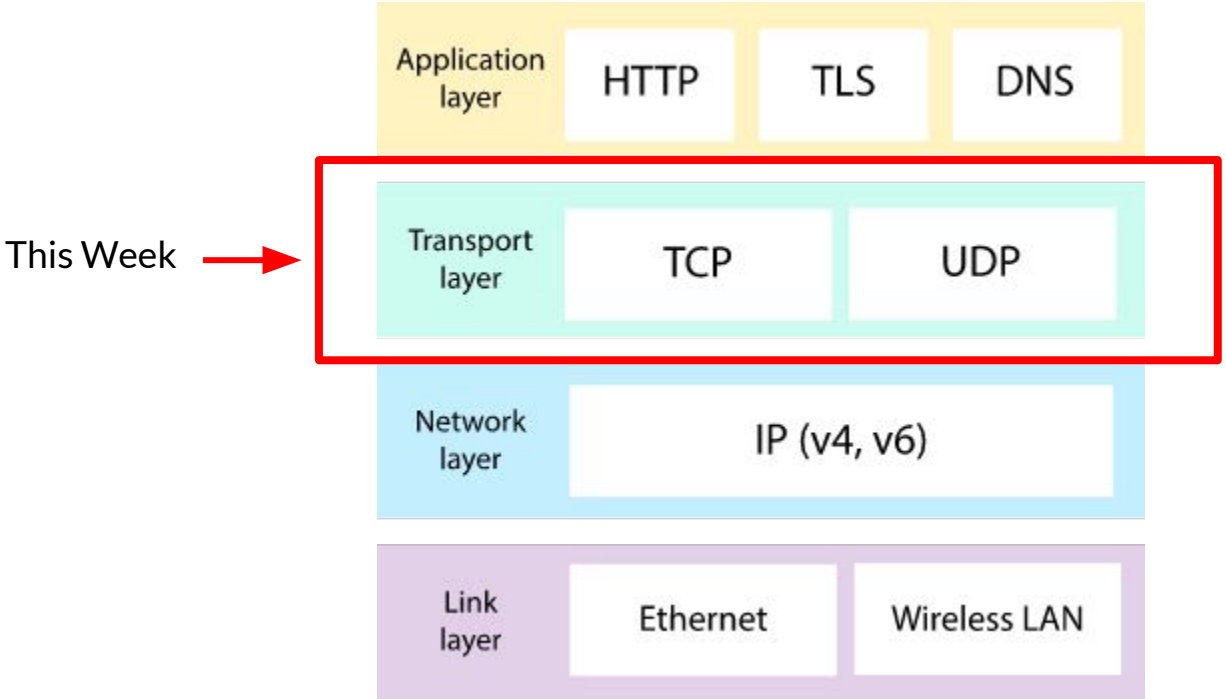
https://cdn.kastatic.org/ka-perseus-images/6a0cd3a5b7e709c2f637c959ba98705ad21e4e3c.svg

Where in the Stack Are We?



https://cdn.kastatic.org/ka-perseus-images/6a0cd3a5b7e709c2f637c959ba98705ad21e4e3c.svg

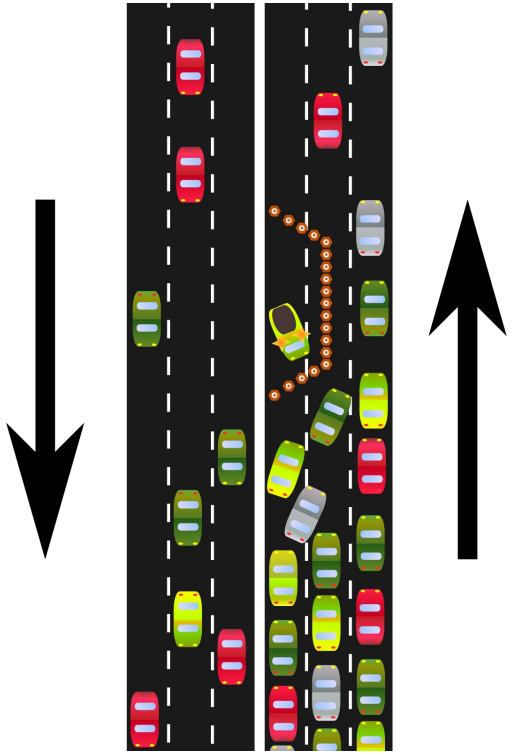
Where in the Stack Are We?



https://cdn.kastatic.org/ka-perseus-images/6a0cd3a5b7e709c2f637c959ba98705ad21e4e3c.svg

Congestion & Bottlenecks

- Congestion: Slowdown due to the network being given more data than it can handle
- Bottleneck: where congestion occurs
- Importance of bottlenecks
  - Sets maximum delivery rate
  - Location of persistent queues

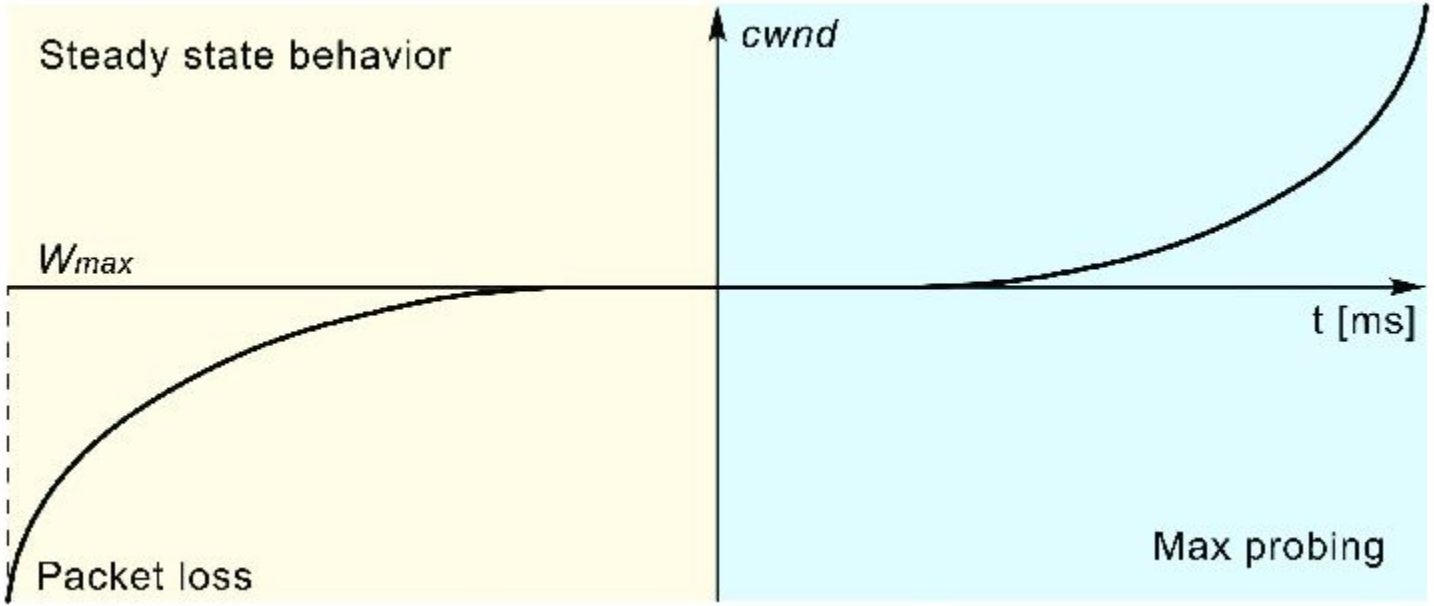


Previous Congestion Control (Loss Based)

1. Set congestion window of `cwnd_size` packets
2. While no loss, keep sending packets and increasing `cwnd_size`
3. If packet loss occurs (i.e. router buffers overflow), reduce `cwnd_size` and retransmit lost packets

CUBIC (Ha, et al.)

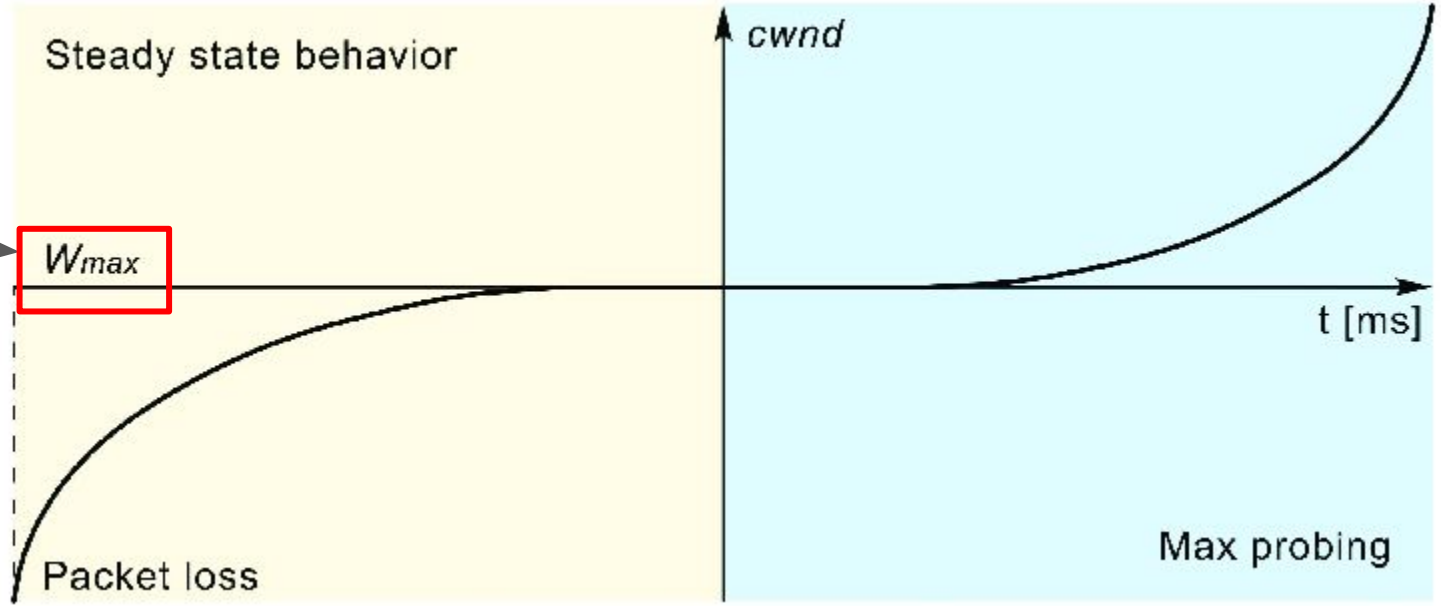
CUBIC (Ha, et al.)



Ha, Sangtae, Injong Rhee, and Lisong Xu. "CUBIC: a new TCP-friendly high-speed TCP variant." *ACM SIGOPS operating systems review* 42.5 (2008): 64-74.
https://www.researchgate.net/profile/Josip-Lorincz/publication/352912159/figure/fig1/AS:1041236991414272@1625261652299/TCP-CUBIC-window-growth-function-after-the-packet-loss-event.png

CUBIC (Ha, et al.)

`cwnd_size` at previously experienced congestion

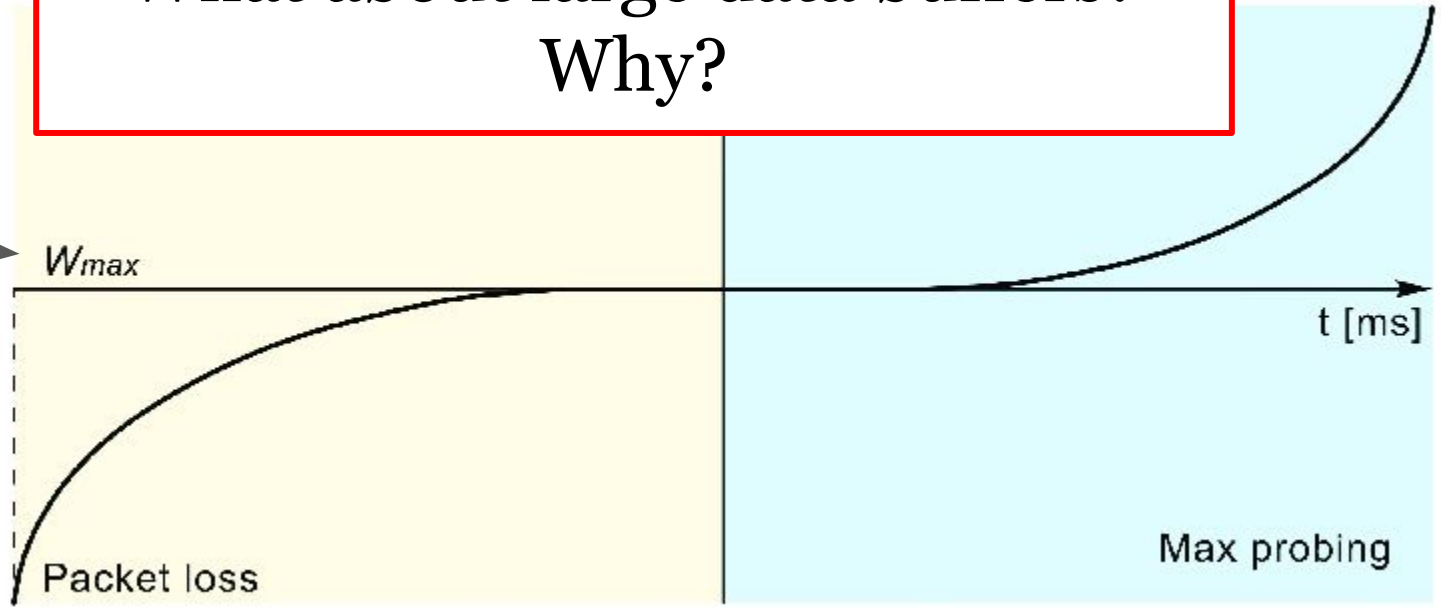


Ha, Sangtae, Injong Rhee, and Lisong Xu. "CUBIC: a new TCP-friendly high-speed TCP variant." *ACM SIGOPS operating systems review* 42.5 (2008): 64-74.
https://www.researchgate.net/profile/Josip-Lorincz/publication/352912159/figure/fig1/AS:1041236991414272@1625261652299/TCP-CUBIC-window-growth-function-after-the-packet-loss-event.png

CUBIC (H

How would CUBIC behave when routers have small data buffers? What about large data buffers? Why?

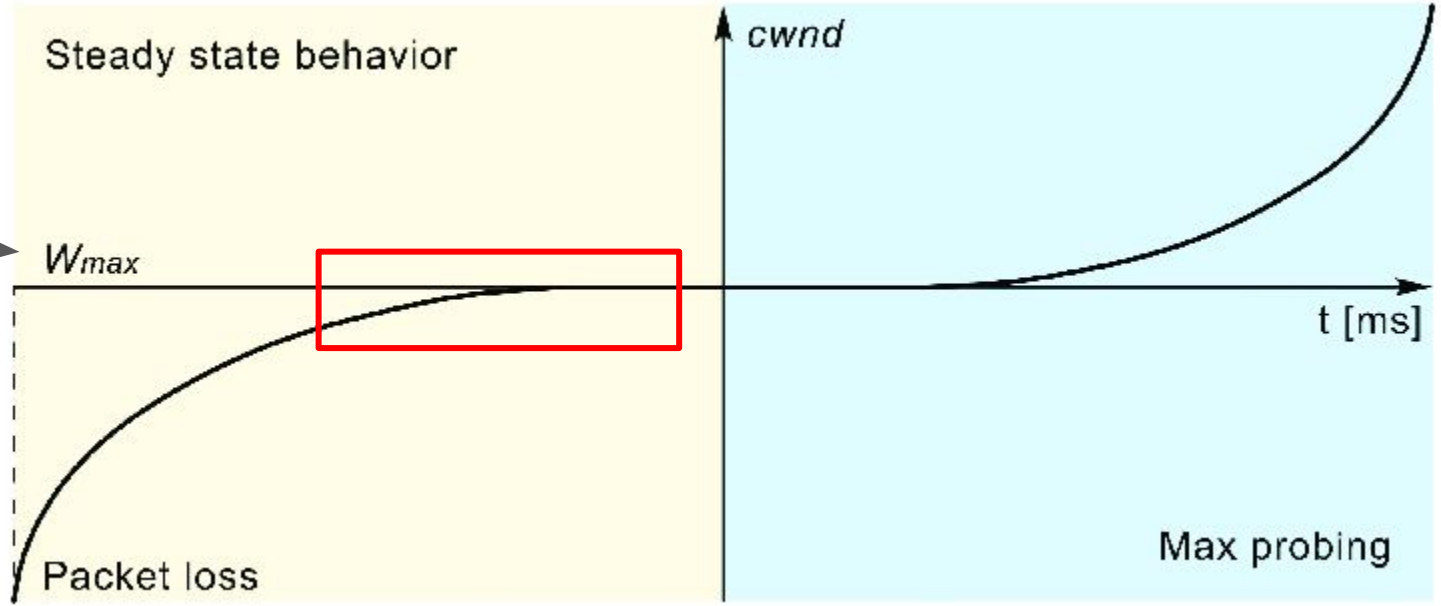
`cwnd_size` at previously experienced congestion



Ha, Sangtae, Injong Rhee, and Lisong Xu. "CUBIC: a new TCP-friendly high-speed TCP variant." *ACM SIGOPS operating systems review* 42.5 (2008): 64-74.
https://www.researchgate.net/profile/Josip-Lorincz/publication/352912159/figure/fig1/AS:1041236991414272@1625261652299/TCP-CUBIC-window-growth-function-after-the-packet-loss-event.png

Where large buffers get stuck:

`cwnd_size` at previously experienced congestion



Ha, Sangtae, Injong Rhee, and Lisong Xu. "CUBIC: a new TCP-friendly high-speed TCP variant." *ACM SIGOPS operating systems review* 42.5 (2008): 64-74.
https://www.researchgate.net/profile/Josip-Lorincz/publication/352912159/figure/fig1/AS:1041236991414272@1625261652299/TCP-CUBIC-window-growth-function-after-the-packet-loss-event.png

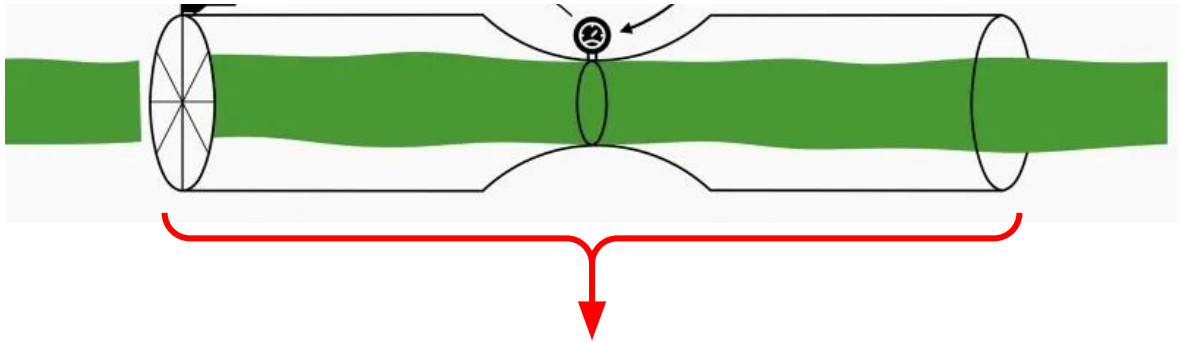
Physical Properties/Constraints of a Link

1. Round Trip Propagation (RTProp)
2. Bottleneck Bandwidth (BtlBw)

https://i0.wp.com/cdn-images-1.medium.com/max/800/1*jav9zG8B-cQIUrkqmDyVLA.jpeg?w=900&ssl=1

Physical Properties/Constraints of a Link

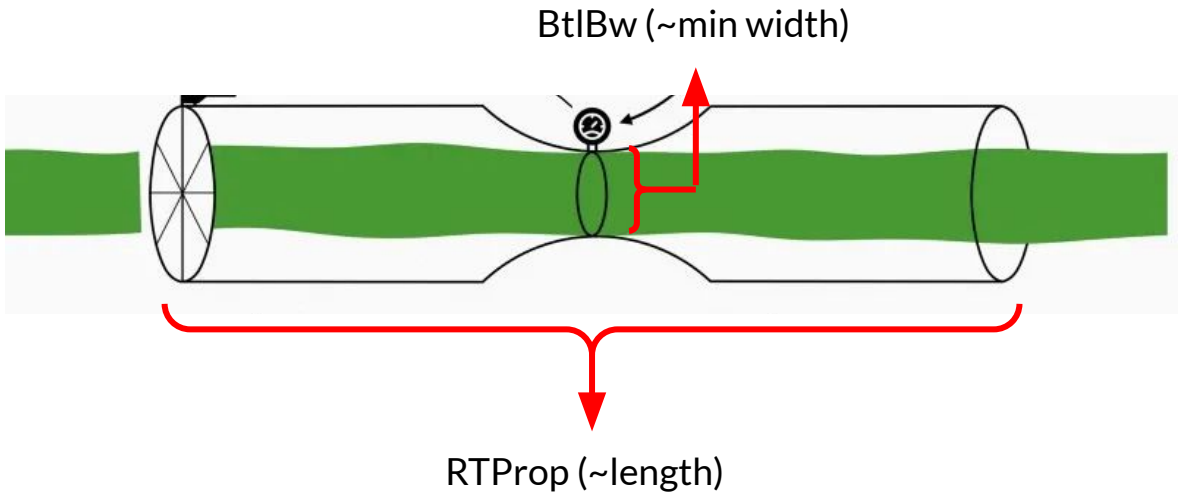
1. Round Trip Propagation (RTProp)
2. Bottleneck Bandwidth (BtlBw)



https://i0.wp.com/cdn-images-1.medium.com/max/800/1*jav9zG8B-cQIUrkqmDyVLA.jpeg?w=900&ssl=1

Physical Properties/Constraints of a Link

1. Round Trip Propagation (RTProp)
2. Bottleneck Bandwidth (BtlBw)

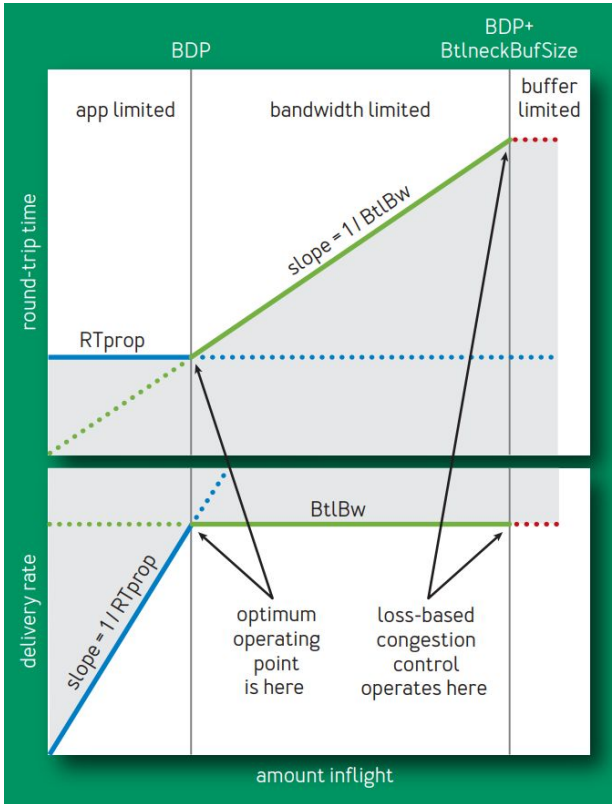


https://i0.wp.com/cdn-images-1.medium.com/max/800/1*jav9zG8B-cQIUrkqmDyVLA.jpeg?w=900&ssl=1

Physical Constraints & Measurable Properties

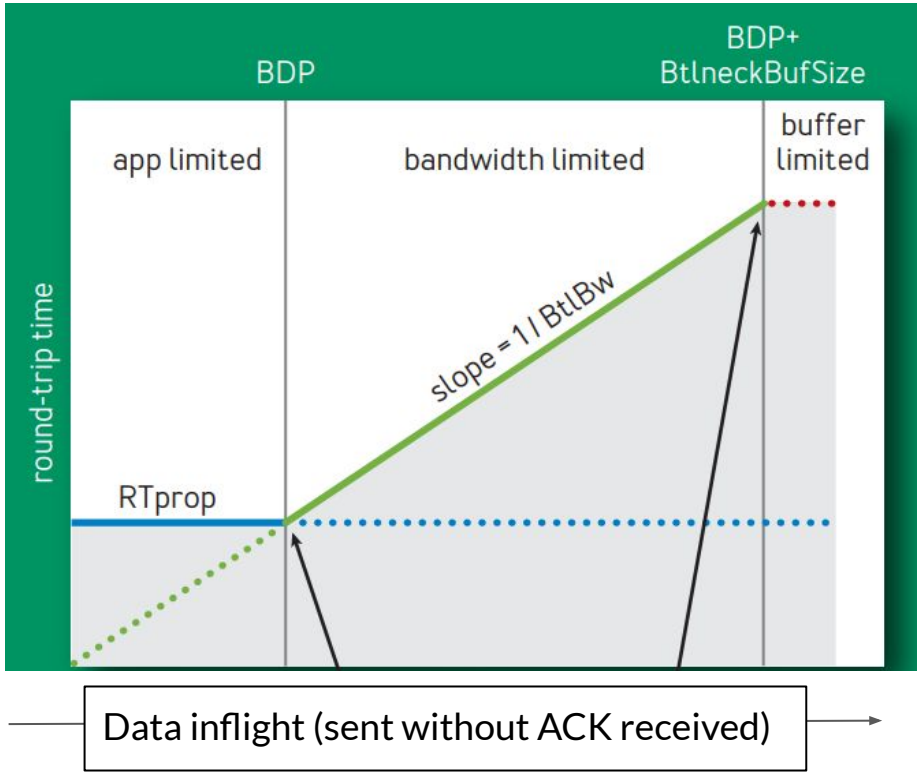
1. Round Trip Propagation (RTProp) → Round Trip Time (RTT)
2. Bottleneck Bandwidth (BtlBw) → Delivery Rate

Physical Constraints & Measurable Properties



Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

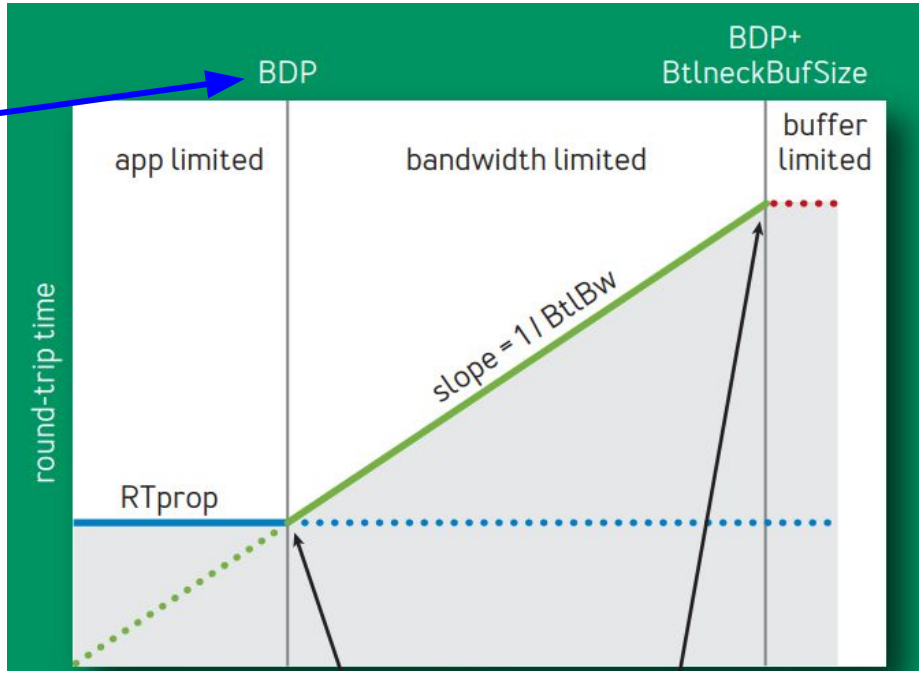
Physical Constraints & Measurable Properties



Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

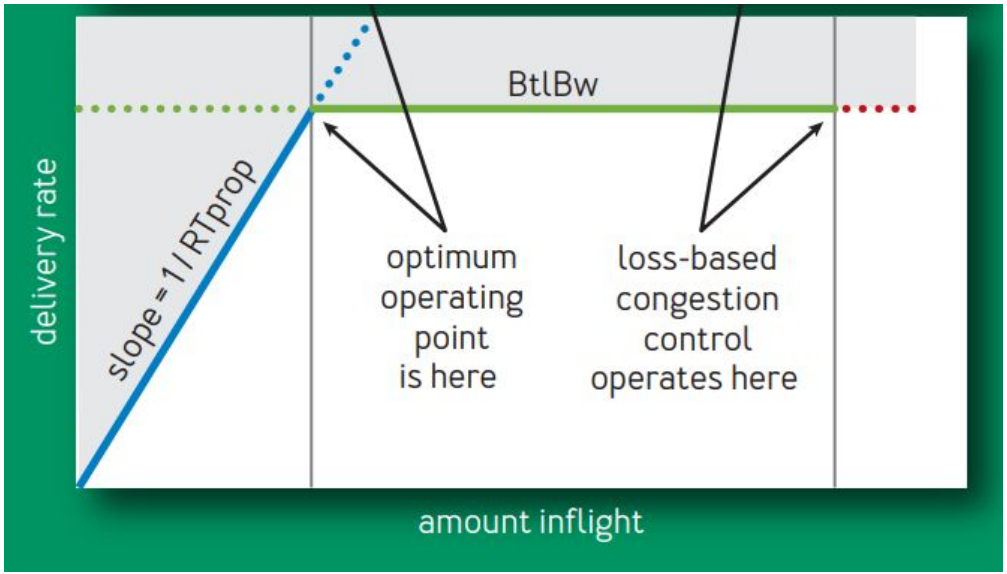
Physical Constraints & Measurable Properties

Bandwidth Delay Product



Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

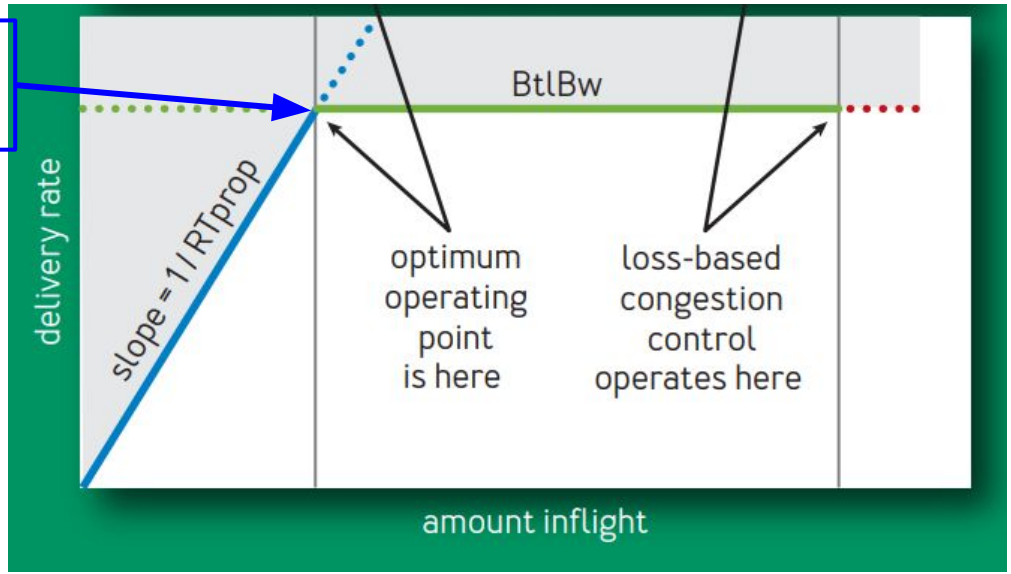
Physical Constraints & Measurable Properties



Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

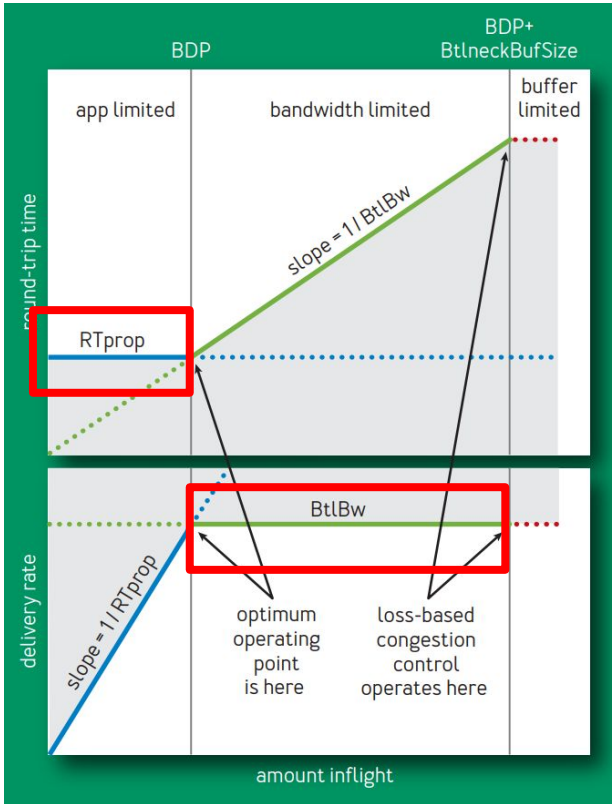
Physical Constraints & Measurable Properties

Bandwidth Delay Product



Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

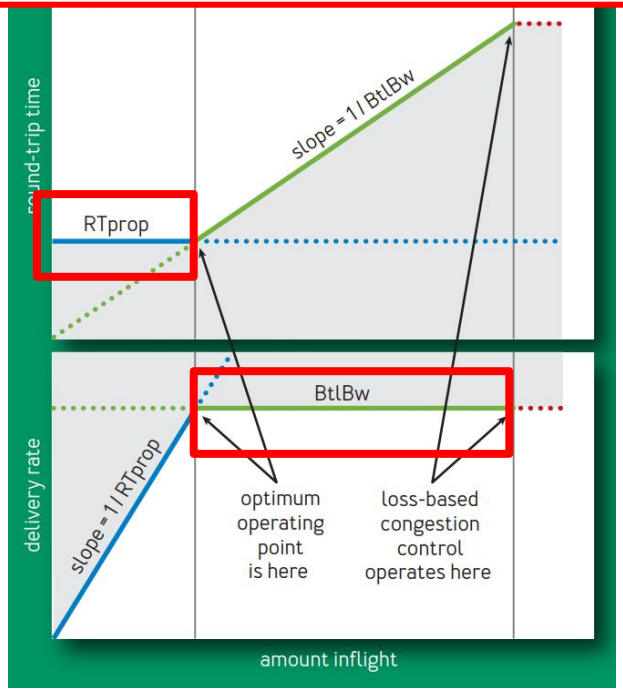
Uncertainty Principle



Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

Uncertain

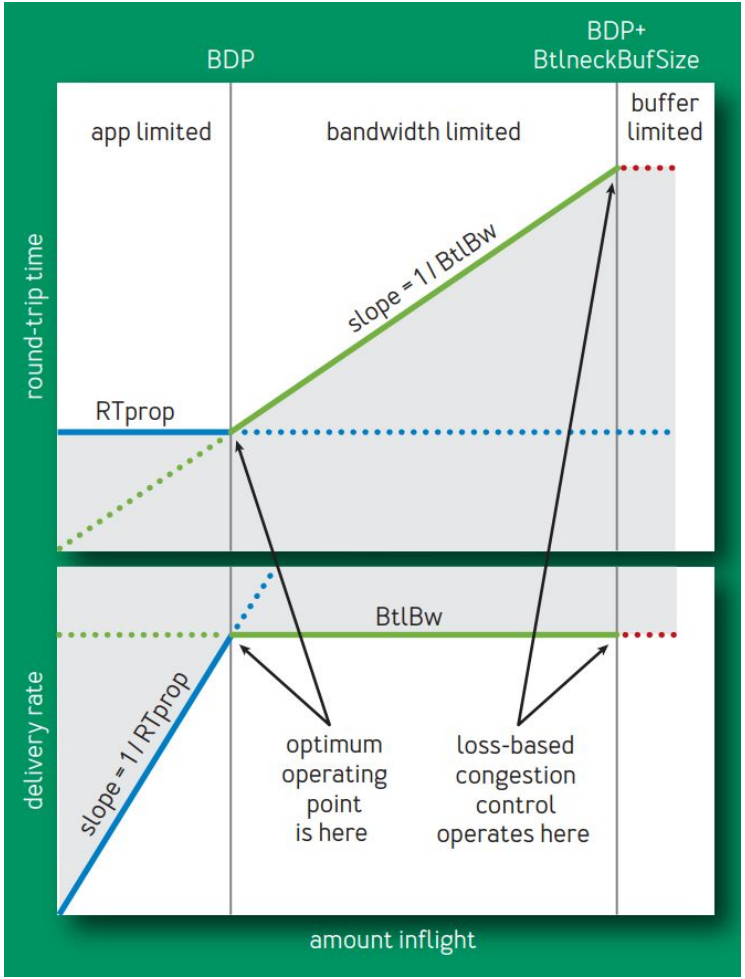
Why is BDP the Optimal Point? Why do loss based methods operate to the right of BDP?



Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

Answers (there can be more!)

- BDP minimizes RTT (round trip time) while maximizing delivery rate
- Loss based methods kick in when the congestion overflows buffers, not when congestion first occurs

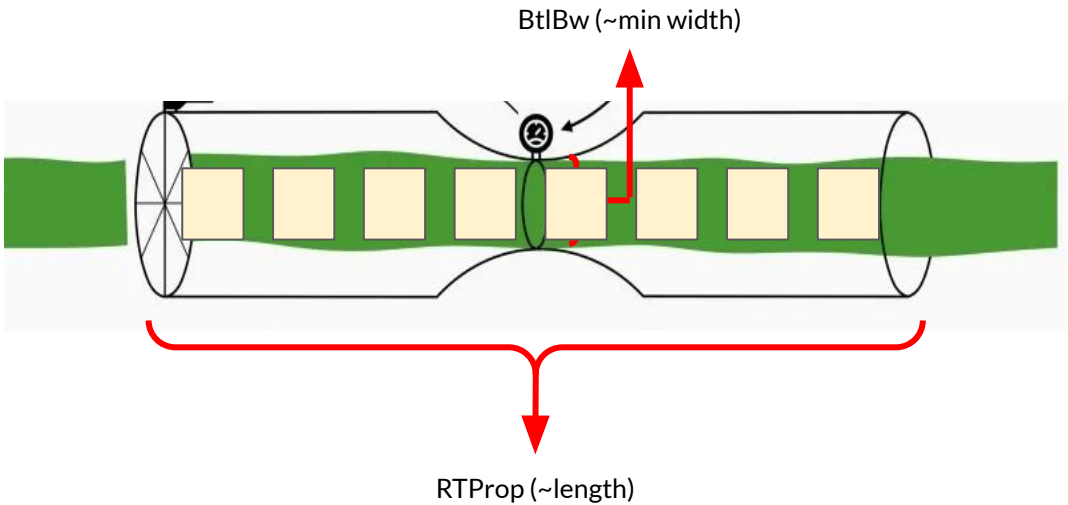


BDP as the Operating Point

Super cool guy, Dr. Leonard Kleinrock!

Conditions for Congestion Control

Goal: **Highest throughput and lowest delay**

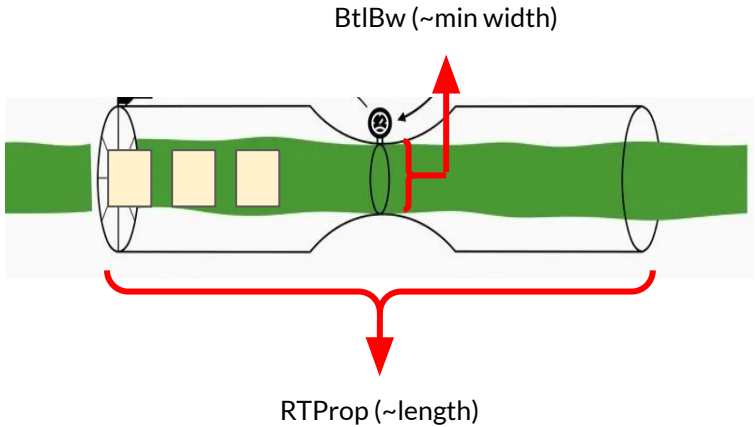


https://i0.wp.com/cdn-images-1.medium.com/max/800/1*jav9zG8B-cQIUrkqmDyVLA.jpeg?w=900&ssl=1

Conditions for Congestion Control

Goal: **Highest throughput and lowest delay**

- **Rate Balance**
  Packet arrival @ bottleneck = BtlBw
  -> Full utilization at the Bottleneck



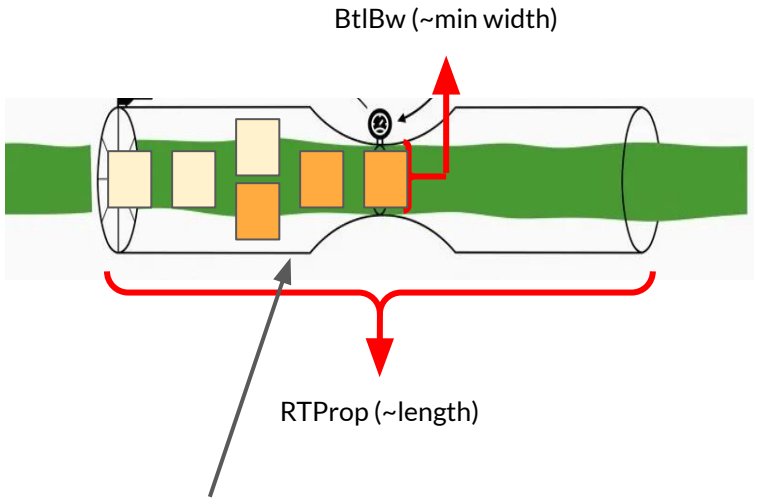
https://i0.wp.com/cdn-images-1.medium.com/max/800/1*jav9zG8B-cQIUrkqmDyVLA.jpeg?w=900&ssl=1

Conditions for Congestion Control

Goal: **Highest throughput and lowest delay**

- **Rate Balance**
  Packet arrival @ bottleneck = BtlBw
  -> Guarantees full utilization at the Bottleneck

BUT: Does not take care of standing queues, no dissipation possible!



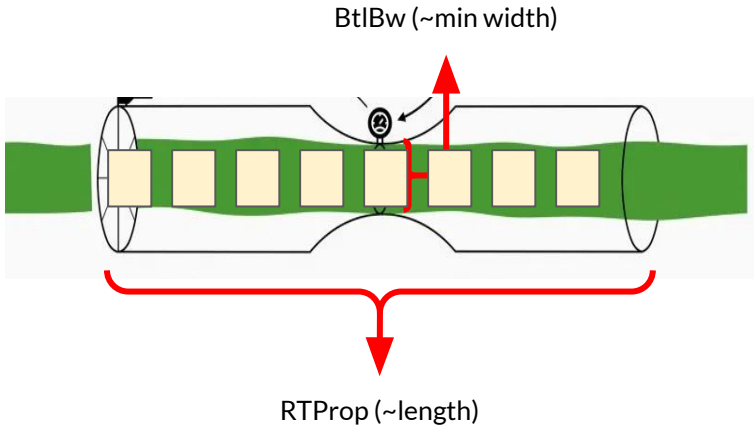
Queue e.g. initial window **> BtlBw**

https://i0.wp.com/cdn-images-1.medium.com/max/800/1*jav9zG8B-cQIUrkqmDyVLA.jpeg?w=900&ssl=1

Conditions for Congestion Control

Goal: **Highest throughput and lowest delay**

- **Full Pipeline**
  Data in flight = BDP (= BtlBW x RTprop)
  -> Guarantees no bottleneck starvation



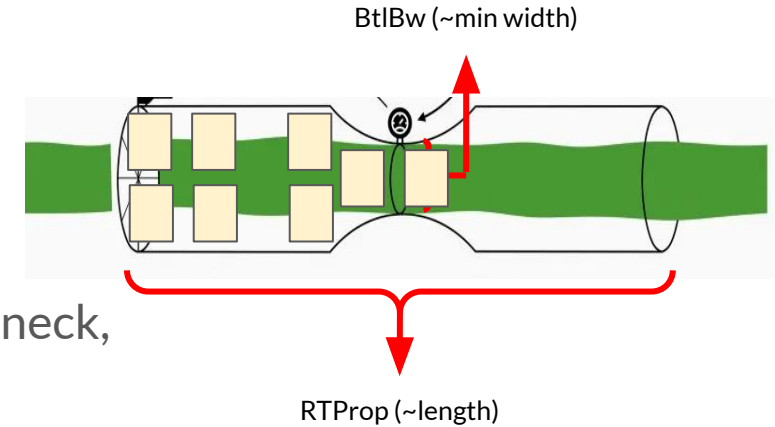
https://i0.wp.com/cdn-images-1.medium.com/max/800/1*jav9zG8B-cQIUrkqmDyVLA.jpeg?w=900&ssl=1

Conditions for Congestion Control

Goal: **Highest throughput and lowest delay**

- **Full Pipeline**
  Data in flight = BDP (= BtlBW x RTprop)
  -> Guarantees no bottleneck starvation

BUT: Does not guarantee no queue at Bottleneck,
can send to big packet bursts!



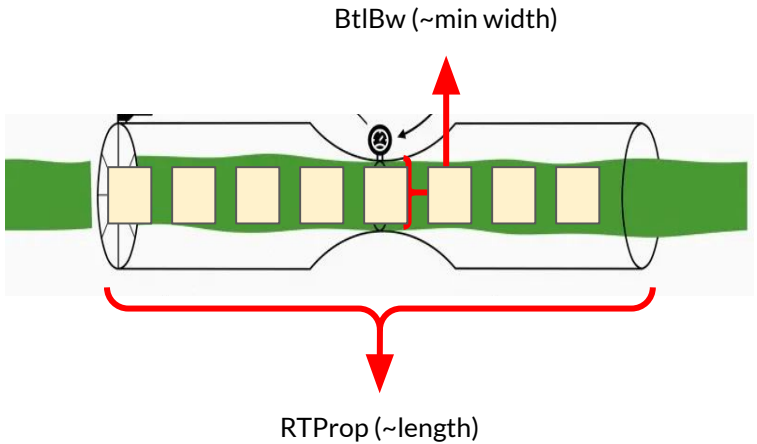
https://i0.wp.com/cdn-images-1.medium.com/max/800/1*jav9zG8B-cQIUrkqmDyVLA.jpeg?w=900&ssl=1

Conditions for Congestion Control

Goal: **Highest throughput and lowest delay**

Meet both conditions!

1. Packet arrival @ bottleneck = BtlBw
2. Data in flight = BDP (= BtlBW x RTprop)



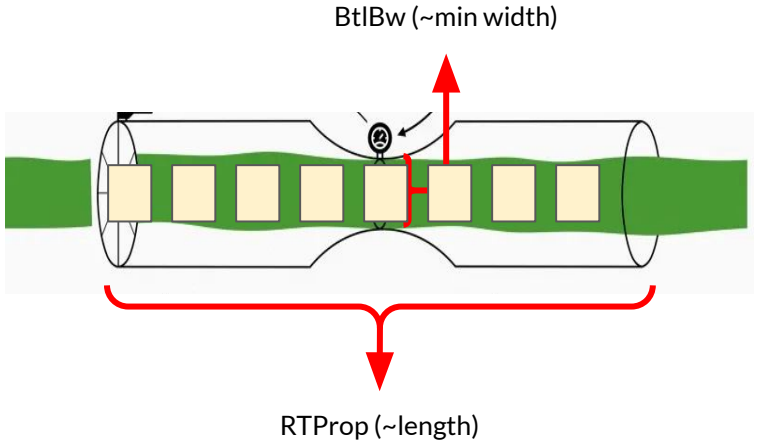
https://i0.wp.com/cdn-images-1.medium.com/max/800/1*jav9zG8B-cQIUrkqmDyVLA.jpeg?w=900&ssl=1

Conditions for Congestion Control

Goal: **Highest throughput and lowest delay**

Meet both conditions!

1. Packet arrival @ bottleneck = BtlBw
2. Data in flight = BDP (= BtlBW x RTprop)



https://i0.wp.com/cdn-images-1.medium.com/max/800/1*jav9zG8B-cQIUrkqmDyVLA.jpeg?w=900&ssl=1

Characterizing the Bottleneck

- **RTT** is the **measured** round-trip time
- Can be decomposed in **RTprop**, the fixed **physical property** of the specific **path**, and **noise** $\eta \geq 0$ “time in queues” and other noise

$$RTT_t$$

Characterizing the Bottleneck

Estimator for RTprop

- Assumption path change in time scale >> RTprop

$$RTT_t = RTprop_t + \eta_t$$

Characterizing the Bottleneck

Estimator for RTprop

- Assumption path change in time scale >> RTprop
- ***RTT without noise = RTprop***

$$RTprop + \eta_t = RTT_t$$

Characterizing the Bottleneck

Estimator for RTprop

- Assumption path change in time scale >> RTprop (W_R in 10 sec to minutes)
- ***RTT without noise = RTprop***
- Noise is not measurable: the best estimation is measured at minimal noise

$$\widehat{RTprop} = RTprop + min\left(\eta_t\right) = min\left(RTT_t\right)$$

$$\forall t \in [T - W_R, T]$$

Characterizing the Bottleneck

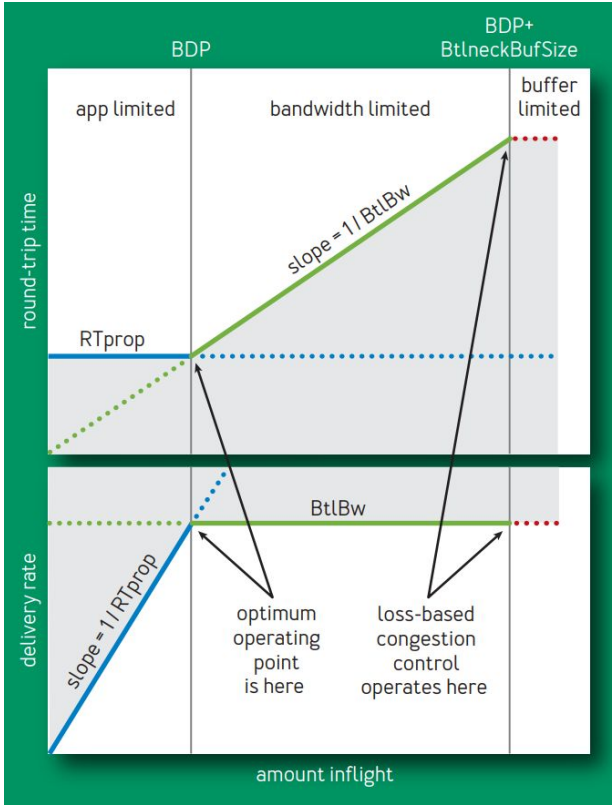
- TCP does not require to measure any bottleneck bandwidth
- But the average $deliveryRate = \frac{\Delta delivered}{\Delta t}$ can be computed with
  - Δt from the RTT
  - $\Delta delivered$ as the data delivered in Δt
- … which is upper-bounded by the bottleneck and therefore:

$$\widehat{BtlBw} = max\left(deliveryRate\right)$$

$$\forall t \in [T - W_B, T]$$

Characterizing the Bottleneck

- Each ack is used as a measurement for RTT and the delivery rate
- Proposed estimator/filters convert to RTprop and BtlBw
- Uncertainty principle does not allow a measurement at the same time
- Operating point specific tracking of RTprop OR BtlBw
- Detailed algorithm…



Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

Core BBR Algorithm

- When an ack is received
- When data is sent

onAck (main points)

1. When an ack is received, new RTT and delivery rate measurements are provided
2. New measurements are used to update RTprop and BtlBw estimates

```
function onAck(packet)
  rtt = now - packet.sendtime
  update_min_filter(RTpropFilter, rtt)
  delivered += packet.size
  delivered_time = now
  deliveryRate = (delivered - packet.delivered)
                 /(now - packet.delivered_time)
  if (deliveryRate > BtlBwFilter.currentMax
      || ! packet.app_limited)
      update_max_filter(BtlBwFilter,
                        deliveryRate)
  if (app_limited_until > 0)
      app_limited_until - = packet.size
```

Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

Updates RTprop

Updates BtlBW

```
function onAck(packet)
  rtt = now - packet.sendtime
  update_min_filter(RTpropFilter, rtt)
  delivered += packet.size
  delivered_time = now
  deliveryRate = (delivered - packet.delivered)
                 /(now - packet.delivered_time)
  if (deliveryRate > BtlBwFilter.currentMax
      || ! packet.app_limited)
      update_max_filter(BtlBwFilter,
                        deliveryRate)
  if (app_limited_until > 0)
      app_limited_until - = packet.size
```

Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

```
function onAck(packet)
  rtt = now - packet.sendtime
  update_min_filter(RTpropFilter, rtt)
  delivered += packet.size
  delivered_time = now
  deliveryRate = (delivered - packet.delivered)
                 /(now - packet.delivered_time)
  if (deliveryRate > BtlBwFilter.currentMax
      || ! packet.app_limited)
      update_max_filter(BtlBwFilter,
                        deliveryRate)
  if (app_limited_until > 0)
      app_limited_until - = packet.size
```

Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

send (main points)

1. BBR paces packets so that the packet-arrival rate matches the bottleneck link's departure rate
2. BBR's primary and secondary control parameters are pacing_rate and cwnd_gain

```
function send(packet)
    bdp = BtlBwFilter.currentMax
          * RTpropFilter.currentMin
    if (inflight >= cwnd_gain * bdp)
        // wait for ack or timeout
        return
    if (now >= nextSendTime)
        packet = nextPacketToSend()
        if (! packet)
            app_limited_until = inflight
            return
        packet.app_limited =
                 (app_limited_until > 0)
        packet.sendtime = now
        packet.delivered = delivered
        packet.delivered_time = delivered_time
        ship(packet)
        nextSendTime = now + packet.size /
               (pacing_gain *
                BtlBwFilter.currentMax)
    timerCallbackAt(send, nextSendTime)
```

Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

Bounds inflight to handle common network and receiver issues (i.e. delayed and stretched acks)

```
function send(packet)
    bdp = BtlBwFilter.currentMax
          * RTpropFilter.currentMin
    if (inflight >= cwnd_gain * bdp)
        // wait for ack or timeout
        return
    if (now >= nextSendTime)
        packet = nextPacketToSend()
        if (! packet)
            app_limited_until = inflight
            return
        packet.app_limited =
                 (app_limited_until > 0)
        packet.sendtime = now
        packet.delivered = delivered
        packet.delivered_time = delivered_time
        ship(packet)
        nextSendTime = now + packet.size /
               (pacing_gain *
                BtlBwFilter.currentMax)
    timerCallbackAt(send, nextSendTime)
```

Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

application-limited

```
function send(packet)
    bdp = BtlBwFilter.currentMax
          * RTpropFilter.currentMin
    if (inflight >= cwnd_gain * bdp)
        // wait for ack or timeout
        return
    if (now >= nextSendTime)
        packet = nextPacketToSend()
        if (! packet)
            app_limited_until = inflight
            return
        packet.app_limited =
                 (app_limited_until > 0)
        packet.sendtime = now
        packet.delivered = delivered
        packet.delivered_time = delivered_time
        ship(packet)
        nextSendTime = now + packet.size /
               (pacing_gain *
                BtlBwFilter.currentMax)
    timerCallbackAt(send, nextSendTime)
```

Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

```
function send(packet)
    bdp = BtlBwFilter.currentMax
          * RTpropFilter.currentMin
    if (inflight >= cwnd_gain * bdp)
        // wait for ack or timeout
        return
    if (now >= nextSendTime)
        packet = nextPacketToSend()
        if (! packet)
            app_limited_until = inflight
            return
        packet.app_limited =
                (app_limited_until > 0)
        packet.sendtime = now
        packet.delivered = delivered
        packet.delivered_time = delivered_time
        ship(packet)
        nextSendTime = now + packet.size /
               (pacing_gain *
                BtlBwFilter.currentMax)
    timerCallbackAt(send, nextSendTime)
```

Pacing to match the bottleneck rate

pacing_rate

Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

Pacing_gain

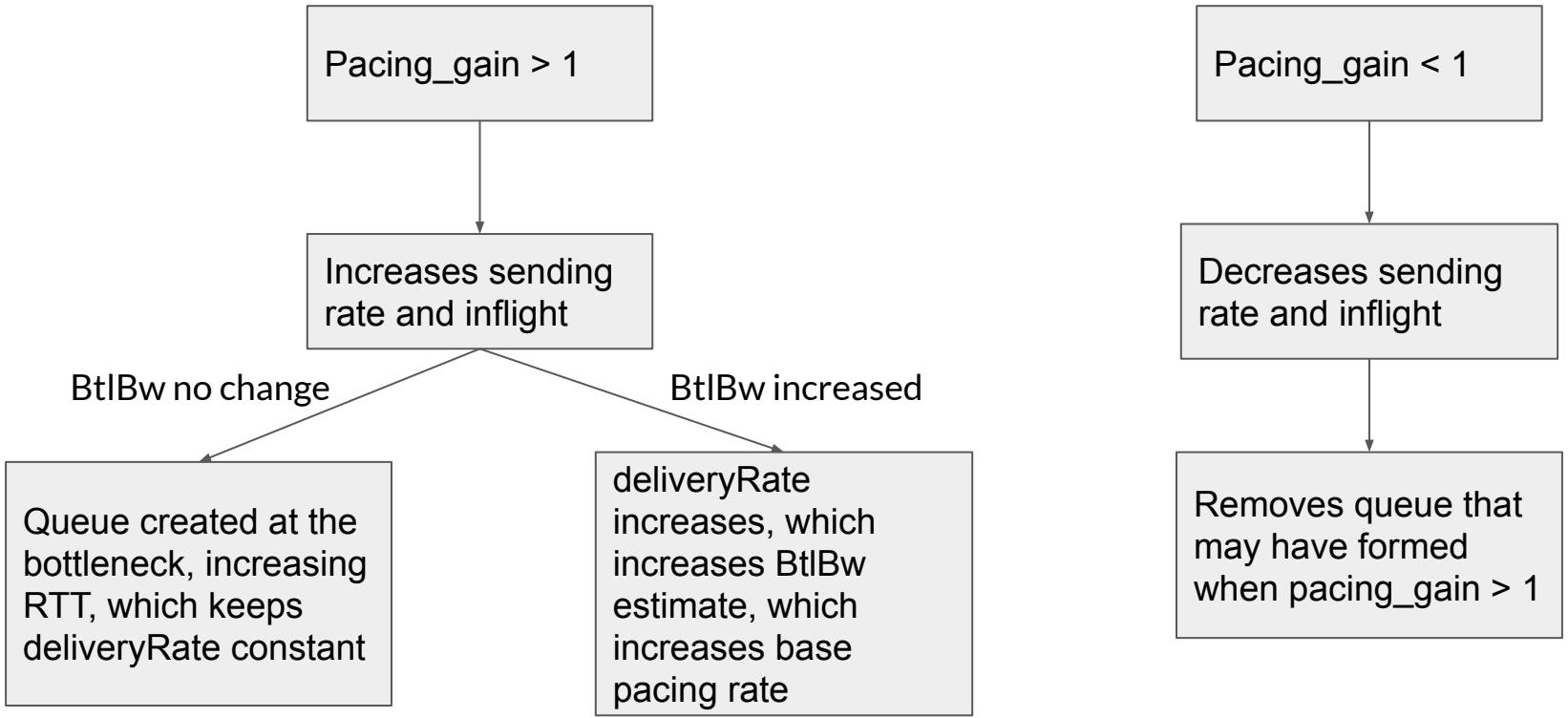
- Pacing_gain > 1 increasing inflight and decreases packet inter-arrival time
  - Pacing _gain < 1 does the opposite
- Pacing_gain is used to implement a state machine for sequential probing
  - Either tests for higher bandwidths or lower round-trip times
  - BBR sends faster to check for BtlBw increases
  - BBR sends slower to check for RTprop decreases

Pacing_gain

- Paci arrival time
  -
- Pacing_gain is used to implement a state machine for sequential probing
  - Either tests for higher bandwidths or lower round-trip times
  - BBR sends faster to check for BtlBw increases
  - BBR sends slower to check for RTprop decreases

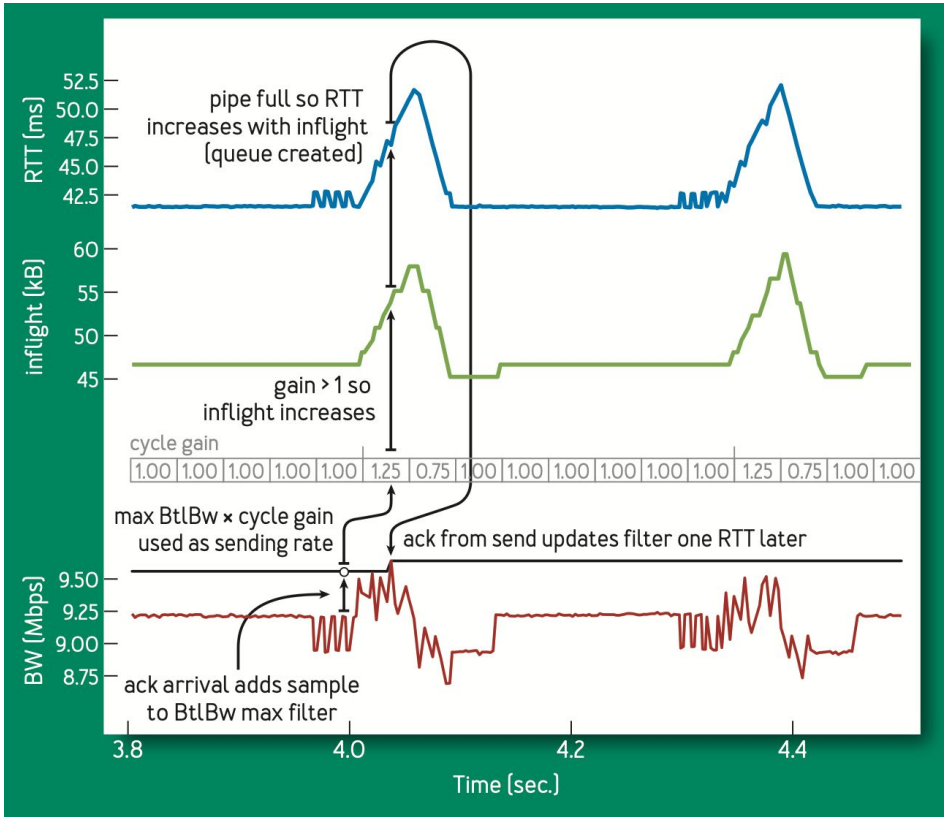
Why do we not have to check for BtlBw decreases or RTprop increases?

Pacing_gain


Pacing_gain > 1
Increases sending rate and inflight
BtlBw no change
BtlBw increased
Queue created at the bottleneck, increasing RTT, which keeps deliveryRate constant
deliveryRate increases, which increases BtlBw estimate, which increases base pacing rate
Pacing_gain < 1
Decreases sending rate and inflight
Removes queue that may have formed when pacing_gain > 1

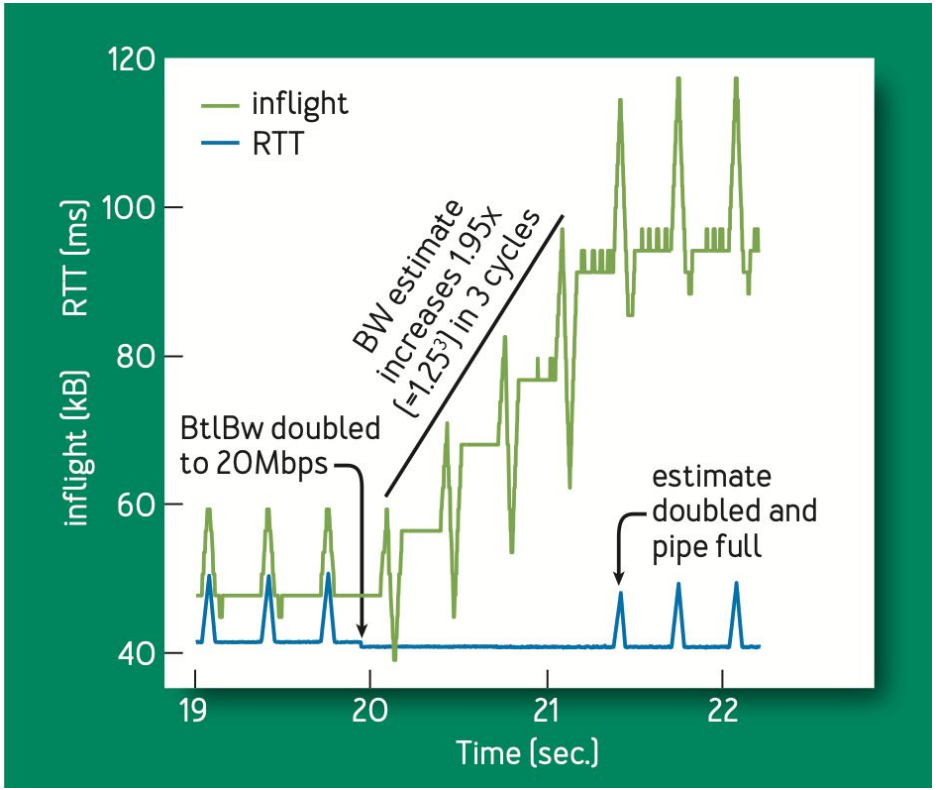
Steady-state Behavior

- The rate and amount BBR sends is a function of the estimated RTprop and BtlBw values

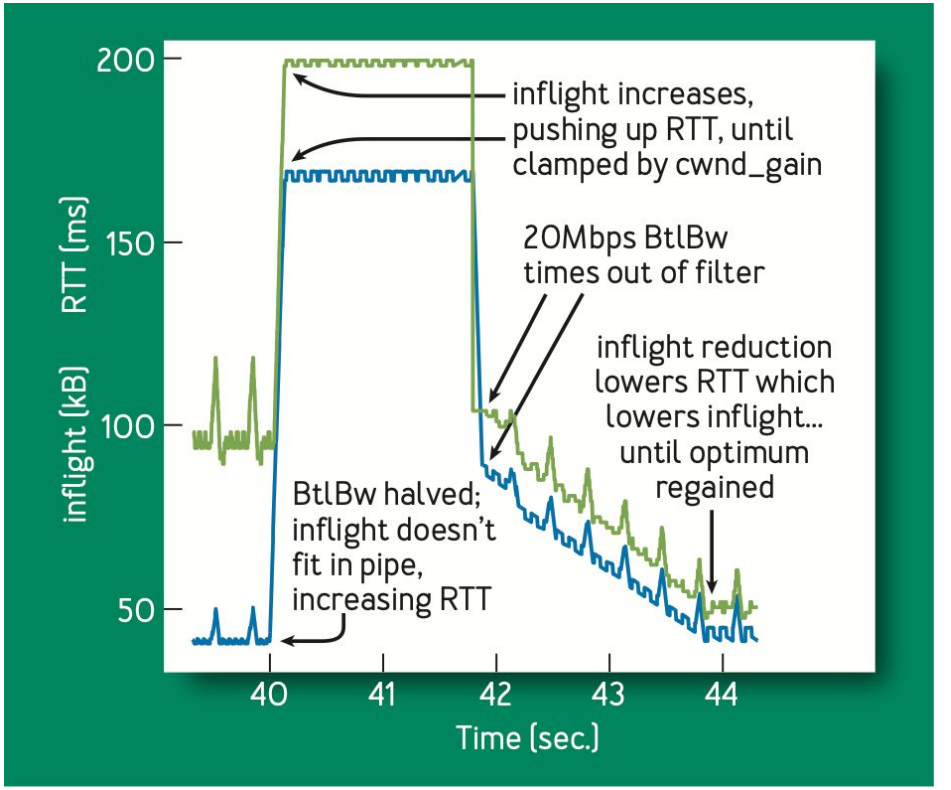


RTT
Inflight
Delivery rate

Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.



a.



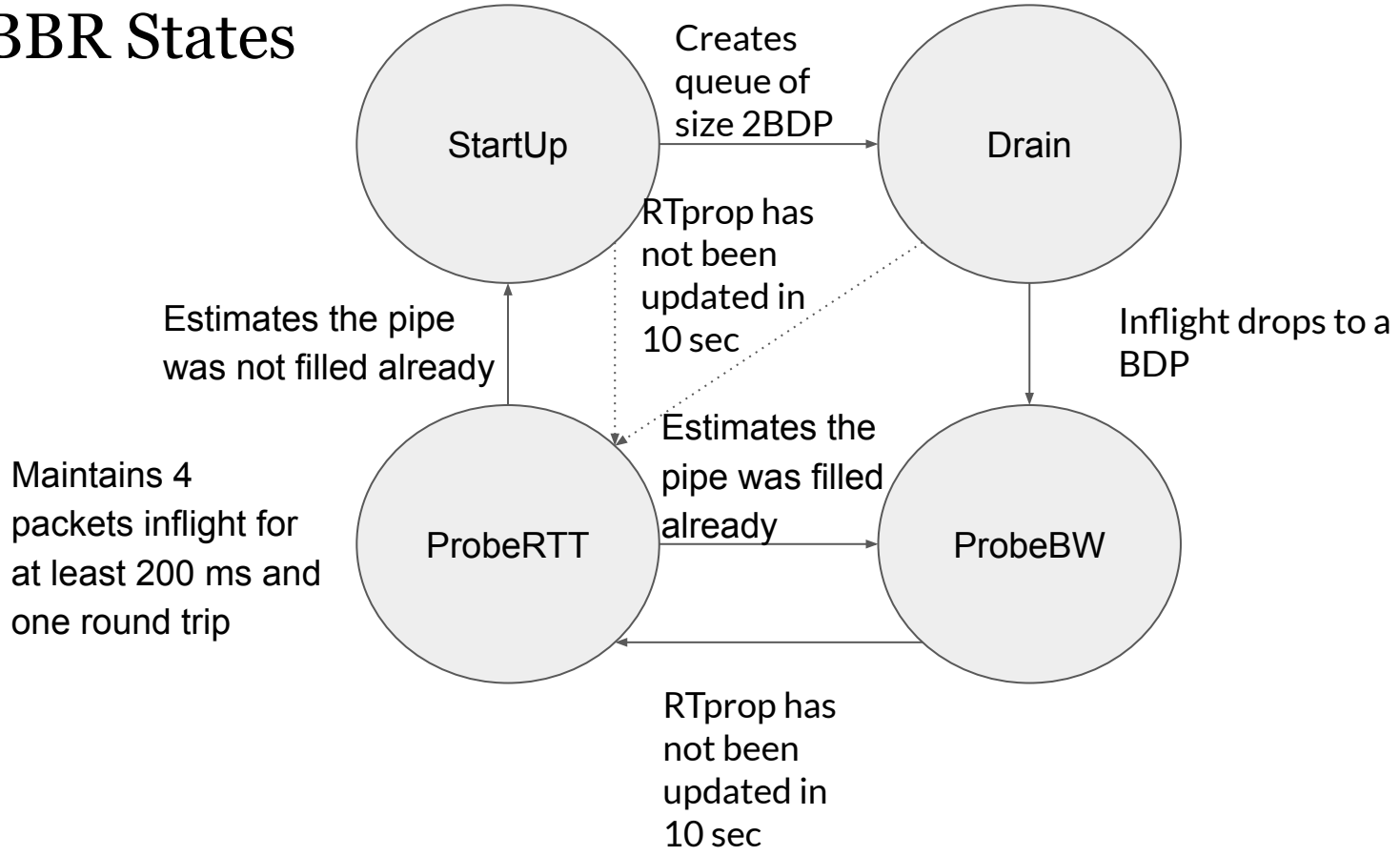
b.

Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

Single BBR Flow Startup Behavior

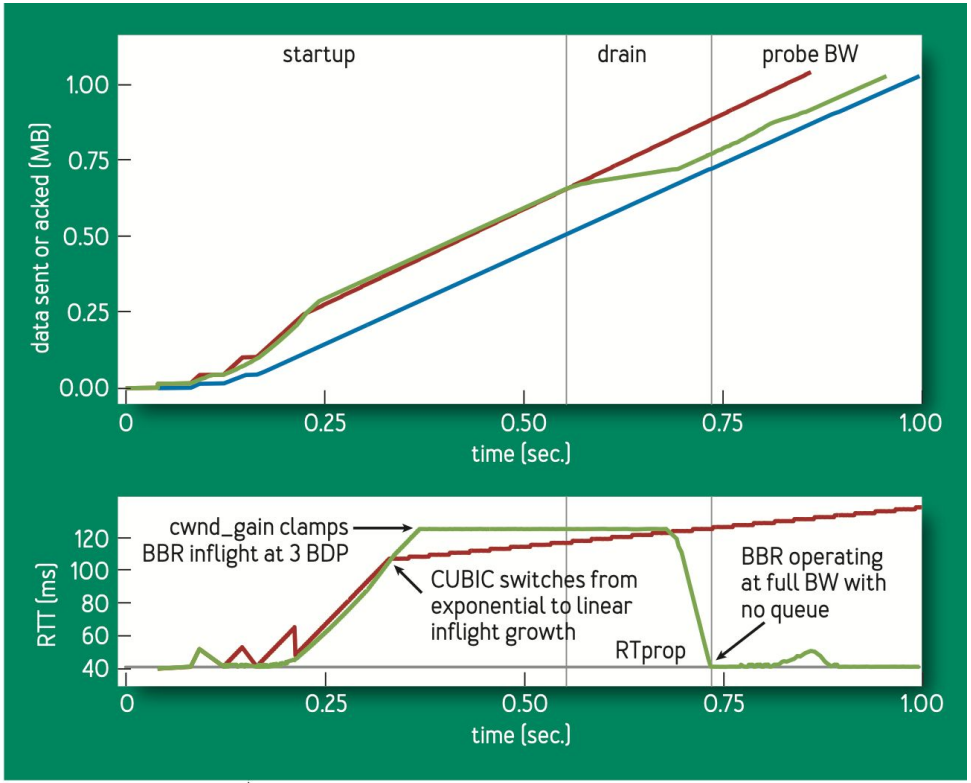
1. Startup, shutdown, and loss recovery do not require event-specific algorithms
2. Events are handled by cycling through a set of states, which are each defined by tables that have fixed gains and exit criteria
3. Most time is sent in the ProbeBW state
4. Startup and Drain states are employed sequentially when a connection starts

BBR States


StartUp
Creates queue of size 2BDP
Drain
RTprop has not been updated in 10 sec
Estimates the pipe was not filled already
Inflight drops to a BDP
Maintains 4 packets inflight for at least 200 ms and one round trip
ProbeRTT
Estimates the pipe was filled already
ProbeBW
RTprop has not been updated in 10 sec

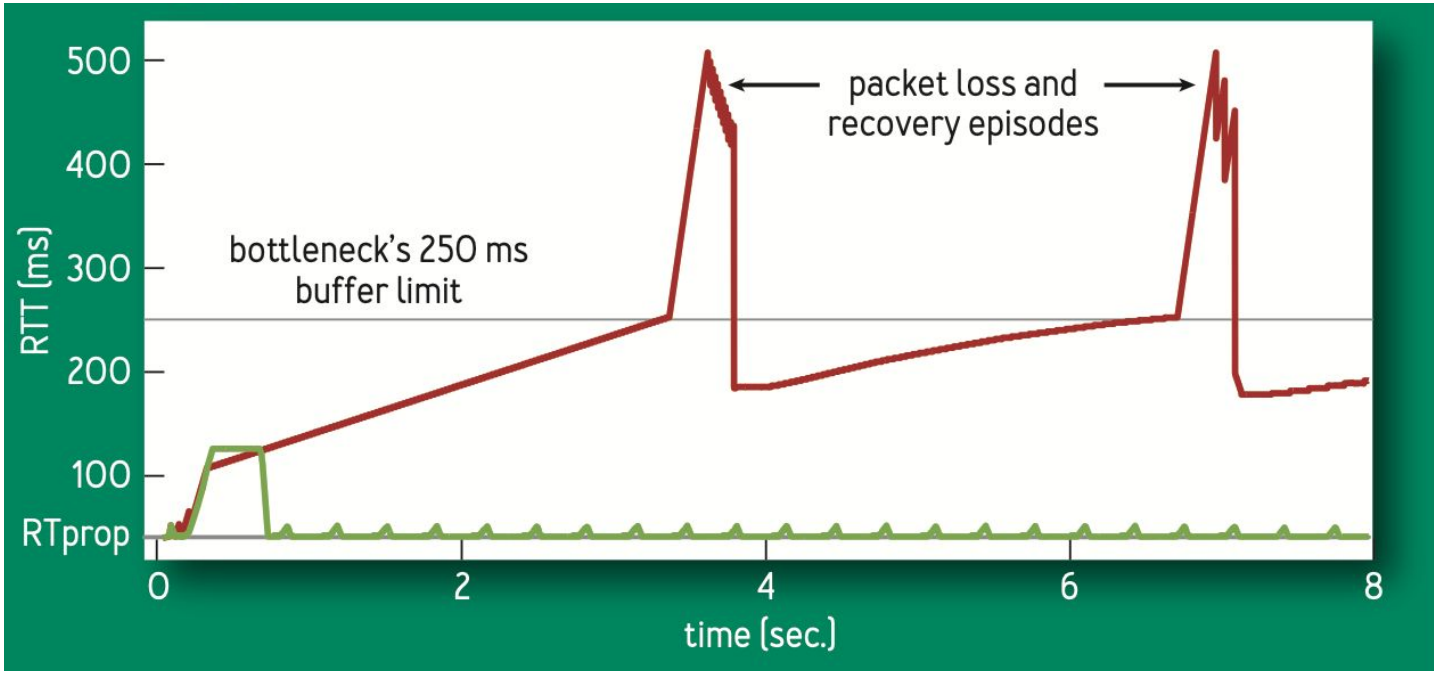
BBR vs. CUBIC

Receiver
Sender
CUBIC Sender



Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

BBR vs. CUBIC



BBR Sender
CUBIC Sender

Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

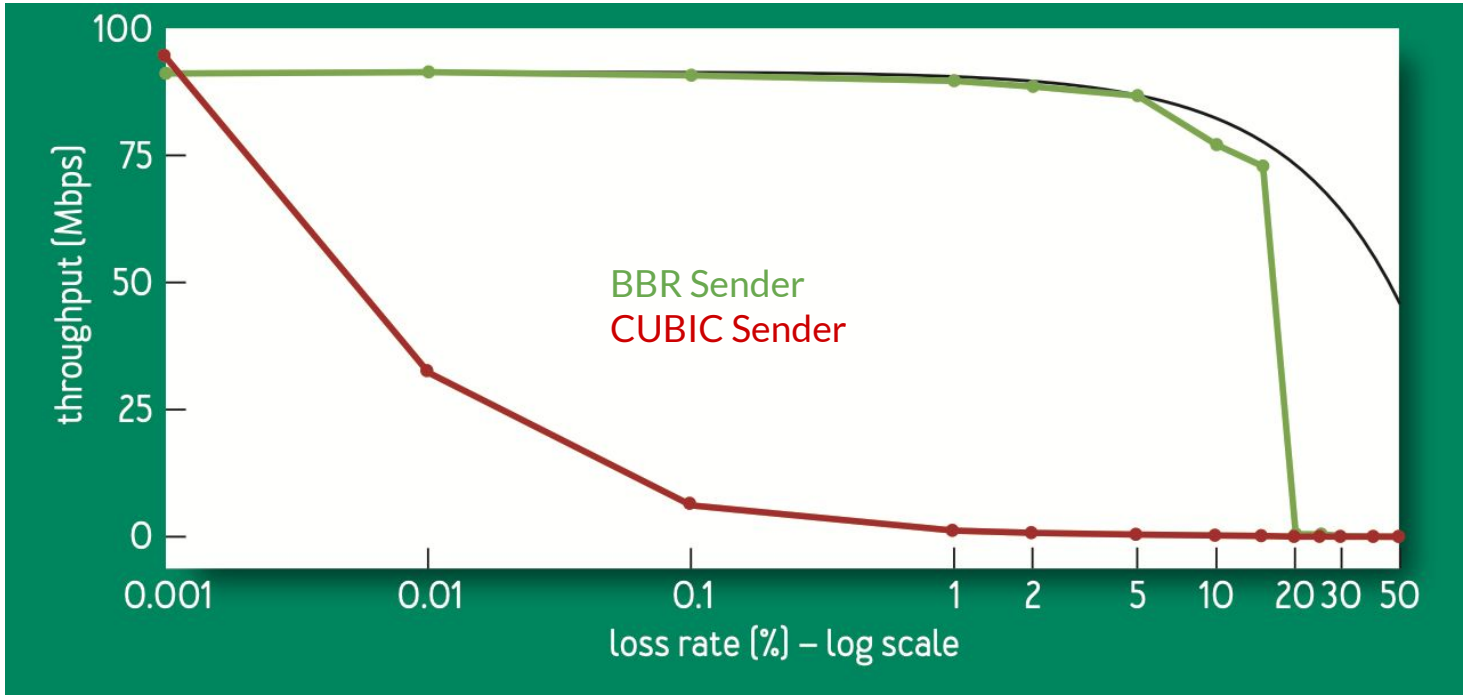
Additional Experiments

1. Google B4 WAN deployment experience
2. Youtube Edge deployment experience

Google B4 WAN Deployment Experience

- Switched from CUBIC to BBR
- Experienced no issues or regressions
- BBR's throughput is consistently 2 to 25 times greater than CUBIC's
  - Reached 133 times relative improvement when utilizing a path that was not limited by a receiver buffer that was deliberately set low to prevent CUBIC from flooding the network
- Attributed success to BBR not using loss as a congestion indicator

Google B4 WAN Deployment Experience



Youtube Edge Deployment Experience

- Playbacks using BBR had improvements in all of Youtube's quality-of-experience metrics
- BBR reduces median RTT by 53% on average globally, and by >80% on average in the developing world
    - BBR keeps queue to a near minimum, independent of bottleneck buffer size and number of active flows
    - CUBIC flows always fill the buffer, causing delay to grow linearly with buffer size

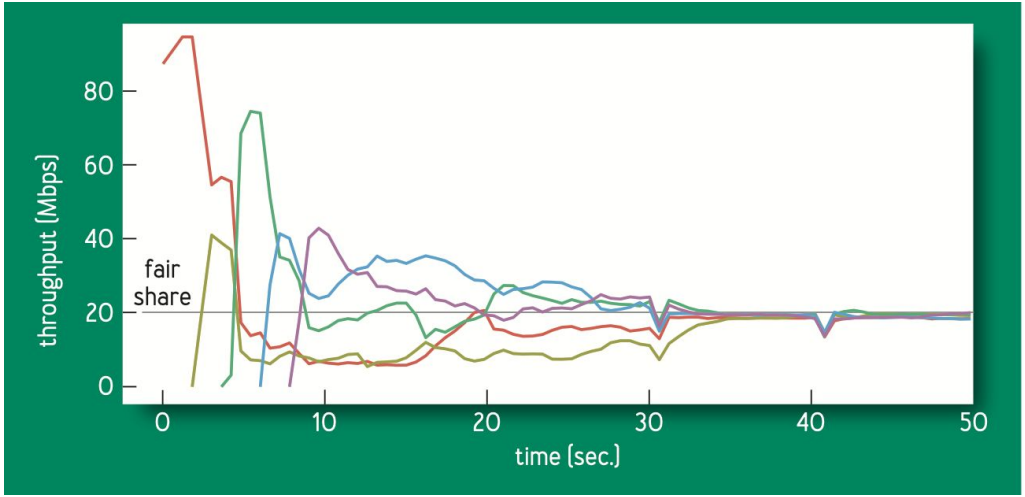
Youtube Edge Deployment Experience

- Playbacks using BBR had improvements in all of Youtube's quality-o
- BBR redu on average in the de
  - BBR number of activ
  - CUBI e

What are limitations of these tests? Are you satisfied that BBR is an improvement over CUBIC in the real world based on these tests?

Bandwidth Fairness

Setup: 5 flows, same RTTmin, different start times and initial states

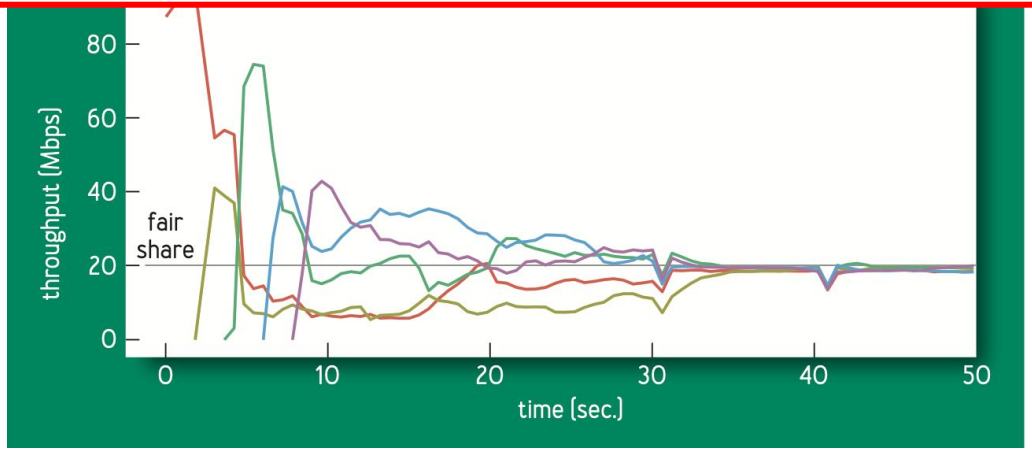


Convergence towards a fair share among all flows

Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

Bandwidth

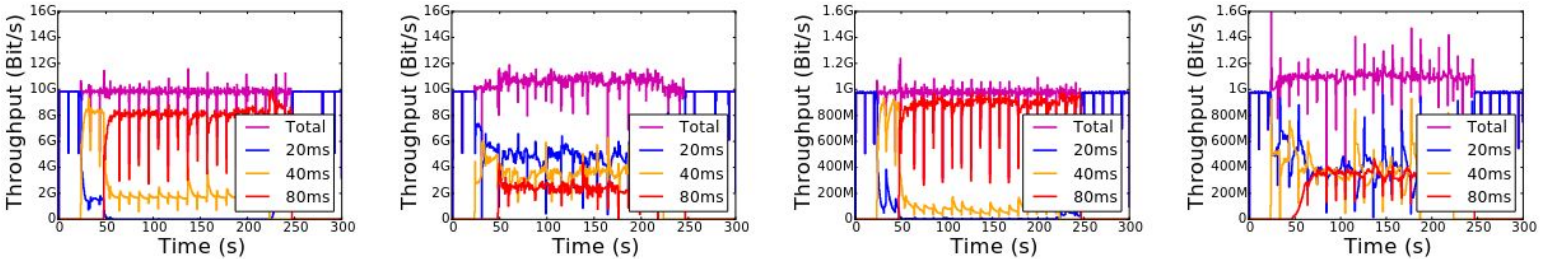
What are limitations and assumptions of these experiments? What other experiments would you want to see?



Cardwell, Neal, et al. "BBR: congestion-based congestion control." *Communications of the ACM* 60.2 (2017): 58-66.

Bandwidth Fairness (External Experiments)

- Three BBR flows with different RTTmin
- Flow with highest RTTmin surprasses other flows in systems with large buffer
- Most likely limited by their inflight data/*BDP* estimation
- Smaller Buffer don't show those results -> BtlBw limited



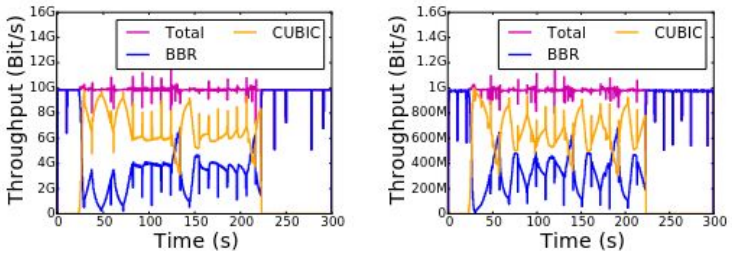
(a) 10 Gbit/s large buffer (b) 10 Gbit/s small buffer (c) 1 Gbit/s large buffer (d) 1 Gbit/s small buffer

Fig. 14: Three BBR flows with different RTT_{min} (20 ms, 40 ms, 80 ms)

Hock, Mario, Roland Bless, and Martina Zitterbart. "Experimental evaluation of BBR congestion control." 2017 IEEE 25th international conference on network protocols (ICNP). IEEE, 2017.

Bandwidth Fairness (External Experiments)

- What about systems with CUBIC and BBR flows?
- Share leaning towards CUBIC flow for larger buffer at the bottleneck, where CUBIC fills the buffer
- Overestimation of RTTmin leads to an increasing behavior of the BBR flow after ProbeRTT



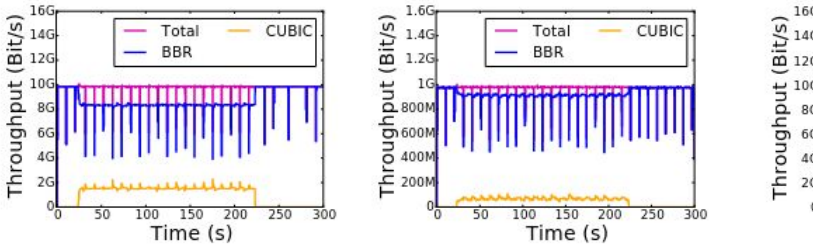
(a) 10 Gbit/s – throughput (b) 1 Gbit/s – throughput

Fig. 15: BBR vs. CUBIC TCP (large buffer)

Hock, Mario, Roland Bless, and Martina Zitterbart. "Experimental evaluation of BBR congestion control." 2017 IEEE 25th international conference on network protocols (ICNP). IEEE, 2017.

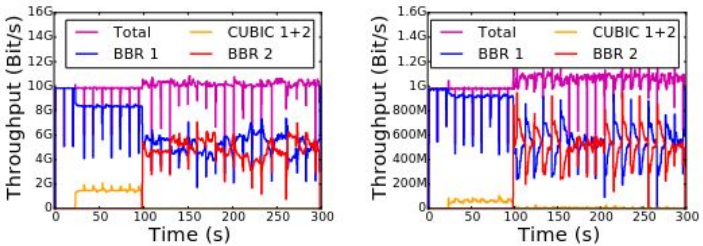
Bandwidth Fairness (External Experiments)

- Competing BBR flows with a buffer is smaller than one bdp -> packet loss for BBR flows
- CUBIC TCP assumes congestion and reduces flow



(a) 10 Gbit/s (b) 1 Gbit/s

Fig. 16: BBR vs. CUBIC TCP (small buffer)



(a) 10 Gbit/s (b) 1 Gbit/s

Fig. 17: 2 BBR vs. 2 CUBIC TCP (small buffer)

Hock, Mario, Roland Bless, and Martina Zitterbart. "Experimental evaluation of BBR congestion control." 2017 IEEE 25th international conference on network protocols (ICNP). IEEE, 2017.

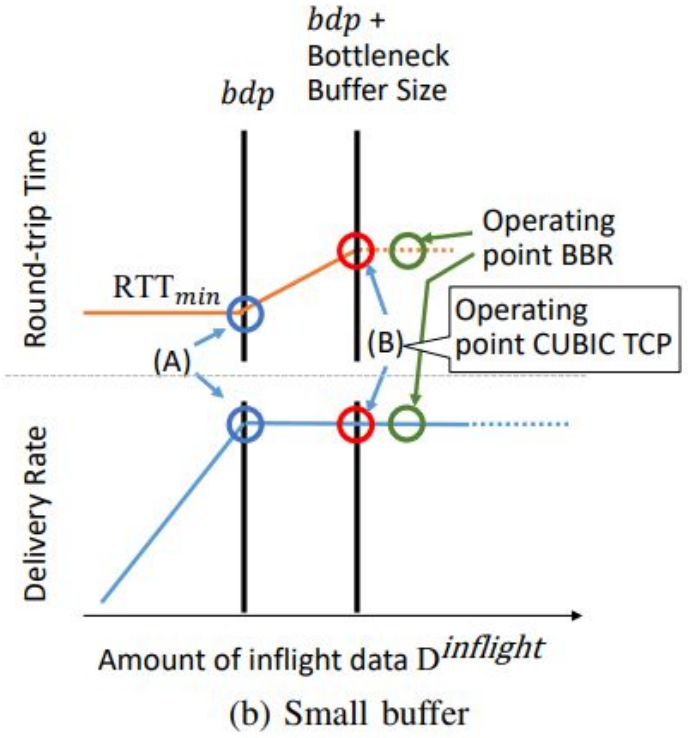
Packet Loss/Retransmission Rate

Packet Los

What would you guess is the packet loss/retransmission rate of BBR compared to CUBIC?

Packet Loss/Retransmission Rate

- Small buffer sizes ⇒ high packet loss
- Recall BBR keeps data inflight = 2* BDP
- Packet loss incurred before reaching BBR operating point
- BBR v2 attempts to alleviate this by including packet loss



Hock, Mario, Roland Bless, and Martina Zitterbart. "Experimental evaluation of BBR congestion control." 2017 IEEE 25th international conference on network protocols (ICNP). IEEE, 2017.

To summarize

- Current loss based congestion control increase congestion with large buffers
- Estimate RTProp using RTT and BtlBw with Delivery Rate
- 4 States
  - Startup
  - Drain
  - ProbeRTT
  - ProbeBW
- Limitations of BBR
  - Fairness
  - Packet Loss
  - (There are more)

- Why large buffers favor loss based competitors?

Further Discussion

- How is BBR's retransmission rate?
  - Terrible, especially with small buffers
  - BBRv2 uses aims to reduce this
  - BBR-Advanced (BBR-A) reduces congestion window and packet loss → reduce retransmissions and increase fairness

Fifty Shades of Congestion Control (Tukovic, et al.)

- BBR vs loss-based ⇒ loss-based dominates (fig. 10, 16)
  - When buffers are large. With small buffers we see the opposite.
- BBR shares fairly with delay-based algorithms (fig. 10)
- BBR is not stable with multiple flows (fig. 9)
- Adding more flows → flows with larger throughput claim more (fig. 12, 13, 15)