BBR: Congestion-Based Congestion Control

by Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, Van Jacobson
(paper review)

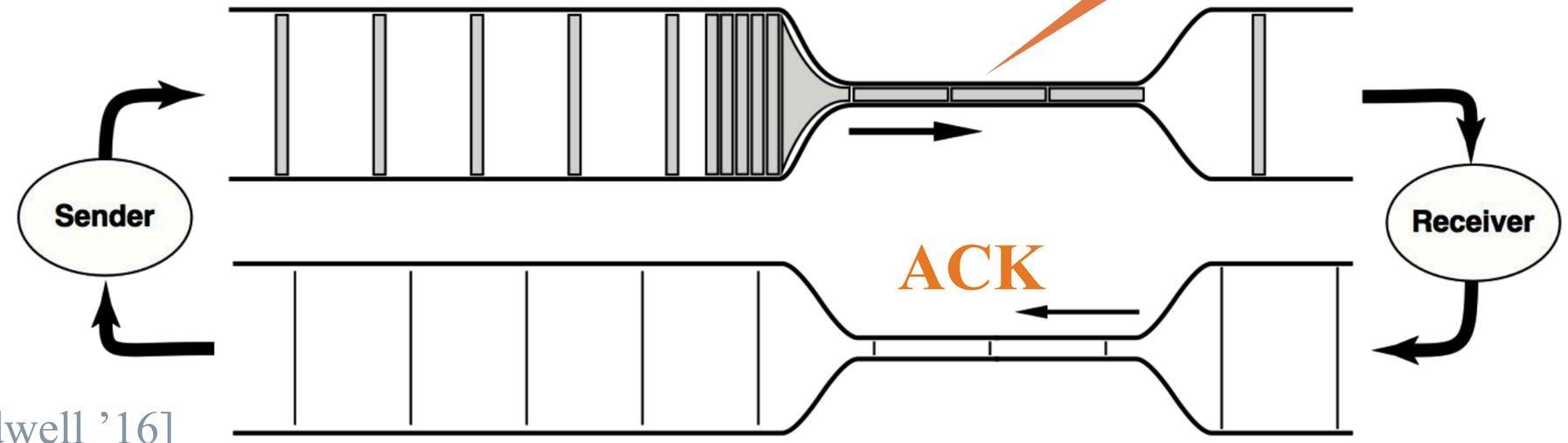
Hongyu Hè

hhy@princeton.edu

September 20, 2024

Context

Congestion Control Algorithms (CCAs)



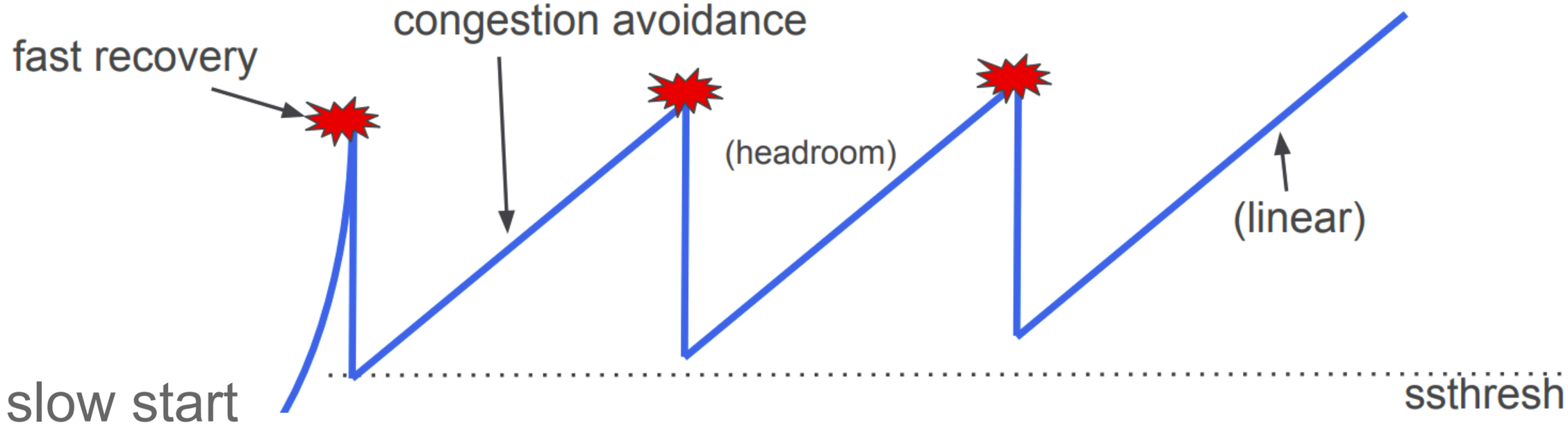
[Cardwell '16]

Motivation (2011–2013)

- NICs w/ more memory → excessive buffering → TCP bufferbloat
- Single-conn HTTPv2 << multi-conn HTTPv1
- Switches w/ shallow buffers have low TCP throughput

Problem Statement

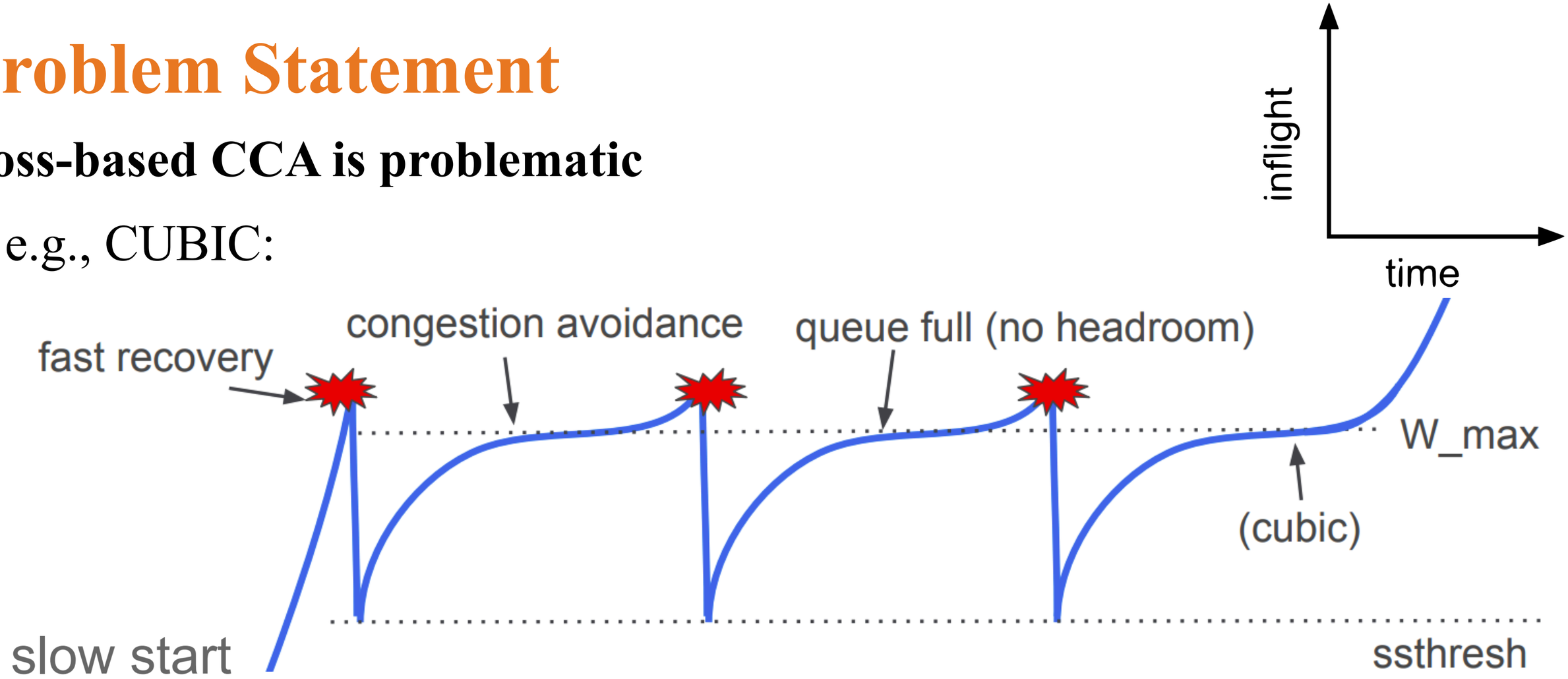
Loss-based CCA is problematic

- e.g., New Reno:
  - 

  - Linear growth: 1000× more BW needs 1000 increases
    - saturate 10Gbps BW; 50ms RTT → ~35min ⇒ loss rate ≤ 5.7e-10

Problem Statement

Loss-based CCA is problematic

- e.g., CUBIC:
  - 

  - Cubic growth: 1000× more BW needs 10 increases
    - saturate 10Gbps BW; 50ms RTT → ~7min ⇒ loss rate ≤ 2.86-e9

Problem Statement

Loss-based CCA is problematic

- Many CCAs were loss-based (e.g., Tahoe, New Reno, and CUBIC)
- Packet loss ⇒ congestion?
- Loss-based CCAs + shallow or deep buffers ⇒ poor performance

BBR: Bottleneck Bandwidth & Round-trip propagation time

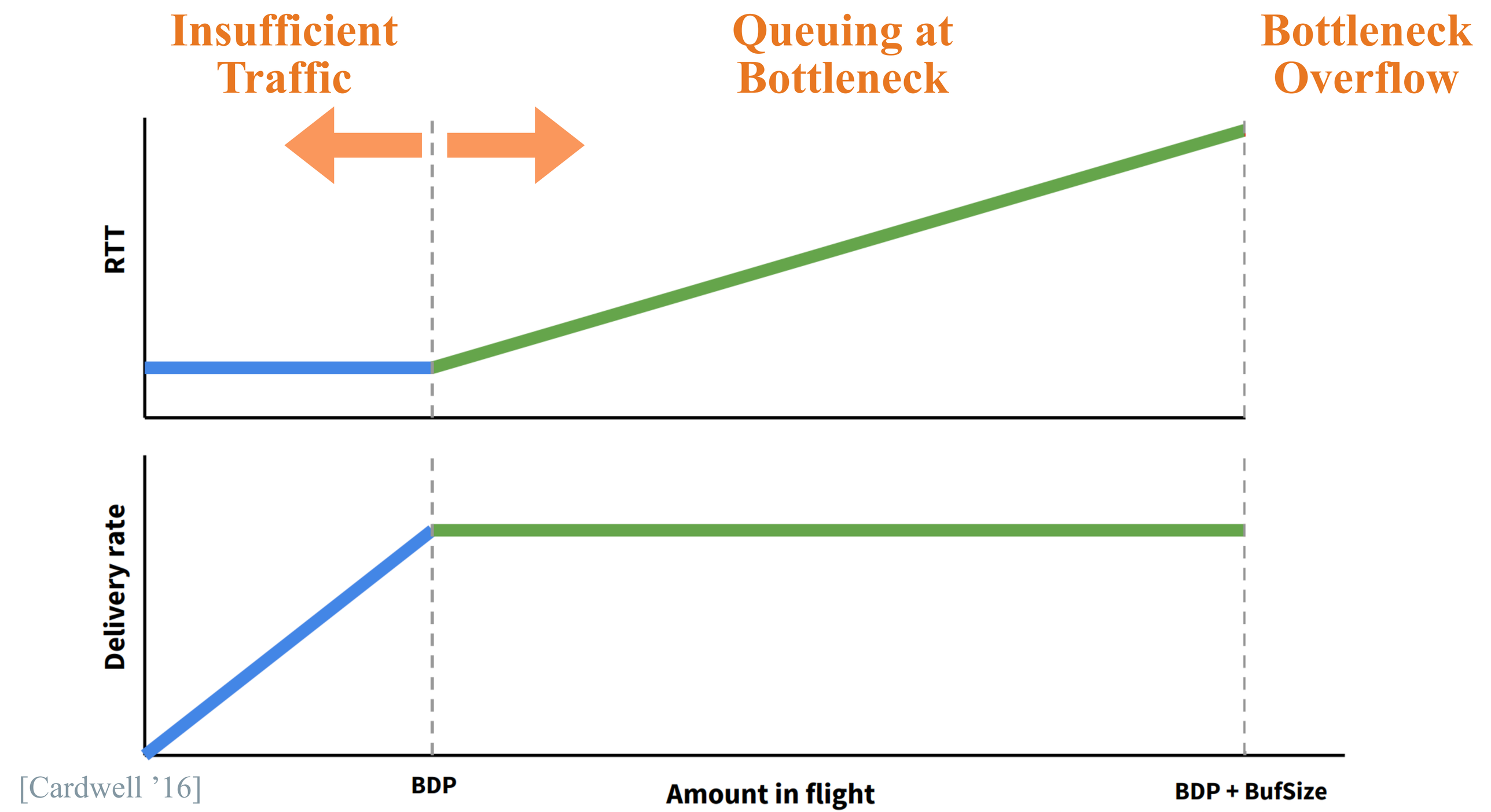
Key Idea:

Explicitly model bottleneck queue by probing the RTT and bottleneck BW periodically to estimate the bandwidth-delay product (BDP)

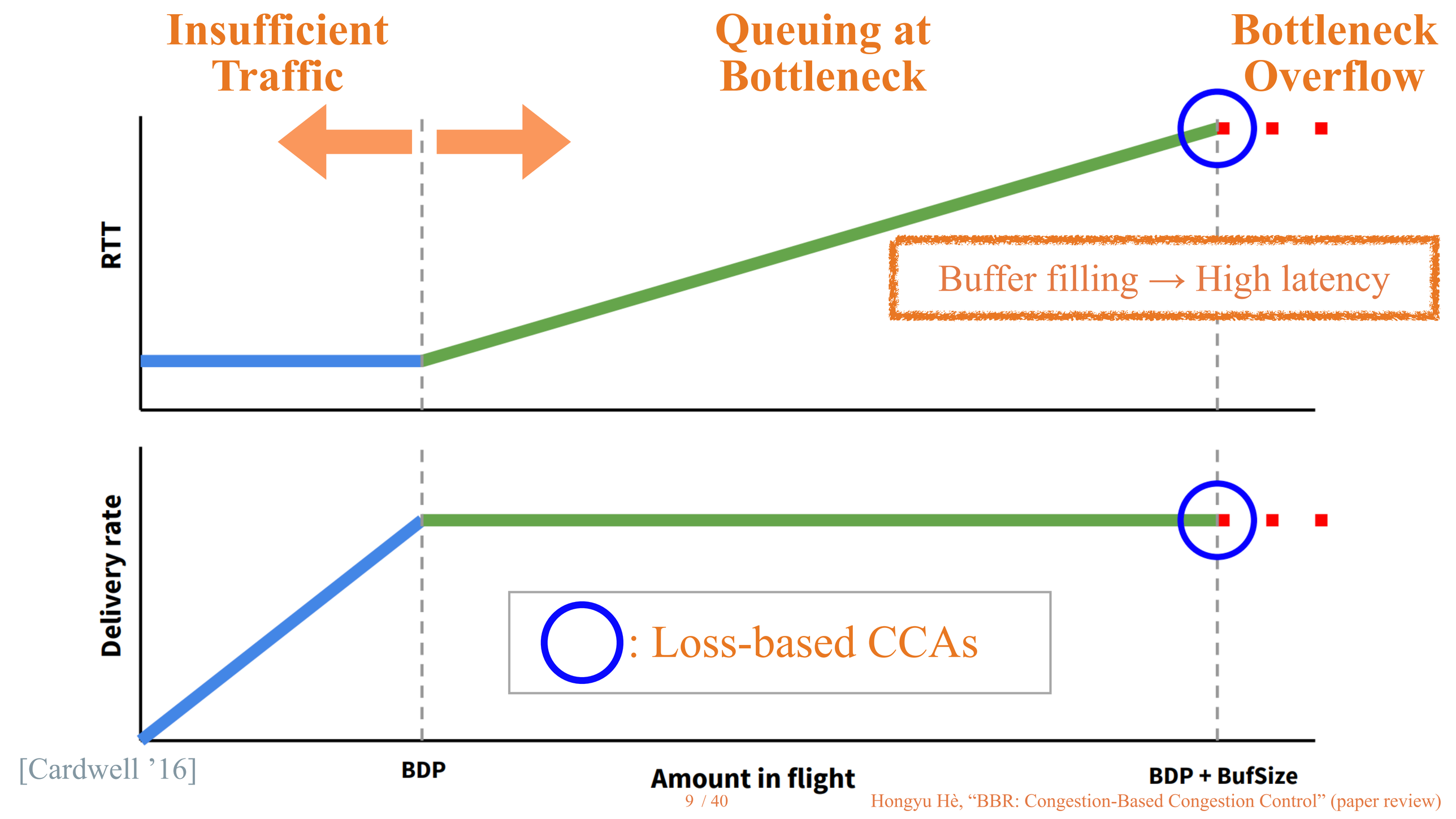
Main Contributions:

1) Identified and addressed a root cause for an internet-scale problem
2) BBR: client-side rate-based CCA, better latency and tput vs. CUBIC
   - Fast search rate $O(\log \text{BDP})$
3) Production deployment, evaluation, and linux integration with LTS

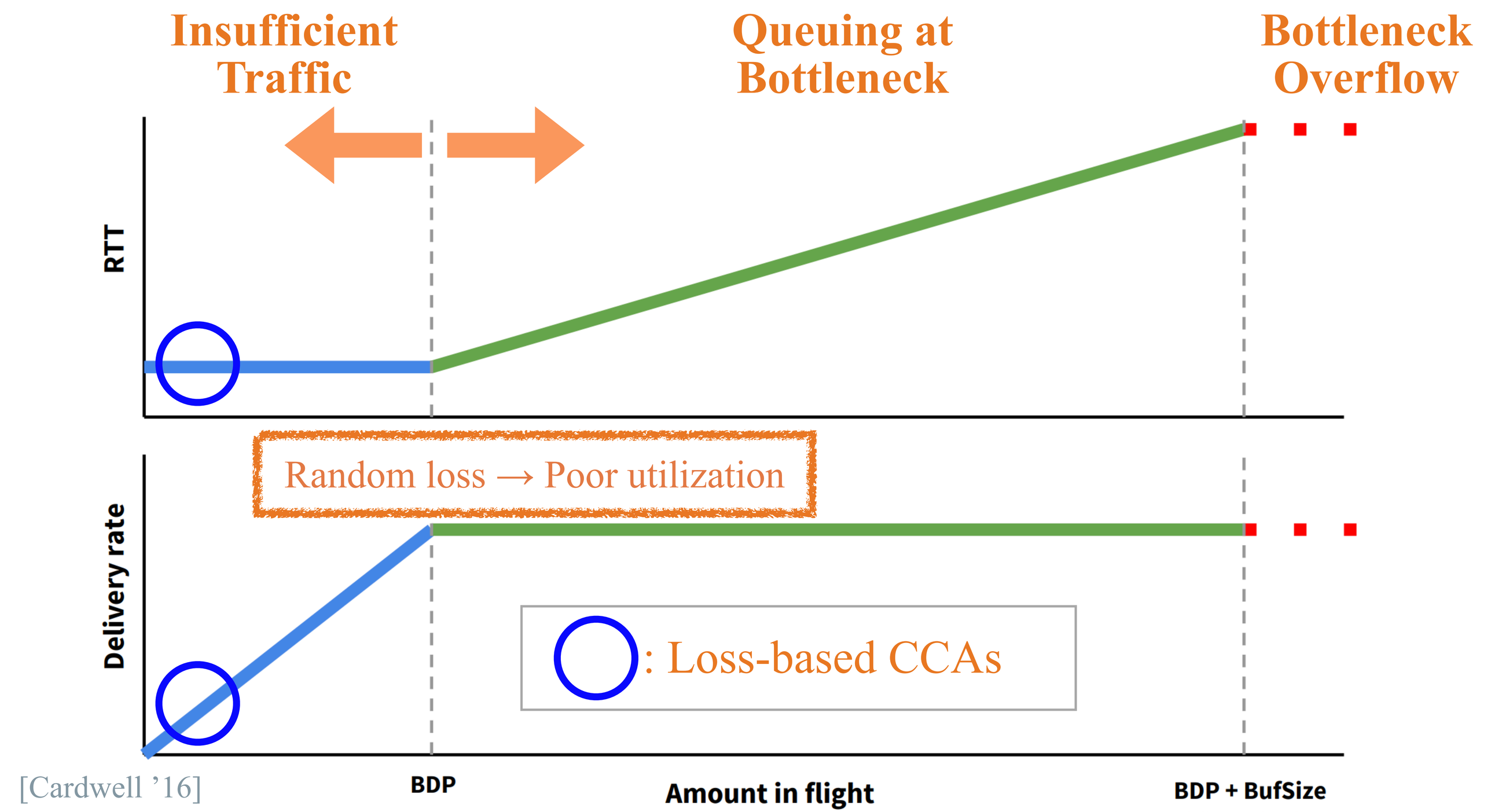
How does BBR work?


Insufficient Traffic
Queuing at Bottleneck
Bottleneck Overflow
RTT
Delivery rate
BDP
Amount in flight
BDP + BufSize

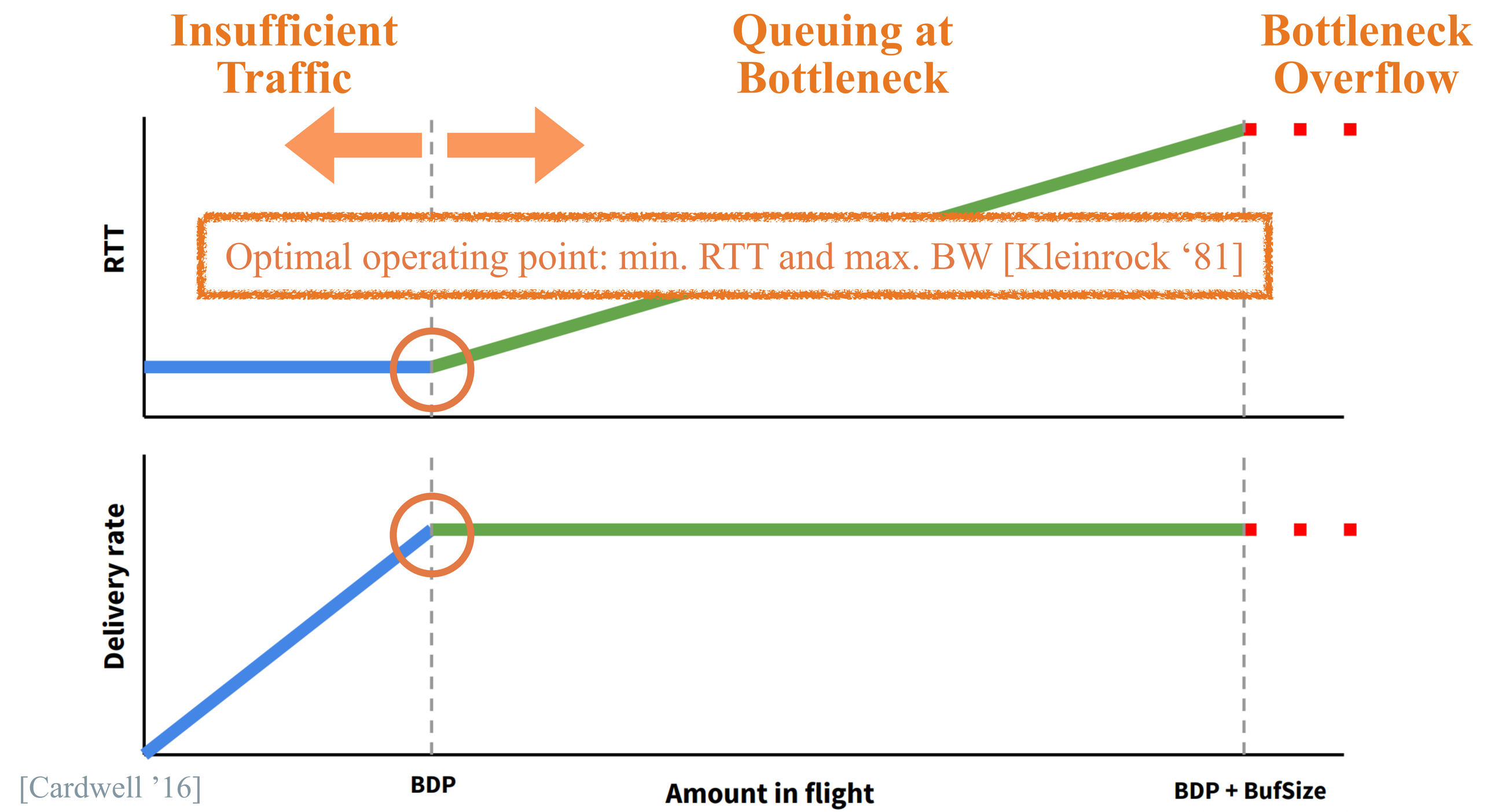
[Cardwell ’16]


Insufficient Traffic
Queuing at Bottleneck
Bottleneck Overflow
RTT
Buffer filling → High latency
Delivery rate
: Loss-based CCAs
BDP
Amount in flight
BDP + BufSize

[Cardwell '16]


Insufficient Traffic
Queuing at Bottleneck
Bottleneck Overflow
RTT
Random loss → Poor utilization
Delivery rate
: Loss-based CCAs
BDP
Amount in flight
BDP + BufSize

[Cardwell '16]


Insufficient Traffic
Queuing at Bottleneck
Bottleneck Overflow
RTT
Optimal operating point: min. RTT and max. BW [Kleinrock ‘81]
Delivery rate
BDP
Amount in flight
BDP + BufSize

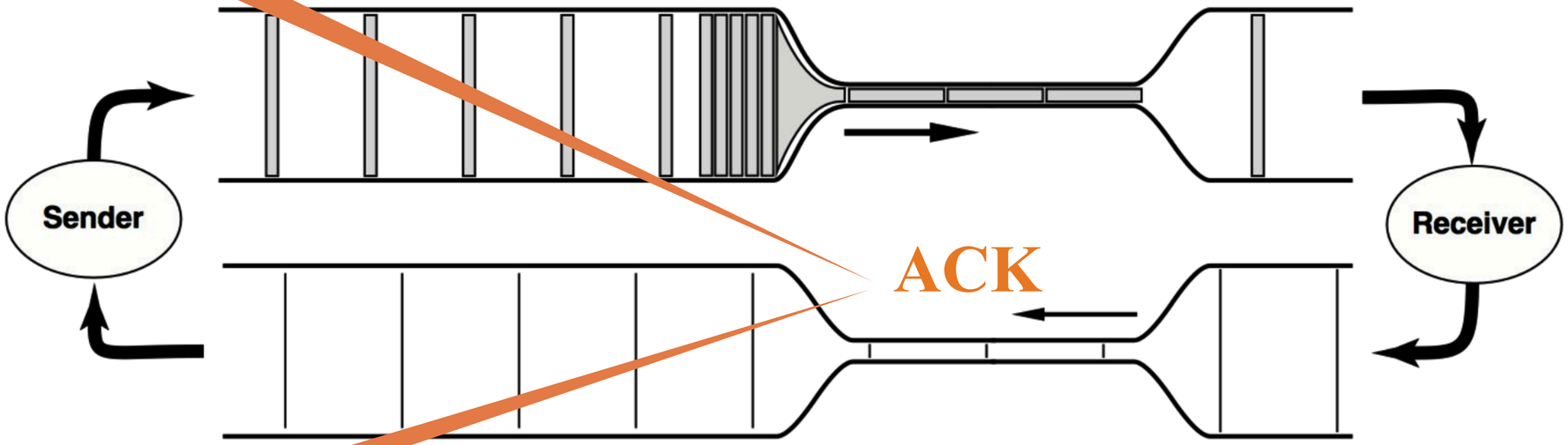
[Cardwell ’16]

BBR’s Design

Dynamically Estimate Windowed Max BW and Min RTT

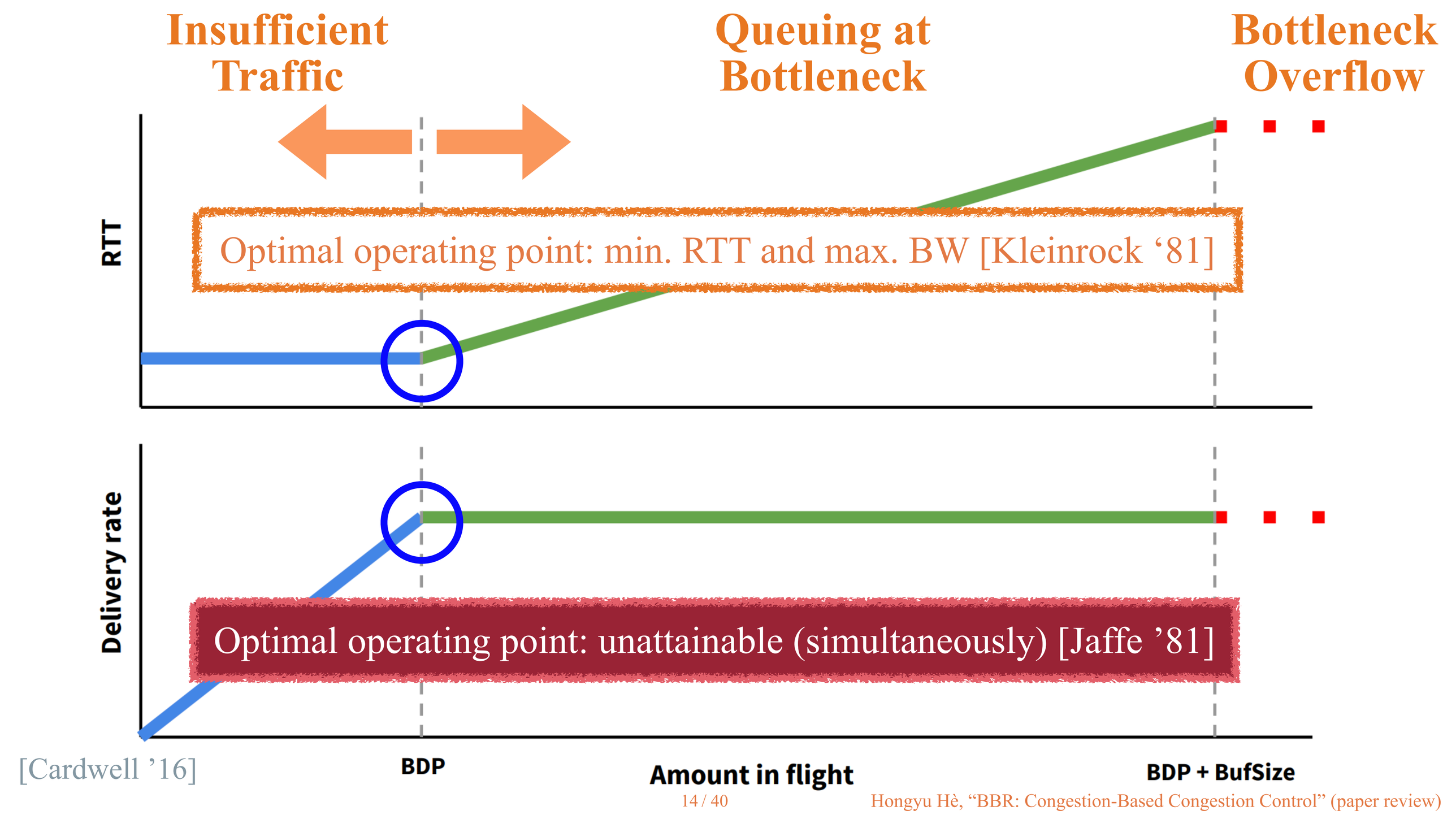
$\hat{\text{BW}}$

$$\forall i \in W_{\text{WB}} : \hat{\text{BW}} \leftarrow \max \left\{ (\Delta \text{delivered} / \Delta t)_i \right\}$$



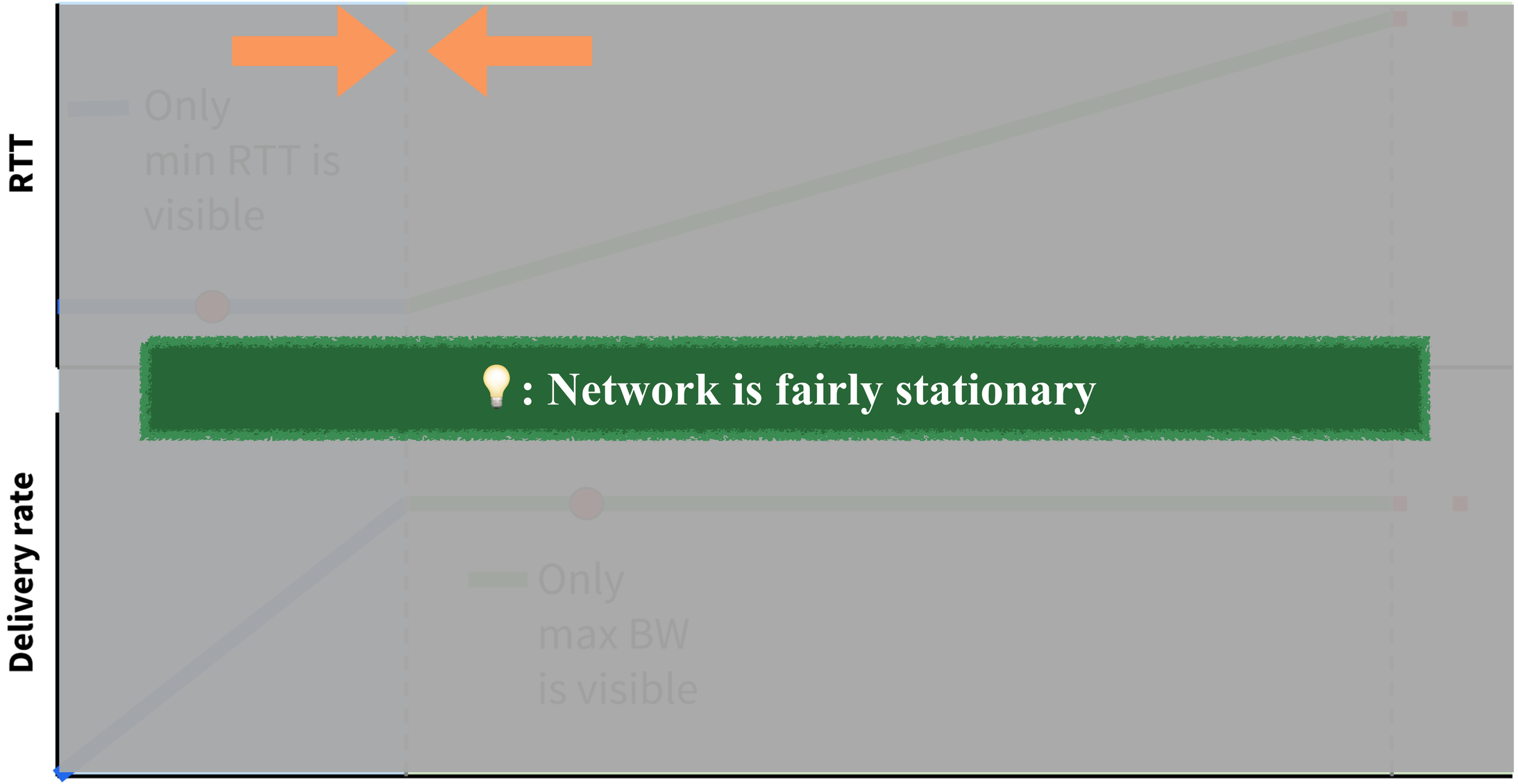
$\hat{\text{RTT}}$

$$\forall i \in W_{\text{RTT}} : \hat{\text{RTT}} \leftarrow \min \left\{ \text{RTT}_i \right\}$$


Insufficient Traffic
Queuing at Bottleneck
Bottleneck Overflow
RTT
Optimal operating point: min. RTT and max. BW [Kleinrock ‘81]
Delivery rate
Optimal operating point: unattainable (simultaneously) [Jaffe ’81]
BDP
Amount in flight
BDP + BufSize

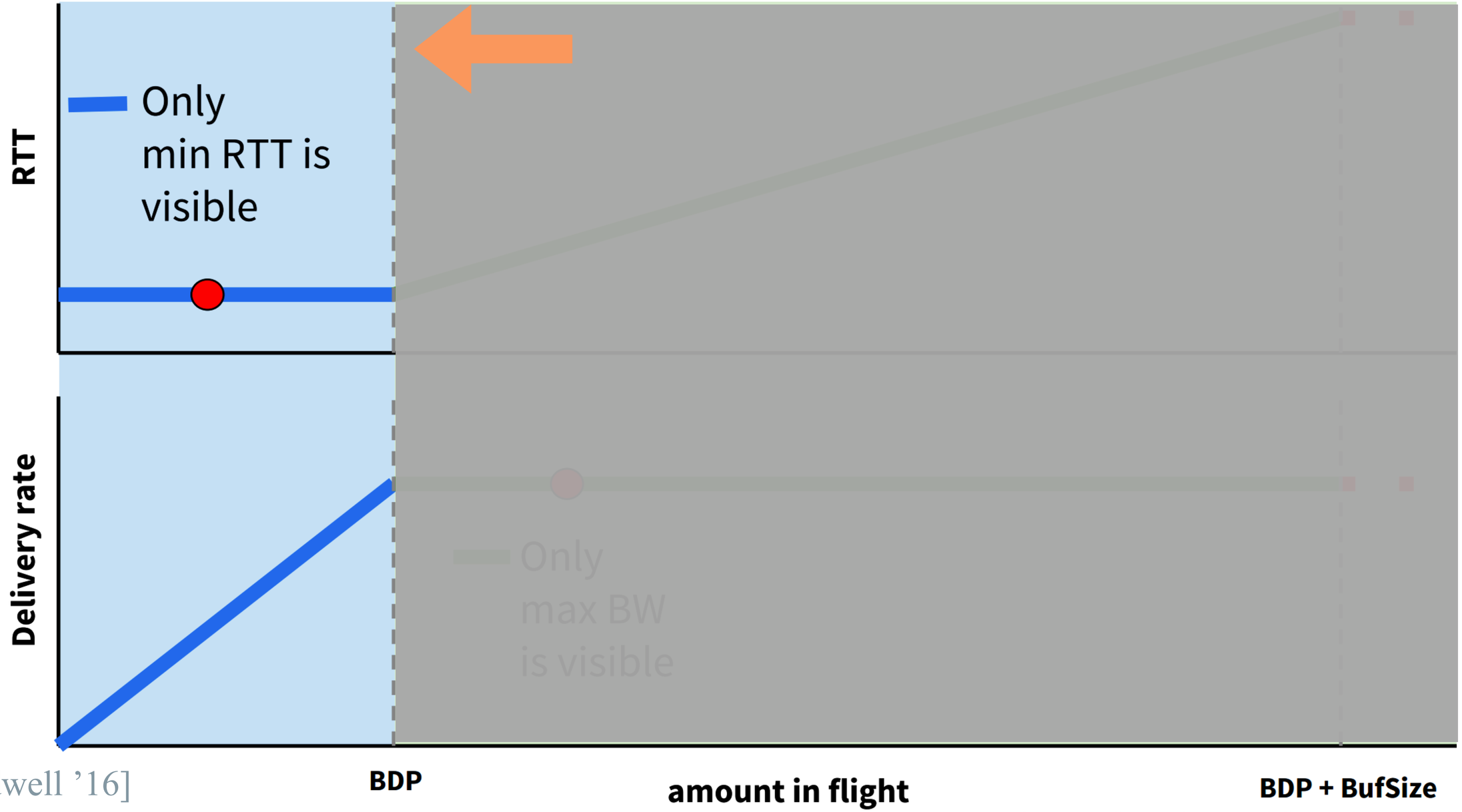
[Cardwell ’16]

Sequentially Probe Max BW and Min RTT



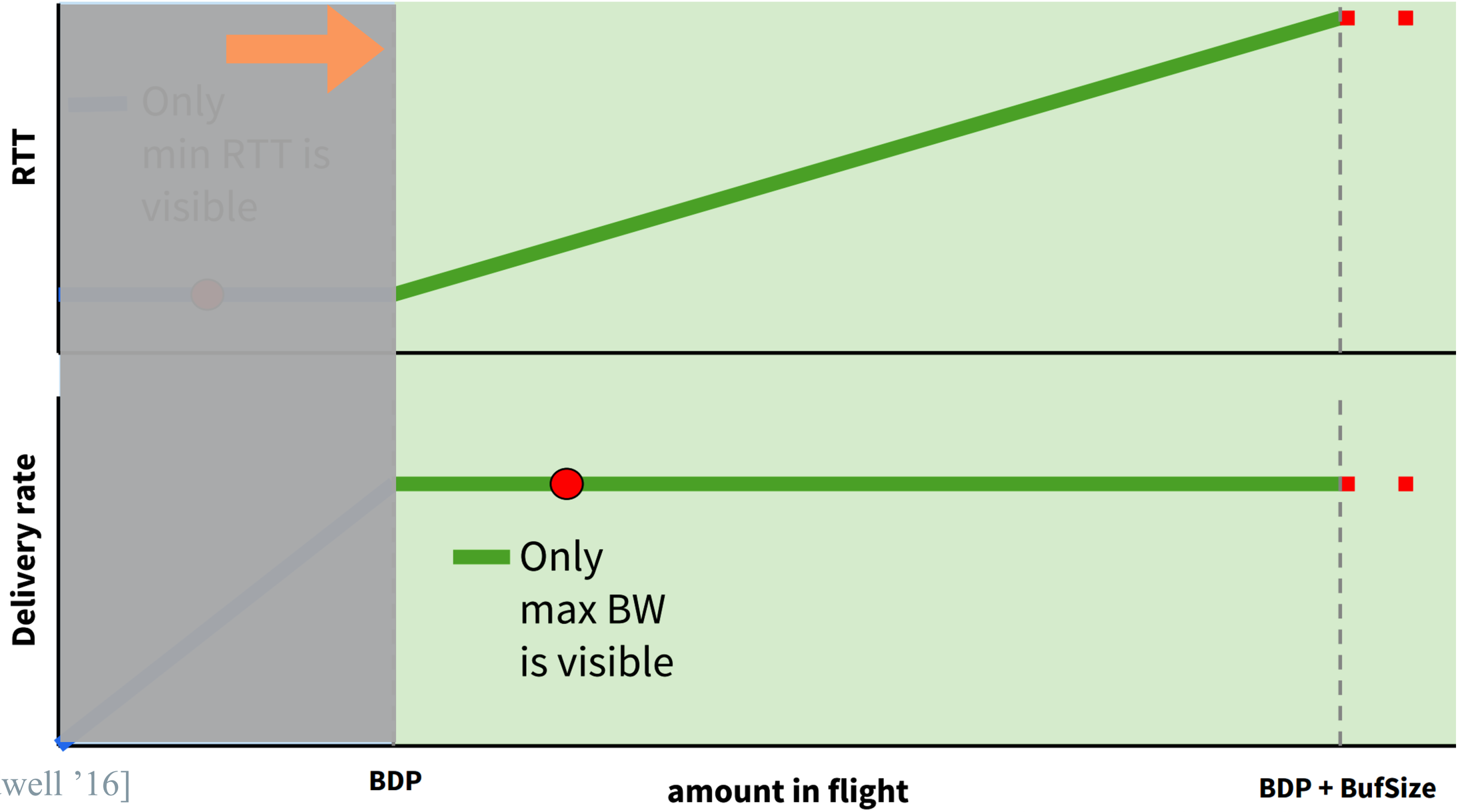
[Cardwell '16]

Sequentially Probe Max BW and Min RTT



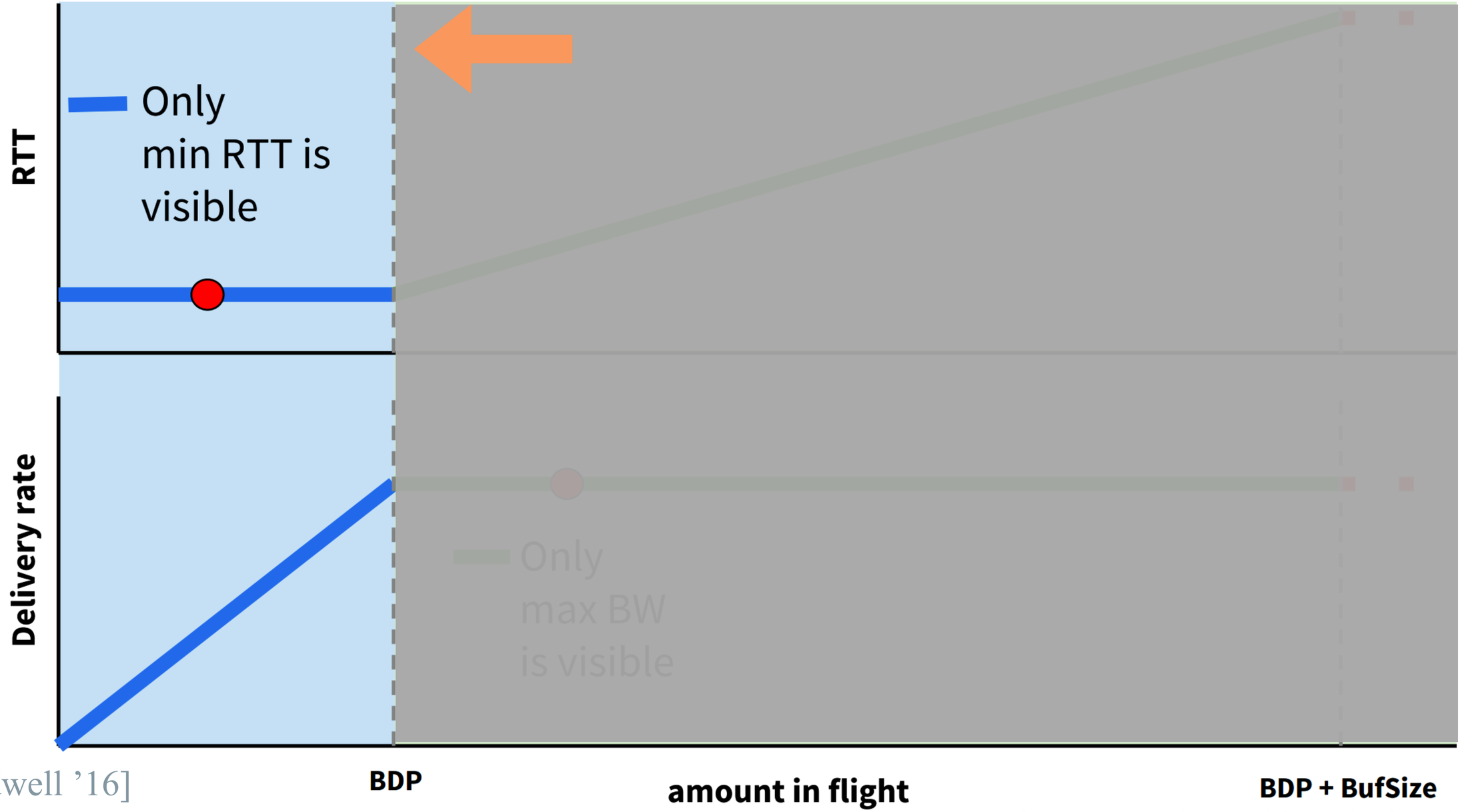
[Cardwell '16]

Sequentially Probe Max BW and Min RTT



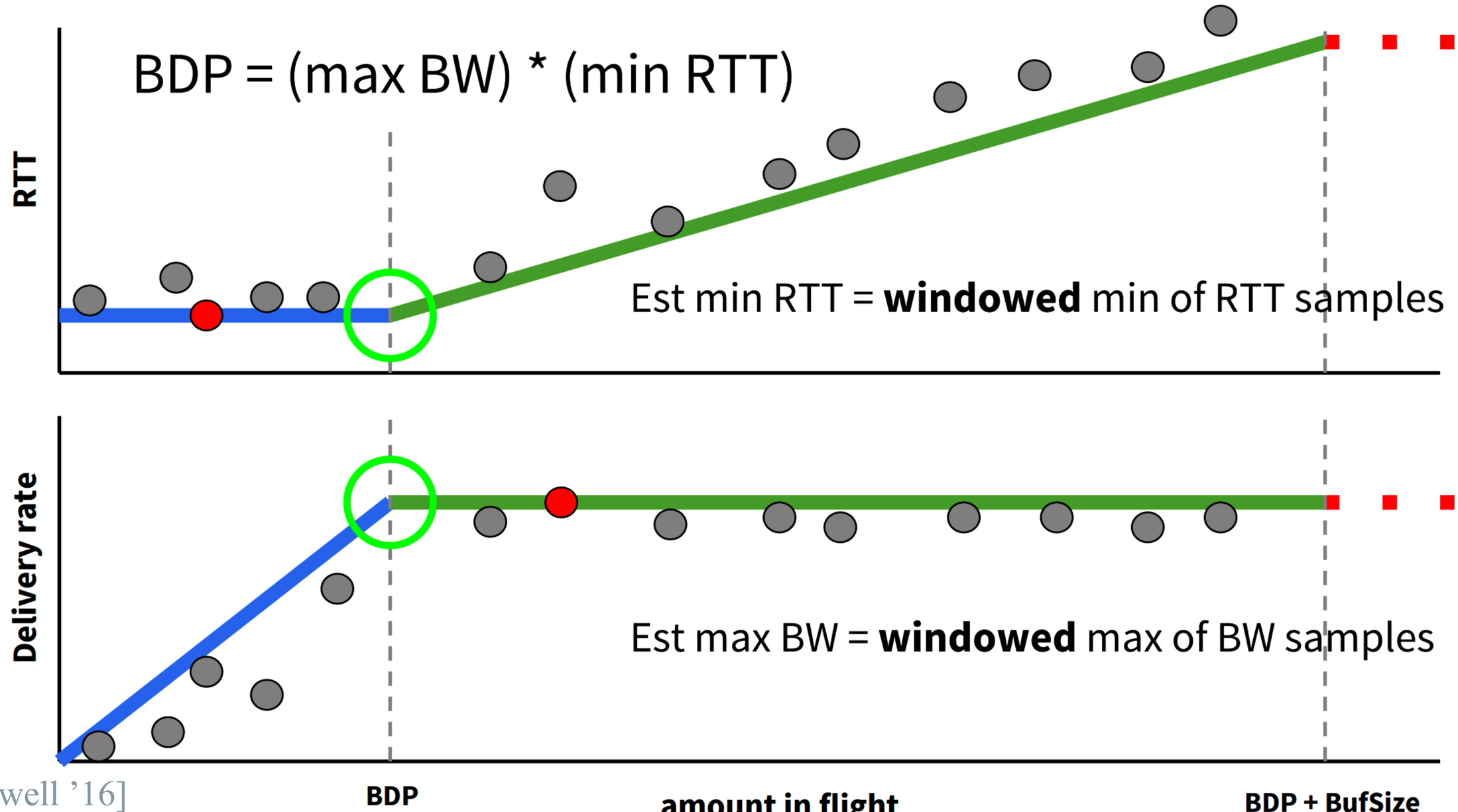
[Cardwell '16]

Sequentially Probe Max BW and Min RTT



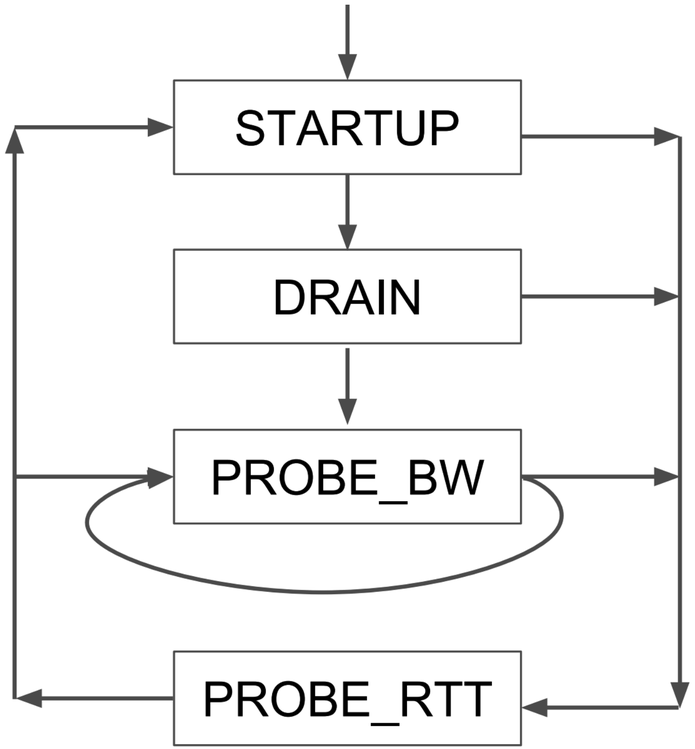
[Cardwell '16]

BDP Estimation



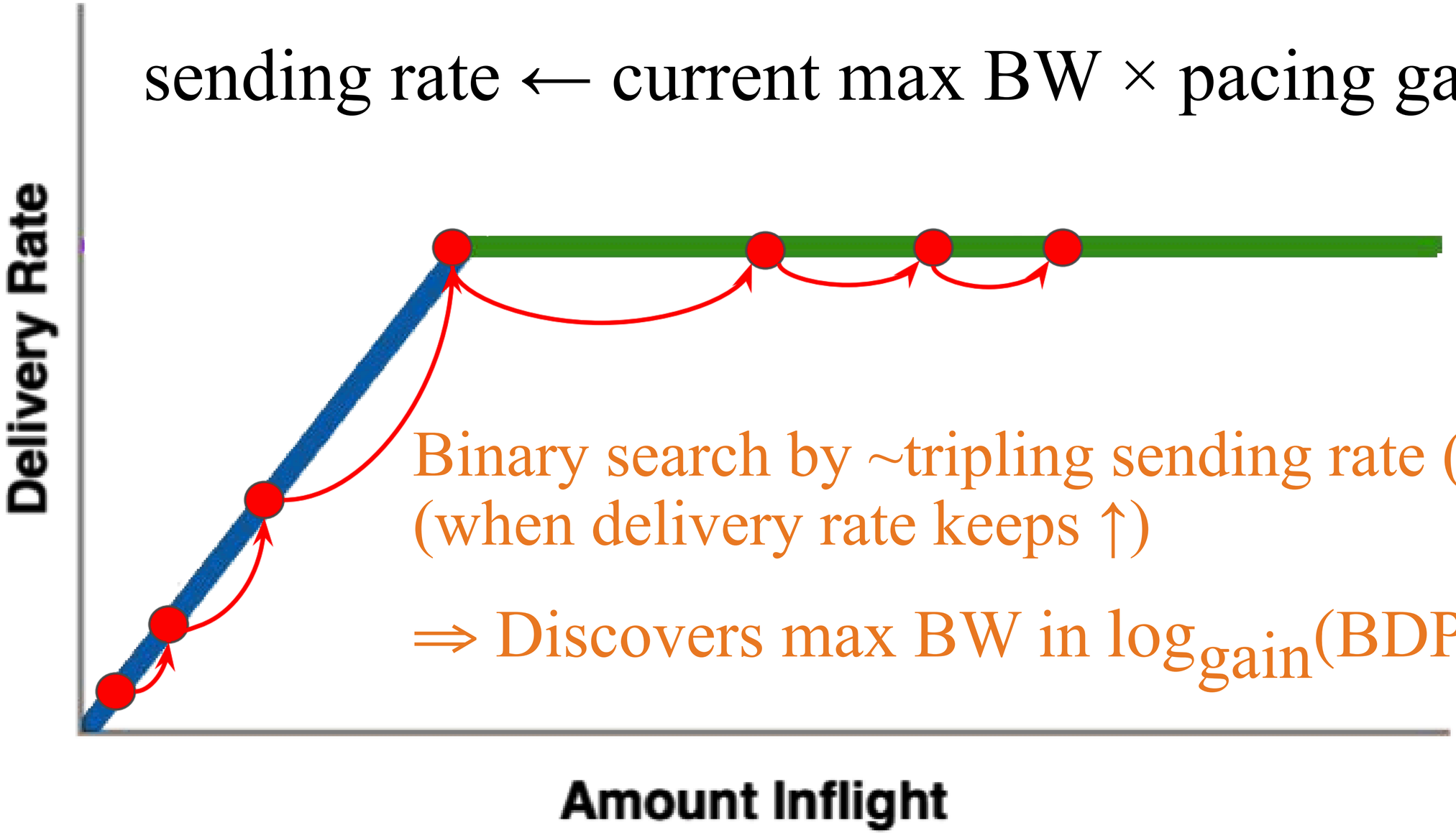
[Cardwell '16]

BBR's State Machine



Startup: Exponential BW search (≈ slow-start)

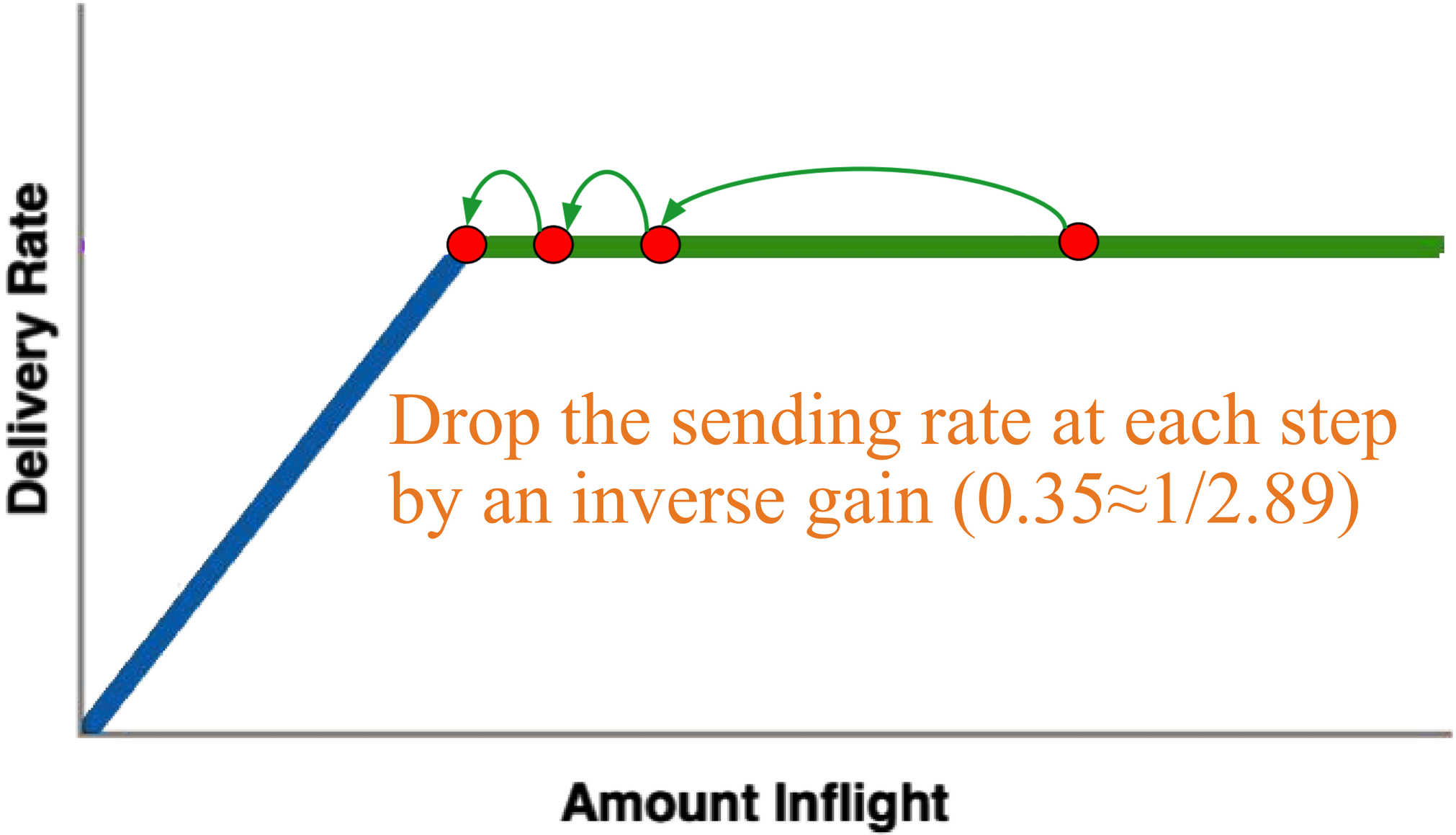
sending rate ← current max BW × pacing gain



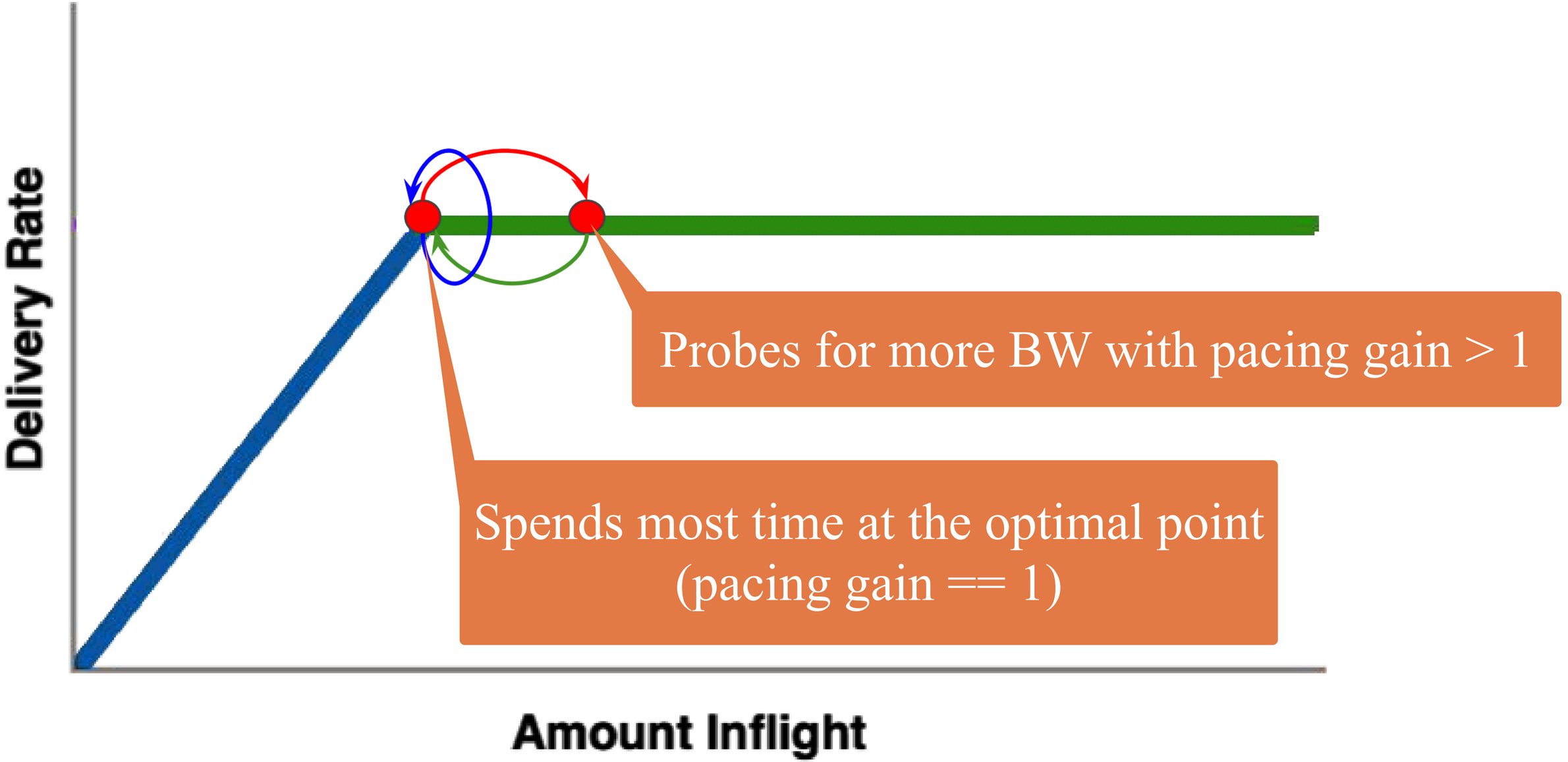
Binary search by ~tripling sending rate (2.89 gain)
(when delivery rate keeps ↑)

⇒ Discovers max BW in $\log_{gain}(BDP)$ RTTs

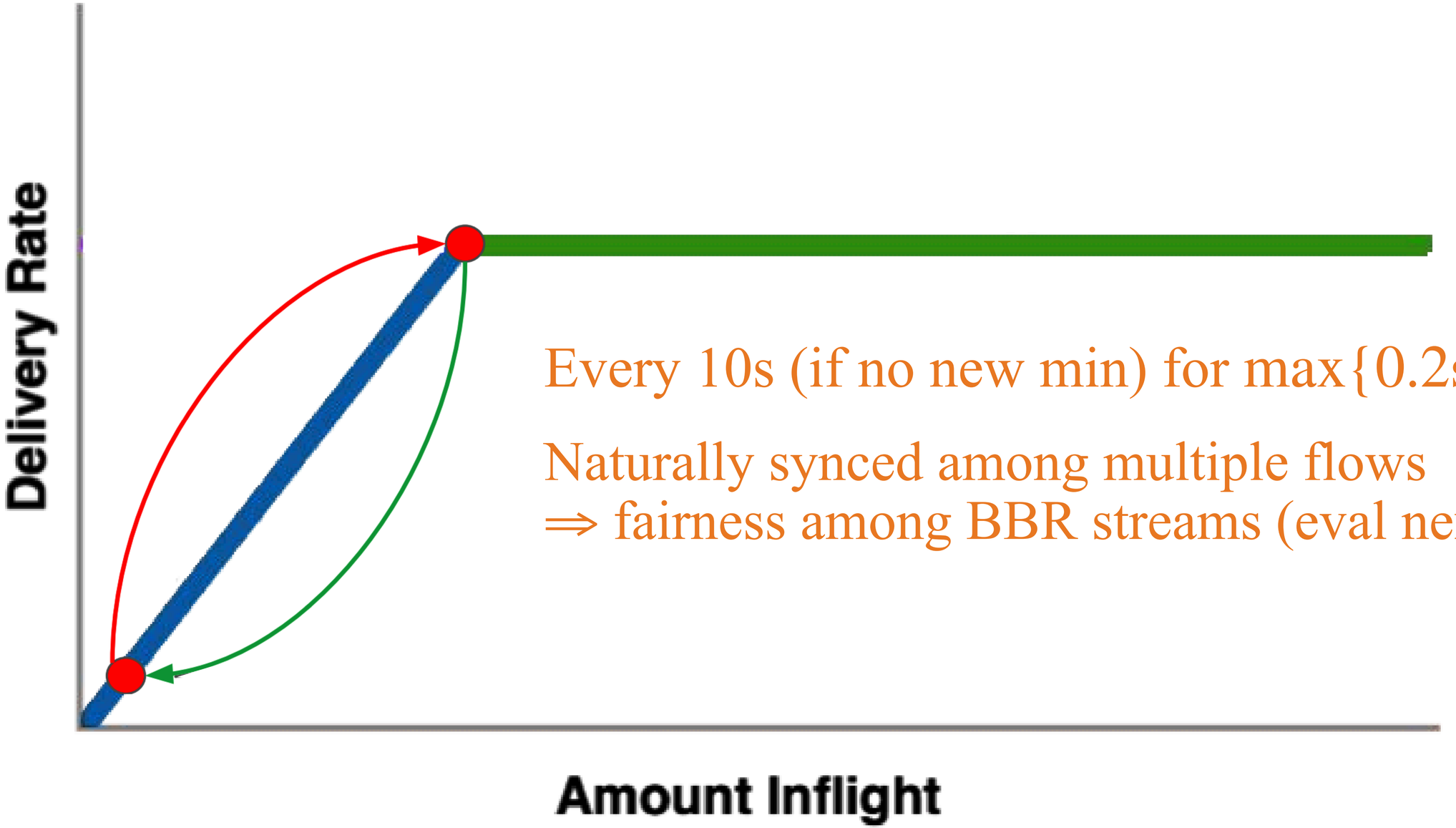
Drain: Depleting queue (bounded by 2×BDP)



Probe BW: Cycling Pacing Gain



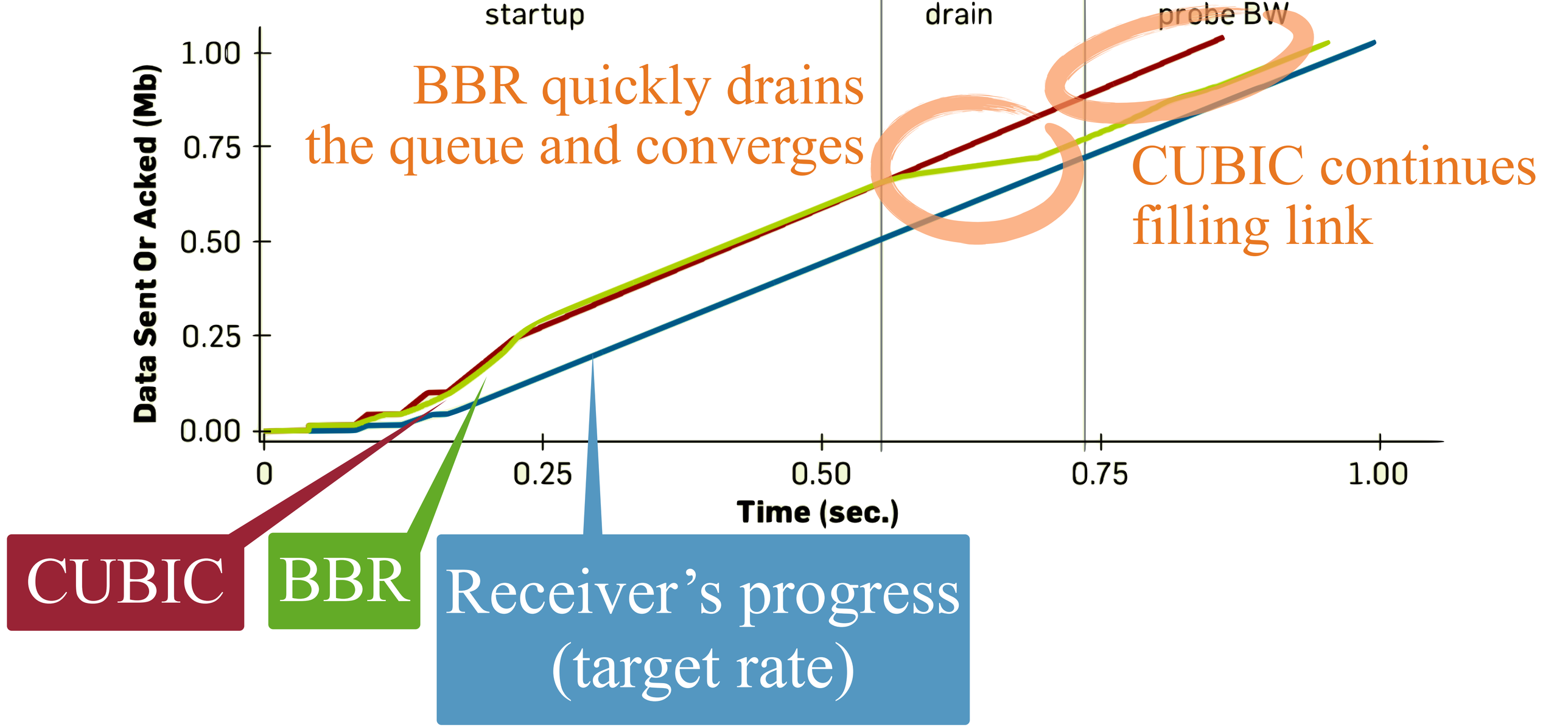
Probe RTT: Periodic back off

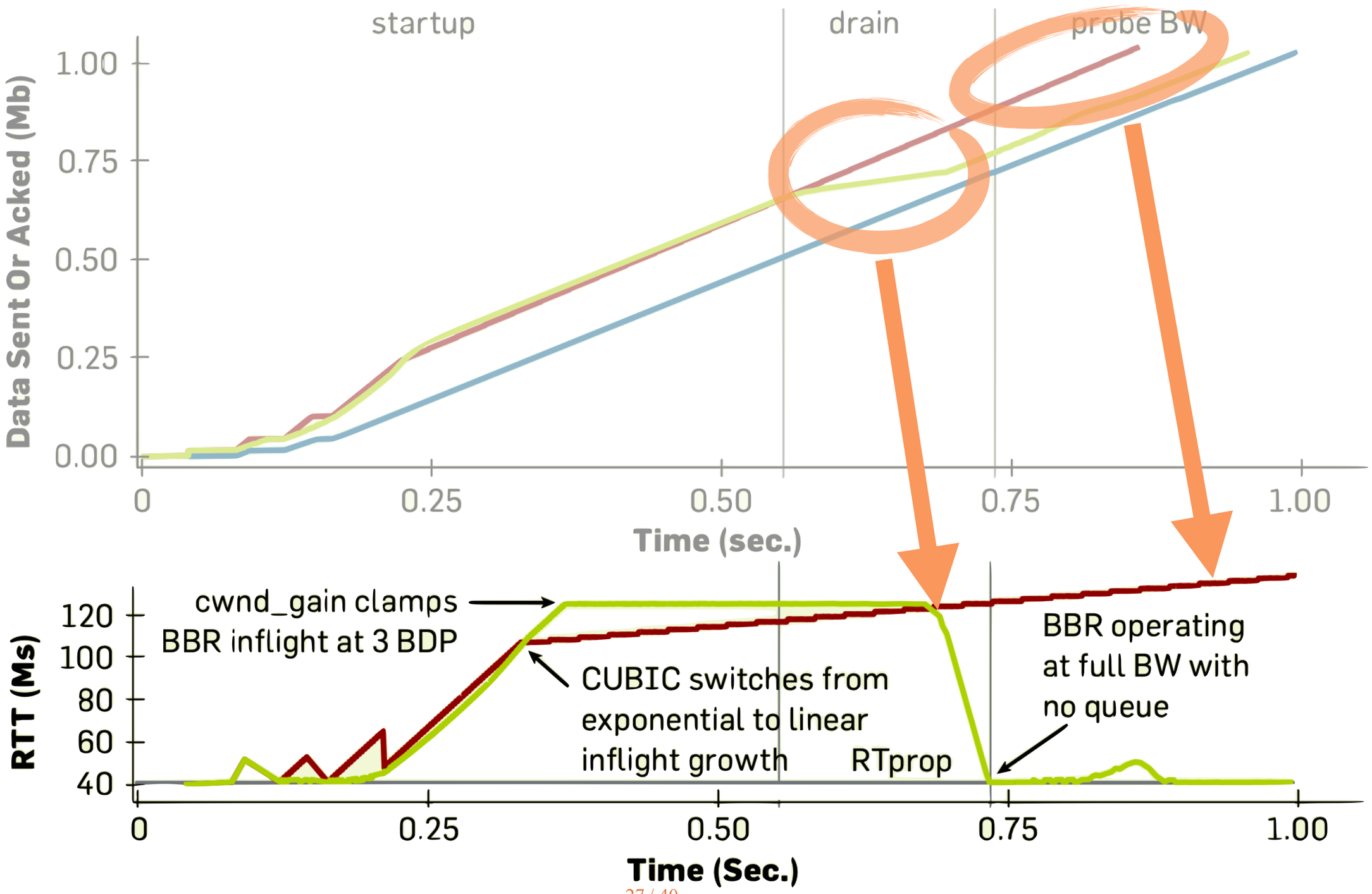


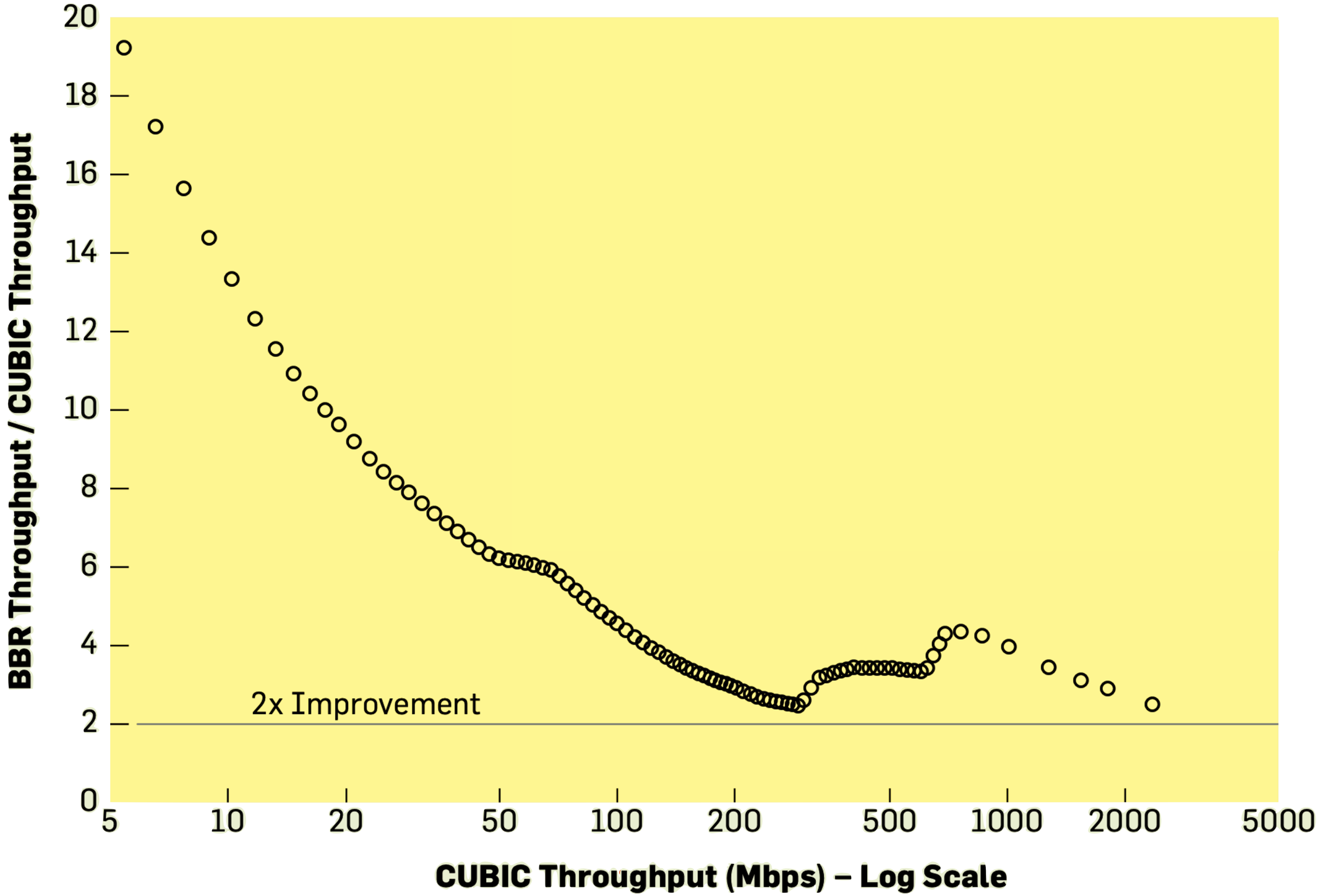
Every 10s (if no new min) for max{0.2s, RTT}

Naturally synced among multiple flows
⇒ fairness among BBR streams (eval next)

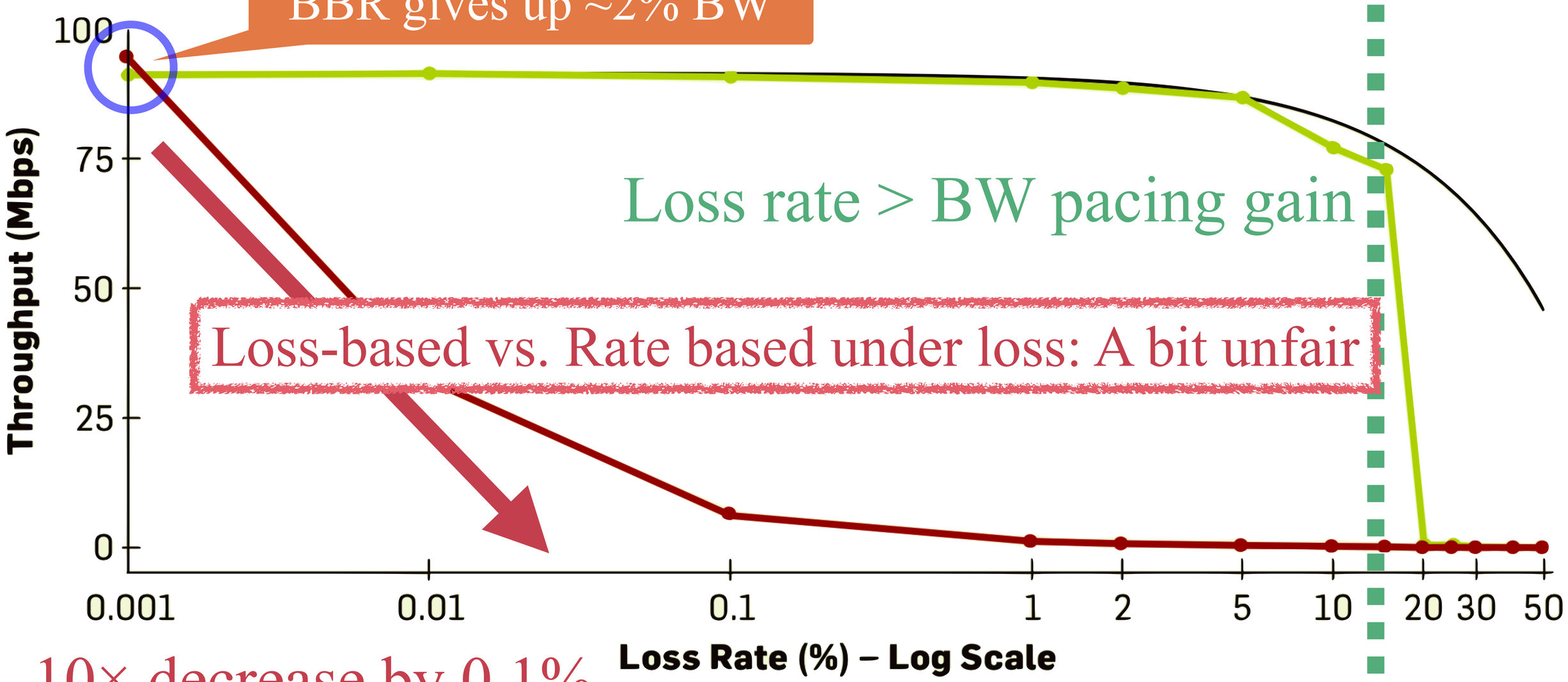
Evaluation


startup
drain
probe BW
BBR quickly drains
the queue and converges
CUBIC continues
filling link
Data Sent Or Acked (Mb)
1.00
0.75
0.50
0.25
0.00
0
0.25
0.50
0.75
1.00
Time (sec.)
CUBIC
BBR
Receiver's progress
(target rate)


startup
drain
probe BW
Data Sent Or Acked (Mb)
1.00
0.75
0.50
0.25
0.00
0
0.25
0.50
0.75
1.00
Time (sec.)
RTT (Ms)
120
100
80
60
40
cwnd_gain clamps
BBR inflight at 3 BDP
CUBIC switches from
exponential to linear
inflight growth
RTprop
BBR operating
at full BW with
no queue
0
0.25
0.50
0.75
1.00
Time (Sec.)


BBR Throughput / CUBIC Throughput
20
18
16
14
12
10
8
6
4
2
0
2x Improvement
5
10
20
50
100
200
500
1000
2000
5000
CUBIC Throughput (Mbps) – Log Scale

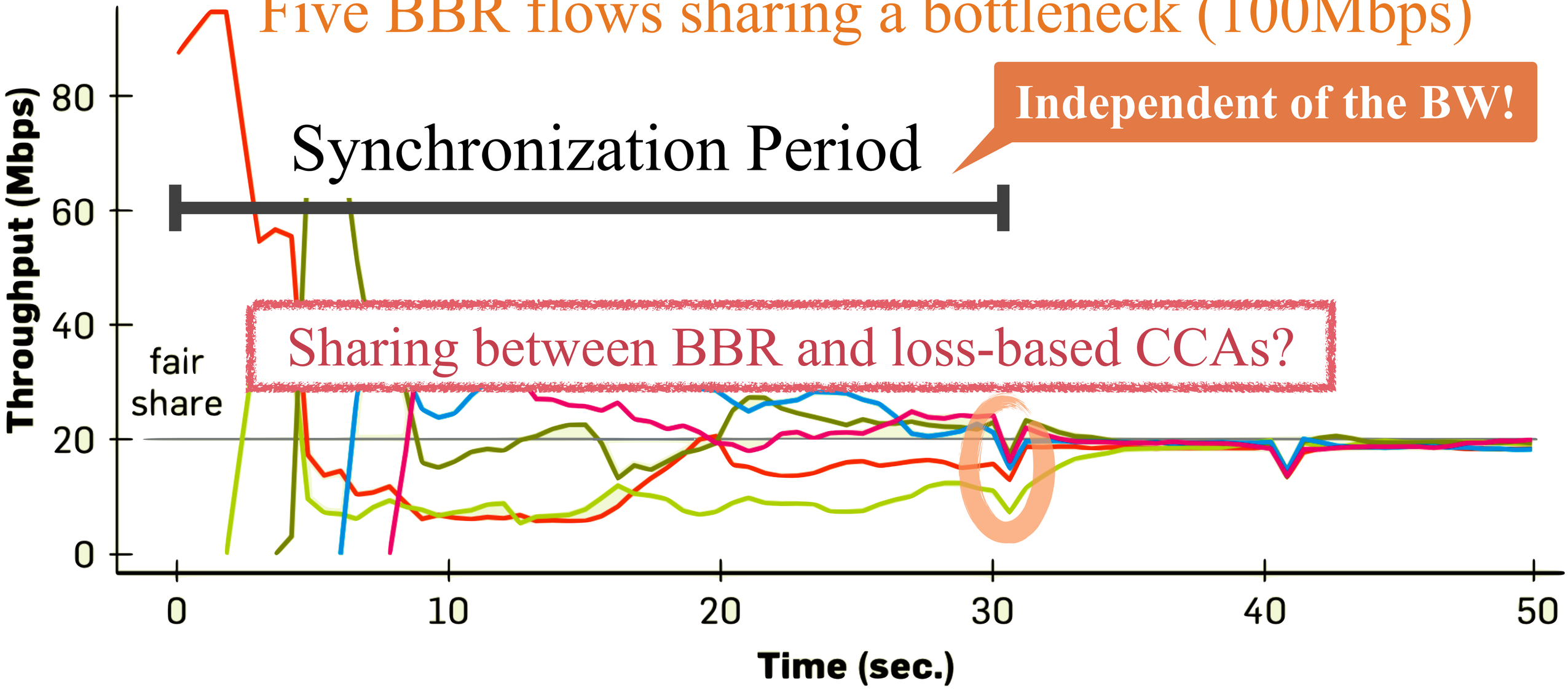
Goodput under Loss BBR vs CUBIC



10× decrease by 0.1%

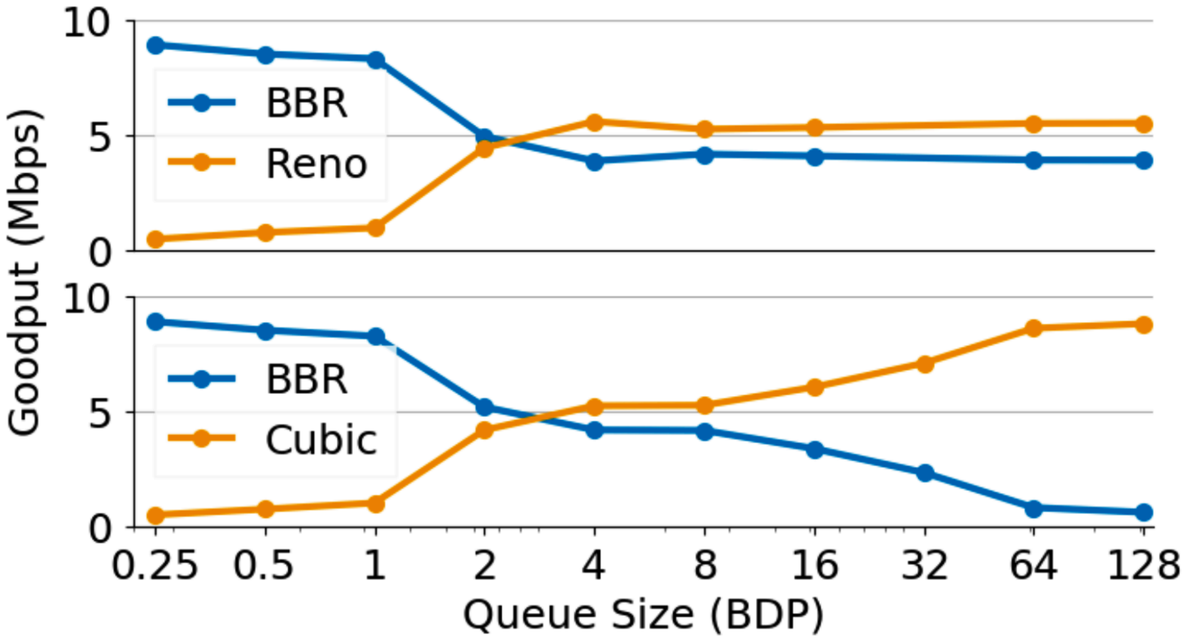
Speaking of Fairness …

Five BBR flows sharing a bottleneck (100Mbps)



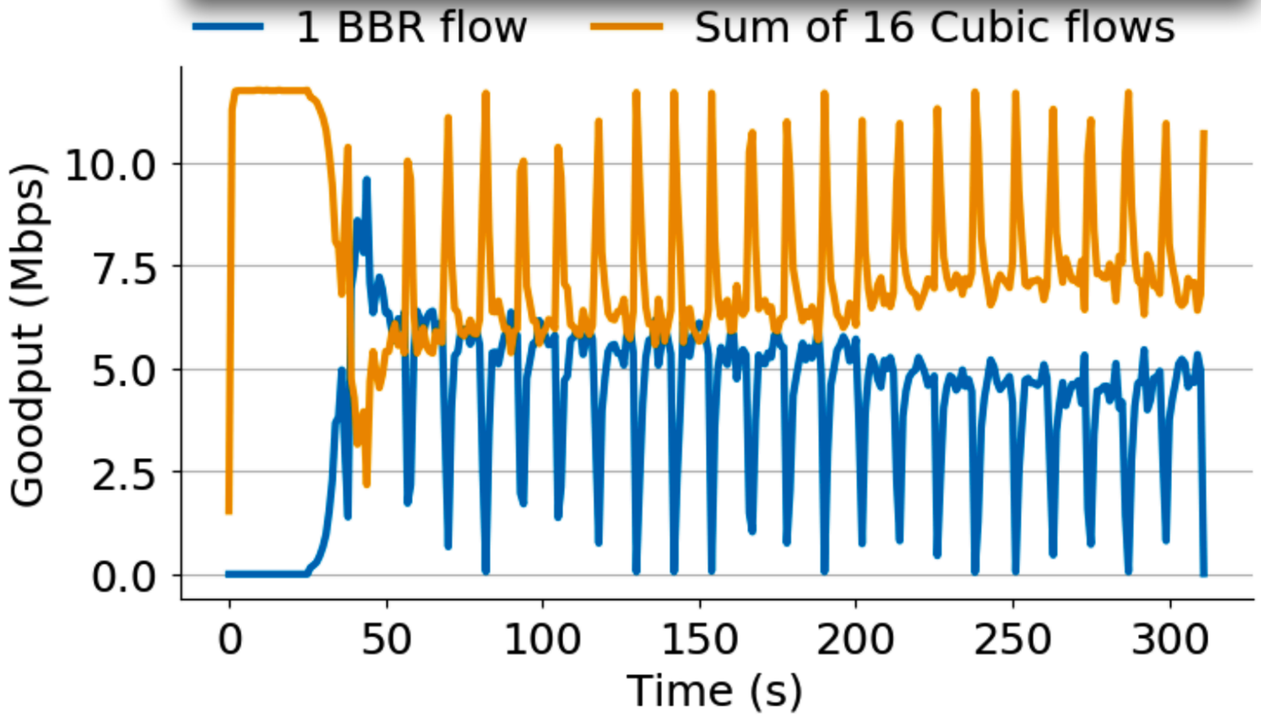
BBR's (Un)Fairness [Ware et al., IMC '19]

BBR is generous



(b) Average goodput for two competing flows over 4 min in a 40ms×10Mbps network with varying queue sizes.

BBR is aggressive

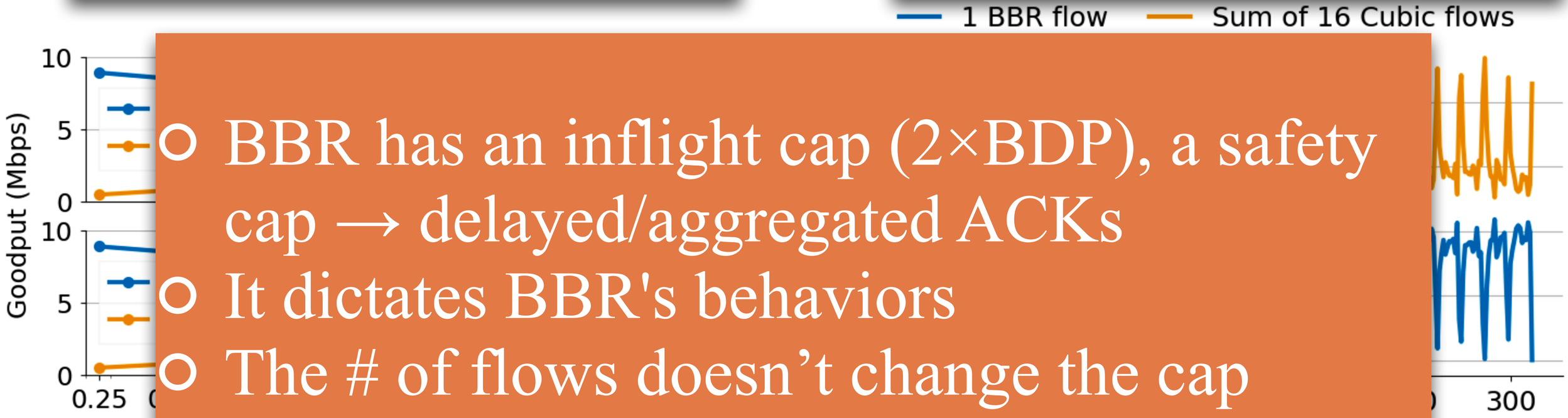


(c) BBR's goodput over time competing with 16 Cubic flows in a 40ms×10Mbps network with a 32 BDP queue.

BBR's (Un)Fairness [Ware et al., IMC '19]

BBR is generous

BBR is aggressive



- BBR has an inflight cap (2×BDP), a safety cap → delayed/aggregated ACKs
- It dictates BBR's behaviors
- The # of flows doesn't change the cap

(More in [Ware et al.])

(b) Avera… flows over 4 min in a 40ms×10Mbps net-work with varying queue sizes.

…mpeting with 16 Cubic flows in a 40ms×10Mbps net-work with a 32 BDP queue.

BBR's Deployment at Google (~2017)

- YouTube: deployed for a small percentage of users
- Internal: test programs for Google data-centers
  - deployed as default TCP congestion control for internal Google traffic

Compared to CUBIC:

- 2% lower latency on google search
- 13% larger Mean Time Between Rebuffers (MTBR) on YouTube
- 32% lower RTT on YouTube
- Loss rate increased from 1 to 2%

Source: https://groups.google.com/d/forum/bbr-dev

Related and Future Work

- New versions of BBR (v2, v3 draft) → backup slides
  - v2 explicitly bounds loss rate
- Other rate based CCAs, e.g., Copa [NSDI ’18] (next session)
- BBR’s (un)fairness [NSDI ‘18; NotNets ‘19; IMC ‘19]
- Optimizing BBR’s retransmission [Bi et al, ATC ’23]

My Review

Strengths

S1. A performant, scalable solution to a fundamental challenge

S2. Solid evaluation, long-term development, and hight impact

Weaknesses

W1. Didn’t push BBR over the edge in the paper (e.g., scaling/unfairness)

W2. Handling token bucket policers [BBRv2], TSO, and middle boxes

W3. (minor) None of the figures have legends :/

Future directions

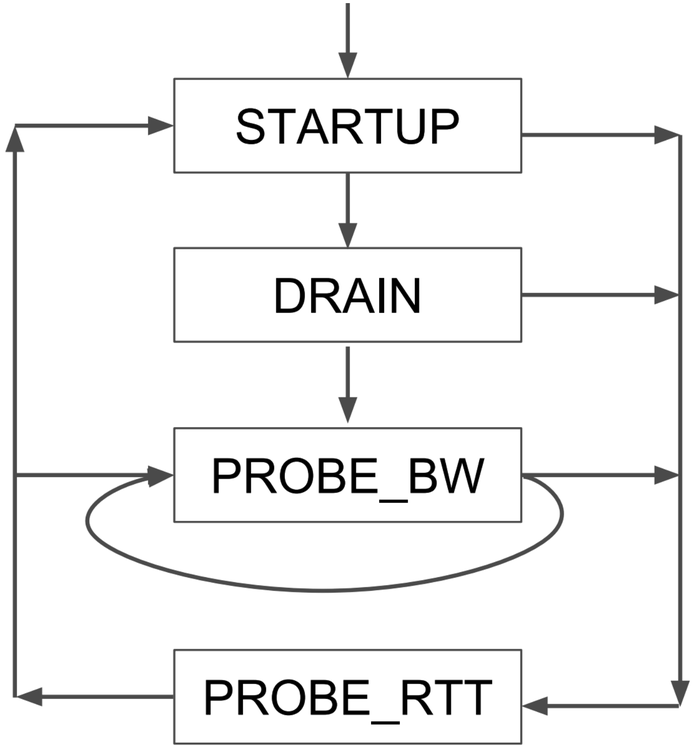
D1. Security vulnerabilities?

D2. Wireless environments (e.g., fairness and work with MPTCP)?

Class Discussion

Backup slides …

BBR's FSM



- STARTUP: exponential growth to quickly fill pipe
  - stop growth when BW estimate plateaus, not on loss or delay
  - pacing_gain = 2.89
- DRAIN: drain the queue created in STARTUP
  - pacing_gain = 0.35 = 1/2.89
- PROBE_BW: cycle pacing gain to explore and fairly share bandwidth
  - [1.25, 0.75, 1, 1, 1, 1, 1, 1] (1 phase per min RTT)
  - pacing_gain = 1.25 ≥ probe for more BW
  - pacing_gain = 1.0 ≥ cruise with full utilization and low, bounded queue
- PROBE_RTT: if needed, occasionally send slower to probe min RTT
  - pacing_gain = 0.75 => drain queue and yield BW to other flows

BBR v2

	CUBIC	BBR v1	BBR v2
Model parameters to the state machine	N/A	Throughput, RTT	Throughput, RTT, max aggregation, max inflight
Loss	Reduce cwnd by 30% on window with any loss	N/A	Explicit loss rate target
ECN	RFC3168 (Classic ECN)	N/A	DCTCP-inspired ECN
Startup	Slow-start until RTT rises (Hystart) or any loss	Slow-start until tput plateaus	Slow-start until tput plateaus or ECN/loss rate > target

BBR's FSM (2)

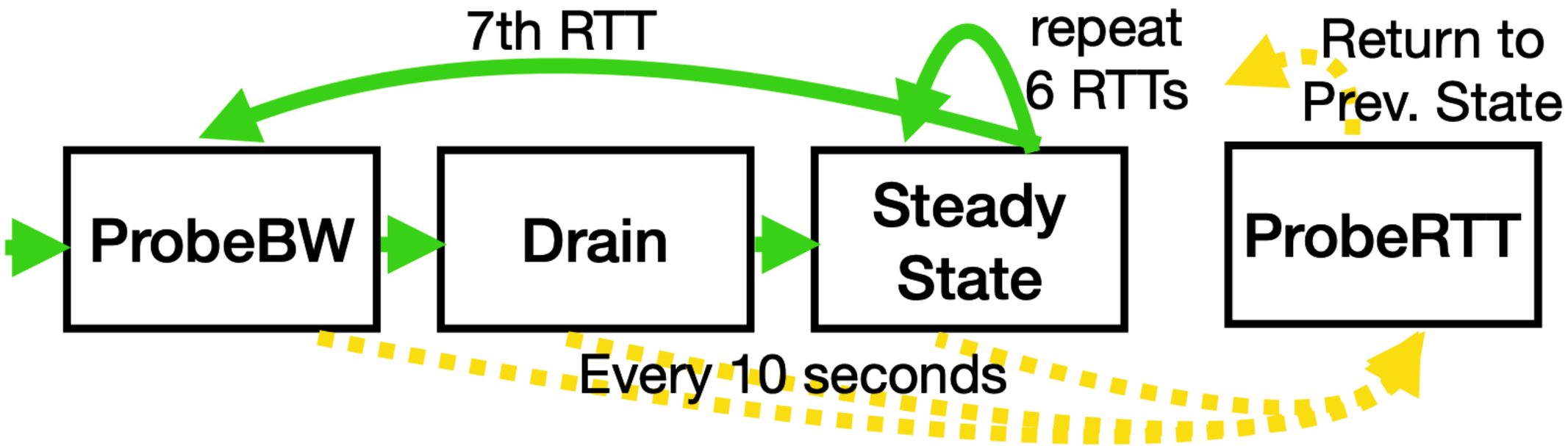


Figure 4: BBR's steady-state operation.

[Ware et al., IMC '19]