Transport Layer: TCP CUBIC, BBR

CS 204: Advanced Computer Networks

Oct 20, 2023

Adapted from Jiasi's CS 204 slides for Spring 23

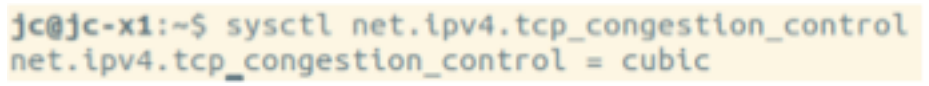
Outline

- TCP Review
- New TCP flavors
    - Multipath-TCP
        - Basics
        - Sequence numbers
        - Congestion control
    - CUBIC
    - BBR
- Mathematical modeling of TCP

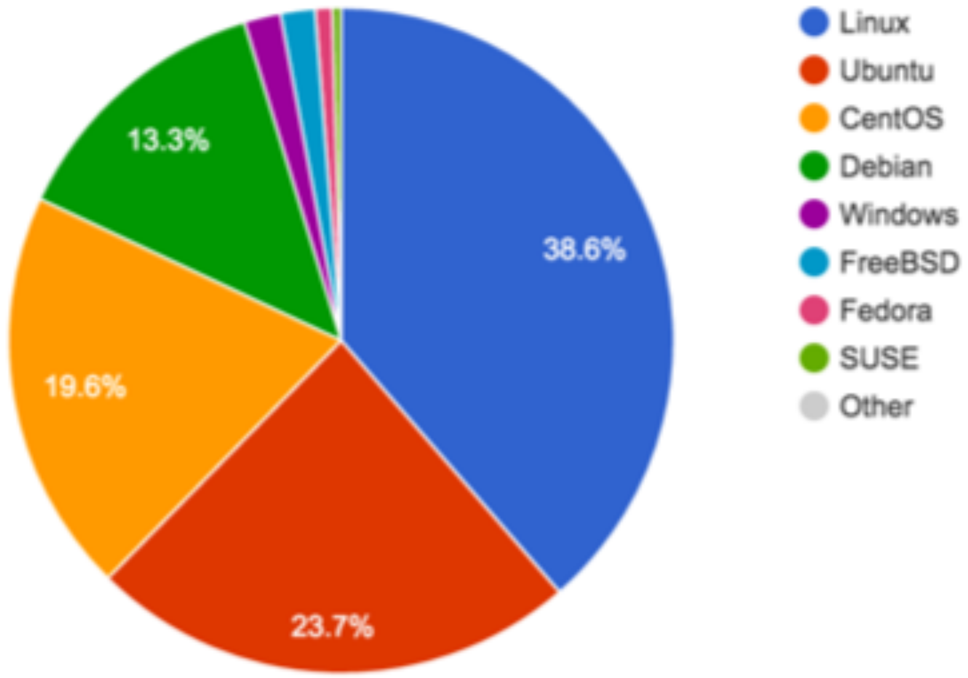
Q: How should flows compete for bandwidth when there is congestion in the network?

Who Uses What?

- Mac OS X: CUBIC (default), New Reno
- Windows: Reno (default), CTCP
- Linux: CUBIC (default), Reno

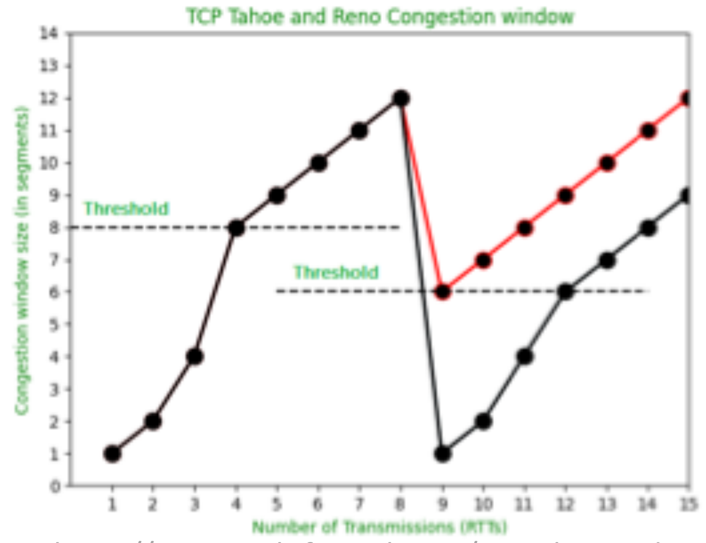


```
jc@jc-x1:~$ sysctl net.ipv4.tcp_congestion_control
net.ipv4.tcp_congestion_control = cubic
```



Web server OS share

TCP Tahoe vs. TCP Reno



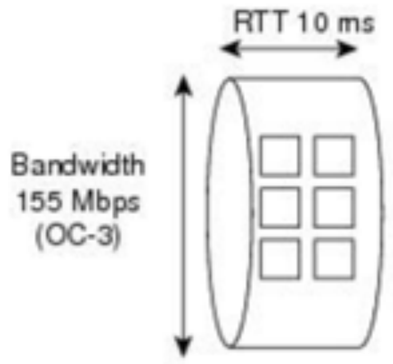
Source: https://www.geeksforgeeks.org/tcp-tahoe-and-tcp-reno/

- Two similar versions of TCP
  - TCP Tahoe (SIGCOMM'88 paper)
  - TCP Reno (1990)
- TCP Tahoe
  - Always repeat slow start after a loss
  - Assign slow-start threshold to half of congestion window
- TCP Reno
  - Repeat slow start after timeout-based loss
  - Divide congestion window in half after triple duplicate ACK

- Slow-start: cwnd ← cwnd + 1
- Congestion avoidance: cwnd ← cwnd + 1/MSS
- Loss: retransmit, ssthresh ← cwnd/2, go to slow-start
  cwnd ← 1

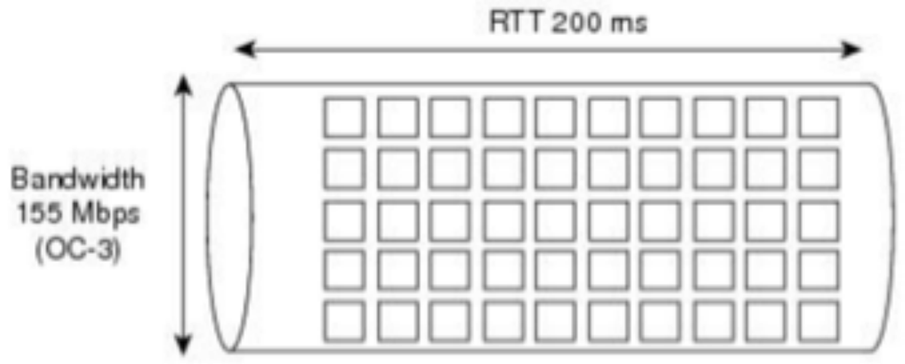
- Slow-start: cnwd ← cwnd + 1
- Congestion avoidance: cwnd ← cwnd + 1/MSS
- Loss: retransmit, ssthresh ← cwnd/2
  cwnd ← cwnd/2, go to congestion avoidance

Bandwidth-Delay Product



155 Mbps * 10 ms = 1550 kb = 104 kB

Measure amount of in-flight data
BDP = bandwidth * RTT

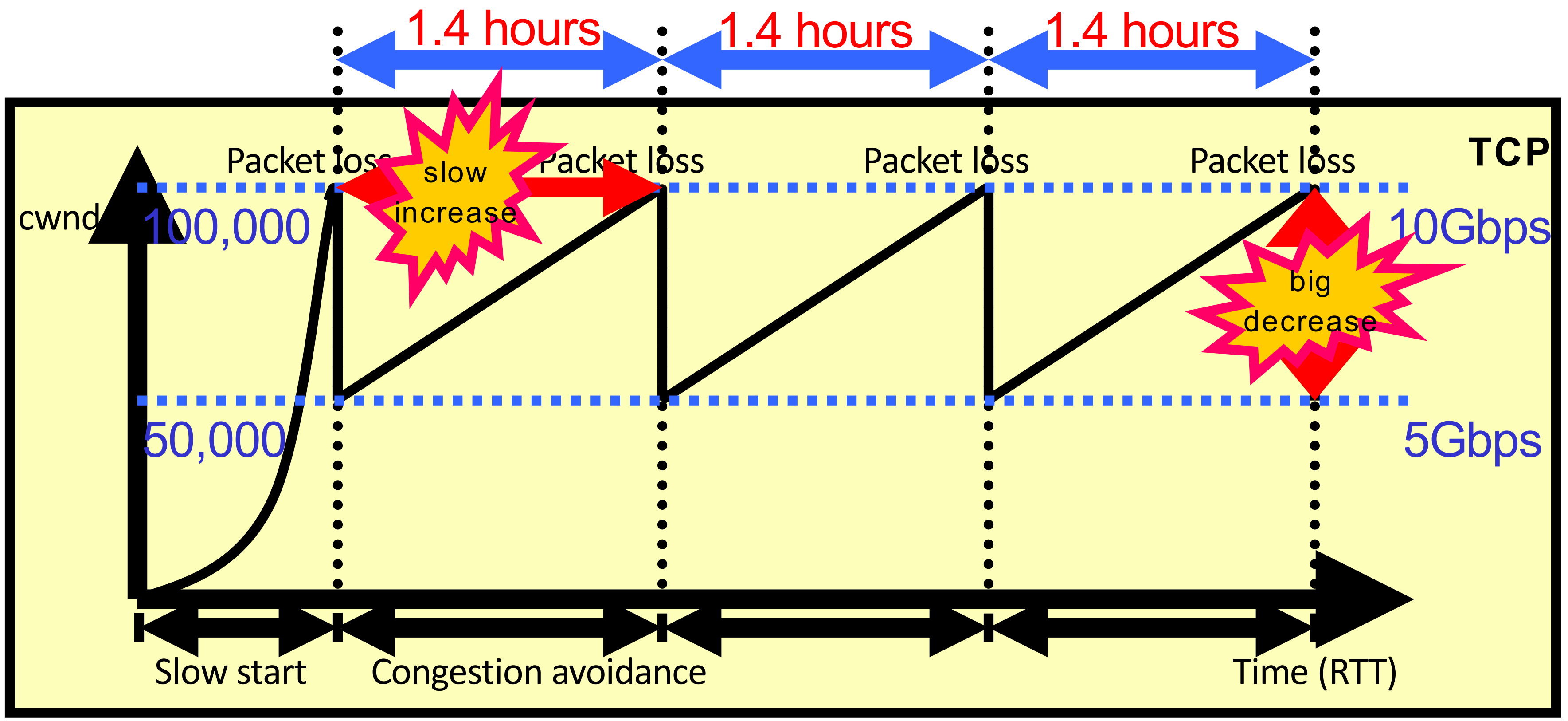


155 Mbps * 200 ms = 31 Mb = 3875 kB

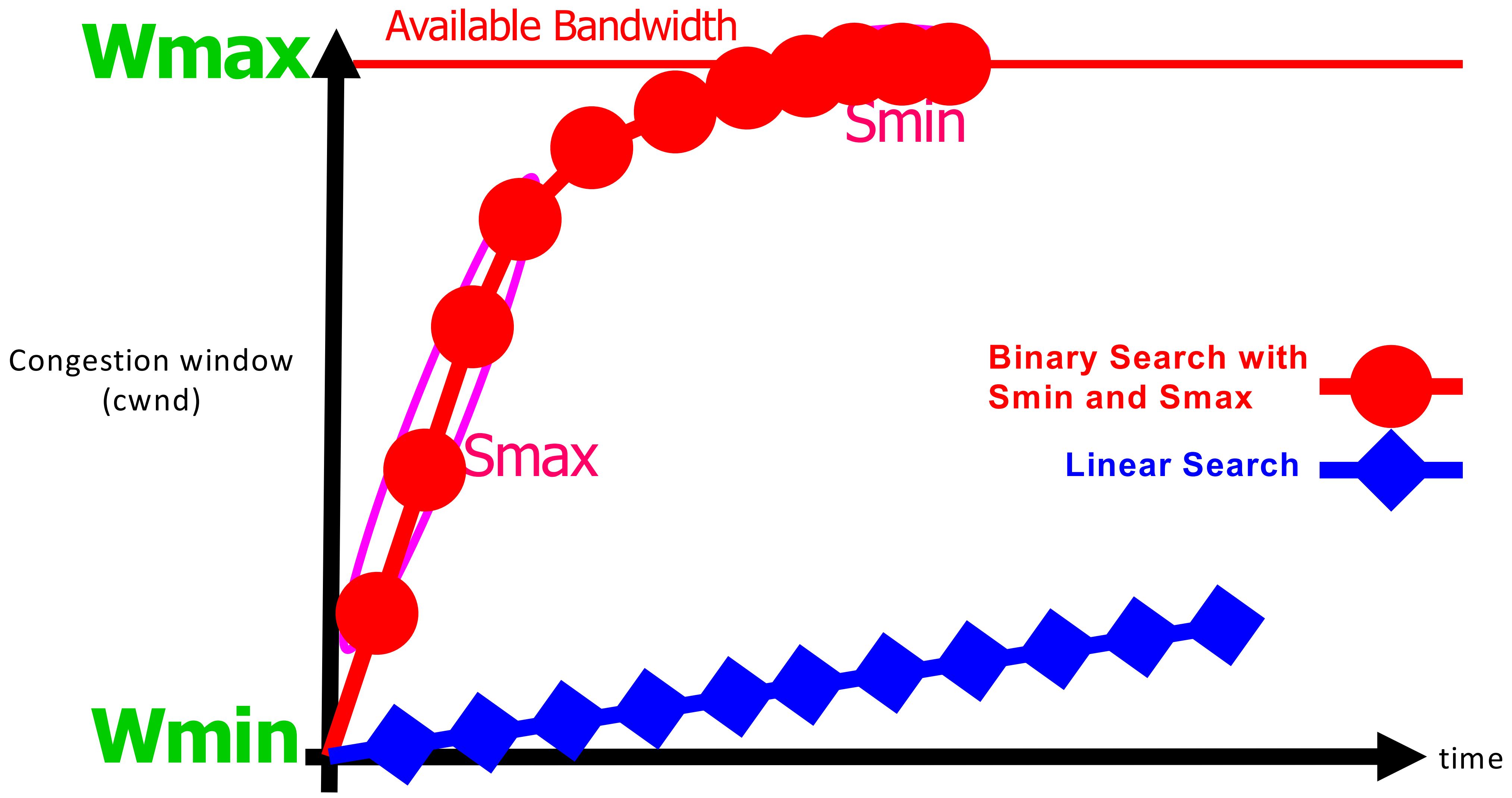
Source: WAN and Application Optimization Technologies, *Cisco*
http://www.cisco.com/c/en/us/td/docs/nsite/enterprise/wan/wan_optimization/wan_opt_sg/chap06.html

Performance Problem – Closer look on what's happening inside

- Slow congestion window growth during congestion avoidance
- A TCP connection with 1250-byte packet size and 100ms RTT running over 10Gbps link.



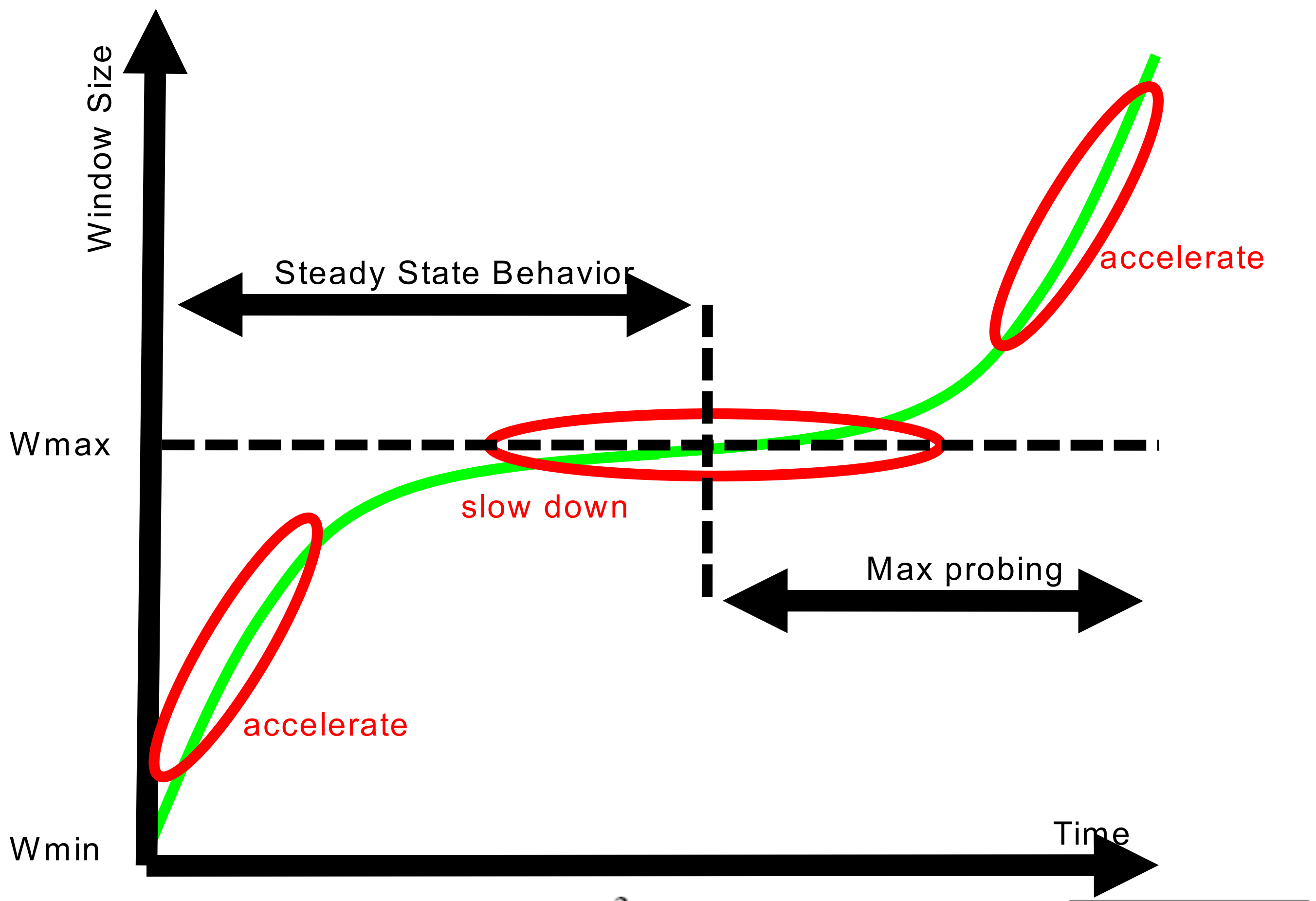
BIC-TCP
Binary Search with Smax and Smin



CUBIC – A new TCP variant

- Why a new protocol?
  - While the window growth of new TCP protocols is scalable, their fairness issue has remained as a major challenge .
  - BIC-TCP shows good utilization and stability, but it lacks TCP friendliness and RTT fairness.
- CUBIC is an enhanced version of BIC
  - Simplifies the BIC window control using a cubic function.
  - Improves its TCP friendliness & RTT fairness.
  - The window growth function of CUBIC is based on real-time (the elapsed time since the last loss event), so that it is independent of RTT.

CUBIC Main Idea



$$W_{cubic} = C(t-K)^3 + W_{\max} \qquad K = \sqrt[3]{W_{\max}\beta/C}$$

where ***C*** is a scaling factor, ***t*** is the elapsed time from the last window reduction, and ***β*** is a constant multiplication decrease factor.

CUBIC TCP Mode

- In short RTT networks, the window growth of CUBIC is slower than TCP since CUBIC is independent of RTT
- Want to ensure that CUBIC is as aggressive as regular TCP

Average sending rate of AIMD = $\frac{1}{RTT}\sqrt{\frac{\alpha}{2}\frac{1+\beta}{1-\beta}\frac{1}{p}}$

α: additive factor
β: multiplicative decrease factor
p: packet loss probability

$\frac{1}{RTT}\sqrt{\frac{3}{2}\frac{1}{p}}$ (TCP). Thus, $\alpha = 3\times\frac{1-\beta}{1+\beta}$

$$W_{tcp} = W_{\max}\beta + 3\frac{1-\beta}{1+\beta}\frac{t}{RTT}$$

The size of TCP window after time t from window reduction.

if W_{tcp} > W_{cubic} : window size = W_{tcp}
Otherwise : window size = W_{cubic}

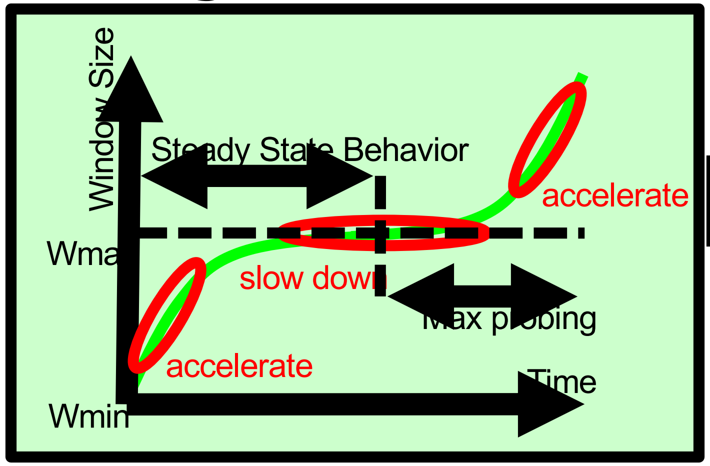
Outline

- TCP Review
- New TCP flavors
  - Multipath-TCP
  - CUBIC
  - BBR
- Mathematical modeling of TCP

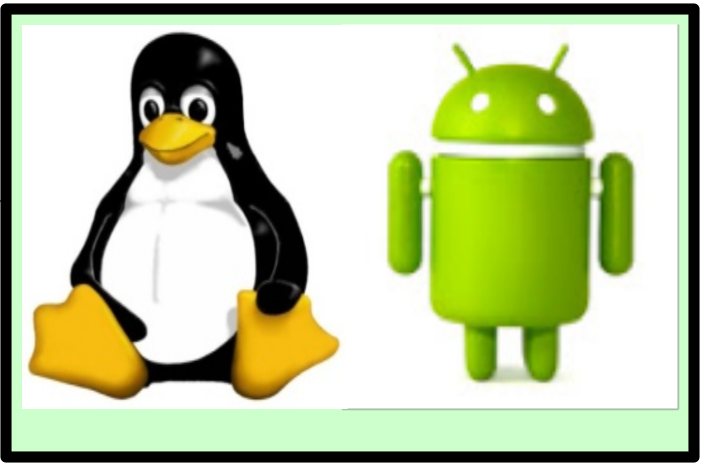
Q: How should flows compete for bandwidth when there is congestion in the network?

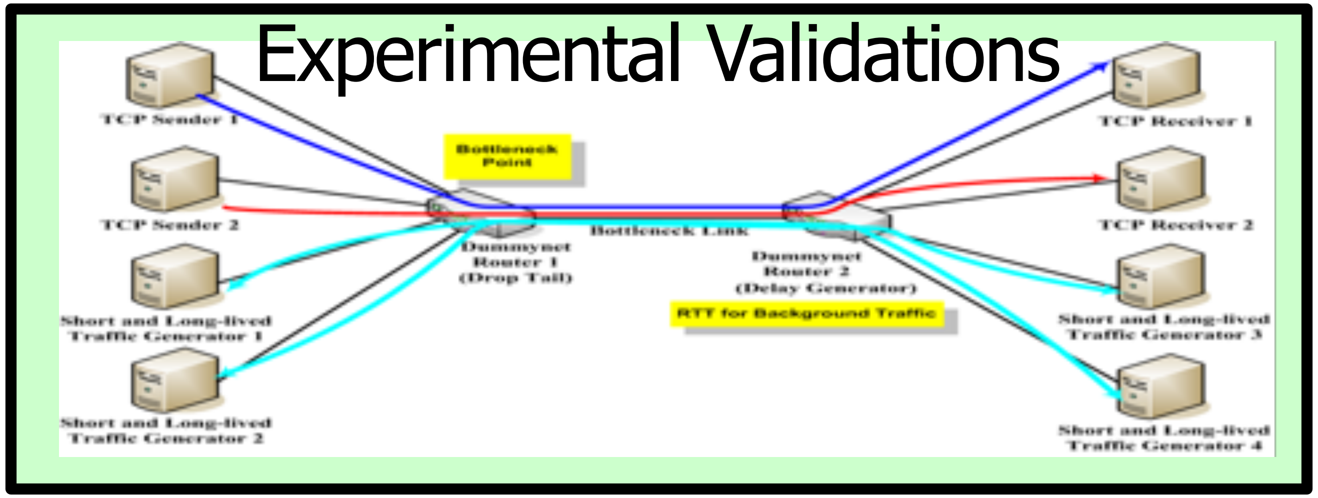
Contributions

Algorithm



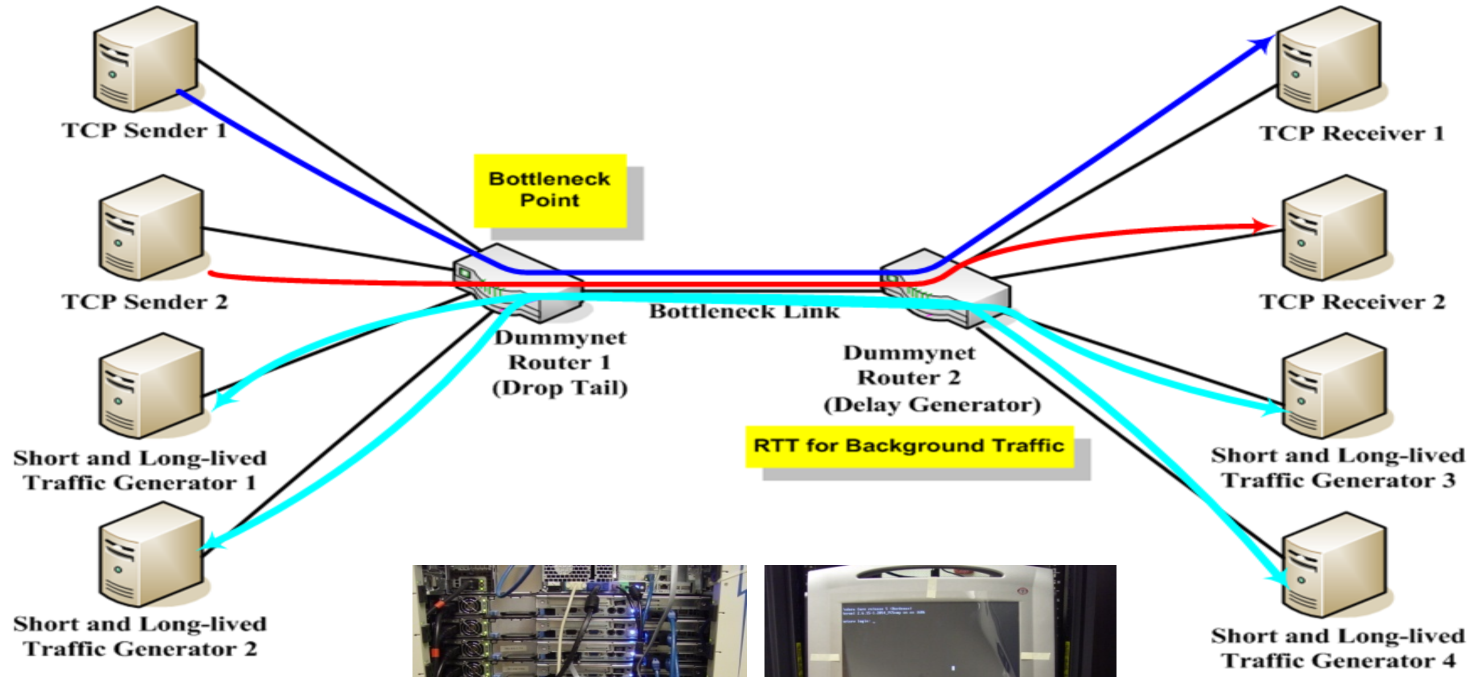
Practical deployment

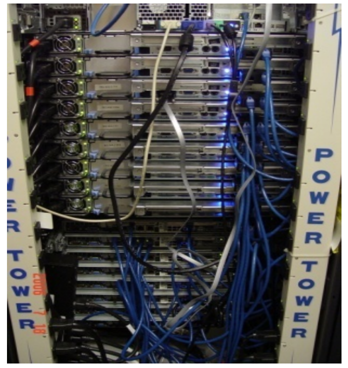



Experimental Validations
TCP Sender 1
TCP Sender 2
Short and Long-lived Traffic Generator 1
Short and Long-lived Traffic Generator 2
Bottleneck Point
Bottleneck Link
Dummynet Router 1 (Drop Tail)
Dummynet Router 2 (Delay Generator)
RTT for Background Traffic
TCP Receiver 1
TCP Receiver 2
Short and Long-lived Traffic Generator 3
Short and Long-lived Traffic Generator 4

Experimental Testbed

http://netsrv.csc.ncsu.edu/wiki/index.php/TCP_Testing


TCP Sender 1
TCP Sender 2
Short and Long-lived Traffic Generator 1
Short and Long-lived Traffic Generator 2
Bottleneck Point
Bottleneck Link
Dummynet Router 1 (Drop Tail)
Dummynet Router 2 (Delay Generator)
RTT for Background Traffic
TCP Receiver 1
TCP Receiver 2
Short and Long-lived Traffic Generator 3
Short and Long-lived Traffic Generator 4

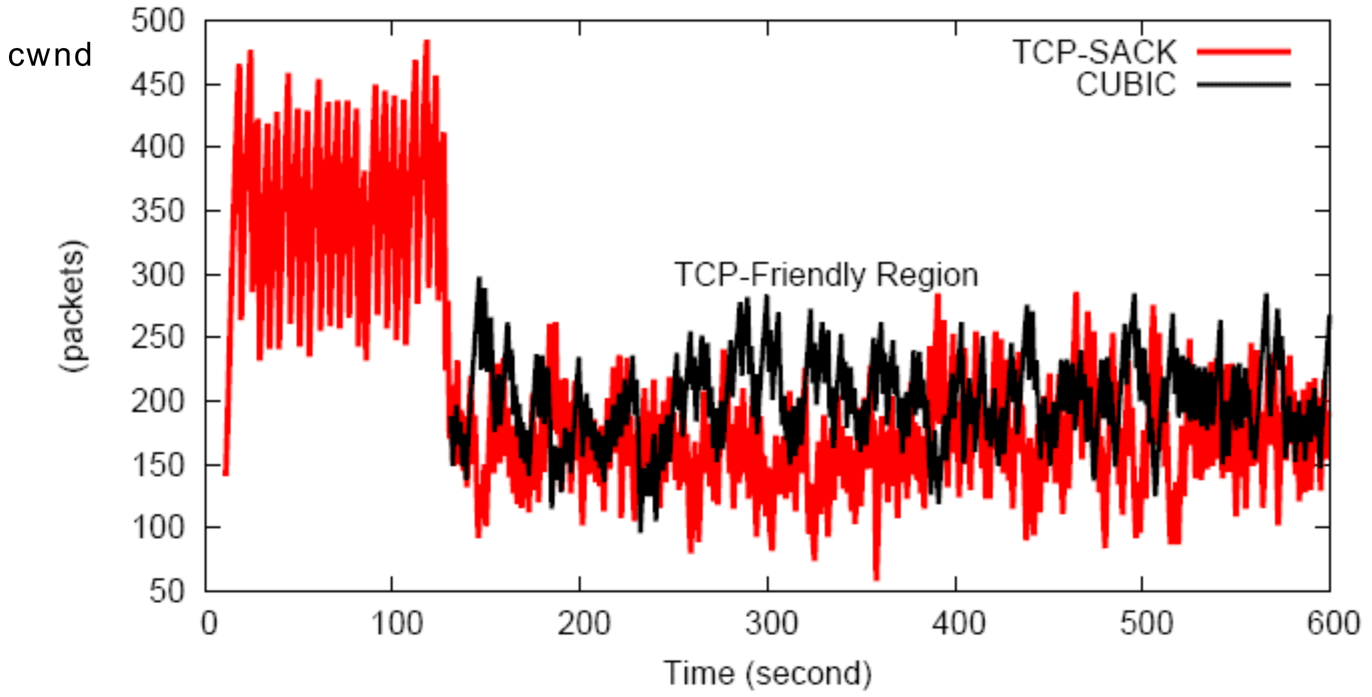


Testbed Setup : Background Traffic Generation

- They use the Internet measurement studies, which have shown complex behaviors and characteristics of Internet traffic.
- TCP RTTs observed in edge routers
  - The exponential mean is set to 66ms (one-way delay), then the CDF is very similar to the CDF of RTT samples shown in the paper, "Variability in TCP Round-trip Times" by J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay in SIGCOMM IMC, 2003.
- Inter-arrival time between two successive TCP connections: Exponential distribution (observed from Floyd and Paxson).
- Short-lived flows (web traffic flows) follow Lognormal (Body) and Pareto (Tail) Distribution.
  - Using the parameters from the paper "Generating Representative Web Workloads for Network and Server Performance Evaluation" by Paul Barford, Mark Crovella in SigMetric 1998.

TCP Friendliness

- Dummynet Testbed: 400Mbps, RTT 10ms, 100% router buffer, and moderate background traffic

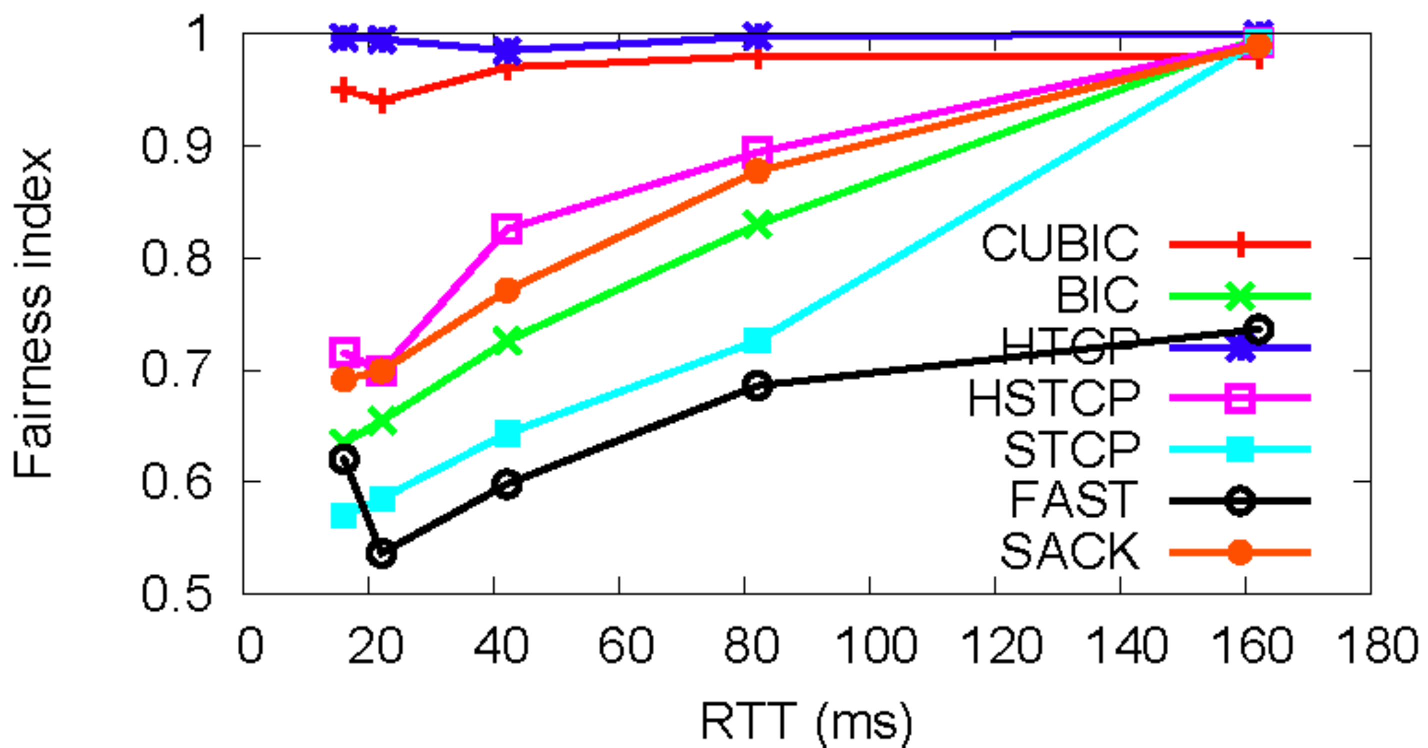


RTT Fairness

- Dummynet Testbed: 400Mbps, 1MB buffer size, background traffic
  - Flow A with RTT 160ms
  - Flow B with RTT 20-160ms

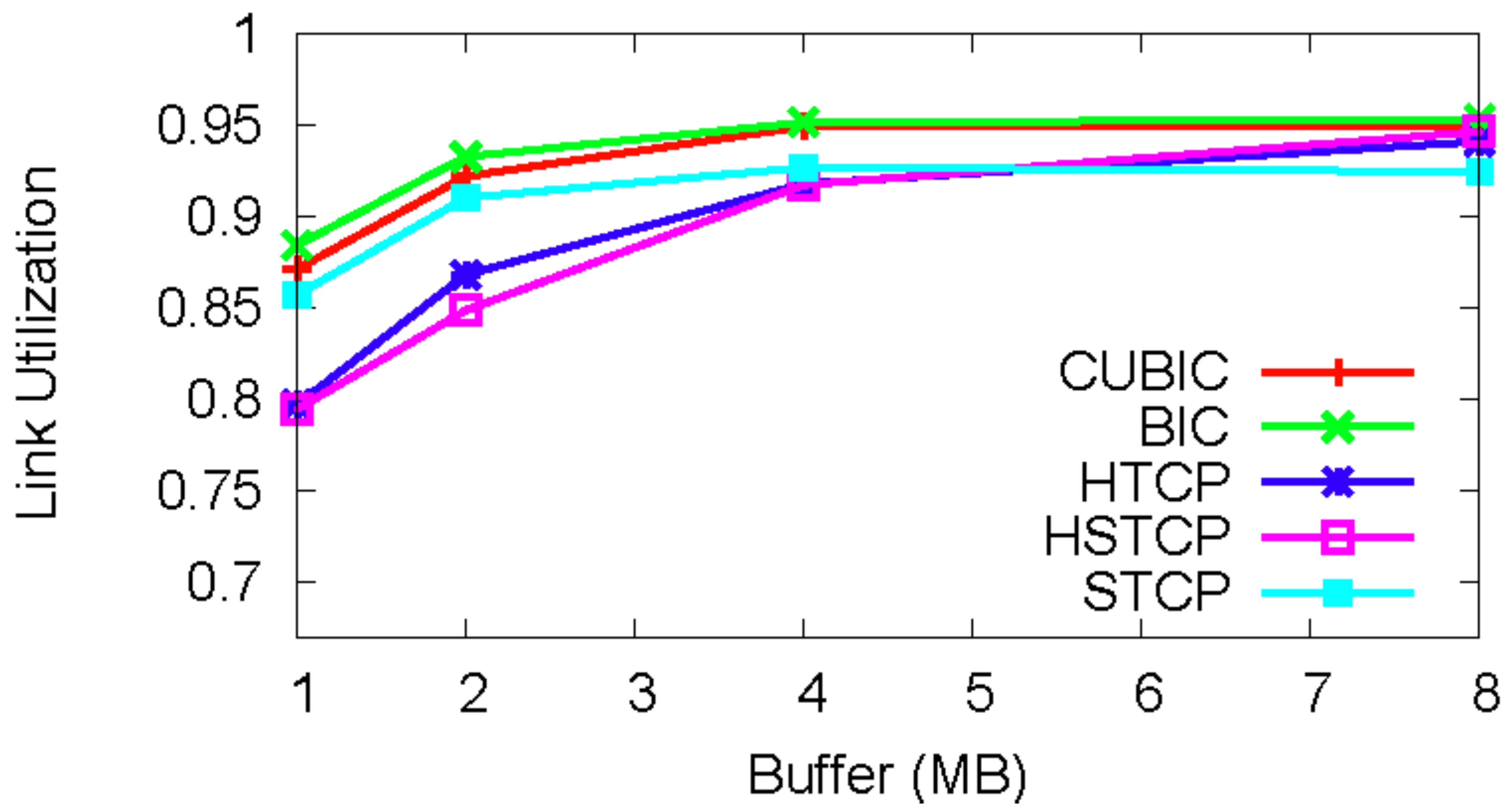
Jain's fairness index

$$= \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} {x_i}^2}$$

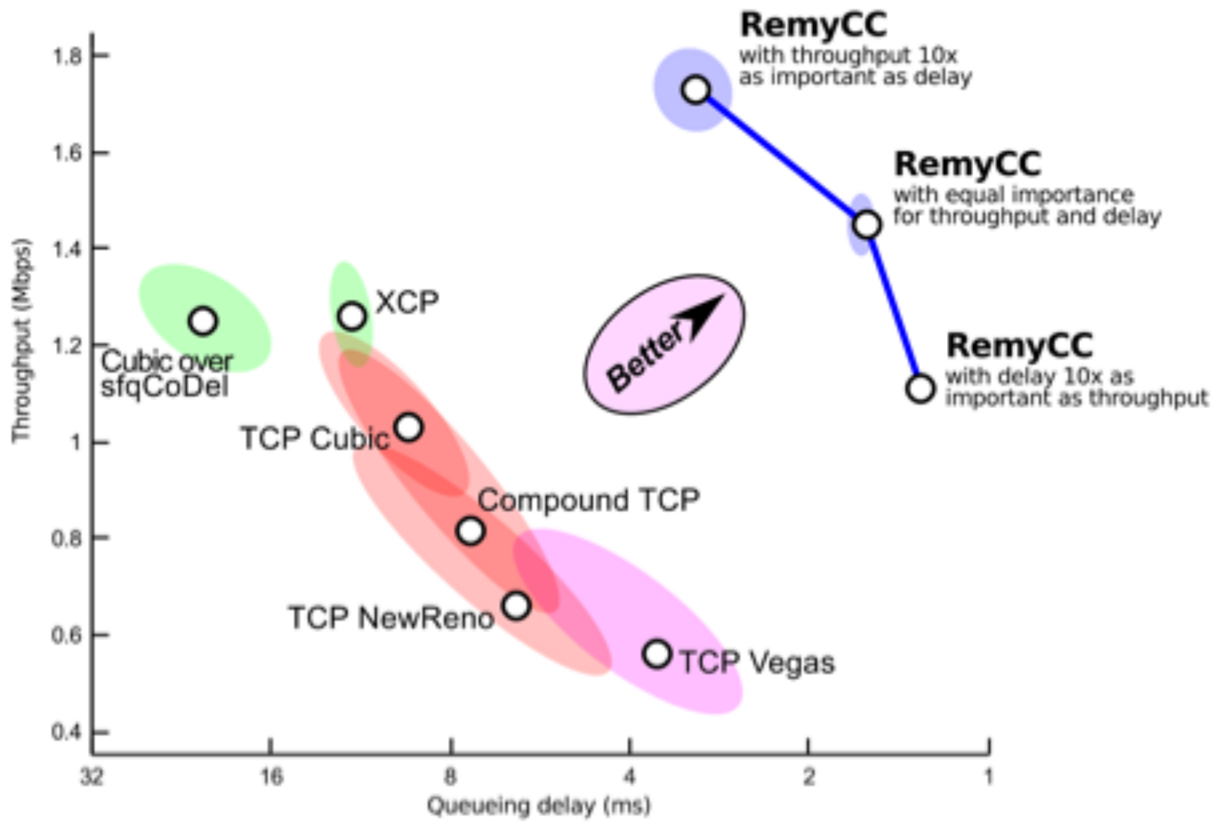


Link Utilization

- Dummynet Testbed : 400Mbps, 4 high-speed TCP flows, background traffic, and vary the buffer size from 1MB to 8MB



Machine Learning for TCP



Shown here: median results and 1-σ ellipses for eight endpoints contending for a 15 Mbps link, RTT = 150 ms, exponentially-distributed flow lengths and pause times.

Source: "TCP Ex Machina," Keith Winstein and Hari Balakrishnan, *SIGCOMM* 2013.

Outline

- TCP Review
- New TCP flavors
  - Multipath-TCP
  - CUBIC
  - BBR
- Mathematical modeling of TCP

Q: How should flows compete for bandwidth when there is congestion in the network?

BBR: the story goes back to 2013...

- Many Google services complained about TCP performance

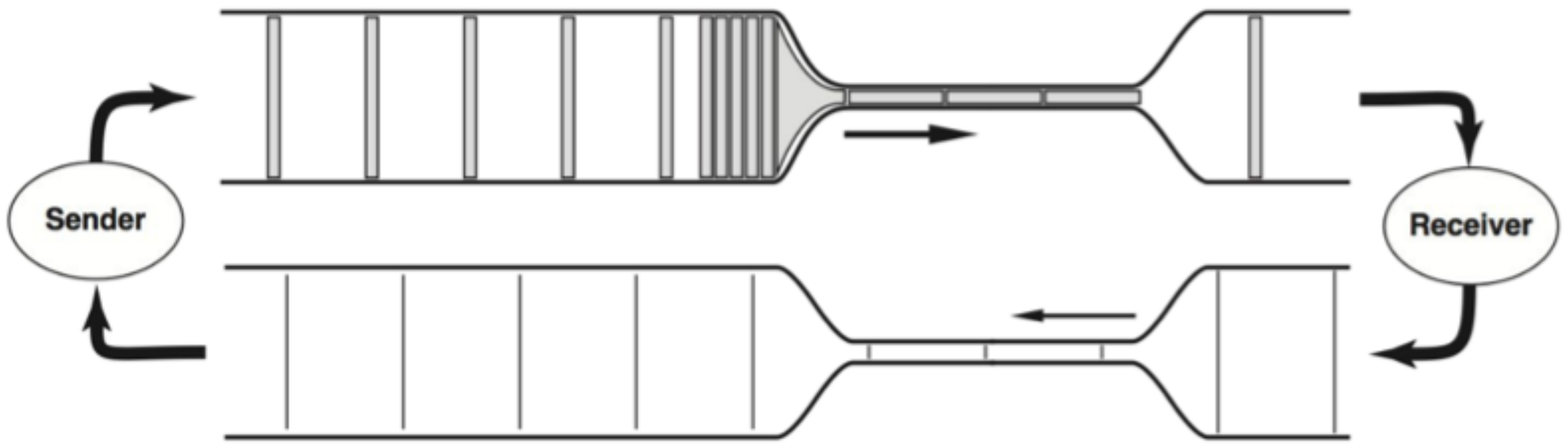
- Internal backbone TCP throughput often < 10 Mbps
- Youtube.com: terrible video quality sometimes, with RTT > 10 s
- Google.com: poor latency in deveoping regions

- Google TCP congestion control was CUBIC (Linux default)
  - Packet loss is the sole signal
  - Services started to “work around” TCP
    - Use parallel connections, tweak TCP knobs, add more network buffers, etc...

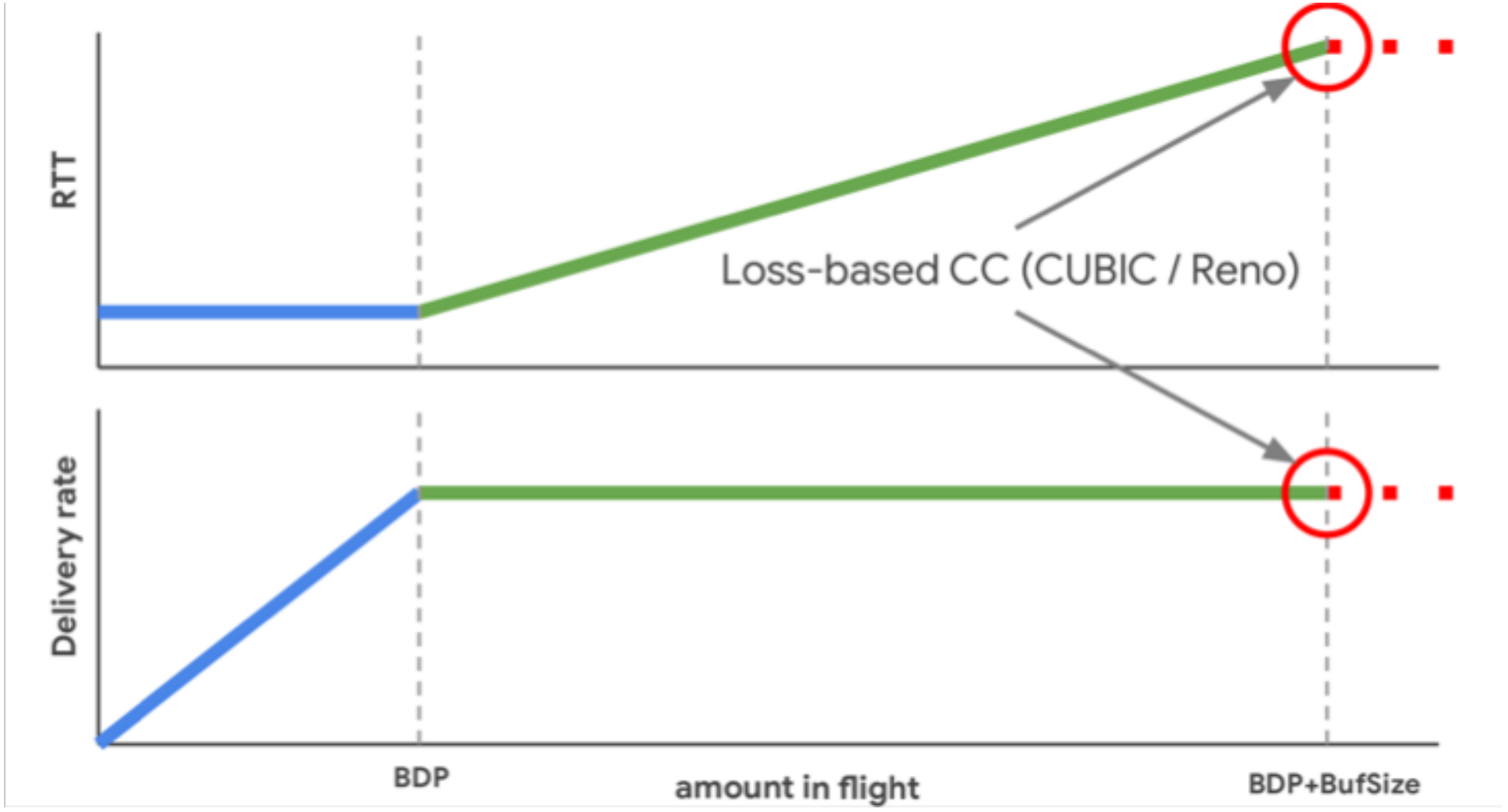
The problem: Loss-based congestion control

- Loss-based congestion control (Reno, CUBIC)
  - Keeps sending until it sees a loss
- But packet loss alone is not a good proxy for congestion
- If packet loss comes before congestion, loss-based CC gets low tput
  - 10 Gbps over 100 ms RTT needs < 0.000003% packet loss (infeasible)
  - 1% loss (feasible) over 100 ms RTT gets only 3 Mbps
- If loss comes after congestion, loss-based CC bloats buffers, suffers high delays

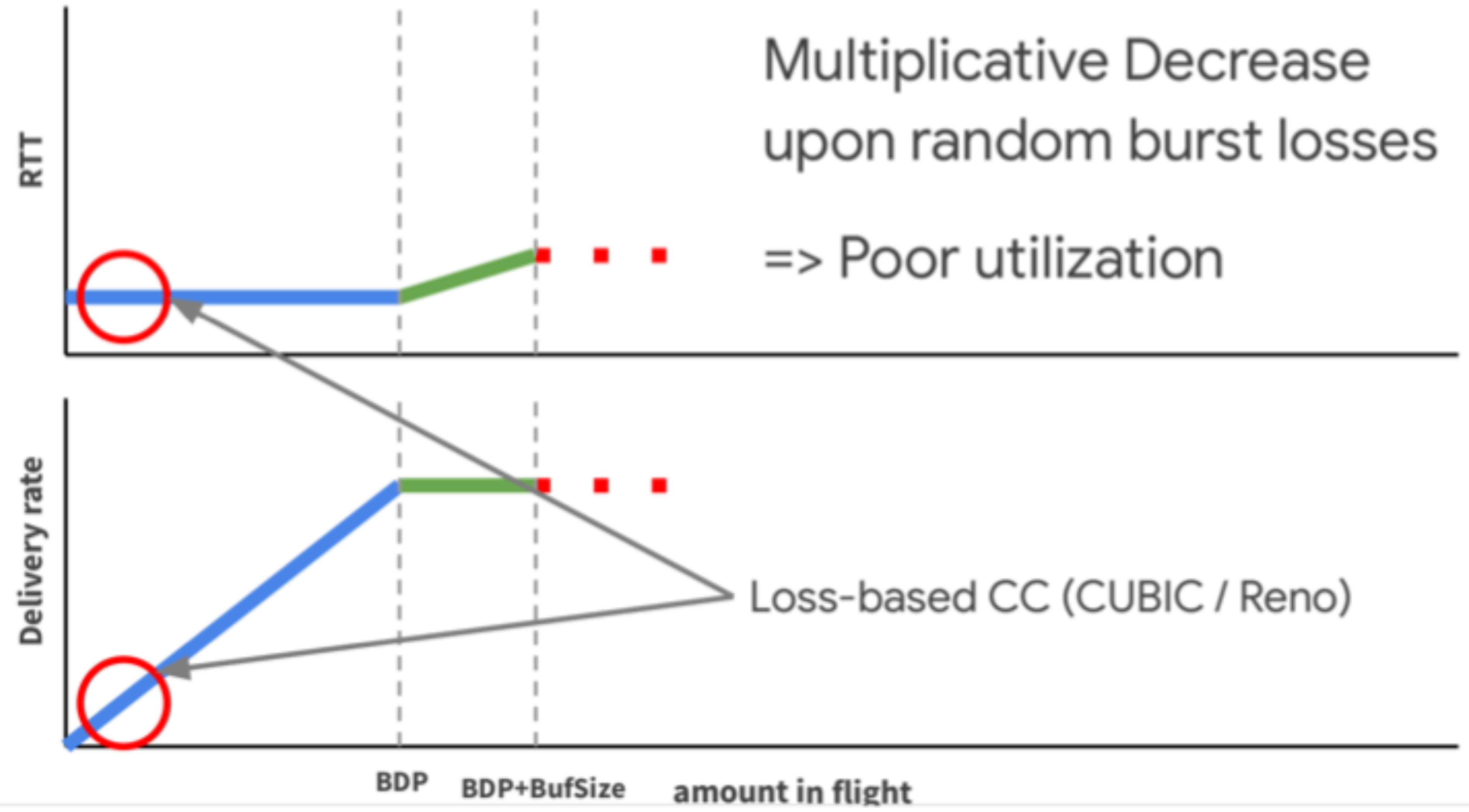
Network congestion and bottlenecks



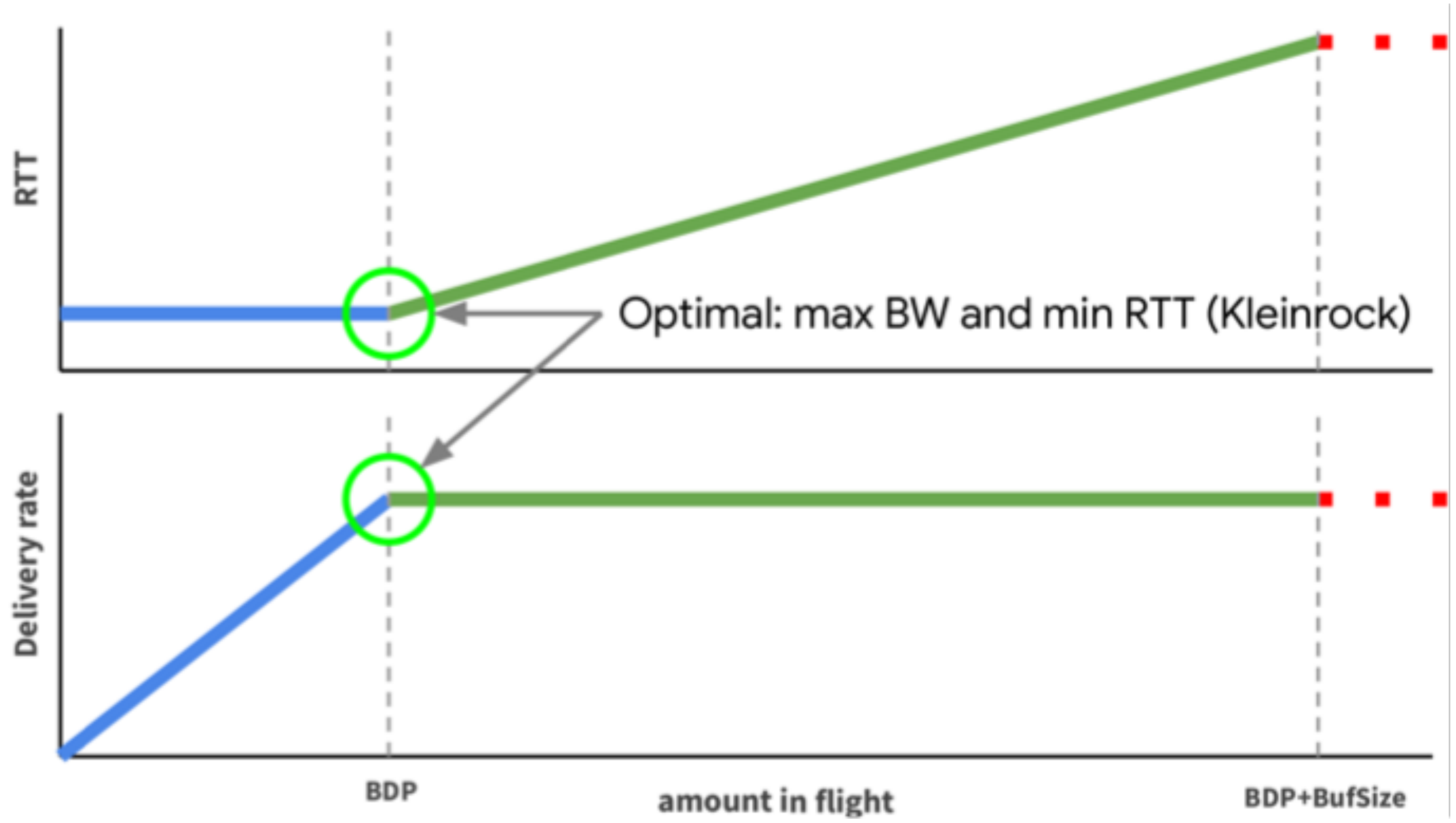
Loss-based congestion control in deep buffers



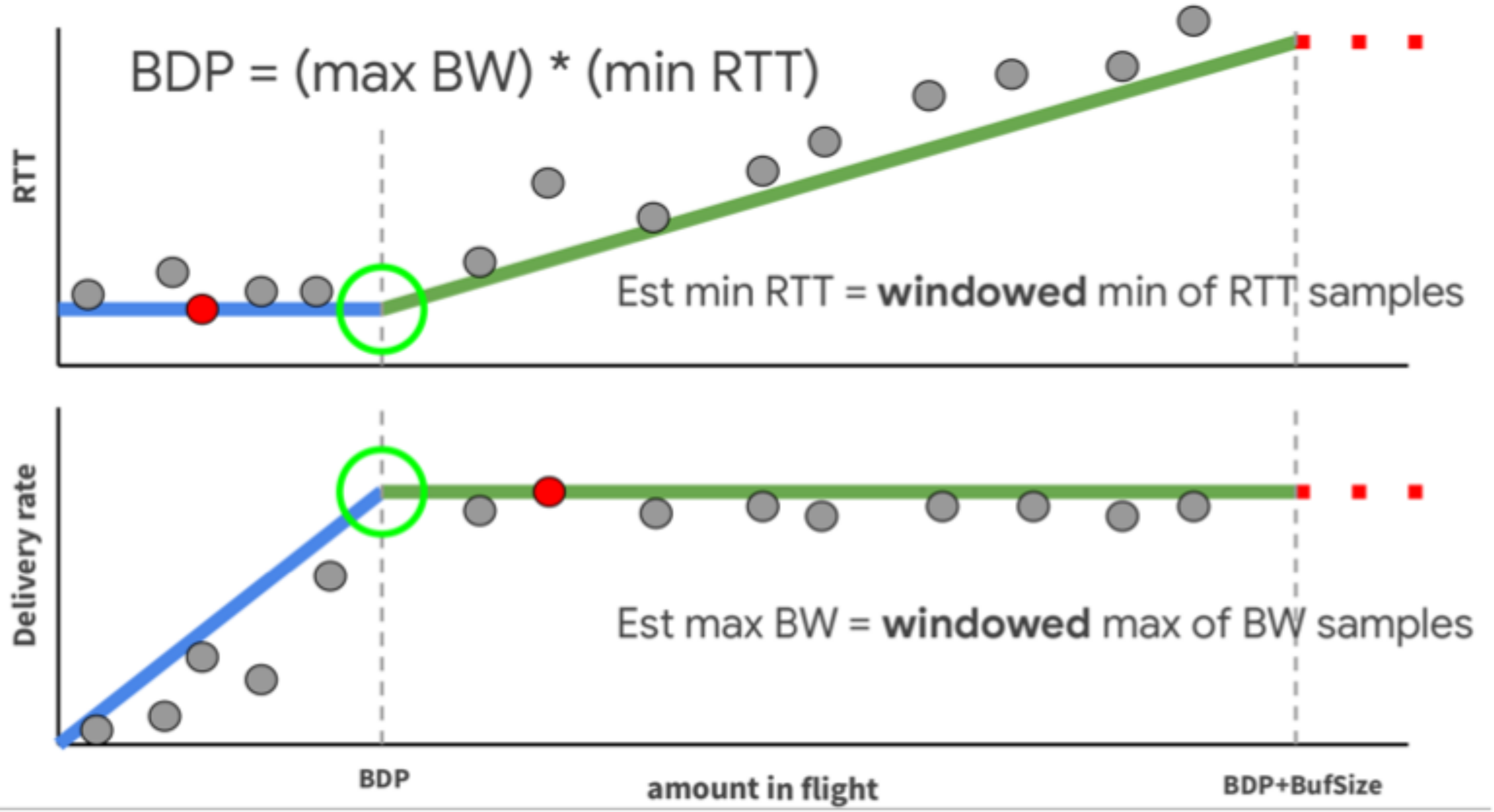
Loss-based congestion control in shallow buffers



Optimal operating point



Estimating optimal point (max BW, min RTT)



The devil is in the details…

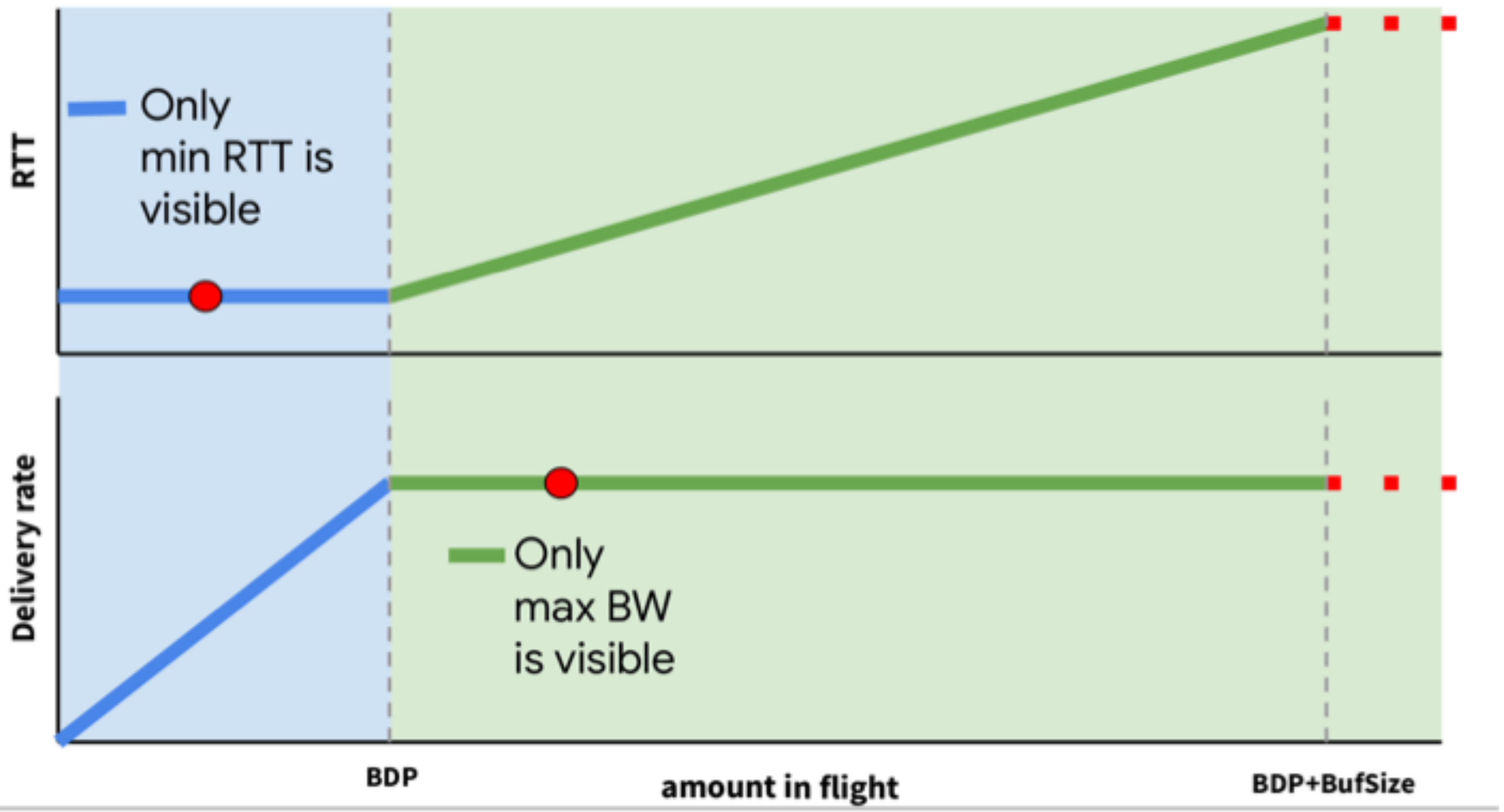
Overall goals

1. Pace yourself: Send with rate equal to the estimated bottleneck bandwidth (BtlBw)
   - cycle_gain parameter
2. Limit yourself: Total data in flight = bandwidth-delay product
   - cwnd_gain parameter

Sending a packet

```
function send(packet)
 bdp = BtlBwFilter.currentMax × RTpropFilter.currentMin
 if (inflight >= cwnd_gain × bdp)
   // wait for ack or retransmission 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 / (cycle_gain × BtlBwFilter.currentMax)
 timerCallbackAt(send, nextSendTime)
```

To see max BW, min RTT: probe both sides of BDP



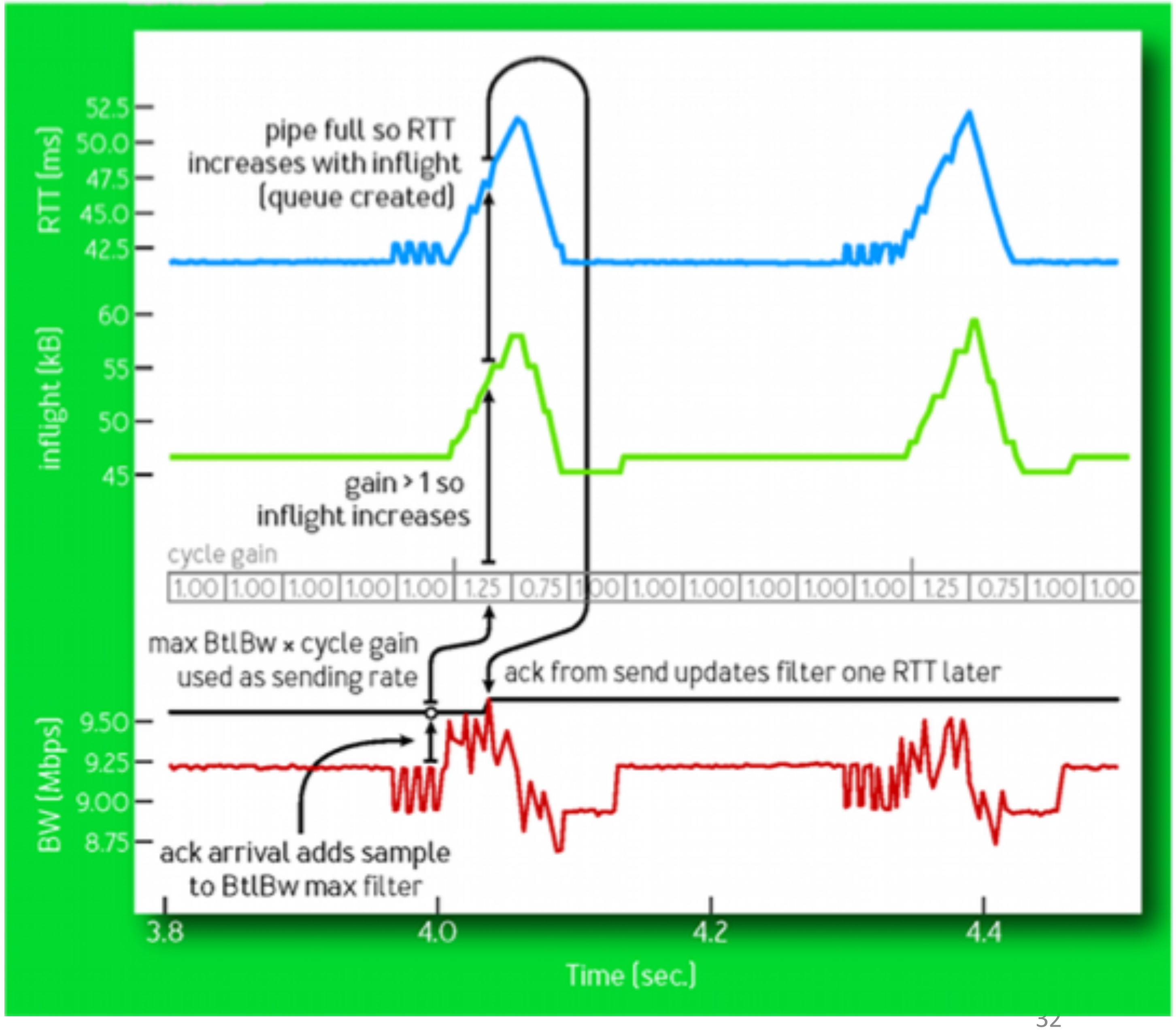
Estimate RTT, BtlBw: On Receiving an ACK

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

Probe Example

- Periodically increase "cycle gain" to probe
- Decrease it after to clear up queues

FIGURE 2: RTT (BLUE), INFLIGHT (GREEN) AND DELIVERY RATE (RED) DETAIL

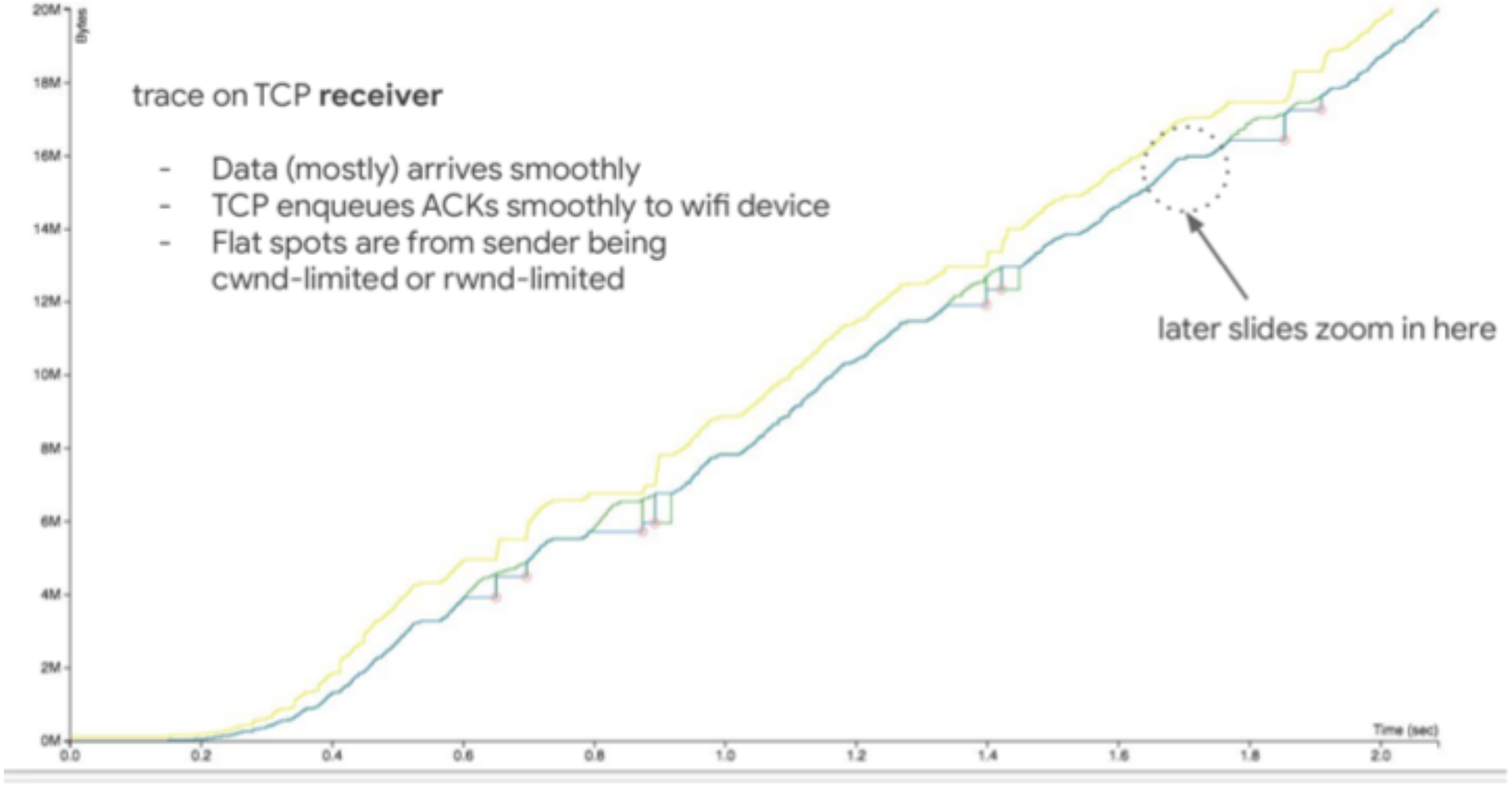


TCP Data and ACK Aggregation

- Commonly used for amortizing overheads to increase efficiency
  - In interrupt processing and hardware/software offload mechanisms (TSO)
  - In shared media like WiFi, cellular, cable modems

- ACK aggregation severely limited throughput in initial BBR release

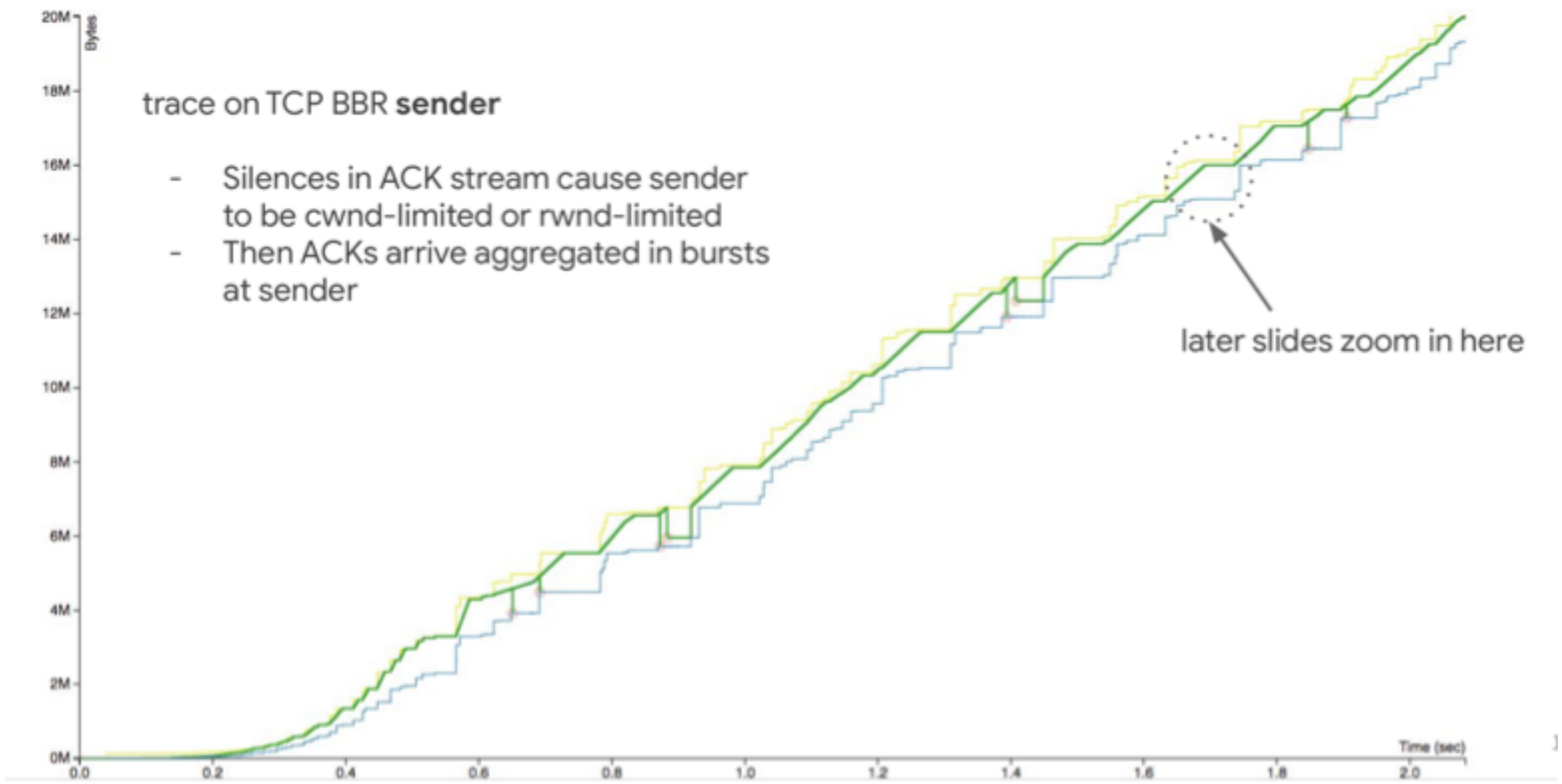
Full WiFi trace: receiver



trace on TCP **receiver**

- Data (mostly) arrives smoothly
- TCP enqueues ACKs smoothly to wifi device
- Flat spots are from sender being cwnd-limited or rwnd-limited

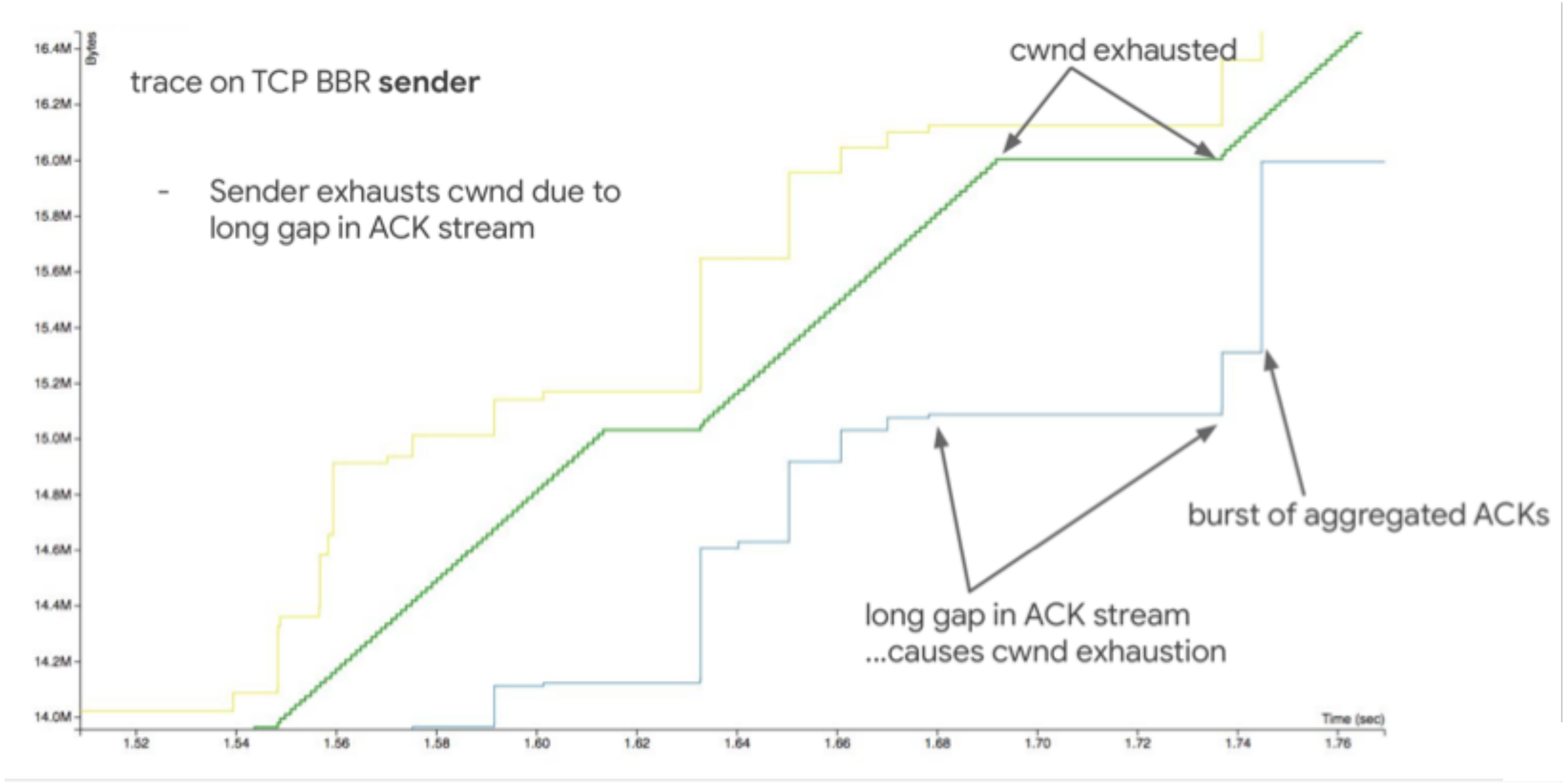
Full WiFi trace: sender



trace on TCP BBR **sender**

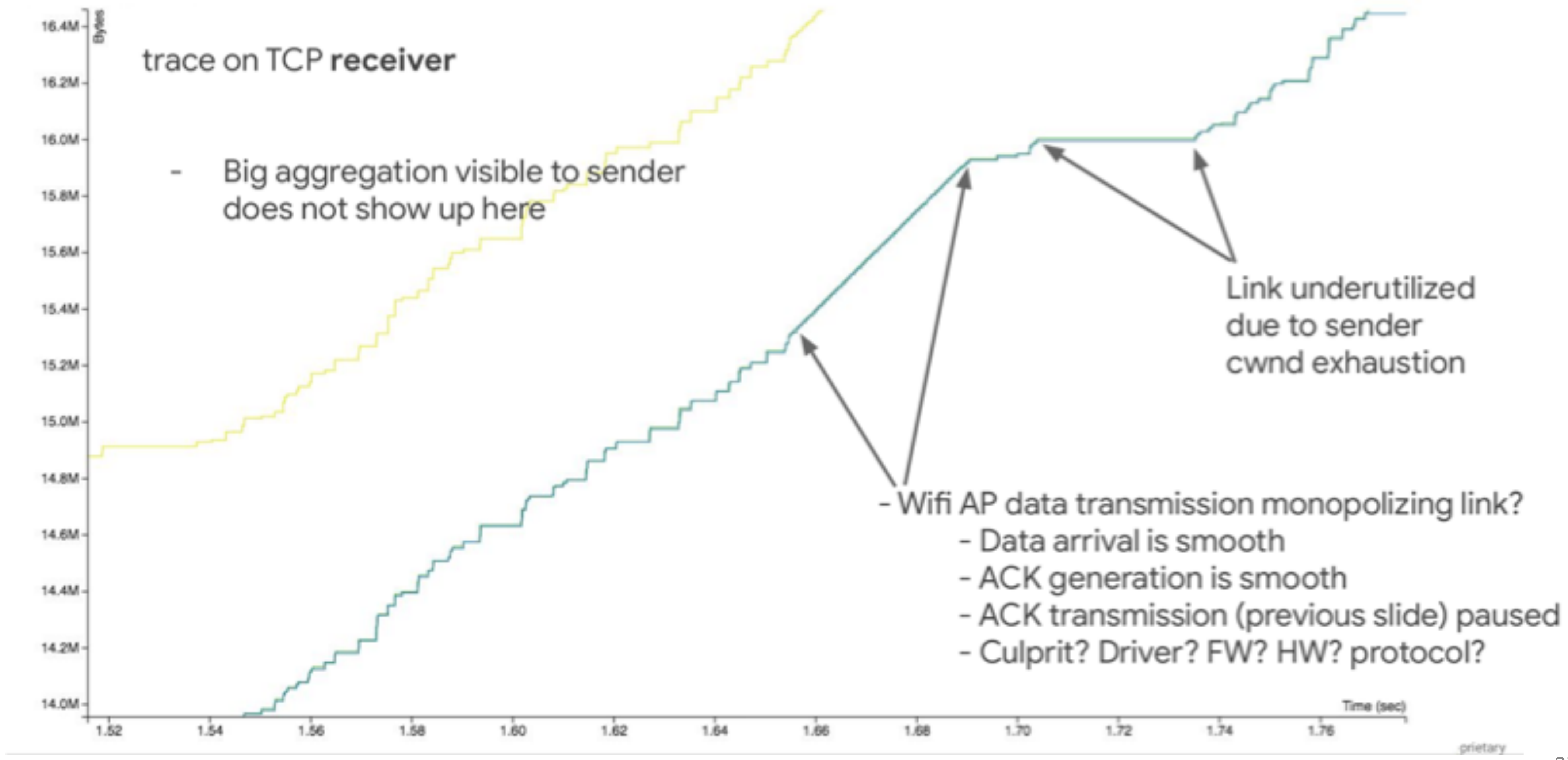
- Silences in ACK stream cause sender to be cwnd-limited or rwnd-limited
- Then ACKs arrive aggregated in bursts at sender

Zoomed-in WiFi trace: sender



- Sender exhausts cwnd due to long gap in ACK stream

Zoomed-in WiFi trace: receiver



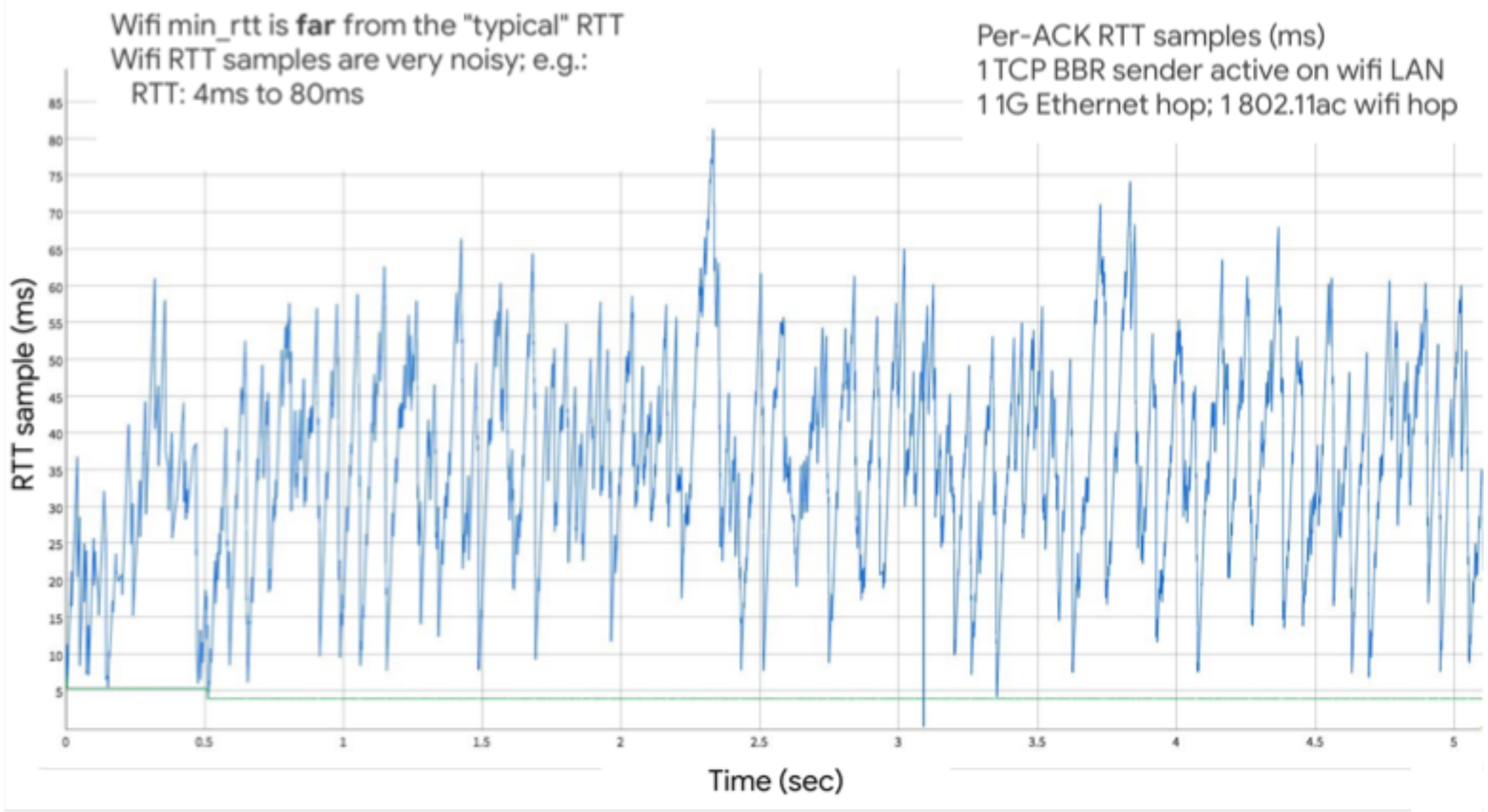
trace on TCP **receiver**

- Big aggregation visible to sender does not show up here

Link underutilized due to sender cwnd exhaustion

- Wifi AP data transmission monopolizing link?
  - Data arrival is smooth
  - ACK generation is smooth
  - ACK transmission (previous slide) paused
  - Culprit? Driver? FW? HW? protocol?

WiFi RTTs



Wifi min_rtt is **far** from the "typical" RTT
Wifi RTT samples are very noisy; e.g.:
RTT: 4ms to 80ms
Per-ACK RTT samples (ms)
1 TCP BBR sender active on wifi LAN
1 1G Ethernet hop; 1 802.11ac wifi hop
RTT sample (ms)
Time (sec)
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

Solution

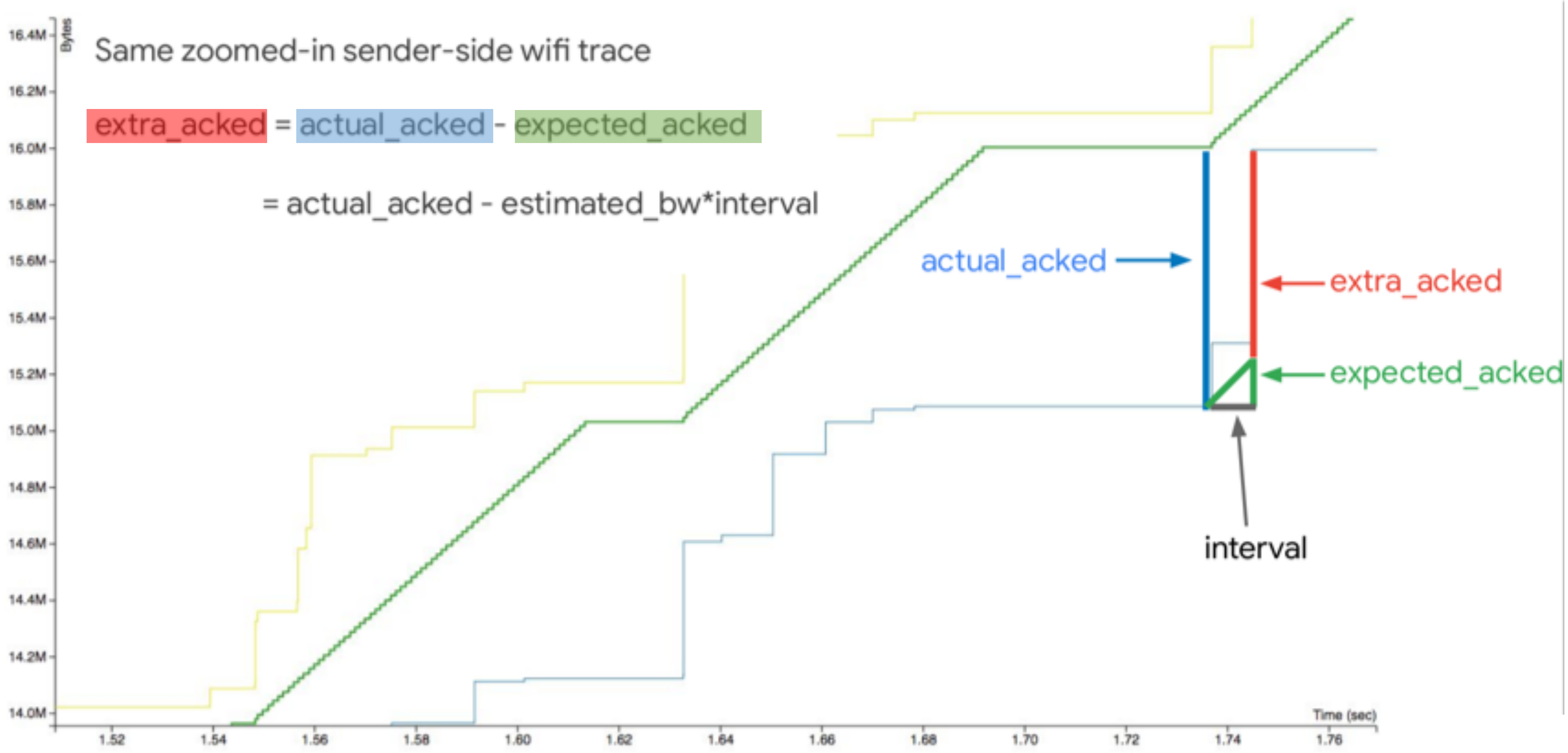
- Provision cwnd based on degree of ACK aggregation (extra_acked)

extra_acked = excess data ACKed beyond expected amount over this time interval

= actual acked – estimated_BW * time_interval

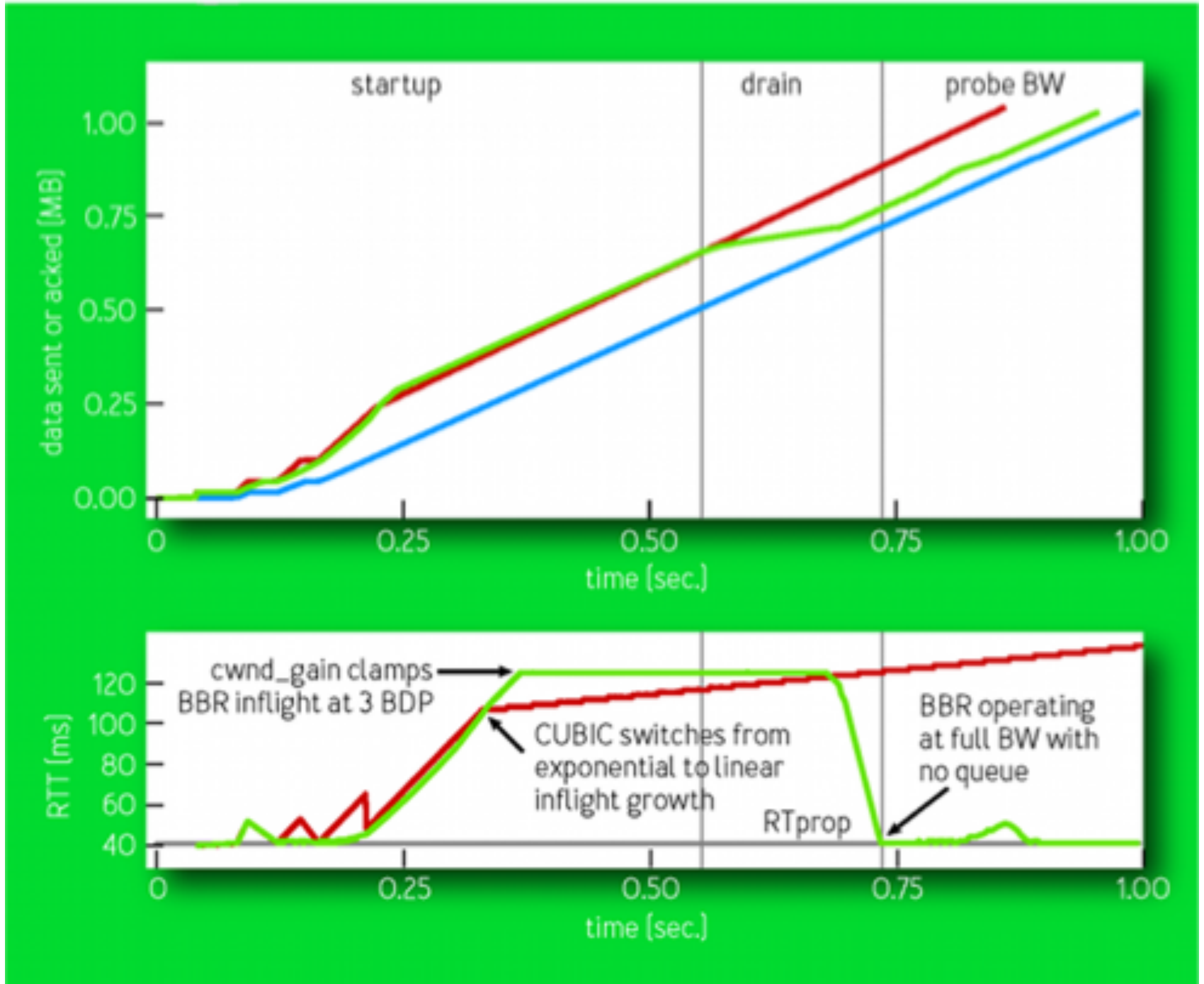
Cwnd = estimated_BW * min_RTT + max(extra_acked, len=10 RTT)

BBR aggregation estimator: visualization



Comparison with TCP CUBIC

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



Lesson learned from developing BBR

- Any algorithm can only work well with clear feedback
  - Delay often not reflecting queuing on the Internet
  - Loss is flaky if induced by transient burst
  - ECN is iffy if routers mark them differently
  - Bandwidth is highly dynamic and vulnerable to ACK compression
- The Internet is (and always will be) full of tricks
  - AQMs & policers assume TCP always backs off on drops
  - Middleboxes delay (or even delete) ACKs for efficiency
- Any congestion control needs continuous refinement to perform well
  - Major performance bugs from unexpected places: delayed ACK, sender/receiver segment offload, socket options, kernel memory management, etc...

Outline

- TCP Review
- New TCP flavors
  - Multipath-TCP
  - CUBIC
  - BBR
- Mathematical modeling of TCP

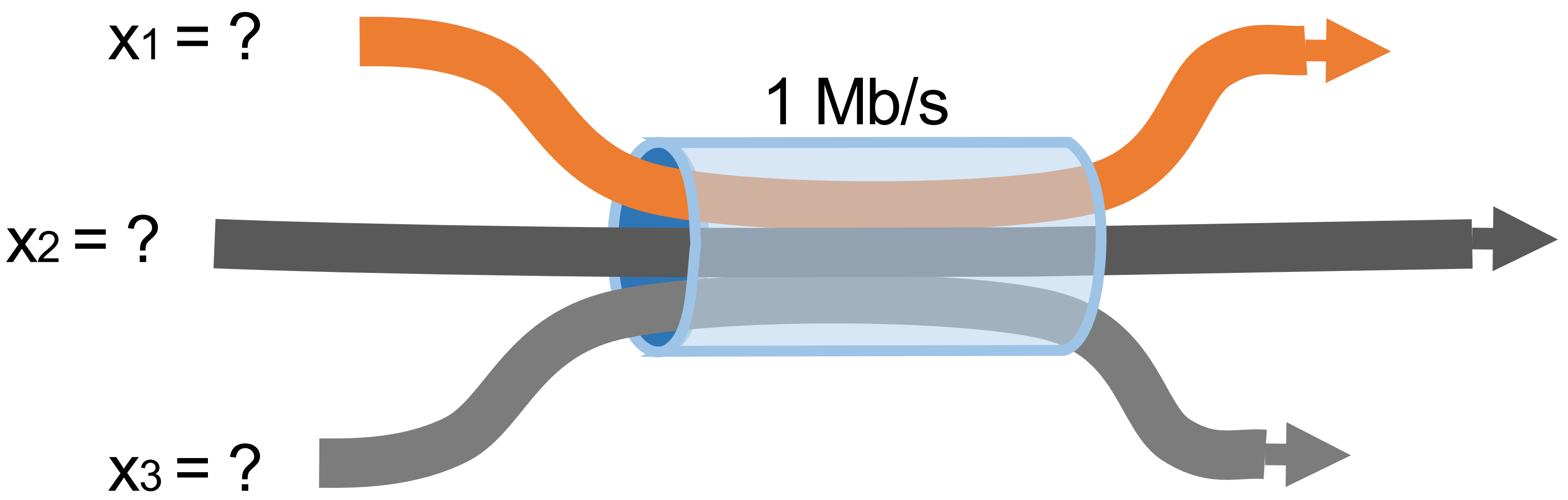
Q: How should flows compete for bandwidth when there is congestion in the network?

What Problem Does a Protocol Solve?

- BGP path selection
  - Select a path that each AS on the path is willing to use
  - Adapt path selection in the presence of failures
- TCP congestion control
  - Prevent congestion collapse of the Internet
  - Allocate bandwidth fairly and efficiently
- But, can we be more precise?
  - Define mathematically what problem is being solved
  - To understand the problem and analyze the protocol
  - To predict the effects of changes in the system
  - To design better protocols from first principles

What Problem is TCP *really* solving?

Max-min rate allocation?

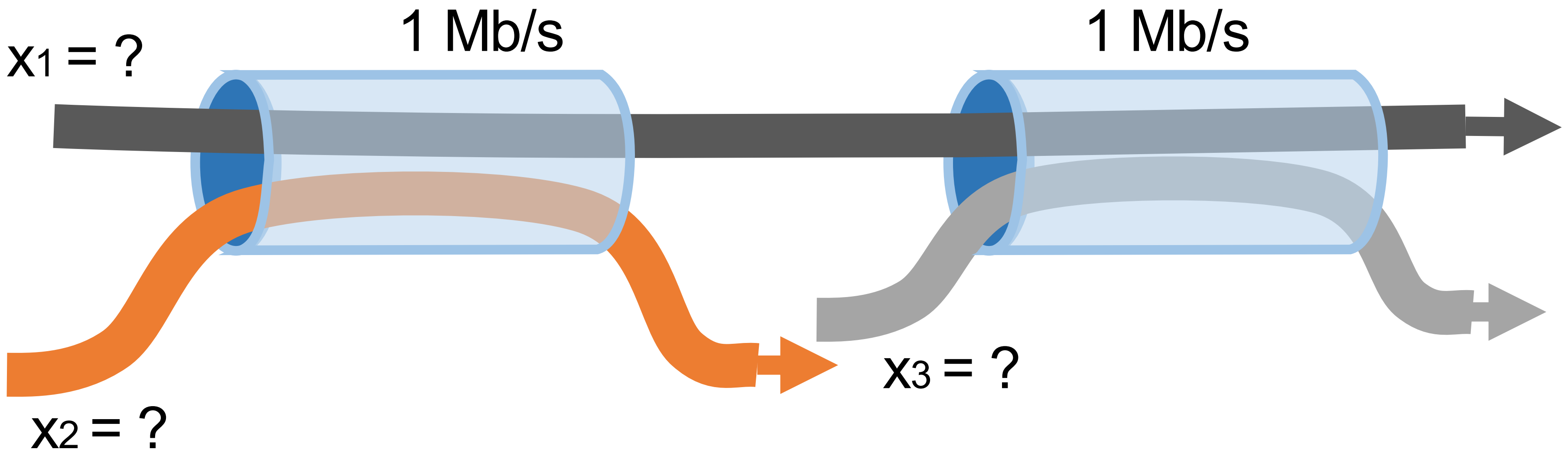


	x_1	x_2	x_3
Max-min	1/3	1/3	1/3
TCP	1/3	1/3	1/3

Assuming equal RTTs

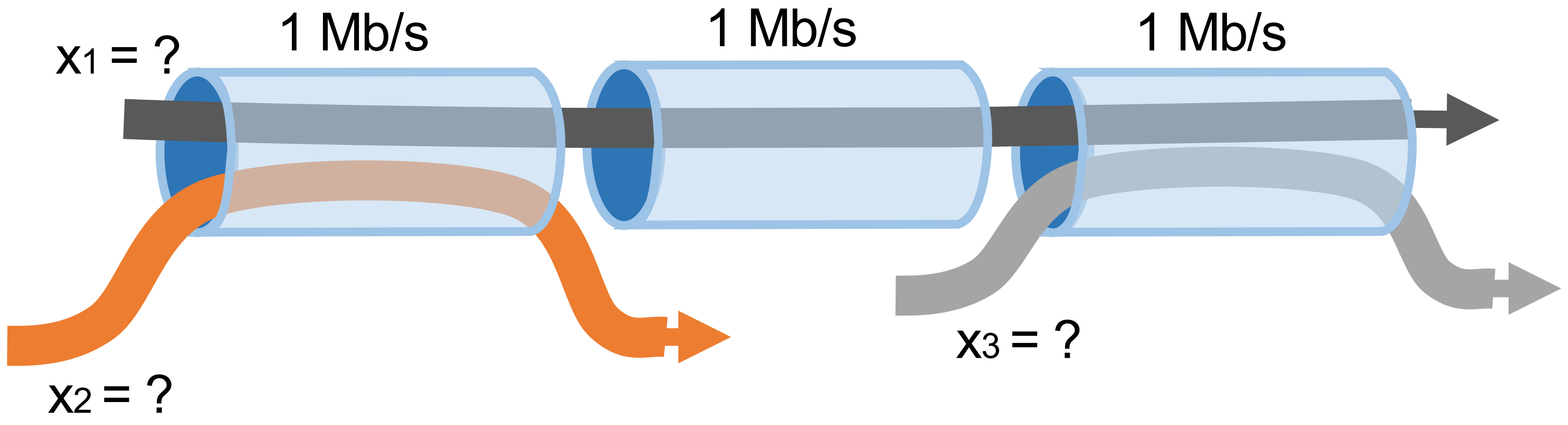
What Problem is TCP *really* solving?

Max-min rate allocation?



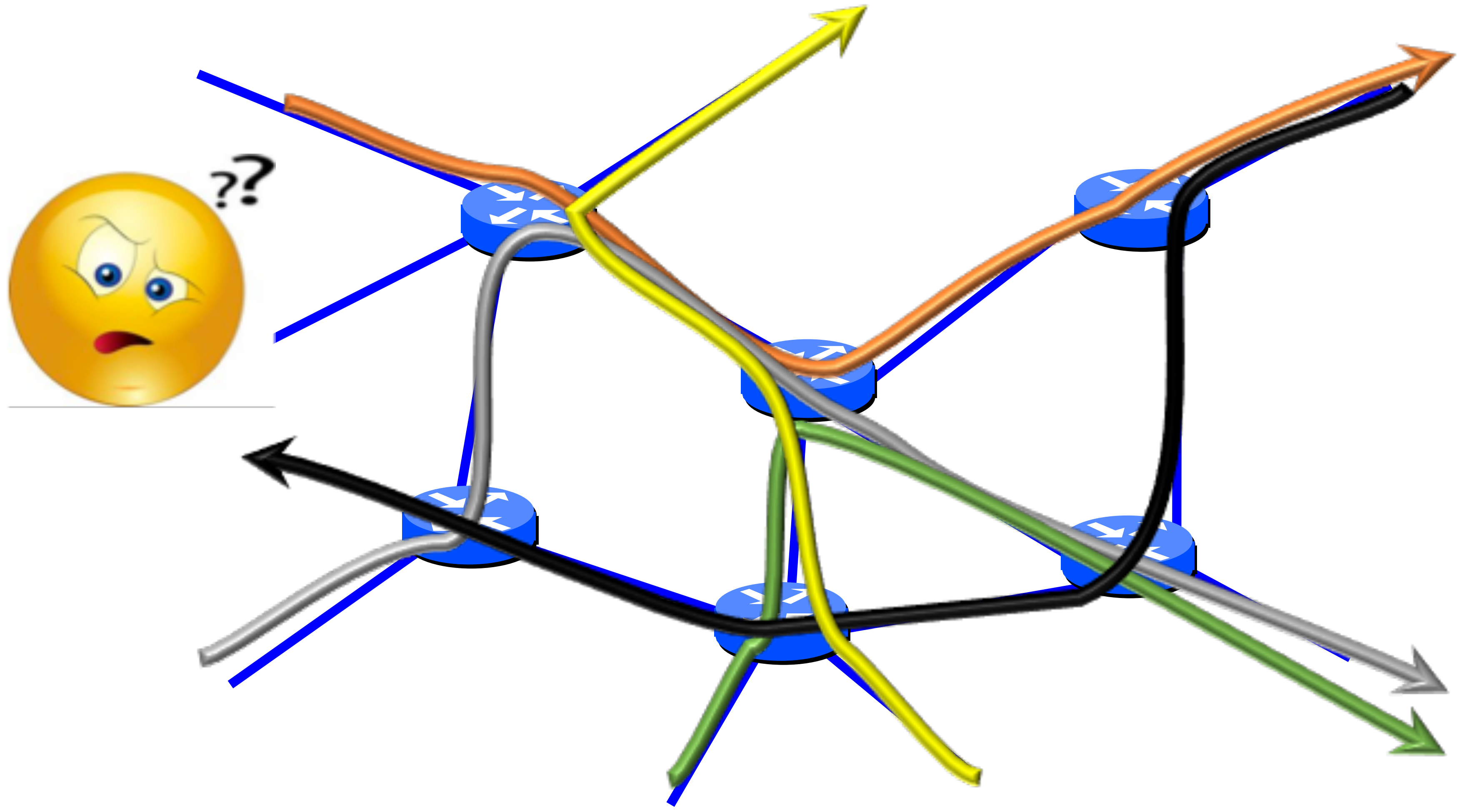
	x_1	x_2	x_3
Max-min	1/2	1/2	1/2
TCP	~0.4	~0.6	~0.6

What Problem is TCP *really* solving?

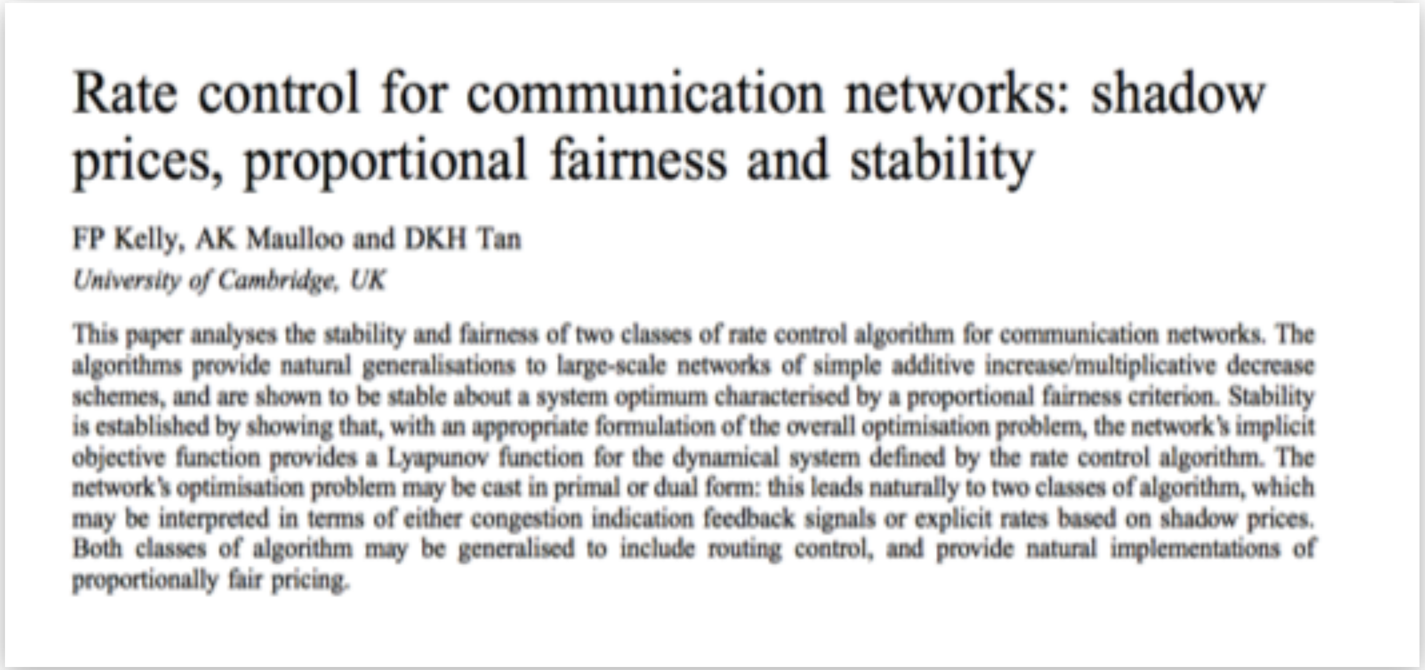


What is the difference between these links?

What Problem is TCP *really* solving?



Network Utility Maximization


Rate control for communication networks: shadow prices, proportional fairness and stability

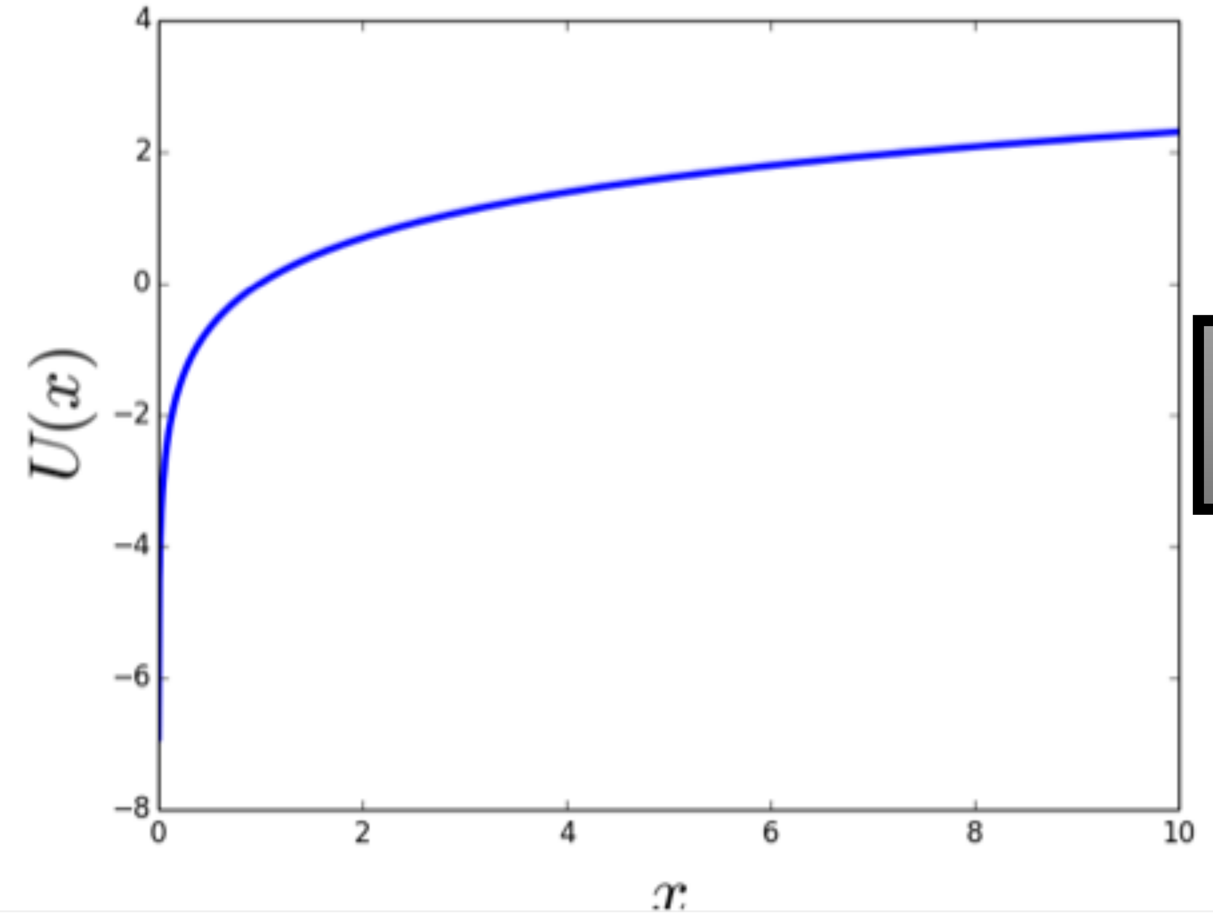
FP Kelly, AK Maulloo and DKH Tan

University of Cambridge, UK

This paper analyses the stability and fairness of two classes of rate control algorithm for communication networks. The algorithms provide natural generalisations to large-scale networks of simple additive increase/multiplicative decrease schemes, and are shown to be stable about a system optimum characterised by a proportional fairness criterion. Stability is established by showing that, with an appropriate formulation of the overall optimisation problem, the network's implicit objective function provides a Lyapunov function for the dynamical system defined by the rate control algorithm. The network's optimisation problem may be cast in primal or dual form: this leads naturally to two classes of algorithm, which may be interpreted in terms of either congestion indication feedback signals or explicit rates based on shadow prices. Both classes of algorithm may be generalised to include routing control, and provide natural implementations of proportionally fair pricing.

- TCP is solving an optimization problem!
- Spurred a lot of research on analyzing and designing network protocols from the lens of optimization
- ~ 6000 citations

Utility Function

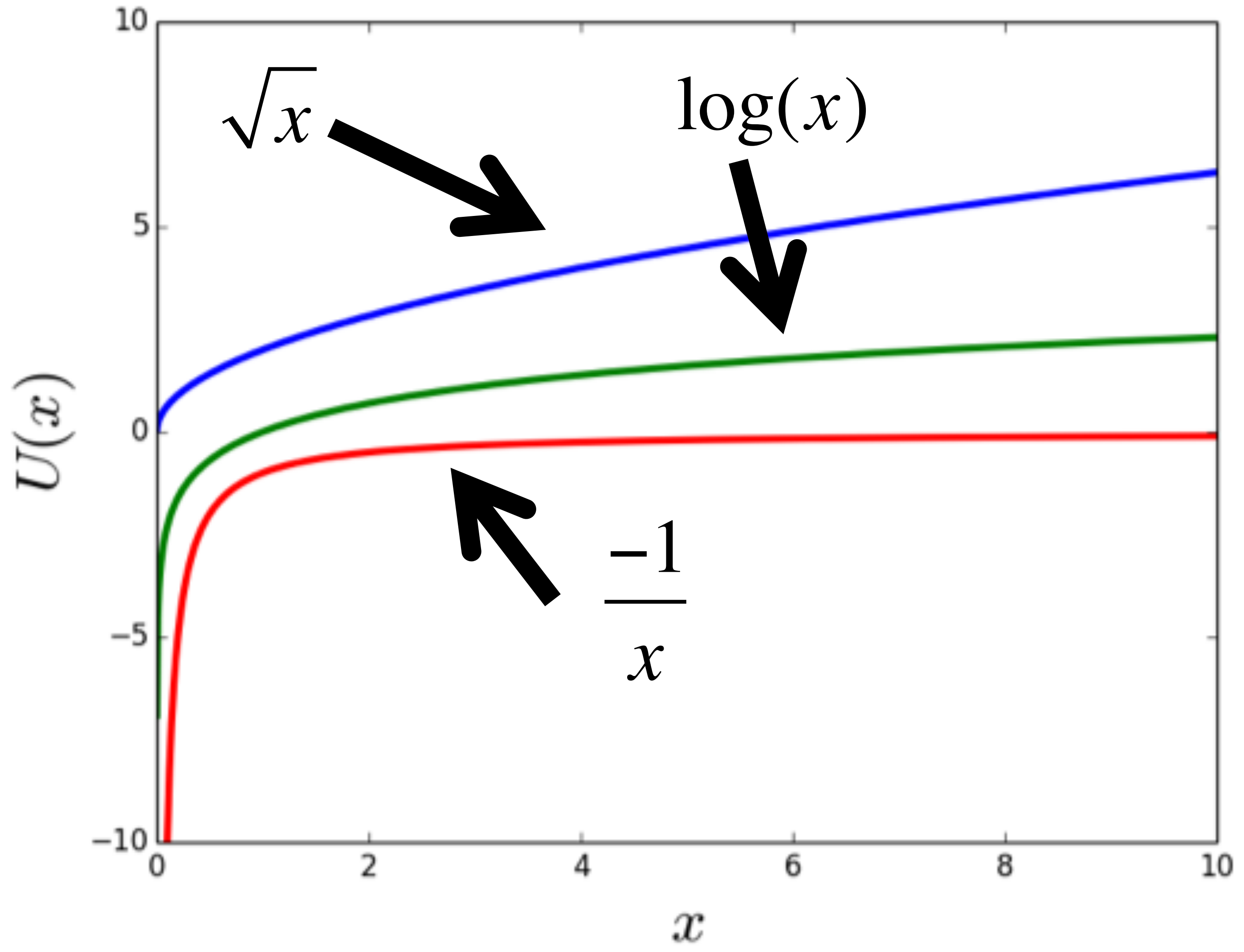


Good model for elastic flows

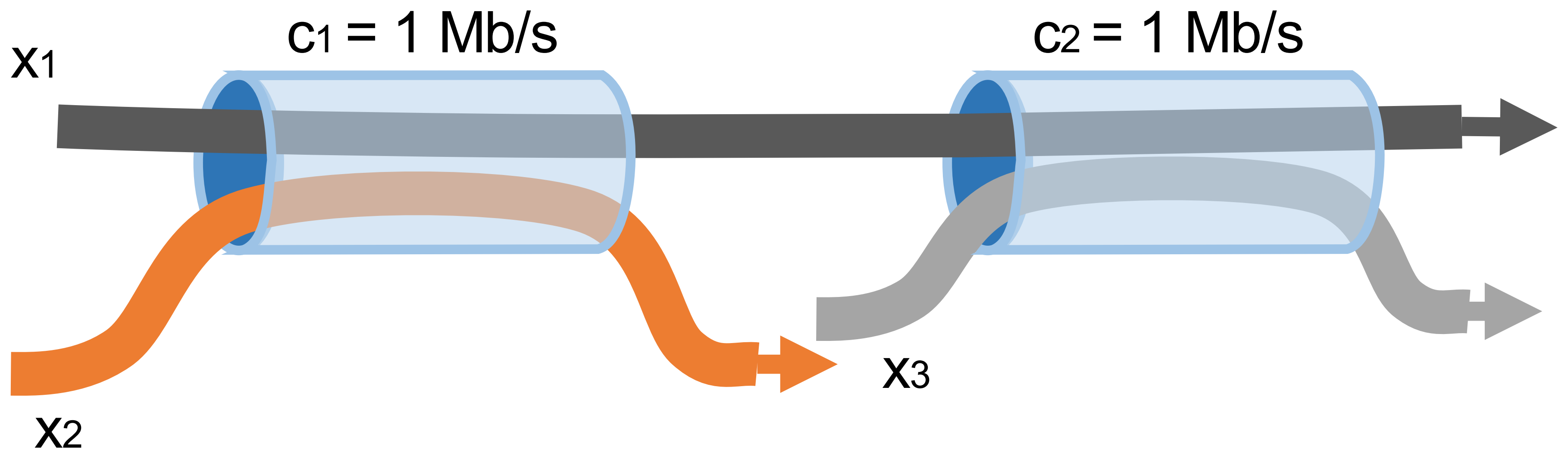
- e.g. file downloads

- The benefit derived from sending at rate *x*
- We'll assume *U(.)* is **increasing** & **concave**

Examples of Utility Functions

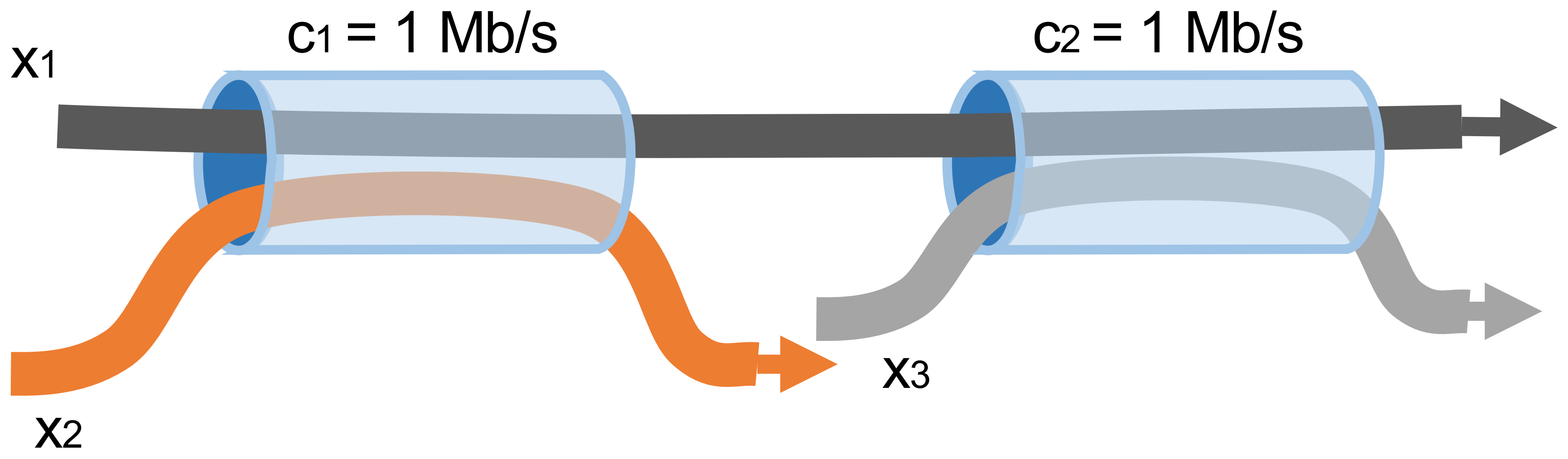


Network Utility Maximization (NUM)



- maximize $\log(x_1) + \log(x_2) + \log(x_3)$
- subject to: $x_1 + x_2 \leq 1$
- $x_1 + x_3 \leq 1$

Network Utility Maximization (NUM)



- maximize $\log(x_1) + \log(x_2) + \log(x_3)$
- subject to:

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

NUM: General Case

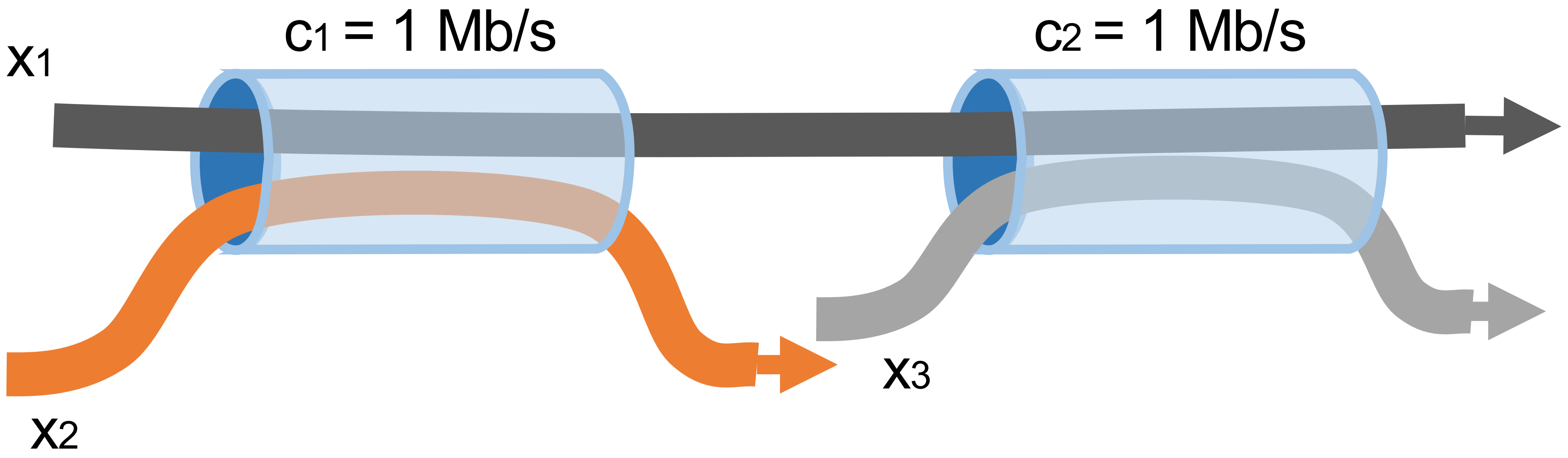
maximize $$\sum_{i=1}^{N} U_i(x_i)$$

- *N flows*
- *L links*

subject to:

$$\overset{R_{LxN}\ \text{routing matrix}}{\begin{pmatrix} 0 & 1 & 0 & \cdots & 1 & 1 \\ 1 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & \vdots \\ 0 & 0 & 1 & \cdots & 1 & 0 \end{pmatrix}} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} \leq \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_L \end{pmatrix}$$

NUM Example 1: Throughput Maximization

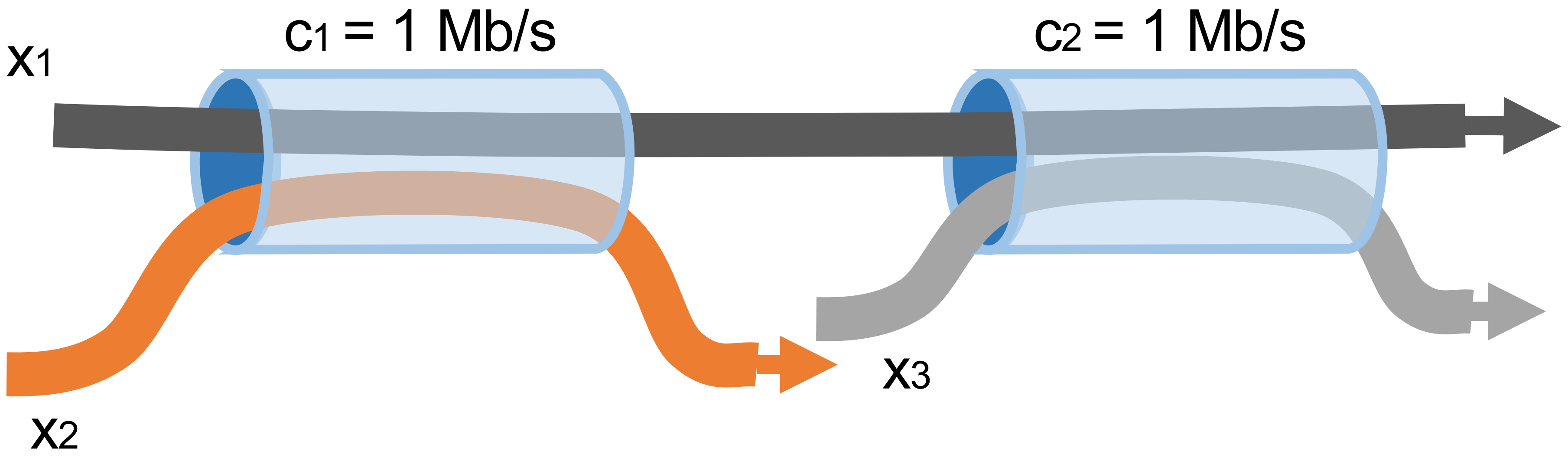


- maximize $x_1 + x_2 + x_3$
- subject to: $x_1 + x_2 \leq 1$

  $x_1 + x_3 \leq 1$

$$\Rightarrow \left\{ \begin{array}{l} x_1^\star = 0 \text{ Mb/s} \\ x_2^\star = 1 \text{ Mb/s} \\ x_3^\star = 1 \text{ Mb/s} \end{array} \right.$$

NUM Example 2: Proportional Fairness

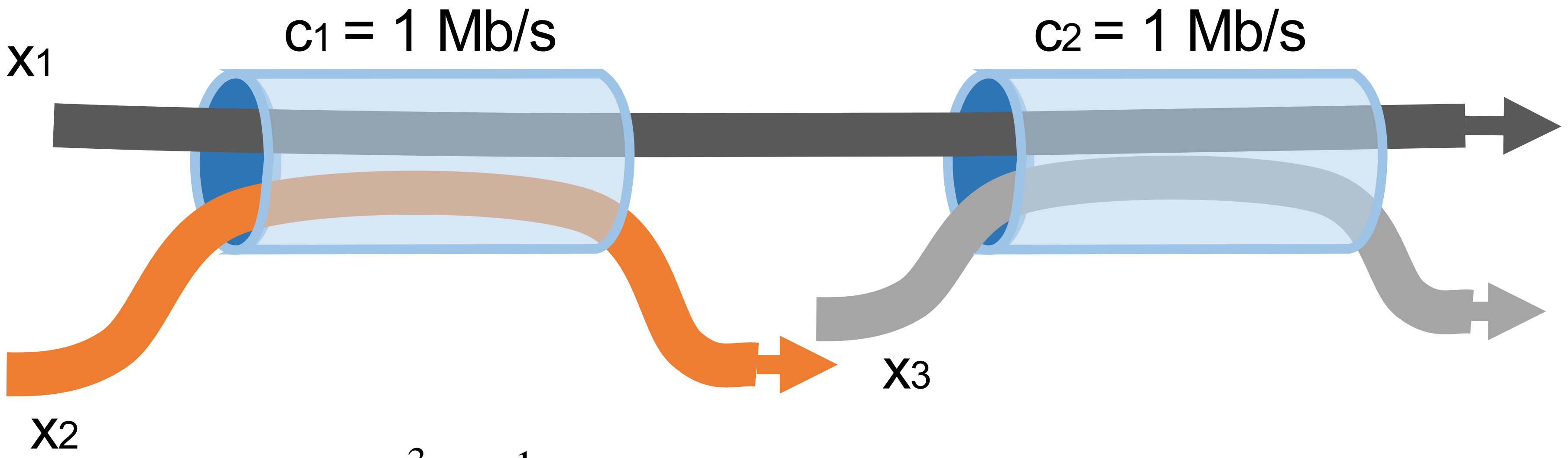


- maximize $\log(x_1) + \log(x_2) + \log(x_3)$
- subject to: $x_1 + x_2 \leq 1$

  $x_1 + x_3 \leq 1$

$$\Rightarrow \left\{ \begin{array}{l} x_1^\star = 1/3 \text{ Mb/s} \\ x_2^\star = 2/3 \text{ Mb/s} \\ x_3^\star = 2/3 \text{ Mb/s} \end{array} \right.$$

NUM Example 3: "α-fairness"



- maximize $\sum_{i=1}^{3} \frac{x_i^{1-\alpha}}{1-\alpha}$
- subject to: $x_1 + x_2 \leq 1$

  $x_1 + x_3 \leq 1$

- $\alpha \geq 0$ (a constant)

alpha	Objective
α = 0	Tput Maximization
α = 1	Proportional Fairness
α ➔infty	Max-min Fairness

What utility function does TCP have?

- Reverse engineering
  - TCP Reno
    - Utilities are arctan(x)
    - Prices are end-to-end packet loss
  - TCP Vegas
    - Utilities are log(x), i.e., proportional fairness
    - Prices are end-to-end packet delays
- Forward engineering
  - Use decomposition to design new variants of TCP
  - E.g., TCP FAST
- Simplifications
  - Fixed set of connections, focus on equilibrium behavior, ignore feedback delays and queuing dynamics

Sources

1. Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., & Jacobson, V. (2016). BBR: Congestion-based congestion control. *Queue*, 14(5), 20-53.
2. BBR slides from Yuchung Cheng, Google
3. CUBIC slides from Sangtae Ha, University of Colorado, Boulder
4. NUM slides from Mohammad Alizadeh, MIT