Why Control **Delay**?

Kathleen Nichols
Pollere LLC
Understanding Latency 2.0
December 13, 2023

- Unlike other talks, this is focused on work from 2011-2012
- Most Internet traffic used Reno or Cubic
- No QUIC

Surprisingly, still some relevance
(and Bjørn requested it!)

AQMs, transport protocols, other elements in the packet path should focus on controlling delay:

1. Because Internet traffic is NOT poisson

- that delusion led to years of attempts to control queue lengths: delay cannot be inferred from queue length
- see refs at https://pollere.net/Codel.html, esp "A Rant on Queues"
- CoDel article and RFC 8289 say these things well so this talk is more motivational than explanatory

2. Because delay is what makes users tear their hair out

- delay is universal - the amount of delay users will accept has been studied
- delay has bounds - upper bounds before protocols start to die

Review (Neal covered this Tuesday)

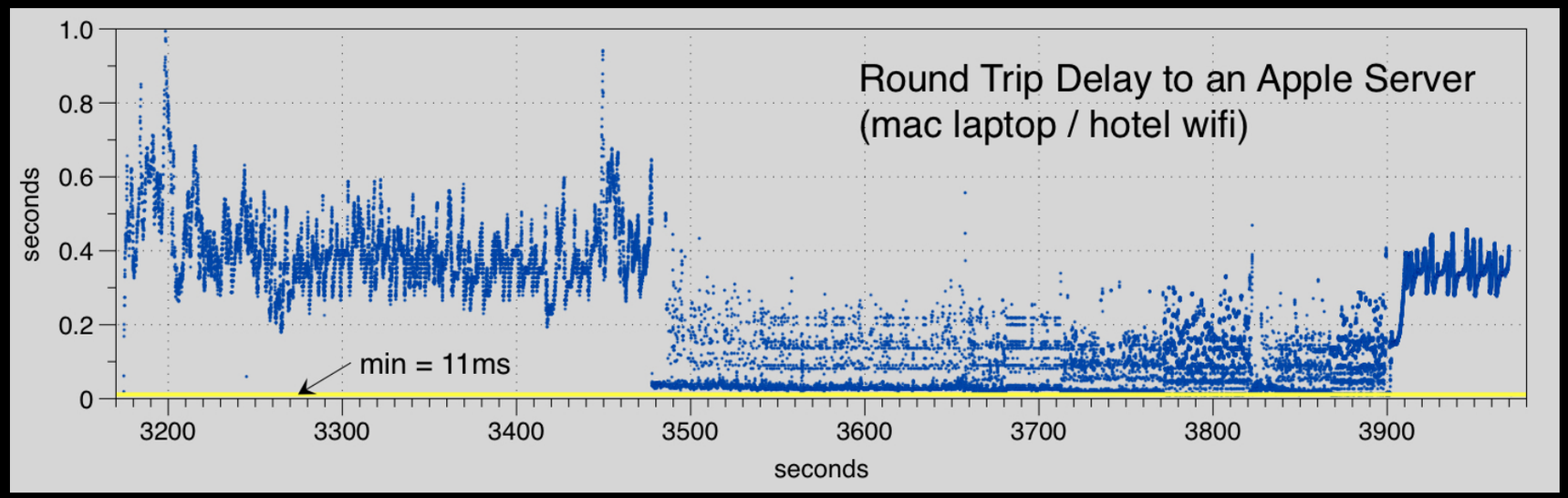
The "full buffer problem" mainly results from transport protocols trying to fill the "pipe" between source and destination by increasing the number of packets in flight until the sender gets an indication the pipe is full

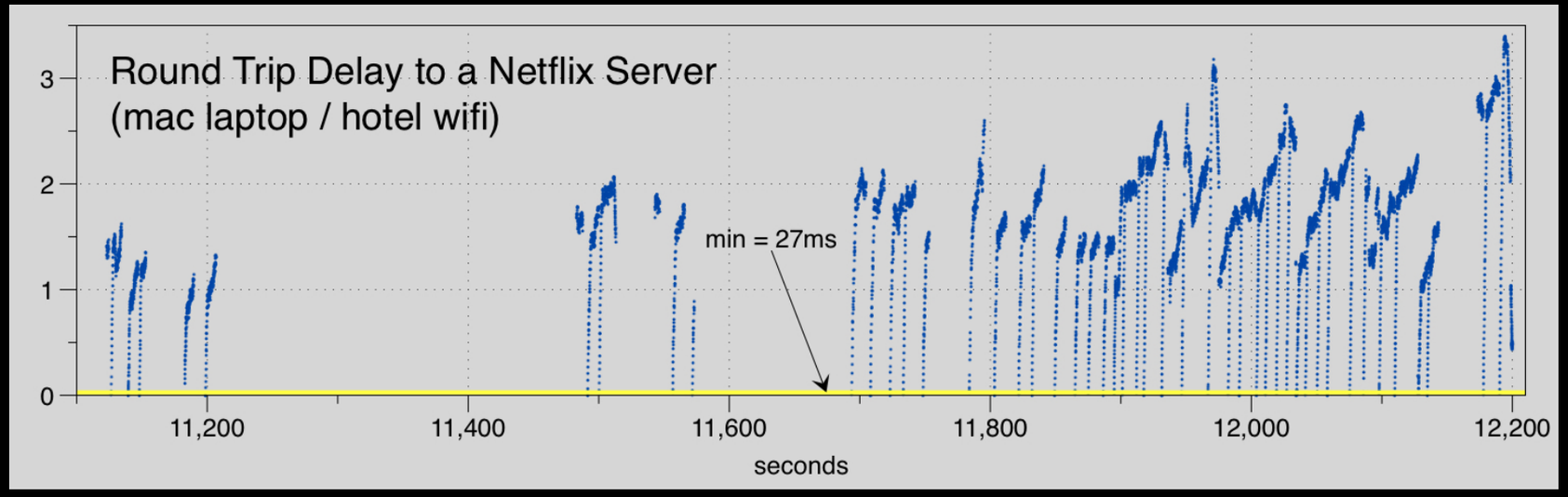
Unfortunately, the "pipe" in this context includes filling up all the buffers along the way, forming a persistent queue of packets

Persistent queues mean added delay (latency) that has no benefit, e.g., data isn't transferred at a higher rate, instead, full buffers leave no room for anyone else

Jim Gettys named the "full buffer problem": **bufferbloat**

The "full buffer problem" or *bufferbloat* in action

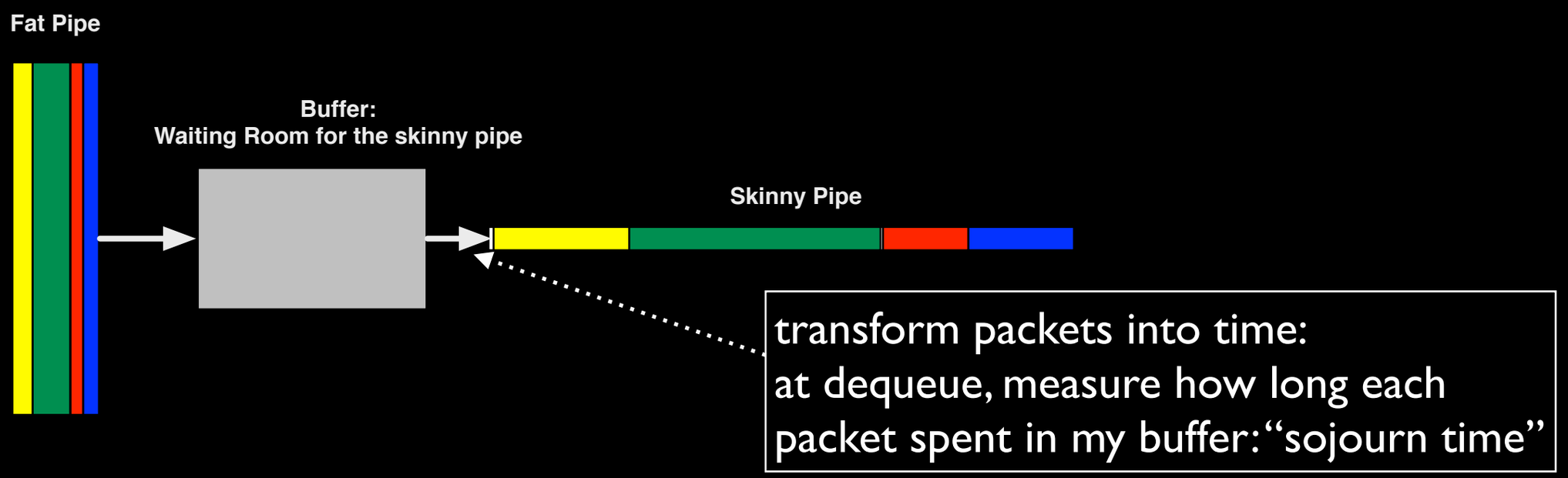




RTTs measured with pping ~2016

The Internet has packet buffers to handle the normal burstiness of statistically multiplexed networks - good

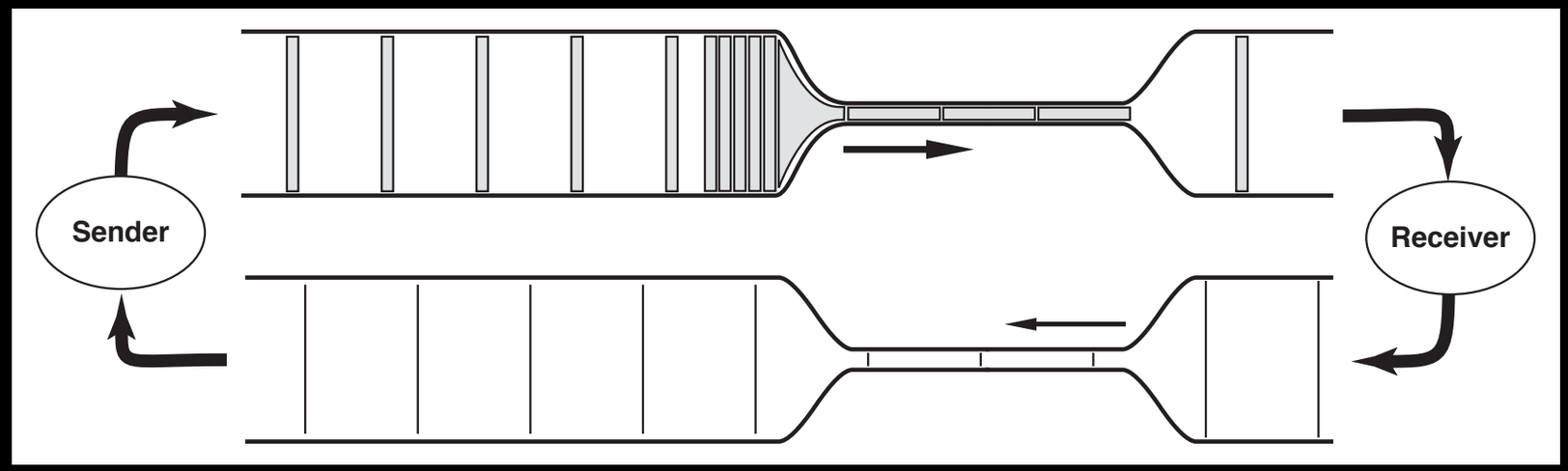
Persistent or standing queues of packets can develop at “fast to slow” transitions - bad



A 1500B packet takes 400 µs @ 30 Mbps, 67 µs @ 180 Mbps, 12 µs at 1 Gbps

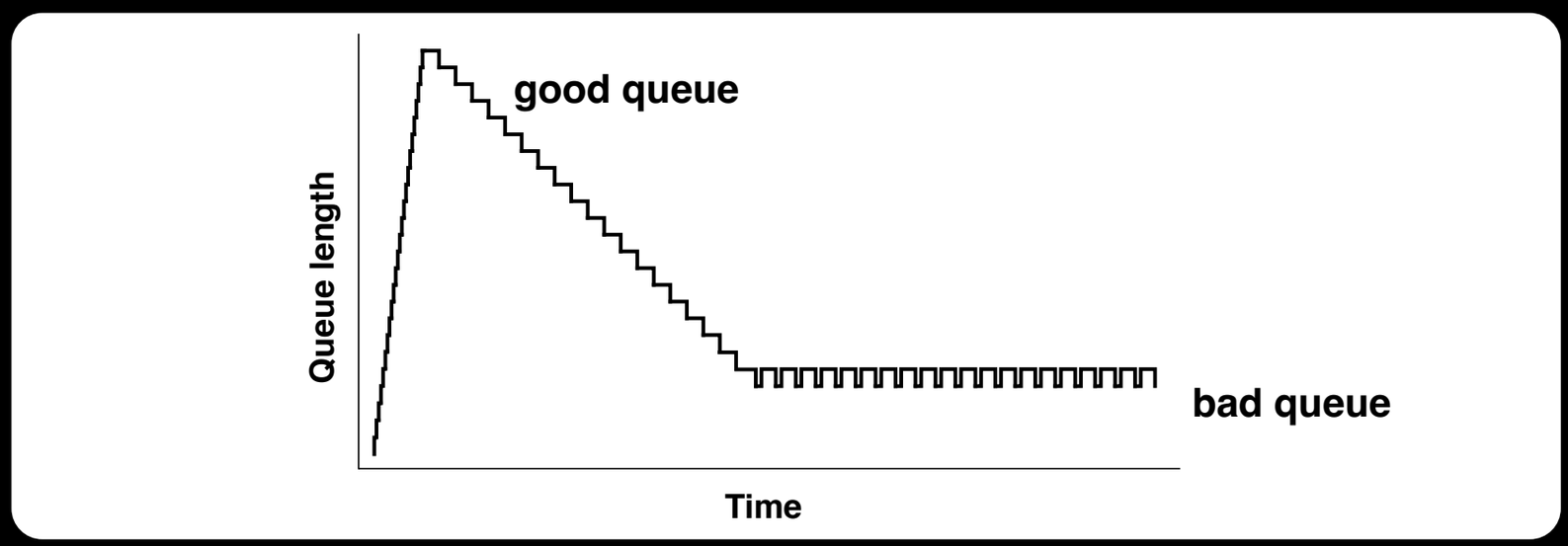
- 100 buffered packets in a “skinny pipe” of 180 Mbps adds 7ms delay; at 30 Mbps, 40ms delay
- The delay through the buffer experienced by dequeued packets is a measure that is both independent of line rates and packet sizes as well as being directly related to something we care about

How do we get standing queue?
Consider a TCP with a 25 packet window into a path with and intrinsic pipe size of 20 packets:

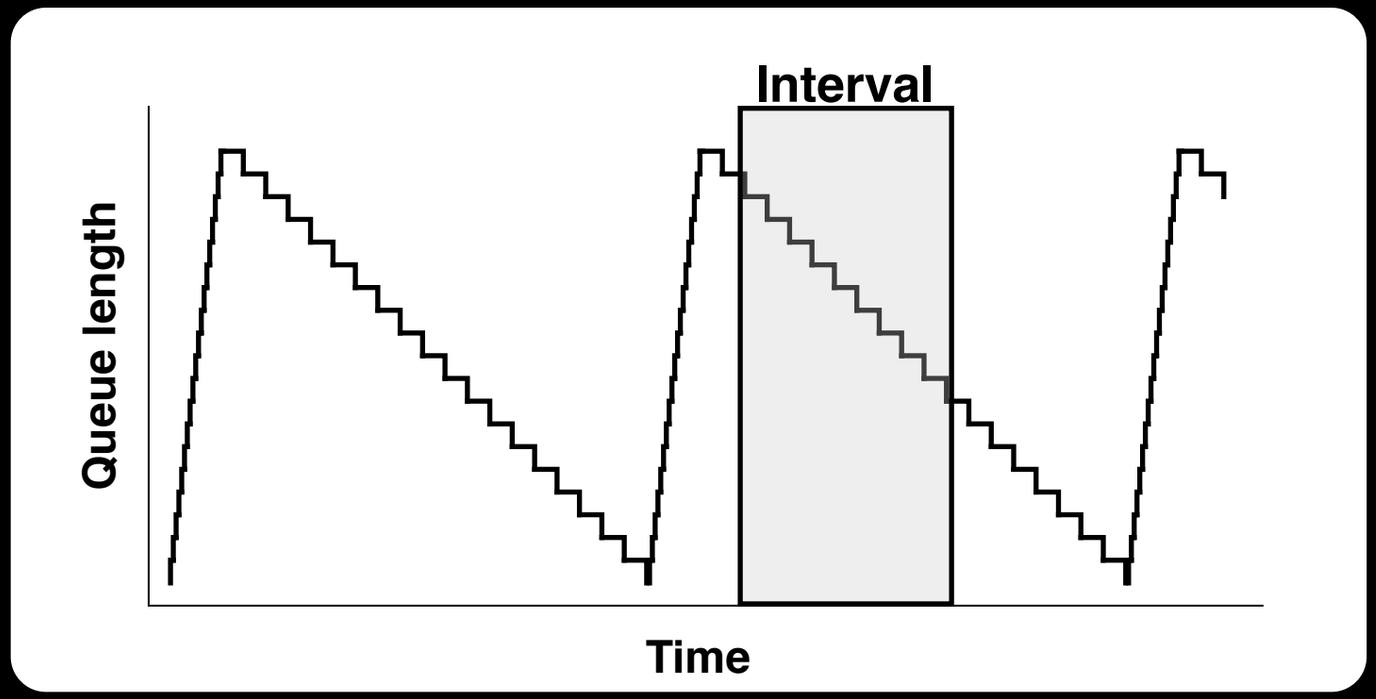


the classic Van Jacobson picture

After the initial burst of packets, the buffer settles into to a 5 (+/-1) packet standing queue:



CoDel's approach to tracking "bad queue" can be summarized as:
"good queue goes away in ~RTT, bad queue hangs around"



Using observed (sojourn) delay experienced by dequeued packets

- ➡ minimum delay during a sliding window interval measures bad queue…
- ➡ … as long as window is at least an RTT wide
- ➡ this delay that "hangs around" is persistent queue

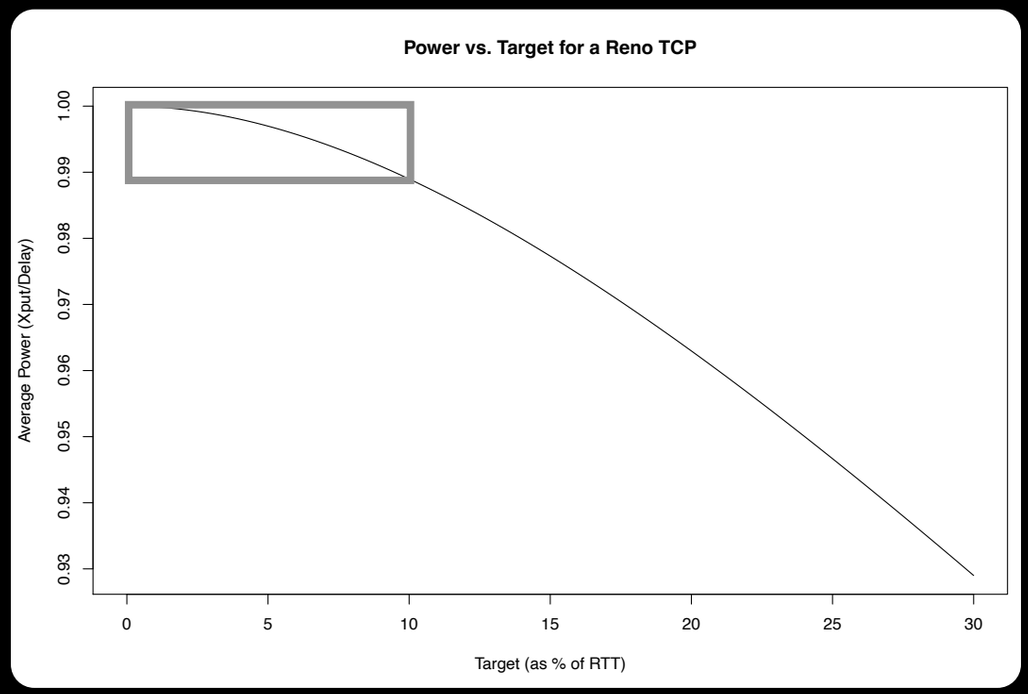
"Interval is loosely related to RTT since it is chosen to give endpoints time to react without being so long that response times suffer." - CoDel CACM paper

- CoDel's default *interval* of 100 ms was chosen for continental Internet-wide traffic (not data center)
- CDNs mean intrinsic RTTs are frequently under 20 ms; should we change the default?
  - A too-short interval can be disastrous, though a small number of long RTT flows in a well-mixed bottleneck should be okay
  - A too-long interval means it takes longer for the dropping to ramp up

Target sojourn time: how low should we go?

- In the sliding interval, the smallest observed sojourn time needs to be under a *target*
- For a single transport connection, if at least one packet has a zero sojourn time during the interval, then there are no bloated packets
- Real life complications: forcing the delay to zero reduces link utilization
- Shouldn’t drop a packet if there is nothing behind it

Kleinrock's network power measure of throughput/ delay applied to a single flow gave a ballpark



This suggested exploring **target** values in the range from 1 to 15 ms

In developing CoDel, this range was explored in simulation with 5 ms consistently the most robust

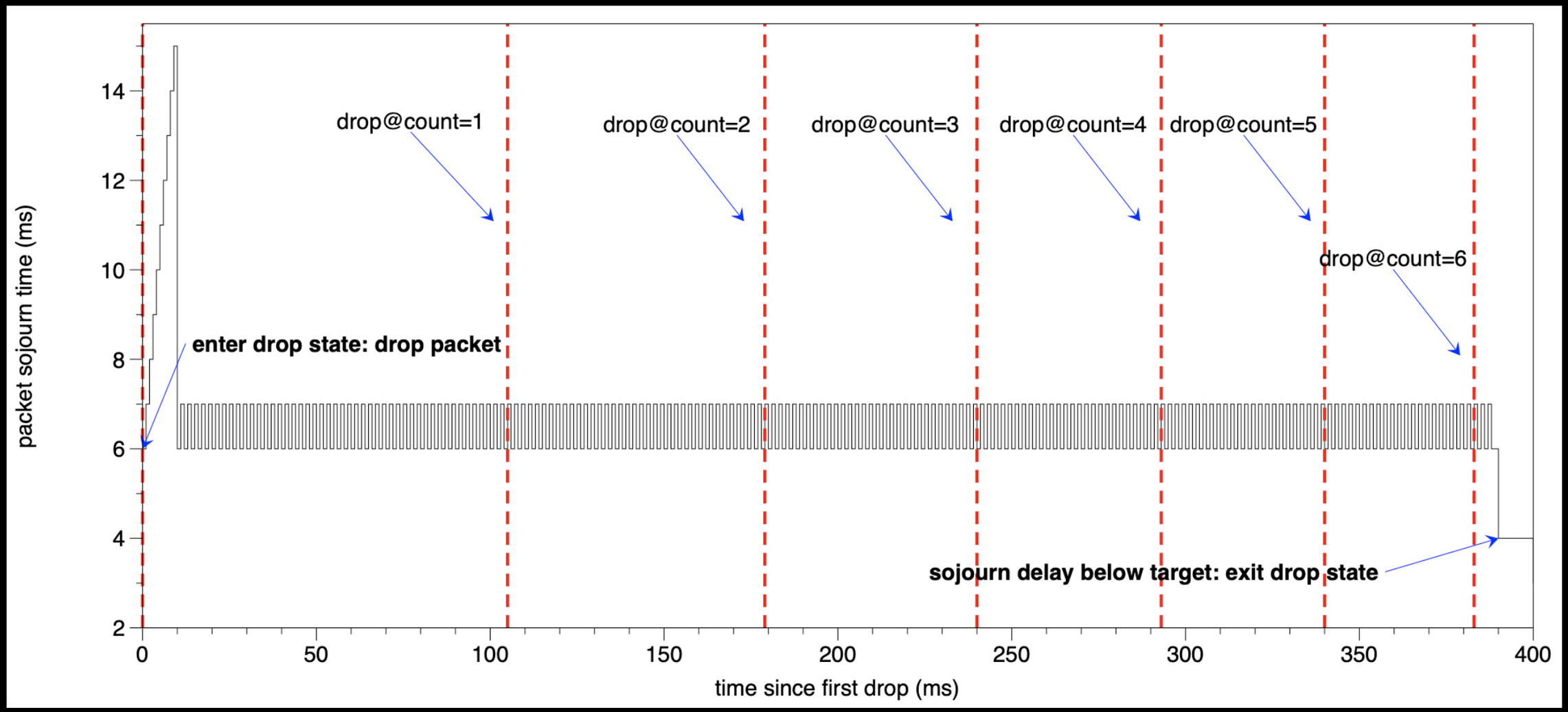
How often should we drop?

- The number of flows, their type and RTT is unknown so proceed with caution in order to "learn" what drop spacing gets the sojourn time under target at least once per interval
- When a persistent queue is detected, enter dropping state and drop a packet (unless it's the only one)
  - persistent queue: measured sojourn time goes over the target and no subsequent sojourn time goes below target for an interval
- next drop time is set ~interval+target
  - a sojourn time under target cancels the drop state
  - after drop decrease next drop time by square root of number of drops since drop state was entered

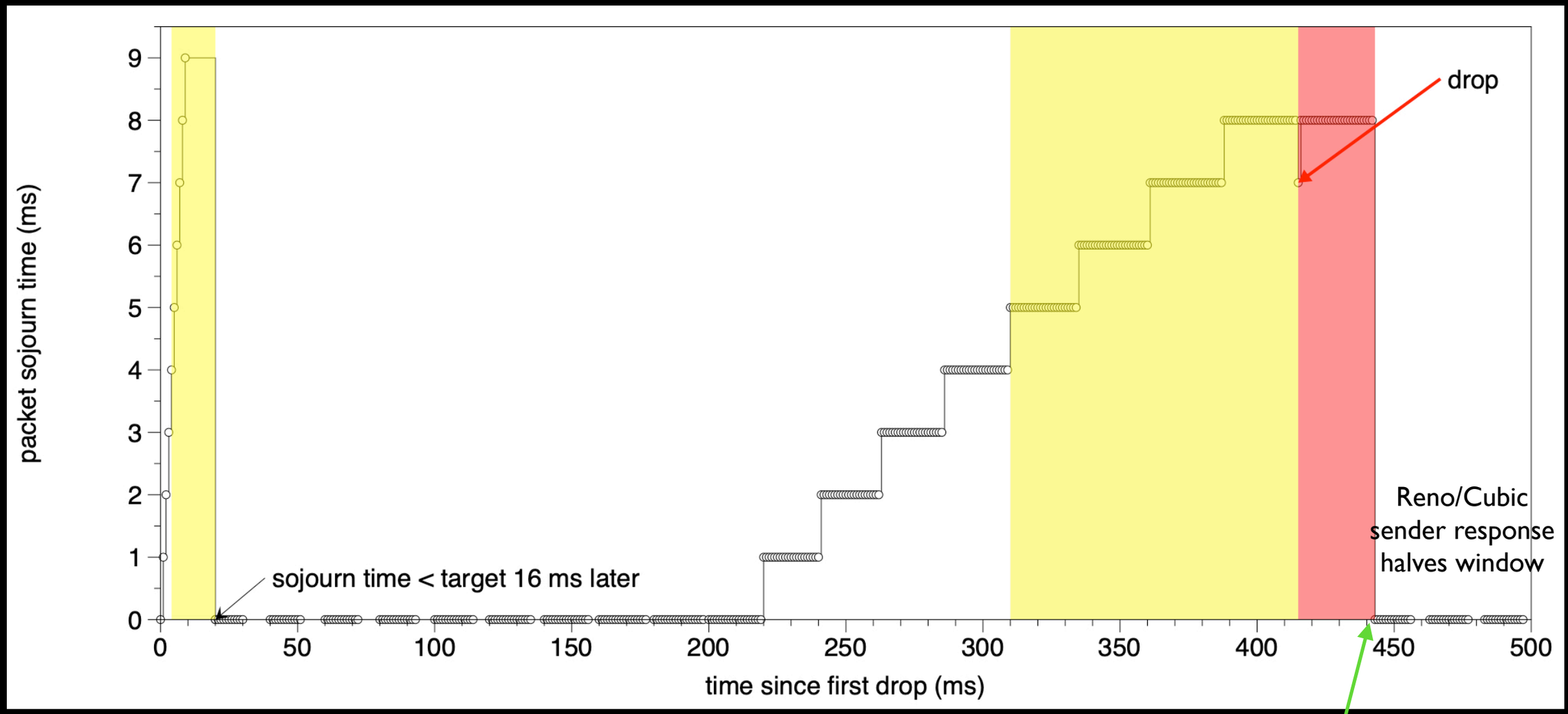
Consider an (apparently) unresponsive traffic mix at a buffer whose measured sojourn times have been >= target for an interval

- graph illustrates start of drop state and subsequent drops
- interval = 100 ms, target = 5 ms
- packet time on bottleneck = 1ms

#drops	next drop Δ
1	105 ms
2	74 ms
3	61 ms
4	53 ms
5	47 ms
6	43 ms



- single transport flow with intrinsic RTT of 20 ms, initial window 10 packets in a burst
- thereafter increases window by a packet every (current) RTT
- Interval = 100 ms, target = 5 ms, packet time on bottleneck = 1ms

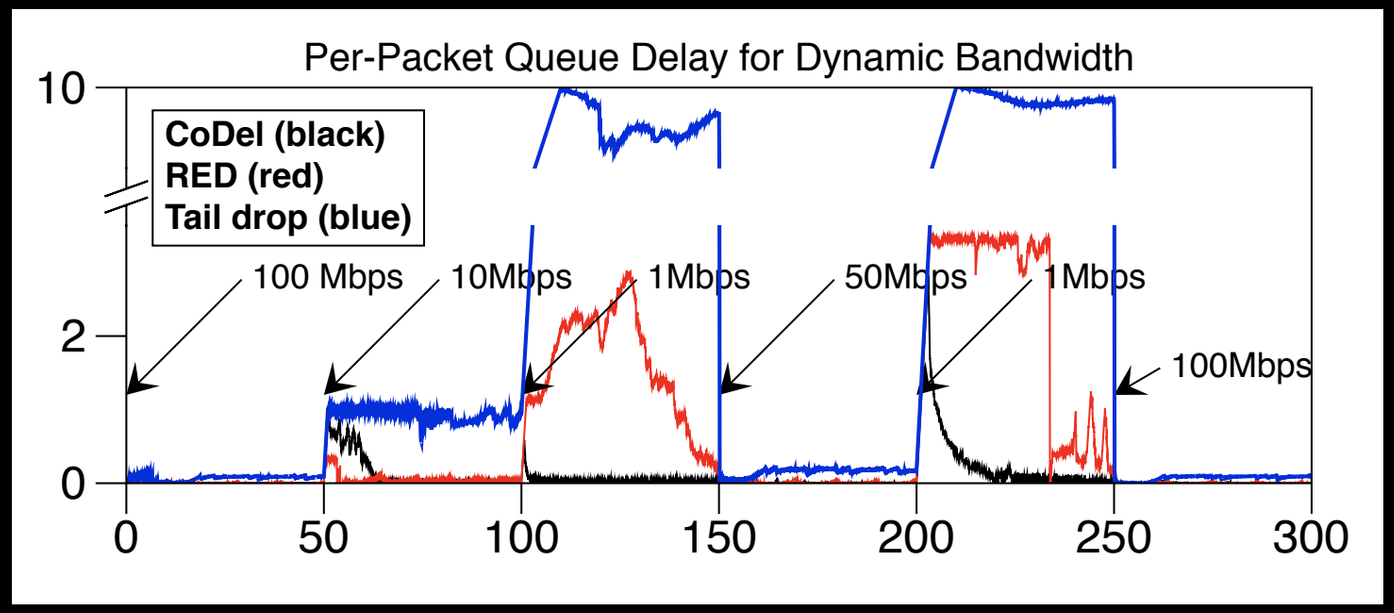


Shorter RTT flow means we "undercontrol"

A longer RTT flow might receive multiple drops before it can respond

➡ reality could be a mixture of flows and RTTs

Cool things about sojourn time as a delay metric: dynamic link

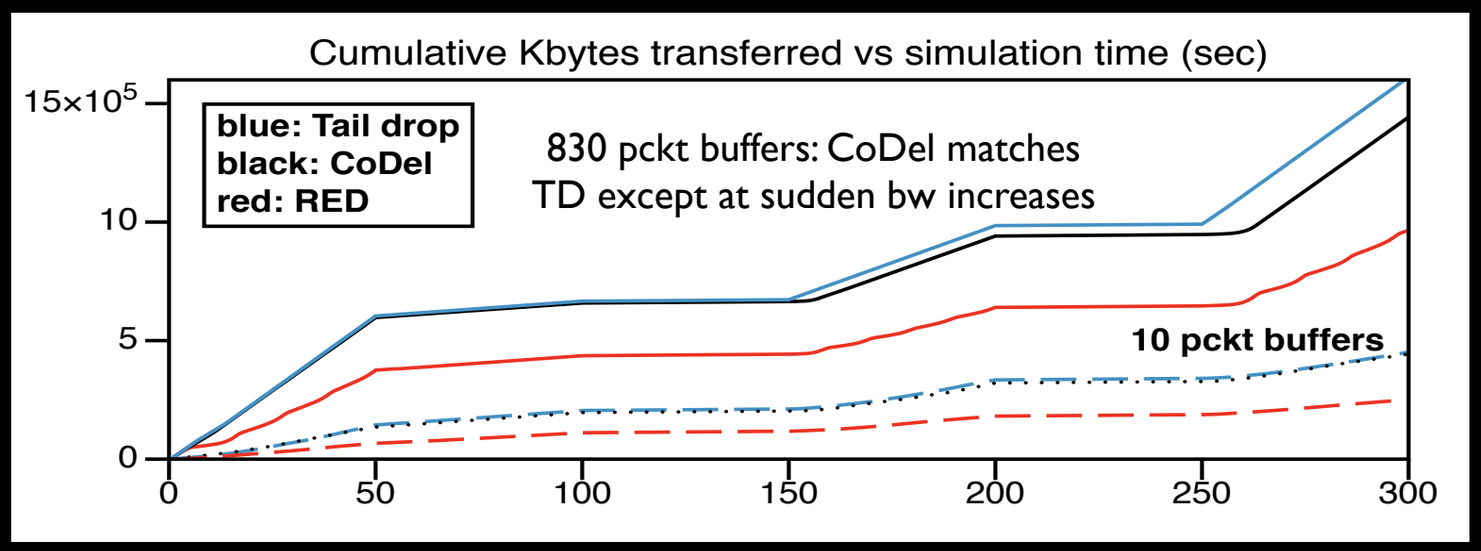


Experimentally duplicated:
https://reproducingnetworkresearch.wordpress.com/2012/06/06/solving-bufferbloat-the-codel-way/

From simulations with rate changing link, buffer size of 830 packets (4 FTPs, 5 packmime connections/sec)

Buffer size is 830 packets (BDP@100Mbps)

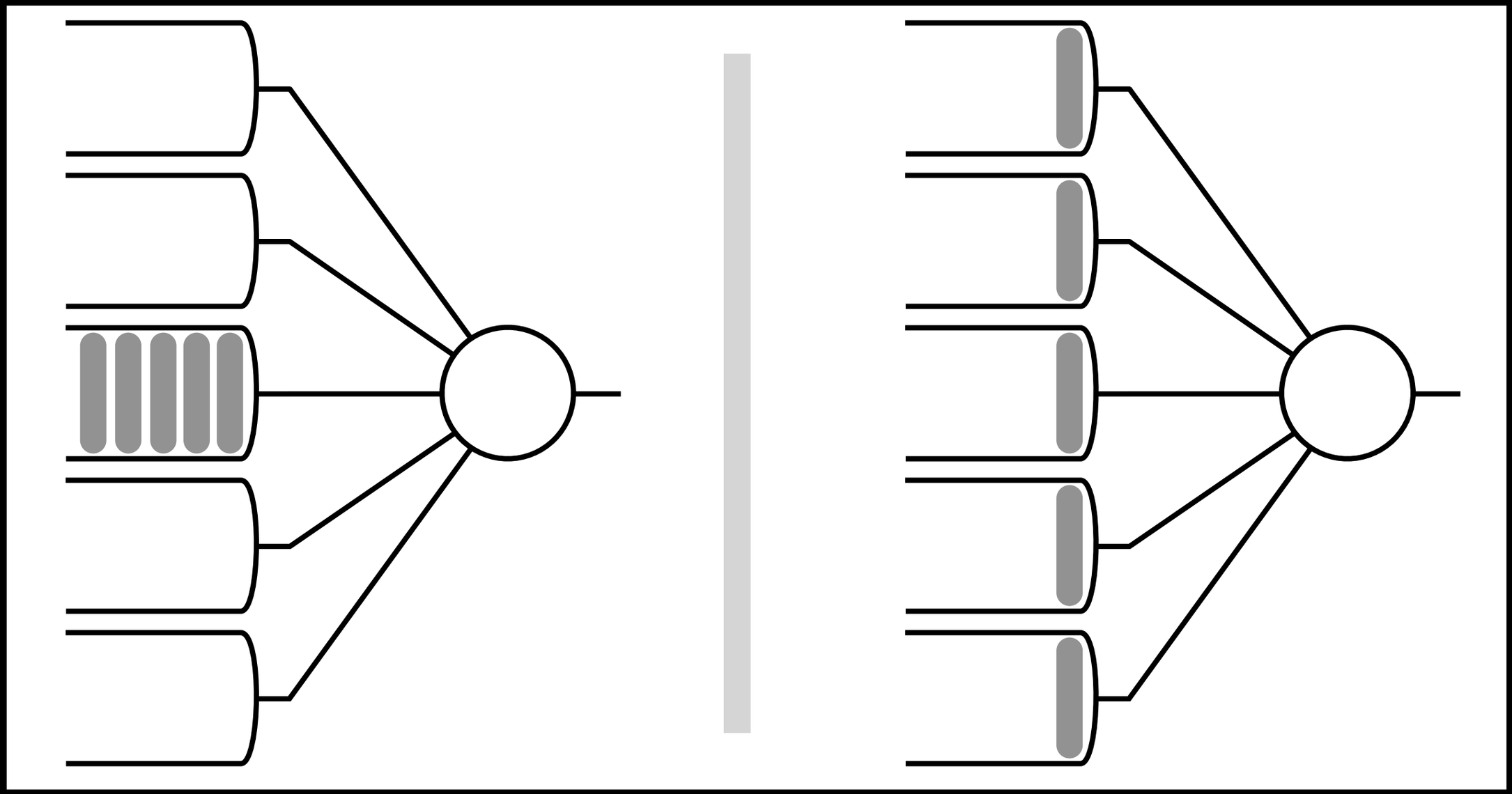
CoDel adapts to rate changes in 100ms



The throughput line for undersized buffer of 10 packets is about 75% less than with the larger buffer

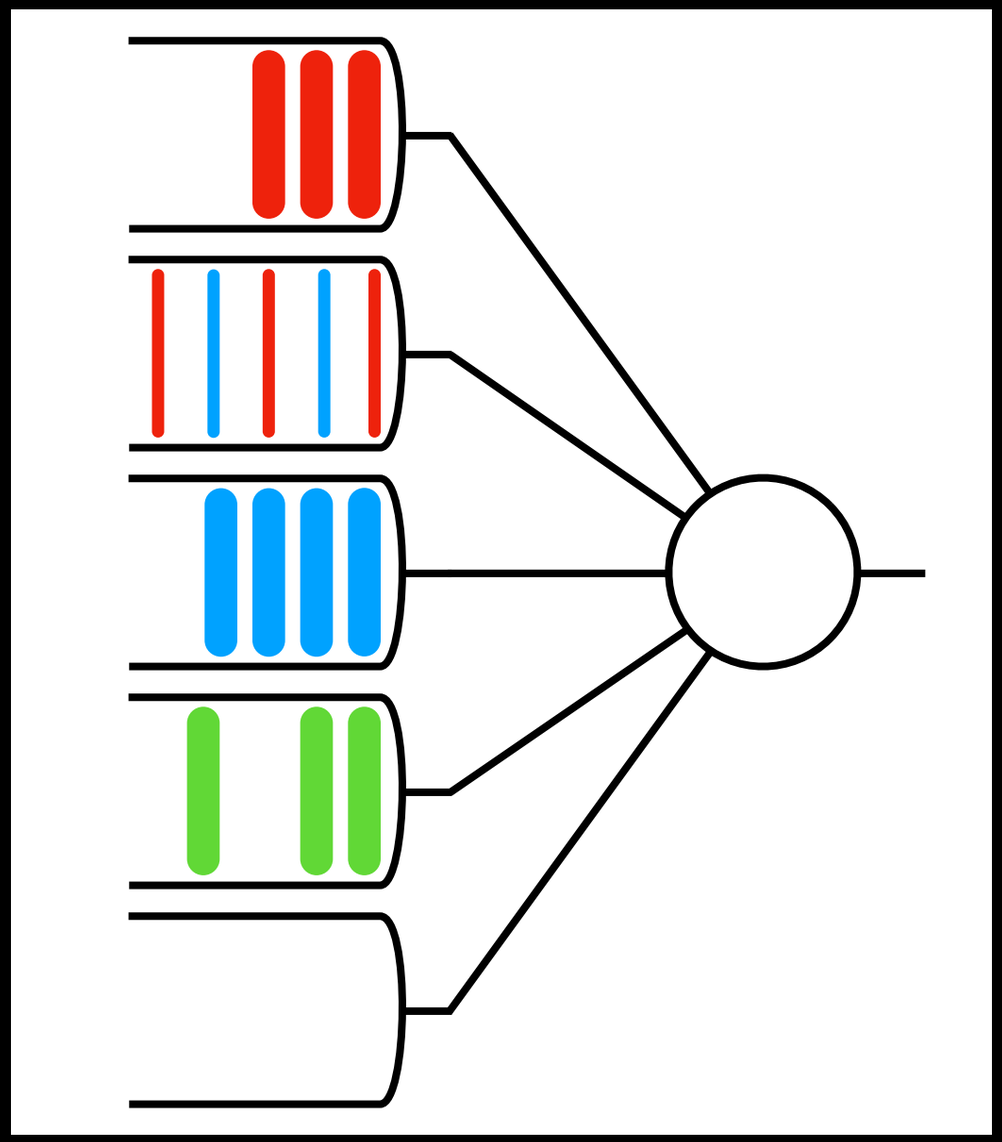
And CoDel has the same throughput as tail drop but only 2.7ms median delay, 5ms 75th percentile

Cool things about sojourn time as a delay metric: multi-queue link



Sojourn time doesn’t care how many queues there are

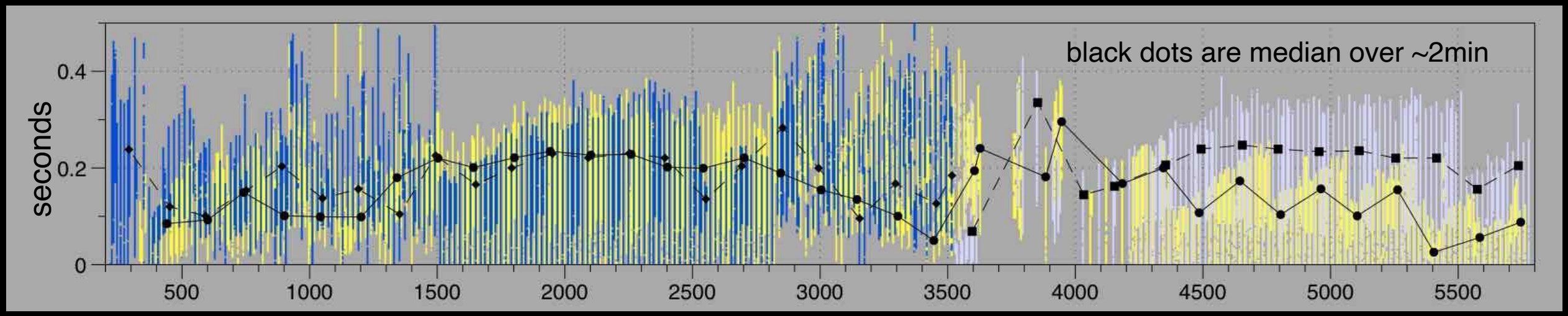
Multiple queues can be employed to mitigate ack compression and unfair treatment of shorter flow (see fq_codel)



Smaller packets avoid waiting behind larger

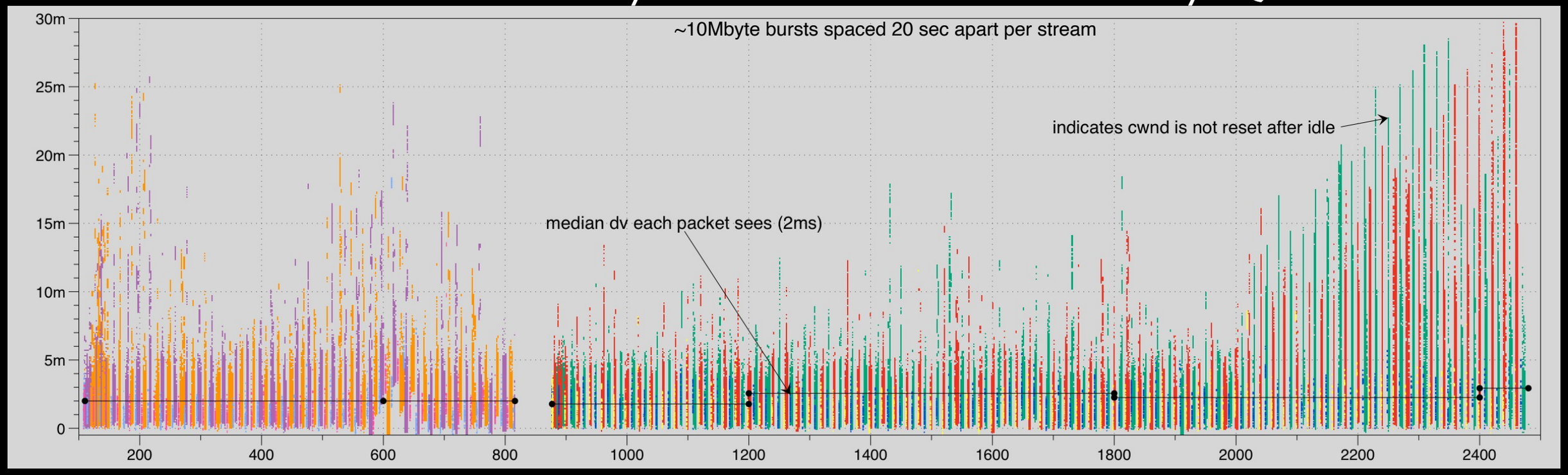
Separating large flows improves fairness

Netflix video on 30 Mbps cable link: captured on home network 03.20.16
Delay variation: server-to-capture point (RTD is ~14.5ms). These are interleaved 3.2 Mbps streams
A delay-aware transport like BBR would be nice. Failing that, CoDel.



Netflix video on 180 Mbps cable downlink: captured 09.11.16 (RTD still ~14.5ms)

Bursts cause shorter delays now but still need a burst-friendly AQM



Warning Label

Actual details of CoDel and fq_codel have evolved over time, new transports should never activate CoDel drop state

This talk used the original approach and was meant to give some intuition on CoDel and motivate measuring and controlling the thing we care about: delay (latency)