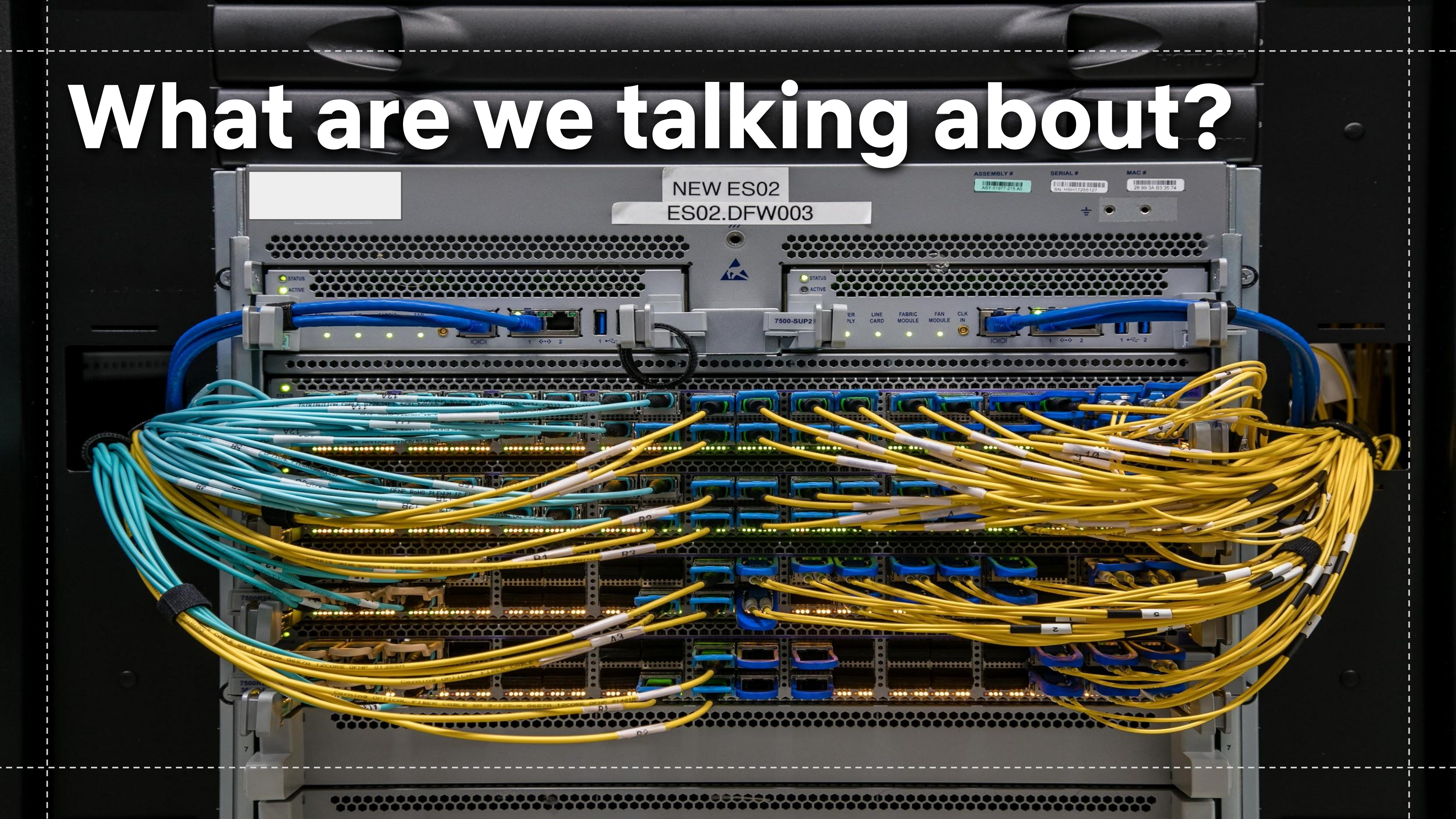
Updating the Theory of Buffer Sizing

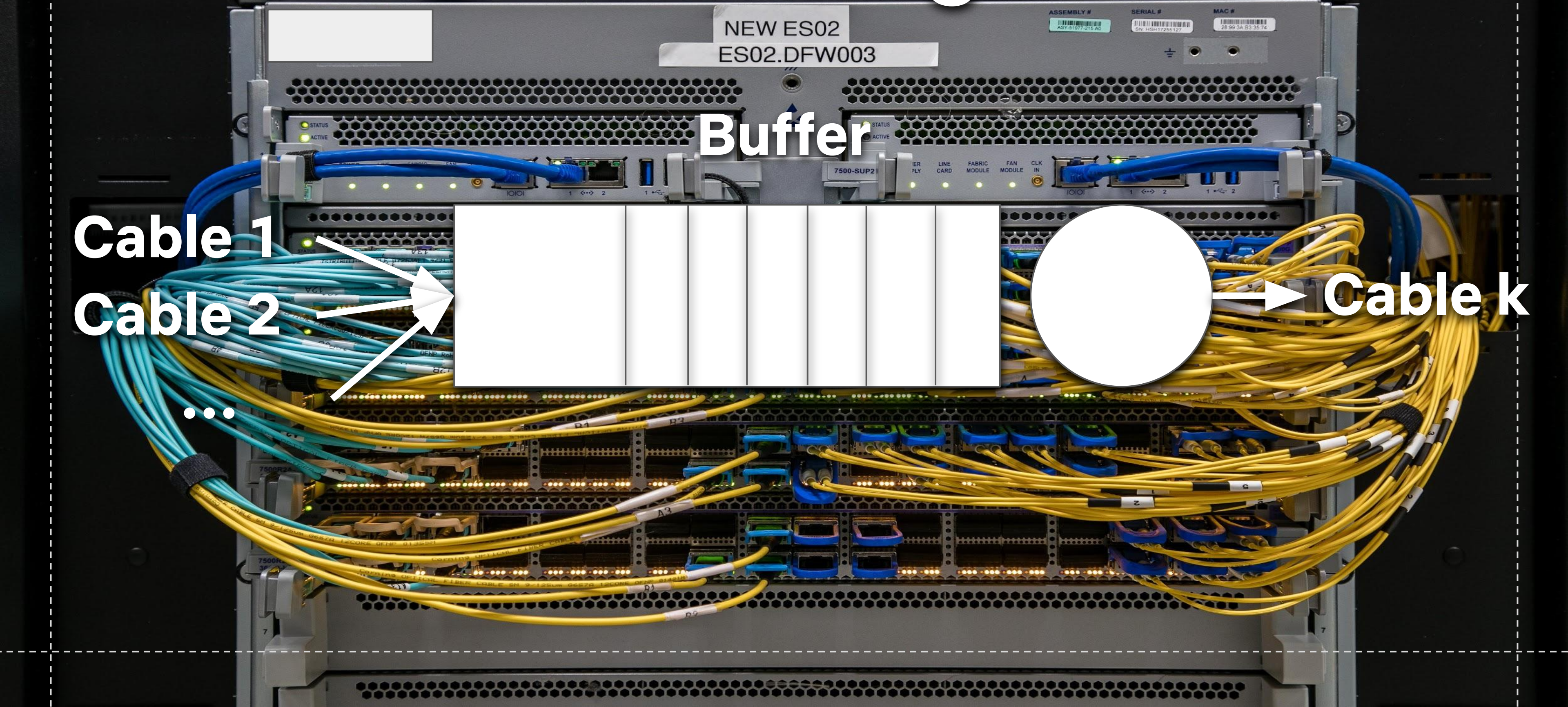
Bruce Spang, Serhat Arslan, Nick McKeown

IFIP Performance 2021

What are we talking about?



What are we talking about?



How big should a buffer be?

For applications:

- **Too big:** packets wait for too long
- **Too small:** can’t handle bursts

For router manufacturers:

- **Too big:** requires off-chip buffers
- **Too small:** people may not buy the router

How big should a buffer be?

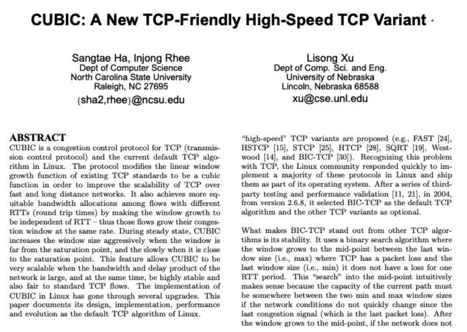
BDP=Bandwidth x Delay
# of packets inflight for full utilization

BDP: Jacobson 90, Villamizar and Song 1994

BDP/√n: Appenzeller, McKeown, Keslassy 2004

Since 2004…

2006: Cubic replaced Reno as default Linux algorithm



CUBIC: A New TCP-Friendly High-Speed TCP Variant

Sangtae Ha, Injong Rhee
Dept of Computer Science
North Carolina State University
Raleigh, NC 27695
{sha2,rhee}@ncsu.edu

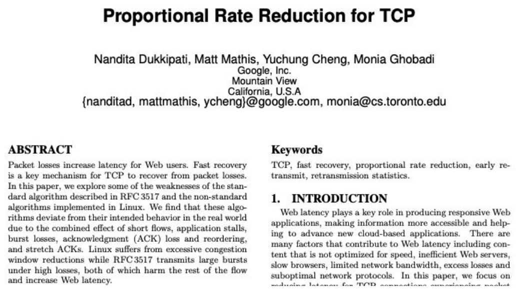
Lisong Xu
Dept of Comp. Sci. and Eng.
University of Nebraska
Lincoln, Nebraska 68588
xu@cse.unl.edu

ABSTRACT
CUBIC is a congestion control protocol for TCP (transmission control protocol) and the current default TCP algorithm in Linux. The protocol modifies the linear window growth function of existing TCP standards to be a cubic function in order to improve the scalability of TCP over fast and long distance networks. It also achieves more equitable bandwidth allocations among flows with different RTTs (round trip times) by making the window growth to be independent of RTT – thus those flows grow their congestion window at the same rate. During steady state, CUBIC increases the window size aggressively when the window is far from the saturation point, and the slowly when it is close to the saturation point. This feature allows CUBIC to be very scalable when the bandwidth and delay product of the network is large, and at the same time, be highly stable and also fair to standard TCP flows. The implementation of CUBIC in Linux has gone through several upgrades. This paper documents its design, implementation, performance and evolution as the default TCP algorithm of Linux.

"high-speed" TCP variants are proposed (e.g., FAST [24], HSTCP [15], STCP [25], HTCP [28], SQRT [19], Westwood [14], and BIC-TCP [30]). Recognizing this problem with TCP, the Linux community responded quickly to implement a majority of these protocols in Linux and ship them as part of its operating system. After a series of third-party testing and performance validation [11, 21], in 2004, from version 2.6.8, it selected BIC-TCP as the default TCP algorithm and the other TCP variants as optional.

What makes BIC-TCP stand out from other TCP algortihms is its stability. It uses a binary search algorithm where the window grows to the mid-point between the last window size (i.e., max) where TCP has a packet loss and the last window size (i.e., min) it does not have a loss for one RTT period. This "search" into the mid-point intuitively makes sense because the capacity of the current path must be somewhere between the two min and max window sizes if the network conditions do not quickly change since the last congestion signal (which is the last packet loss). After the window grows to the mid-point, if the network does not

2011: PRR breaks usual BDP argument for Reno



Proportional Rate Reduction for TCP

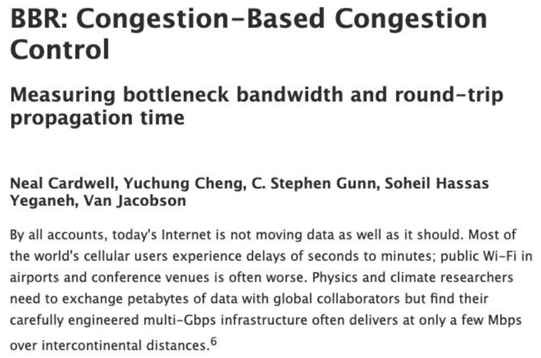
Nandita Dukkipati, Matt Mathis, Yuchung Cheng, Monia Ghobadi
Google, Inc.
Mountain View
California, U.S.A
{nanditad, mattmathis, ycheng}@google.com, monia@cs.toronto.edu

ABSTRACT
Packet losses increase latency for Web users. Fast recovery is a key mechanism for TCP to recover from packet losses. In this paper, we explore some of the weaknesses of the standard algorithm described in RFC 3517 and the non-standard algorithms implemented in Linux. We find that these algorithms deviate from their intended behavior in the real world due to the combined effect of short flows, application stalls, burst losses, acknowledgment (ACK) loss and reordering, and stretch ACKs. Linux suffers from excessive congestion window reductions while RFC 3517 transmits large bursts under high losses, both of which harm the rest of the flow and increase Web latency.

Keywords
TCP, fast recovery, proportional rate reduction, early retransmit, retransmission statistics.

1. INTRODUCTION
Web latency plays a key role in producing responsive Web applications, making information more accessible and helping to advance new cloud-based applications. There are many factors that contribute to Web latency including content that is not optimized for speed, inefficient Web servers, slow browsers, limited network bandwidth, excess losses and suboptimal network protocols. In this paper, we focus on

2016: BBR introduced



BBR: Congestion-Based Congestion Control

Measuring bottleneck bandwidth and round-trip propagation time

Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, Van Jacobson

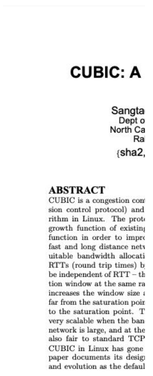
By all accounts, today's Internet is not moving data as well as it should. Most of the world's cellular users experience delays of seconds to minutes; public Wi-Fi in airports and conference venues is often worse. Physics and climate researchers need to exchange petabytes of data with global collaborators but find their carefully engineered multi-Gbps infrastructure often delivers at only a few Mbps over intercontinental distances.[6]

Since 2004…

2006: Cubic replaced Reno as default

2011: PRR breaks usual BDP

2016: BBR introduced



CUBIC: A

Sangta
Dept o
North Ca
Ra
{sha2,

ABSTRACT
CUBIC is a congestion con
sion control protocol) and
rithm in Linux. The prot
growth function of existin
function in order to impr
fast and long distance net
uitable bandwidth allocati
RTTs (round trip times) b
be independent of RTT – th
tion window at the same ra
increases the window size
far from the saturation poi
to the saturation point. T
very scalable when the ban
network is large, and at the
also fair to standard TCP
CUBIC in Linux has gone
paper documents its desig
and evolution as the defaul

ngestion

d round-trip

Gunn, Soheil Hassas

as well as it should. Most of
onds to minutes; public Wi-Fi in
sics and climate researchers
llaborators but find their
en delivers at only a few Mbps

How big should a buffer be today?

Our results

Understanding how buffer sizes interact with choices made by TCP:

- Buffer size for full utilization for modern TCP implementations (PRR, Cubic, BBR, etc…)
- Relationship between buffer size and utilization

Buffers can be made smaller by making better choices

Buffer requirements for a single flow

Algorithm	Full Utilization	90% Utilization
Reno	BDP	0.80 BDP
Cubic	0.42 BDP	0.28 BDP
BBR	0.25 BDP	0.15 BDP
Scalable	0.14 BDP	0.03 BDP

Multiple Reno flows [Appenzeller, McKeown, Keslassy 2004]

If buffer is $\geq$ BDP/$\sqrt{n}$ and [conditions apply] then link will be fully utilized

Conditions: TCP sends data at a rate that is

1. Uniformly distributed between c_1 BDP/n and c_2 BDP/n
2. Independent

Multiple Reno flows [SAM21]

If n flows share a link and [conditions apply] then:

1. If buffer is $\geq BDP/\sqrt{n}$, link will be fully utilized
2. Utilization is at least $1-\Omega(1/\sqrt{n})$, independent of buffer size

Conditions:

1. **Fair:** flows send roughly same amount of data
2. **Desynchronized:** only a few flows decrease windows at same time

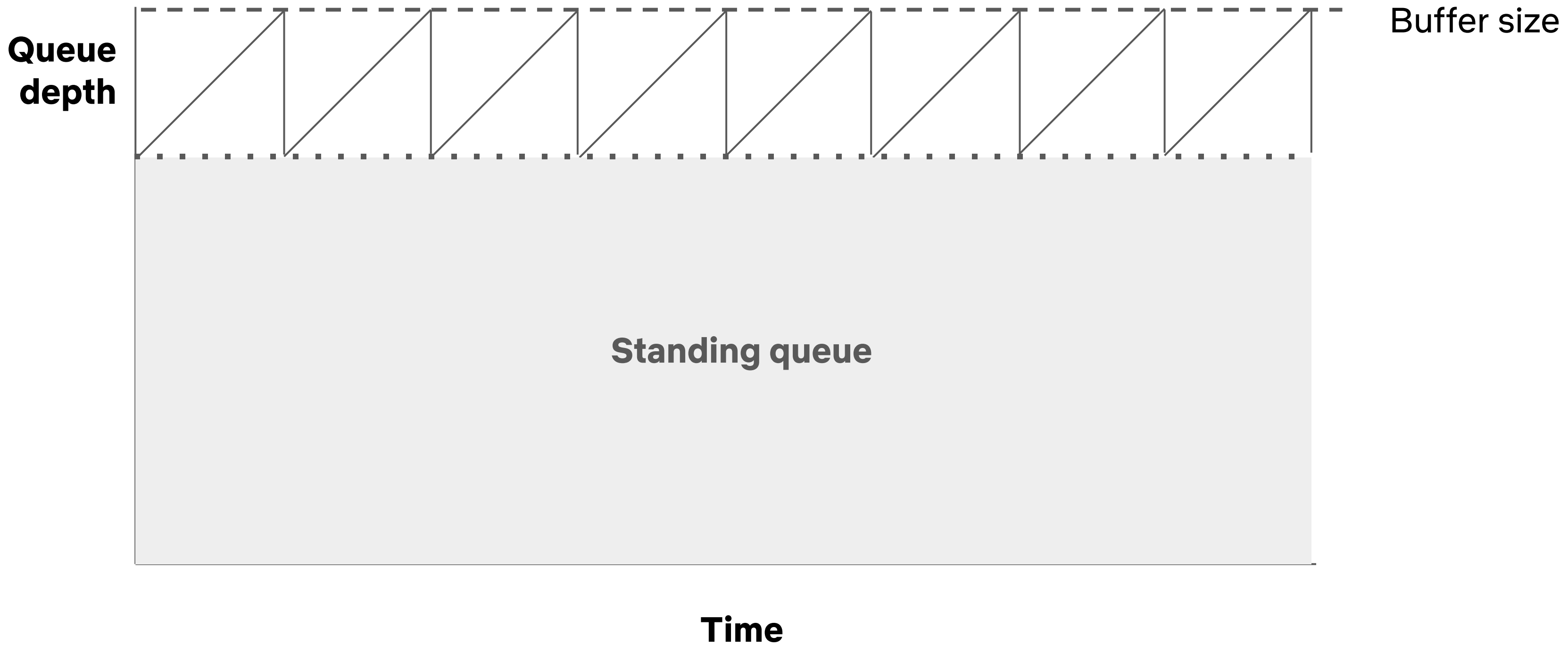
Buffer requirements for 10,000 Reno Flows

1. Full utilization if buffer ≥ BDP/100
2. Always have at least 1-1/100=99% utilization
   (independent of buffer size!)

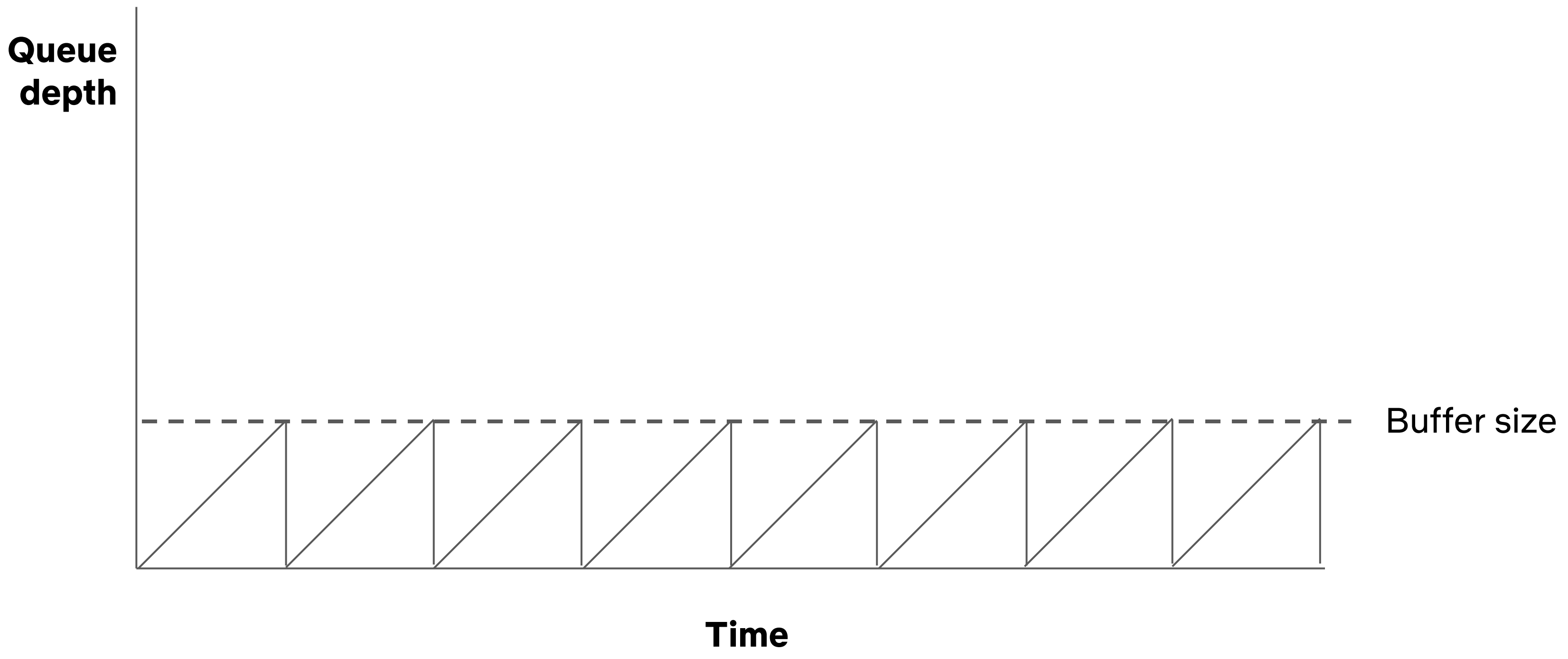
Why do we need the two conditions:

1. Fairness
2. No synchronization

Intuition: buffer only needs to handle variability

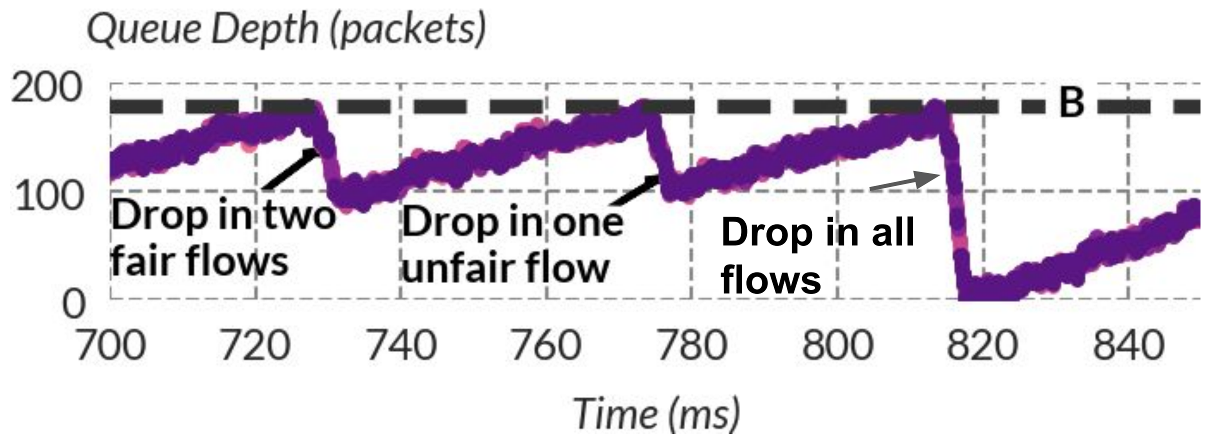


Intuition: buffer only needs to handle variability



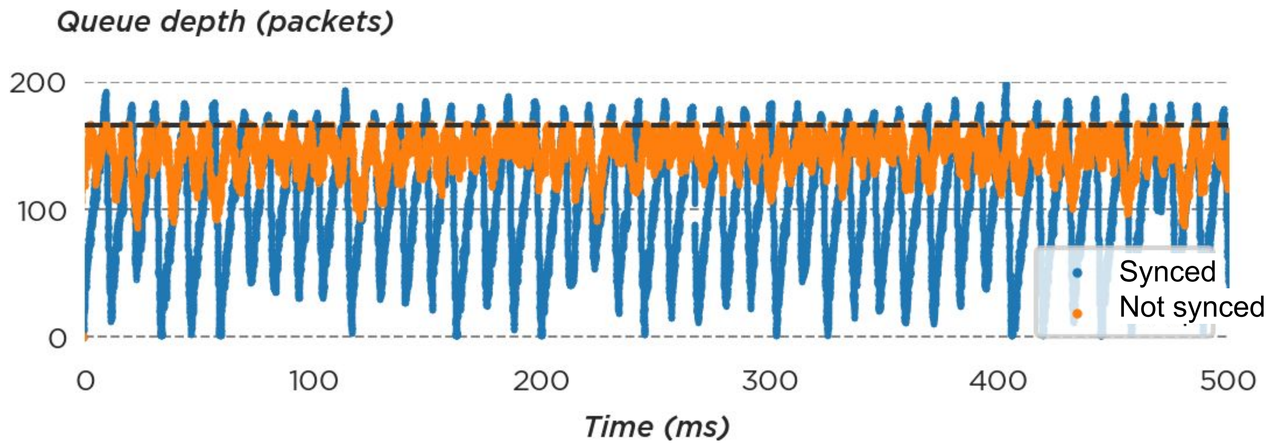
Unfairness increases queue variability

If a TCP flow has more data in flight, it will back off more, causing a larger drop in queue depth (and larger required buffer)

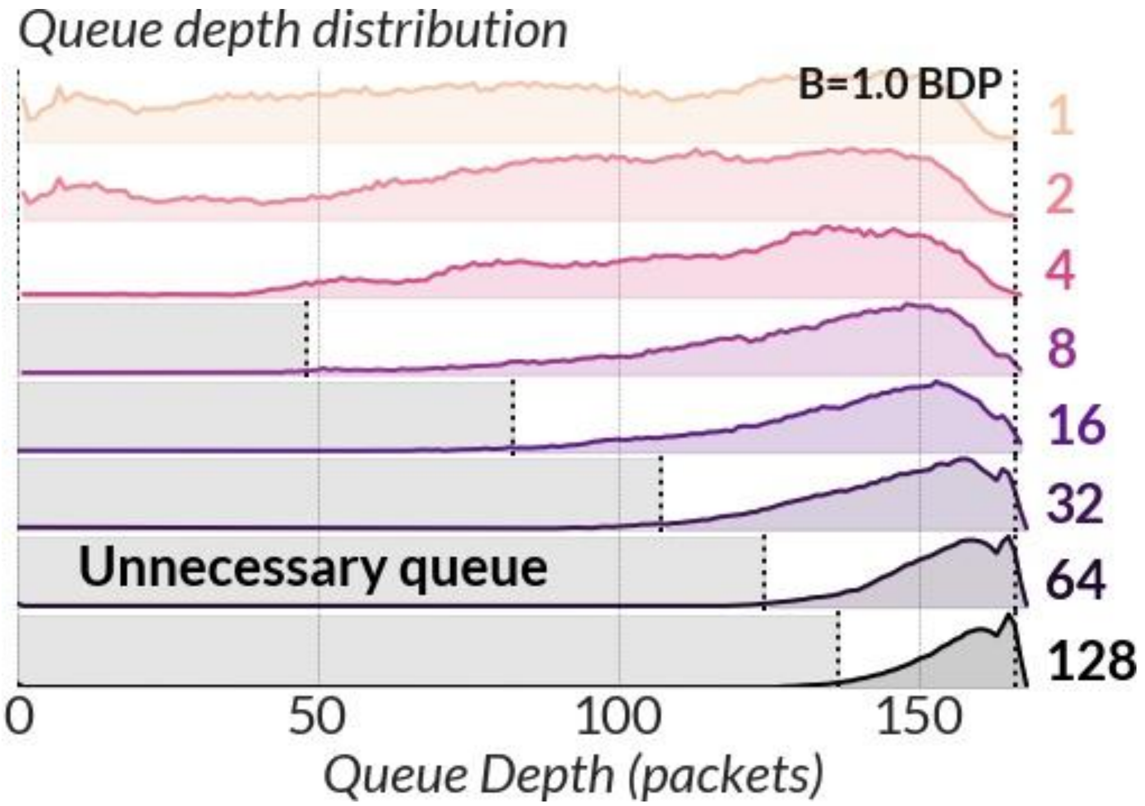


Synchronization increases queue variability

If everyone stops sending data at once, queues will fluctuate more



Queue variability follows √n rule in testbed



Adding randomness reduces synchronization

Can prove $\sqrt{n}$ results without synchronization condition for:

- BBR
- Reno variant which randomly decreases window

We *should* only need small buffers

Modern TCP requires smaller buffers than Reno

Relationship between buffers and utilization is a consequence of congestion control choices

Should be able to get away with buffers of 10-100 packets.

Lots more to understand with buffer sizing!

How are loss and fairness affected by buffer size, even for Reno?

How is application performance impacted by buffer size?

How big a buffer do we need in practice?