CSCI-1680

Congestion Control Mechanics

Nick DeMarinis

Based partly on lecture notes by Rodrigo Fonseca, David Mazières, Phil Levis, John Jannotti

Administrivia

- TCP Milestone I: Sign up for a meeting this week, if you haven't already!
- TCP gearup II TONIGHT (11/2) 5-7pm in CIT68 (+Zoom, +Recorded)
  - Any questions you have
  - Stuff for milestone II
  - How to test
- HW3: Out now, due next Wed => practice for milestone II

Warmup

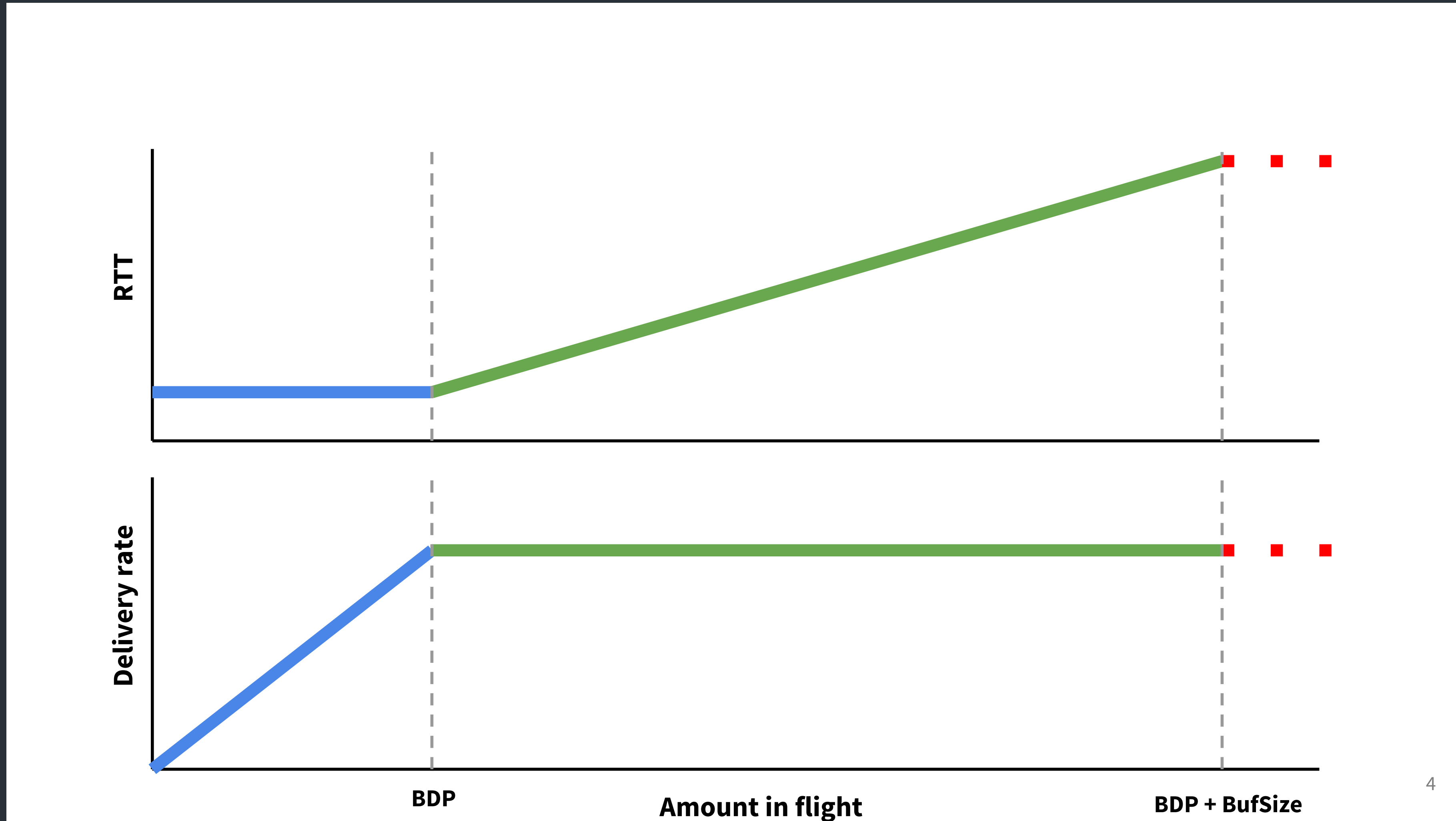
Which of the following contribute to <u>congestion</u>:

a. Packets queueing up at switches
b. High CPU usage on the receiver
c. Many TCP connections on the same link
d. Many UDP connections on the same link
e. Poor wifi connection on the sender

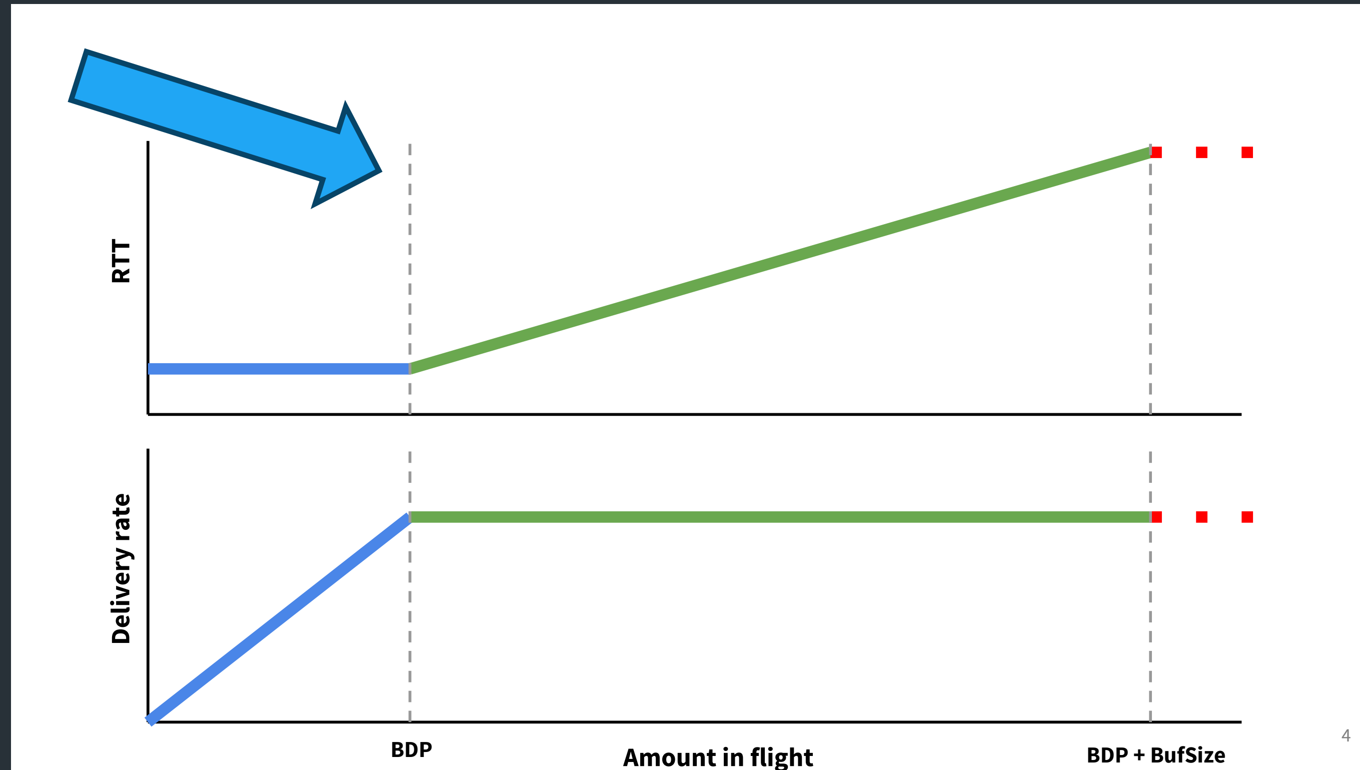
Flow control: making sure we don't overwhelm the receiver

Congestion control: making sure we don't overwhelm the network

Thinking about congestion



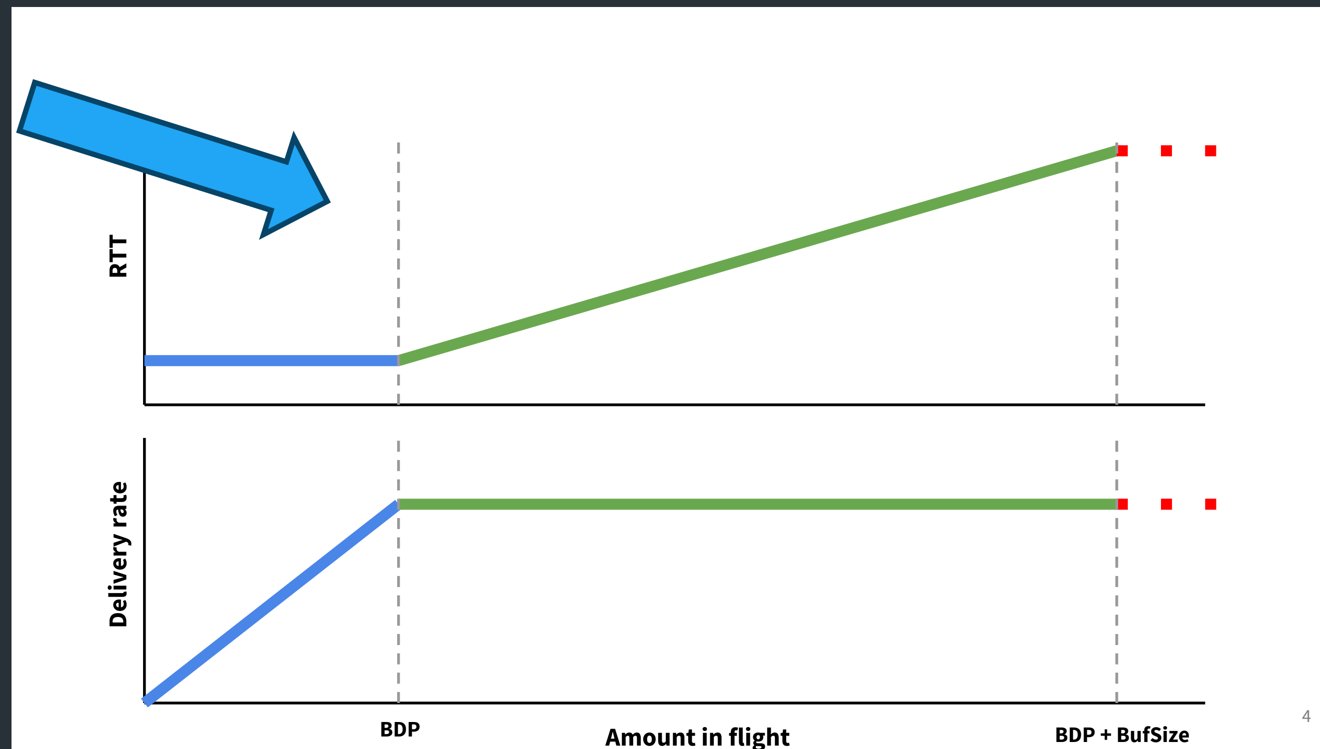
"BBR congestion control"



"BBR congestion control"

Bandwidth-delay product (BDP): maximum amount of data that can be in-transit on a network link at any given time

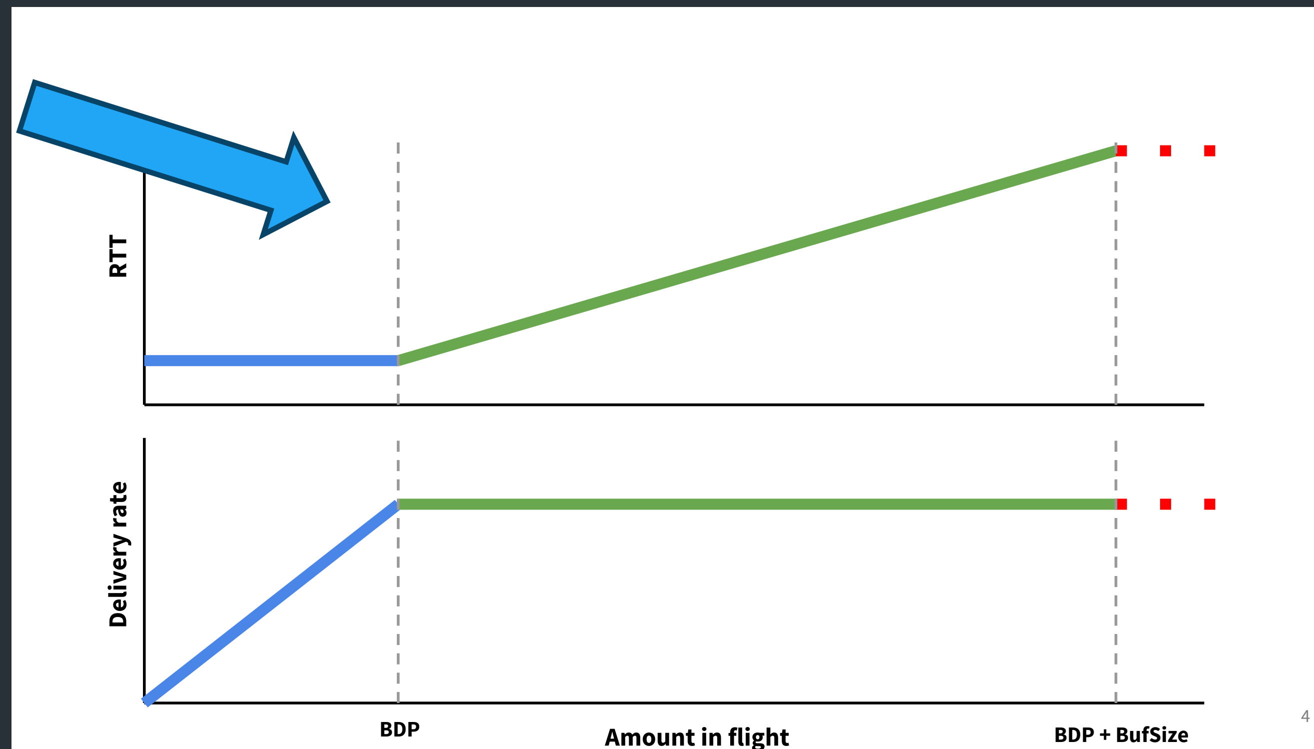
(Link capacity (bits/sec)) * (RTT (sec))
= (bytes)



"BBR congestion control"

Bandwidth-delay product (BDP): maximum amount of data that can be in-transit on a network link at any given time

(Link capacity (bits/sec)) * (RTT (sec))
= (bytes)
Eg. 1Gbps link * 1ms RTT = 125KiB BDP

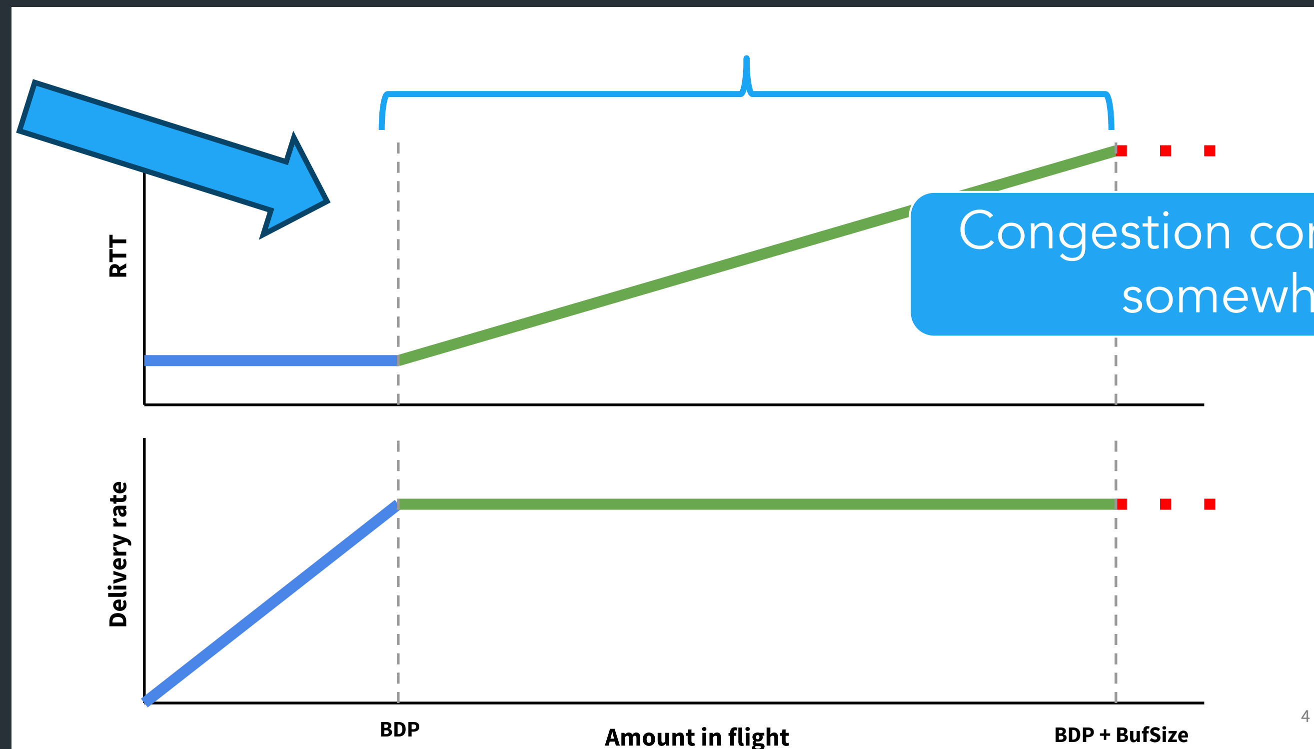


"BBR congestion control"

Bandwidth-delay product (BDP): maximum amount of data that can be in-transit on a network link at any given time

(Link capacity (bits/sec)) * (RTT (sec))
= (bytes)
Eg. 1Gbps link * 1ms RTT = 125KiB BDP

=> After exceeding BDP, network is queueing packets. After queues are full, packets getting dropped due to congestion.



Congestion control algorithms operate somewhere in this region

"BBR congestion control"

Bandwidth-delay product (BDP): maximum amount of data that can be in-transit on a network link at any given time

(Link capacity (bits/sec)) * (RTT (sec))
= (bytes)
Eg. 1Gbps link * 1ms RTT = 125KiB BDP

=> After exceeding BDP, network is queueing packets. After queues are full, packets getting dropped due to congestion.

Why is this hard?

Sender doesn’t know the network capacity

- The network can’t (generally) tell us this

… and the network may change

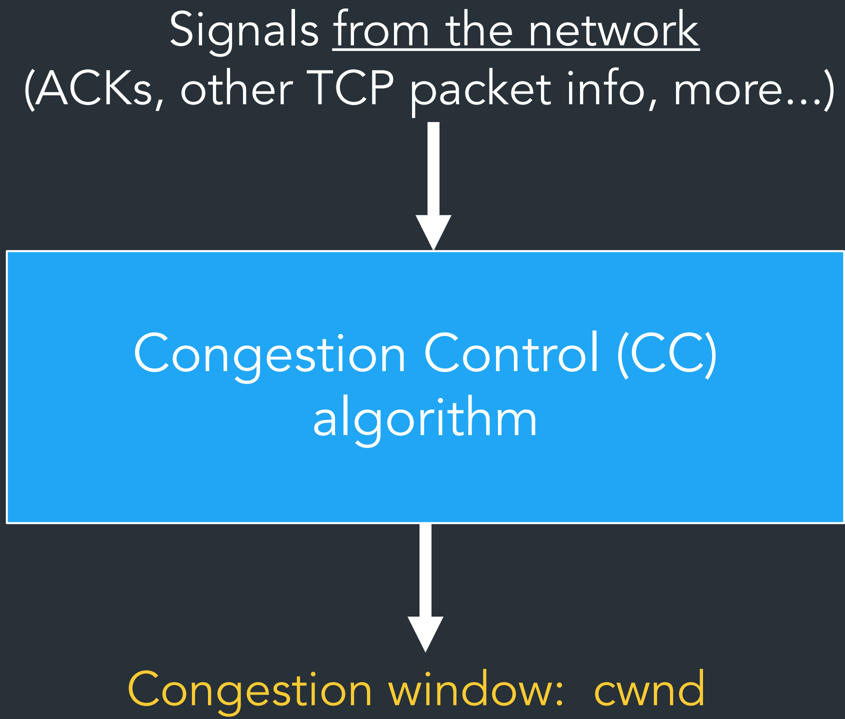
- New connections start up
- Connections end
- Link characteristics may change…

=> Need to measure or model what is going on in the network as we are sending, adapt accordingly

The basic principle

Congestion Control (CC)
algorithm

The basic principle



The basic principle

Signals from the network
(ACKs, other TCP packet info, more…)

Congestion Control (CC) algorithm

Congestion window: cwnd

Sender can send: min(advertised window, cwnd)

(Advertised window: flow control window from receiver)

The basic principle

Signals from the network
(ACKs, other TCP packet info, more...)

Congestion Control (CC)
algorithm

Congestion window: cwnd

Sender can send: min(advertised window, cwnd)

(Advertised window: flow control window from receiver)

⇒ Different CC algorithms use different signals, different techniques for adapting cwnd, but most fit this format

Lots of CC variants designed with different strategies and goals

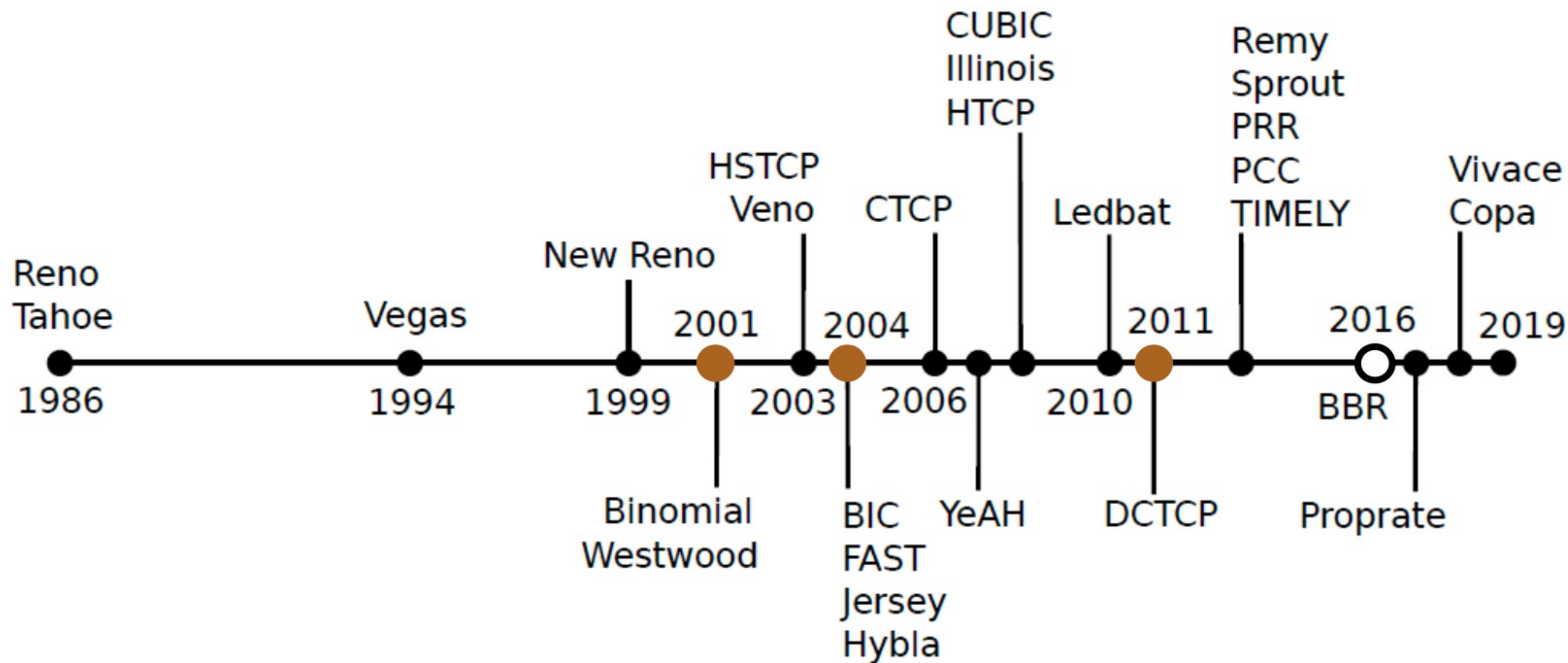
Network Signals

- Packet loss ("loss-based")
- Delay/RTT ("delay-based")
- "Marks" added on packets by routers

Goals

- Maximize throughput
- Recover from packet loss or high RTT
- Short-long "flows"
- Datacenter-specific (low-latency)

⇒ This is a big research area!


Reno
Tahoe
1986
Vegas
1994
New Reno
1999
2001
Binomial
Westwood
HSTCP
Veno
2003
2004
BIC
FAST
Jersey
Hybla
CTCP
2006
YeAH
CUBIC
Illinois
HTCP
Ledbat
2010
2011
DCTCP
Remy
Sprout
PRR
PCC
TIMELY
2016
BBR
Proprate
Vivace
Copa
2019

Variant	Feedback	Required changes	Benefits	Fairness
(New) Reno	Loss	—	—	Delay
Vegas	Delay	Sender	Less loss	Proportional
High Speed	Loss	Sender	High bandwidth	
BIC	Loss	Sender	High bandwidth	
CUBIC	Loss	Sender	High bandwidth	
C2TCP[11][12]	Loss/Delay	Sender	Ultra-low latency and high bandwidth	
NATCP[13]	Multi-bit signal	Sender	Near Optimal Performance	
Elastic-TCP	Loss/Delay	Sender	High bandwidth/short & long-distance	
Agile-TCP	Loss	Sender	High bandwidth/short-distance	
H-TCP	Loss	Sender	High bandwidth	
FAST	Delay	Sender	High bandwidth	Proportional
Compound TCP	Loss/Delay	Sender	High bandwidth	Proportional
Westwood	Loss/Delay	Sender	Lossy links	
Jersey	Loss/Delay	Sender	Lossy links	
BBR[14]	Delay	Sender	BLVC, Bufferbloat	
CLAMP	Multi-bit signal	Receiver, Router	Variable-rate links	Max-min
TFRC	Loss	Sender, Receiver	No Retransmission	Minimum delay
XCP	Multi-bit signal	Sender, Receiver, Router	BLFC	Max-min
VCP	2-bit signal	Sender, Receiver, Router	BLF	Proportional
MaxNet	Multi-bit signal	Sender, Receiver, Router	BLFSC	Max-min
JetMax	Multi-bit signal	Sender, Receiver, Router	High bandwidth	Max-min
RED	Loss	Router	Reduced delay	
ECN	Single-bit signal	Sender, Receiver, Router	Reduced loss	

Wikipedia's list

Congestion control has a long history

- Active research area for ~40 years
- I am <u>nowhere close</u> to being an expert
- My hope is to get you to understand the problems involved

Classical Congestion Control

Loss-based: assume packet loss => congestion

Classical Congestion Control

Loss-based: assume packet loss => congestion

- TCP Tahoe (1988)
  - Slow start, congestion avoidance, fast retransmit

Classical Congestion Control

Loss-based: assume packet loss => congestion

- TCP Tahoe (1988)
  - Slow start, congestion avoidance, fast retransmit
- TCP Reno (1990)
  - TCP Tahoe + Fast recovery
- Many variations developed from this… (see optional readings)

Modes of operation

- Slow start (SS)
  - Determine initial window, recover after loss
- Congestion avoidance (CA)
  - Steady state, slowly probe for changes in capacity

Congestion Avoidance

After finishing a window, recompute cwnd:

- If no losses, cwnd = cwnd + MSS
  - (Often written as cwnd += 1)
- If packets were lost: cwnd = cwnd/2

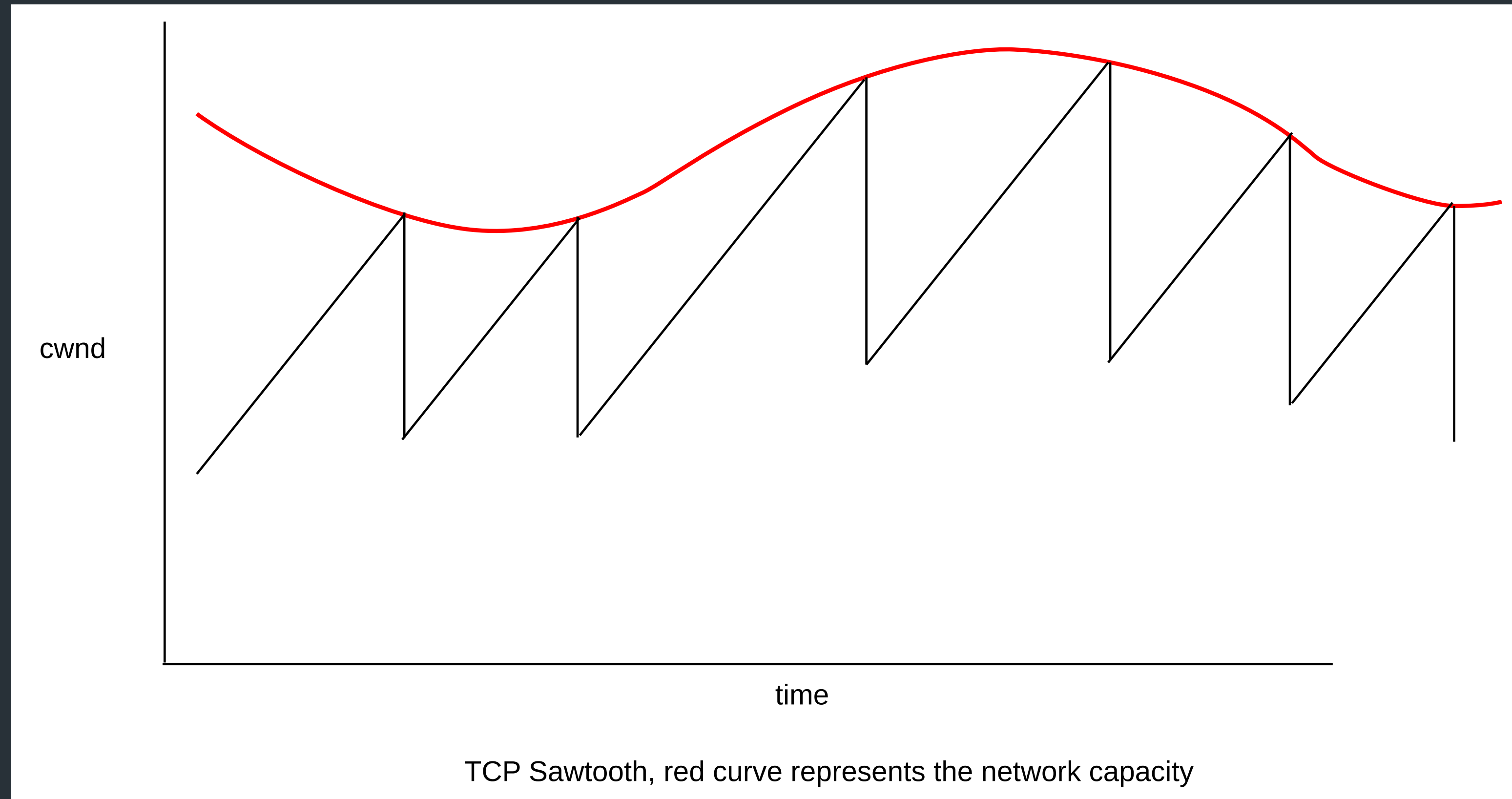
Congestion Avoidance

After finishing a window, recompute cwnd:

- If no losses, cwnd = cwnd + MSS
  - (Often written as cwnd += 1)
- If packets were lost: cwnd = cwnd/2

This is called additive increase, multiplicative decrease (AIMD)
- Slowly increase capacity
- Dramatically scale back on loss

AIMD Example



TCP Sawtooth, red curve represents the network capacity

Slow Start

Turns out AIMD is really slow to start up. So do something faster at connection start…

Slow Start

Turns out AIMD is really slow to start up. So do something faster at connection start…

After finishing a window

- cwnd = cwnd * 2
- Continue doing this until you experience a loss

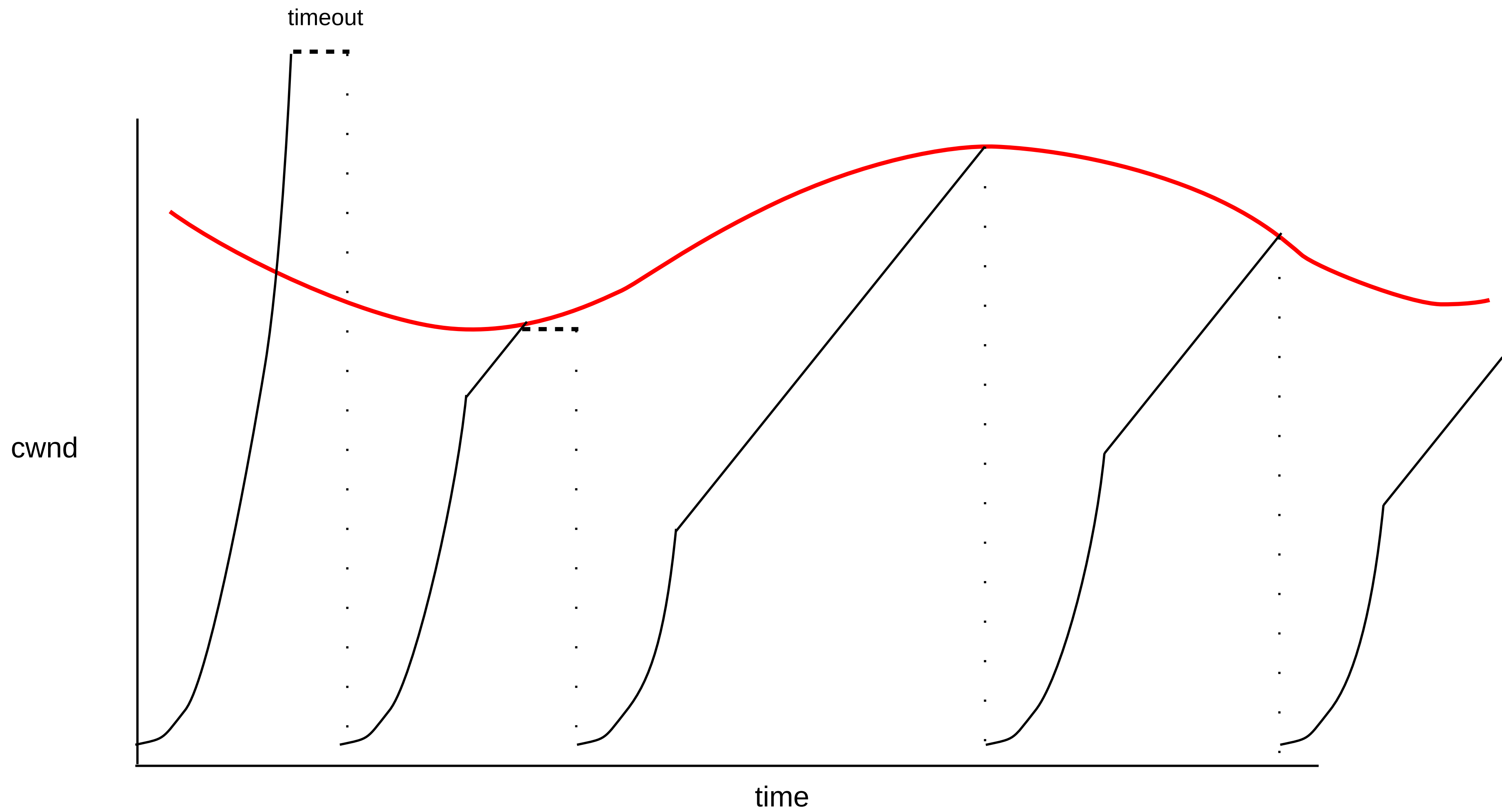
Slow Start

Turns out AIMD is really slow to start up. So do something faster at connection start…

After finishing a window

- cwnd = cwnd * 2
- Continue doing this until you experience a loss

- After first loss, keep slow-start threshold (ssthresh):
  - If window < ssthresh: slow-start
  - If window > ssthresh: congestion avoidance
- After first loss: ssthresh = cwnd / 2

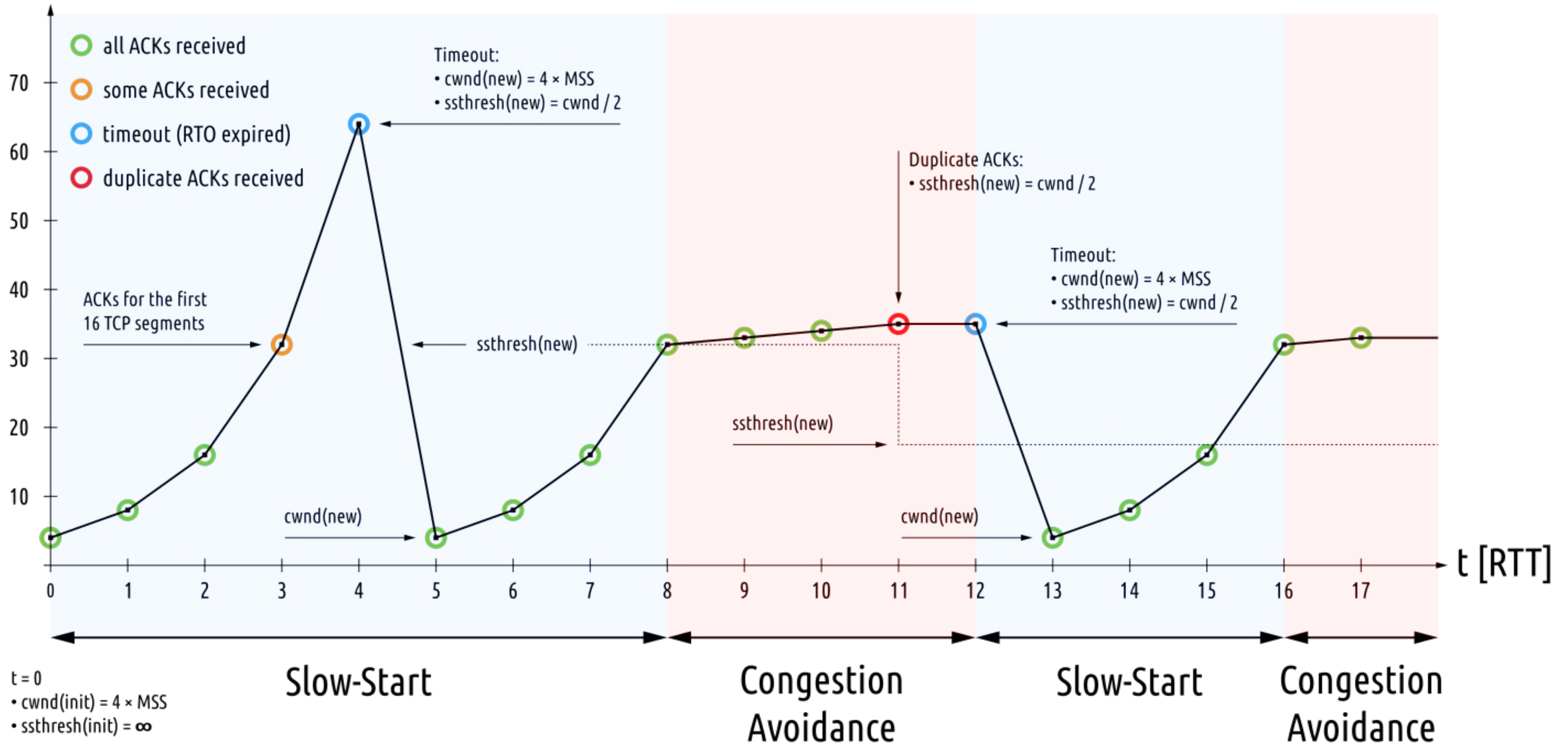


TCP Tahoe Sawtooth, red curve represents the network capacity
Slow Start is used after each packet loss until ssthresh is reached

How to Detect Loss

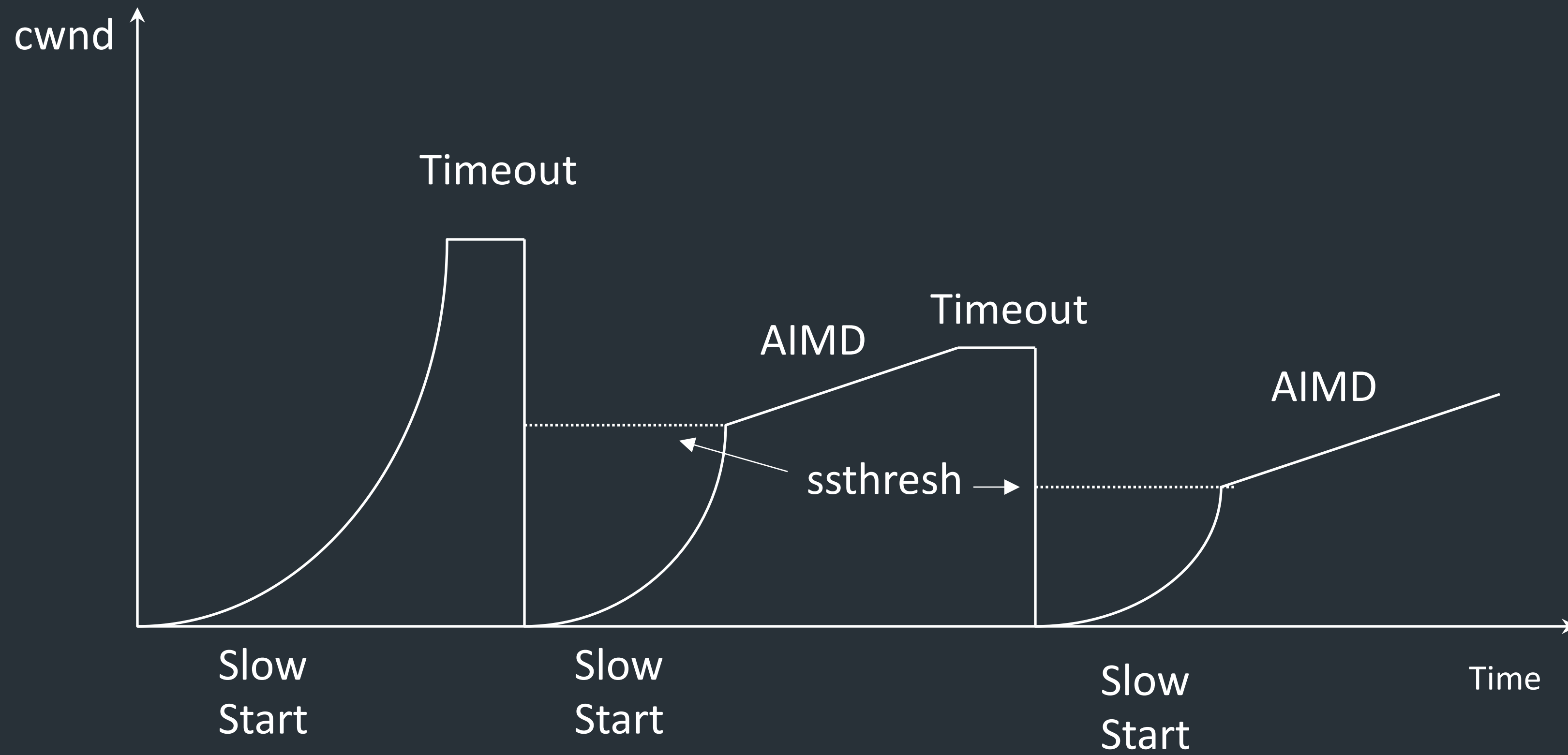
- Timeout
- Any other way?
  - Gap in sequence numbers at receiver
  - Receiver uses cumulative ACKs: drops => duplicate ACKs
- 3 Duplicate ACKs considered loss

- Which one is worse?



cwnd [MSS]
all ACKs received
some ACKs received
timeout (RTO expired)
duplicate ACKs received
Timeout:
• cwnd(new) = 4 × MSS
• ssthresh(new) = cwnd / 2
Duplicate ACKs:
• ssthresh(new) = cwnd / 2
Timeout:
• cwnd(new) = 4 × MSS
• ssthresh(new) = cwnd / 2
ACKs for the first
16 TCP segments
ssthresh(new)
ssthresh(new)
cwnd(new)
cwnd(new)
70
60
50
40
30
20
10
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
t [RTT]
t = 0
• cwnd(init) = 4 × MSS
• ssthresh(init) = ∞
Slow-Start
Congestion Avoidance
Slow-Start
Congestion Avoidance

Putting it all together



Slow start every time?!

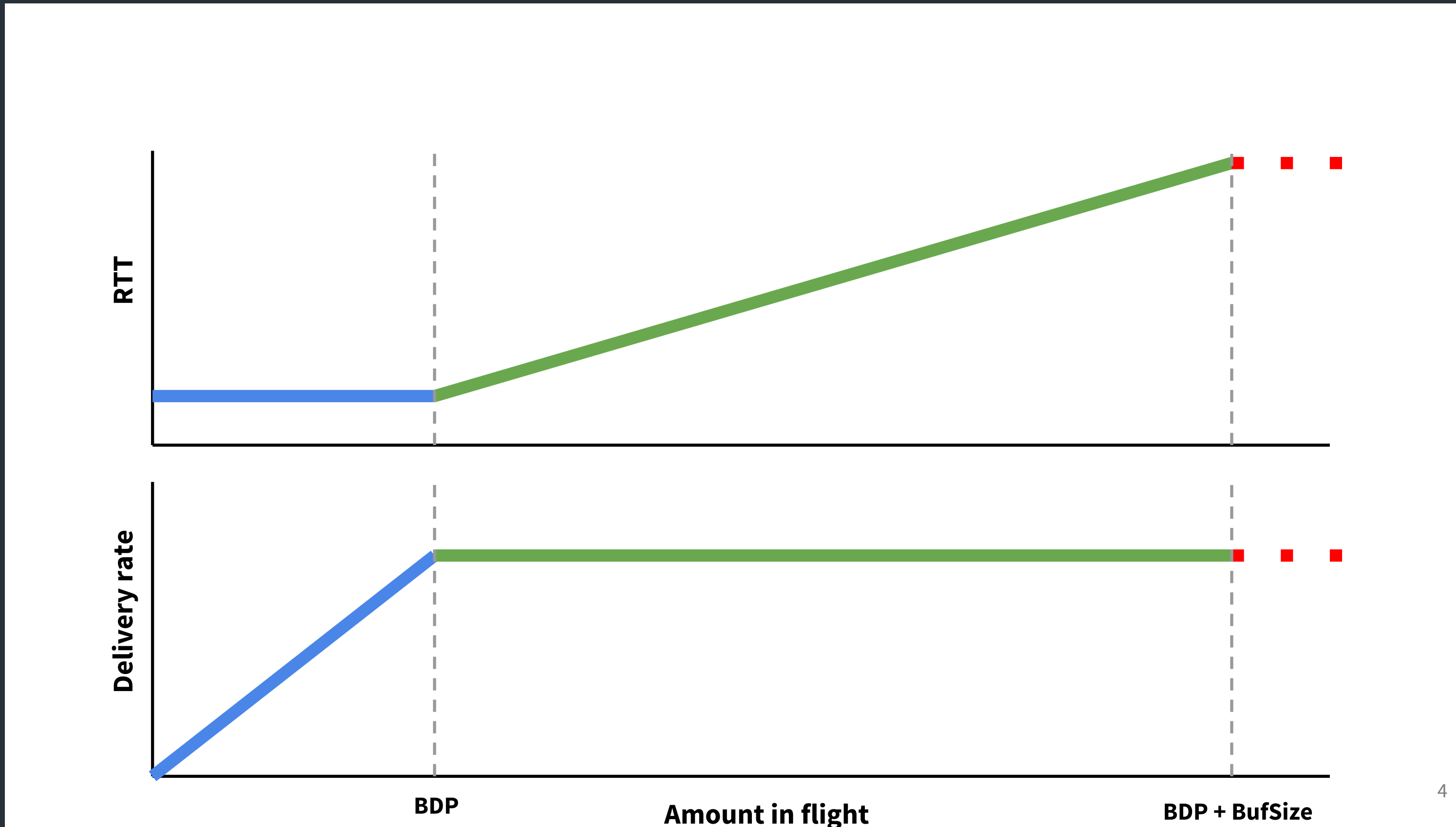
- Losses have large effect on throughput
- Fast Recovery (TCP Reno)
  - Same as TCP Tahoe on Timeout: w = 1, slow start
  - On triple duplicate ACKs: w = w/2
  - Retransmit missing segment (fast retransmit)
  - Stay in Congestion Avoidance mode
- Why 3 dup-acks instead of just 1?

This is just the beginning…

Lots of congestion control schemes, with different strategies/goals:

- Tahoe (1988)
- Reno (1990)
- Vegas (1994): Detect based on RTT
- New Reno: Better recovery multiple losses
- Cubic (2006): Linux default, window size scales by cubic function
- BBR (2016): Used by Google, measures bandwidth/RTT

BBR: what's different



"BBR congestion control"

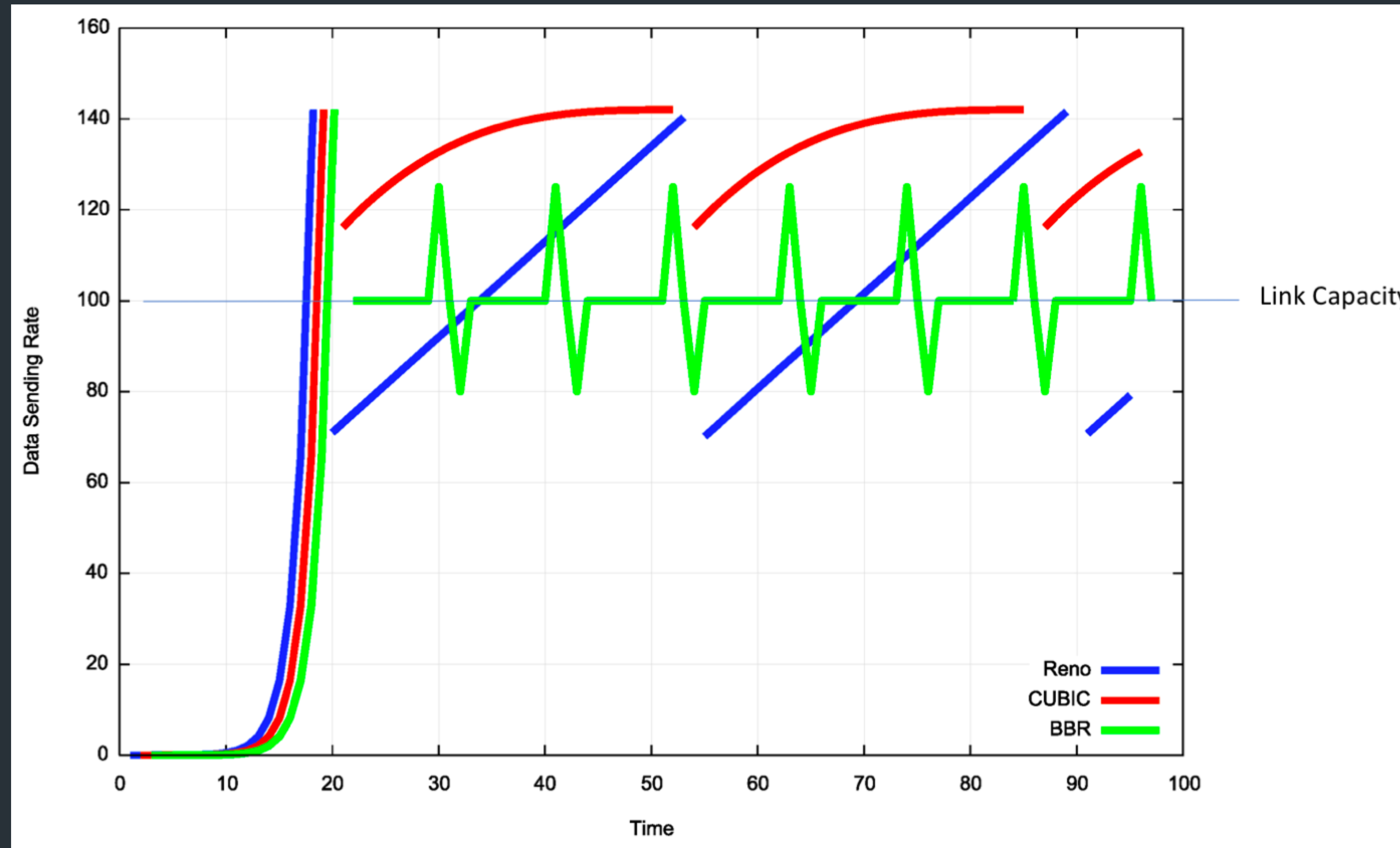
BBR

- Problem: can’t measure both RTT_{prop} and Bottleneck BW at the same time

BBR:

- – Slow start
- – Measure throughput when RTT starts to increase
- – Measure RTT when throughput is still increasing
- – Pace packets at the BDP
- – Probe by sending faster for 1RTT, then slower to compensate

BBR



From: https://labs.ripe.net/Members/gih/bbr-tcp

Another way: ECN

What if we didn't have to drop packets?

- Routers/switches set bits in packet to indicate congestion

- When sender sees congestion bit, scales back cwnd
- Must be supported by both sender and receiver

=>Avoids retransmissions optionally dropped packets

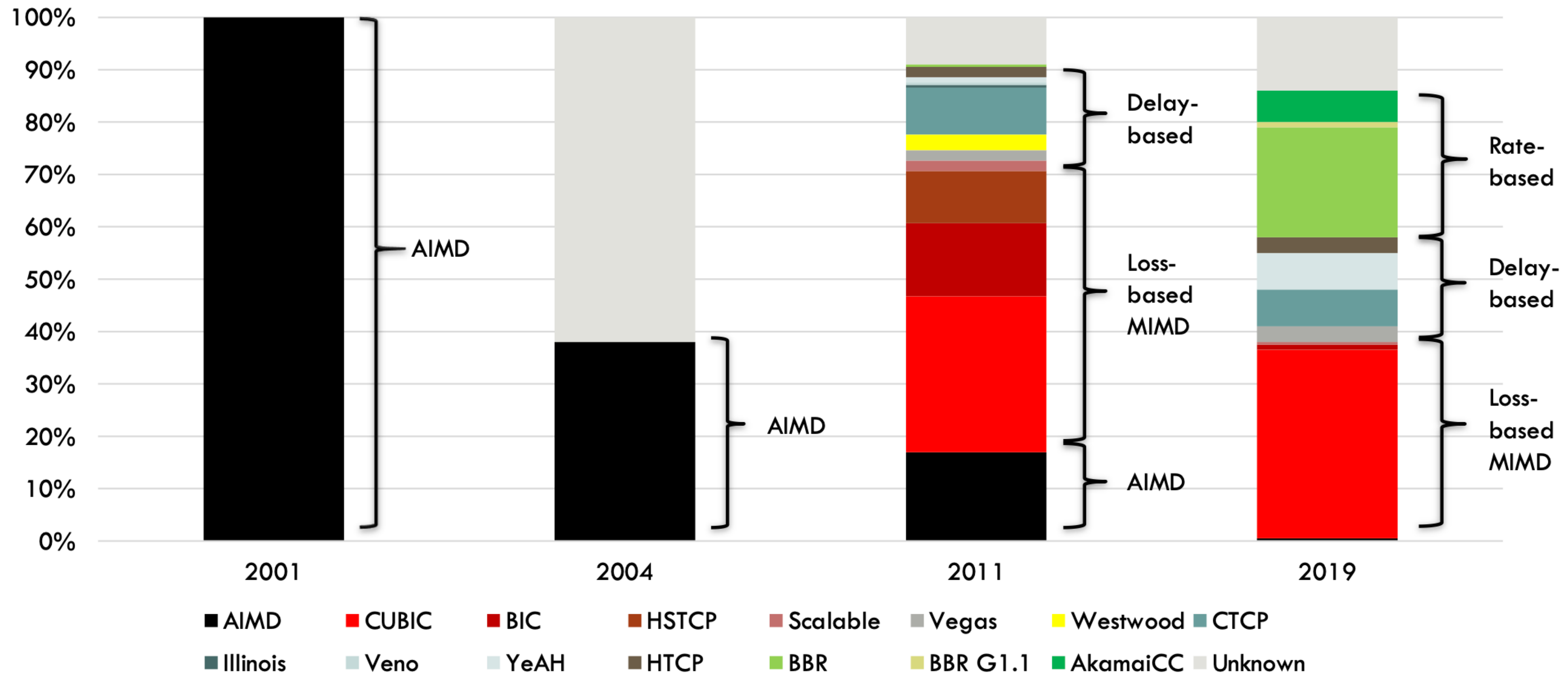
Special purpose example: DCTCP (2010)

Designed for datacenter usage only

- Want to avoid queuing as much as possible
- Routers/switches mark packets with ECN bit in header
- When this happens, senders scale back dramatically

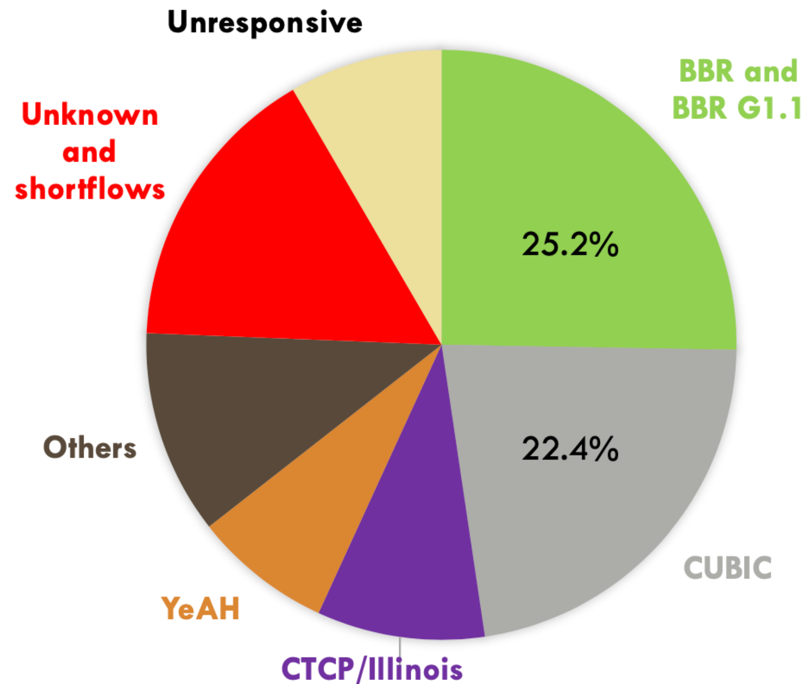
What happens in practice now?

THE EVOLUTION OF THE TCP ECOSYSTEM



"The Great Internet TCP congestion control census" (2019)

DISTRIBUTION BY POPULARITY AND TRAFFIC SHARE



Share of congestion control algorithms deployed by website count in the **Alexa Top 250** websites

- Among the top 250 Alexa websites, BBR has a larger share by website count than Cubic
- In terms of traffic share, BBR is now contributing to **more than 40%** of the downstream traffic on the Internet!

Site	Downstream traffic share	Variant
Amazon Prime	3.69%	CUBIC
Netflix	15%	CUBIC
YouTube	11.35%	BBR
Other Google sites	28%	BBR
Steam downloads	2.84%	BBR

(As measured on **static HTTP webpages**)

"The Great Internet TCP congestion control census" (2019)

LOOKING CLOSER AT THE UNCLASSIFIED VARIANTS

We had a total of **6,330 (31.65%)** of websites that were **unclassified**

We ran a variety of network profiles on these websites to infer something about their congestion control mechanism

Type	React to Packet Loss?	React to BDP?	Websites (share)
AkamaiCC	✗	✓	1,103 (5.52%)
Unknown Akamai	✗	?	157 (0.79%)
Unknown	✗	?	493 (2.47%)
	✓	?	1,782 (8.91%)
Short flows	✓	?	1,493 (7.47%)
Unresponsive	?	?	1,302 (6.51%)
Total			6,330 (31.65%)

"The Great Internet TCP congestion control census" (2019)

A Short History of TCP

- 1974: 3-way handshake
- 1978: IP and TCP split
- 1983: January 1st, ARPAnet switches to TCP/IP
- 1984: Nagle predicts congestion collapses
- 1986: Internet begins to suffer congestion collapses
  - LBL to Berkeley drops from 32Kbps to 40bps
- 1987/8: Van Jacobson fixes TCP, publishes seminal paper*: (TCP Tahoe)
- 1990: Fast transmit and fast recovery added (TCP Reno)

* Van Jacobson and Michael Karels. Congestion avoidance and control. SIGCOMM '88

Congestion Collapse

Nagle, rfc896, 1984

- Mid 1980's: Problem with the protocol *implementations*, not the protocol!
- What was happening?
- If close to capacity, and, e.g., a large flow arrives suddenly…
  - RTT estimates become too short
  - Lots of retransmissions → increase in queue size
  - Eventually many drops happen (full queues)
  - Fraction of useful packets (not copies) decreases

The problem

- https://witestlab.poly.edu/respond/sites/genitutorial/files/tcp-aimd.ogv

Just a few TCP implementations

What's the difference?

General usage

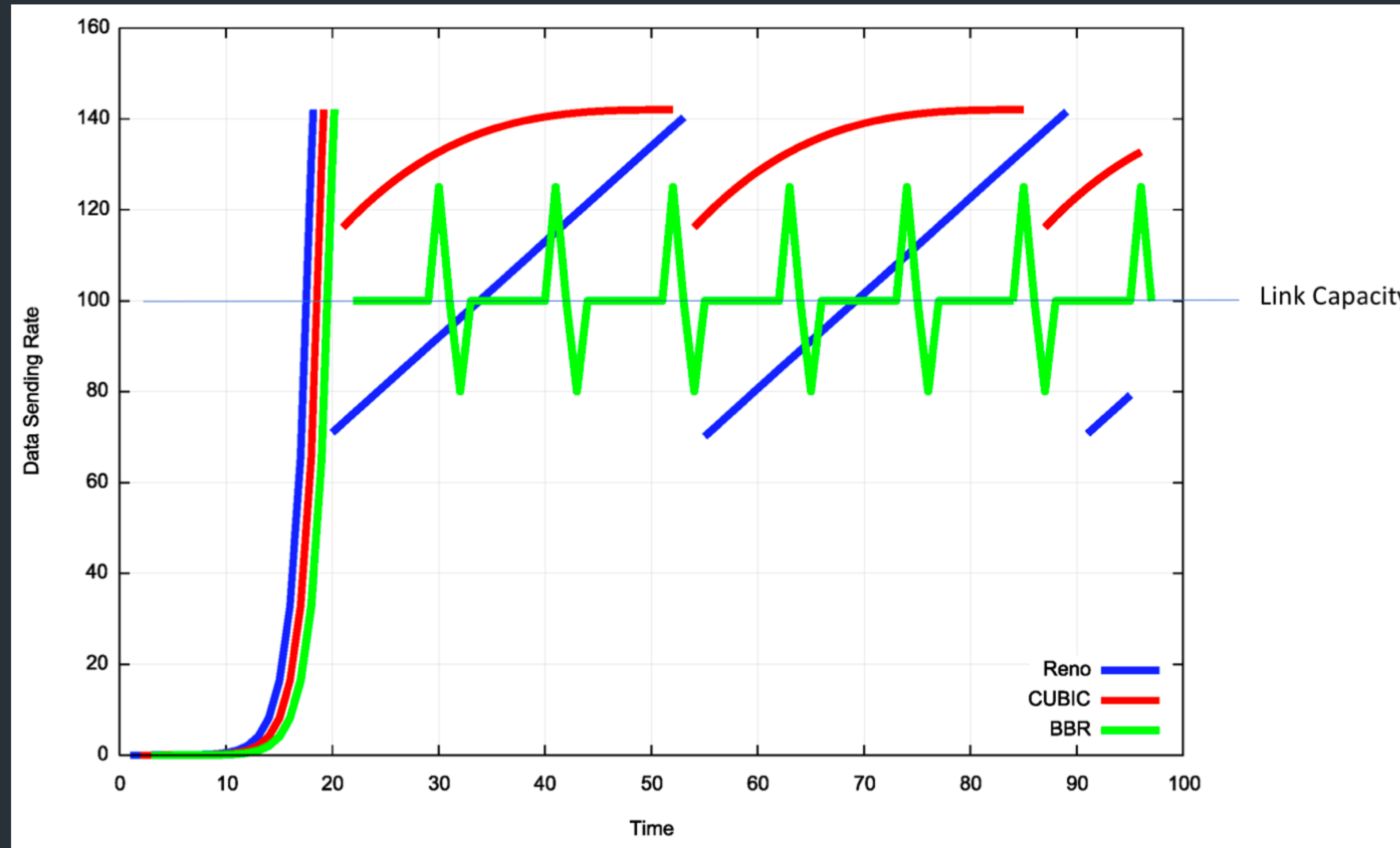
- Reno (1980s)
- Tahoe
- Vegas
- New Vegas
- Westwood
- Cubic
- BBR (2016)
- …

Dealing with Congestion

To start:

- Assume losses are due to congestion
- After a loss, reduce congestion window
  - How much to reduce?
- Idea: conservation of packets at equilibrium
  - Want to keep roughly the same number of packets in network
  - Analogy with water in fixed-size pipe
  - Put new packet into network when one exits

BBR



From: https://labs.ripe.net/Members/gih/bbr-tcp

TCP State Diagram

unusual event
client/receiver path
server/sender path

(Start) CLOSED
CONNECT/SYN (Step 1 of the 3-way-handshake)
CLOSE/-
LISTEN/-
CLOSE/-
LISTEN
(Step 2 of the 3-way-handshake) SYN/SYN+ACK
RST/-
SEND/SYN
SYN RECEIVED
SYN/SYN+ACK (simultaneous open)
SYN SENT
Data exchange occurs
ACK/-
ESTABLISHED
SYN+ACK/ACK
(Step 3 of the 3-way-handshake)
CLOSE/FIN
CLOSE/FIN
FIN/ACK

Active CLOSE
FIN WAIT 1
FIN/ACK
CLOSING
FIN+ACK/ACK
ACK/-
ACK/-
FIN WAIT 2
TIME WAIT
FIN/ACK
Timeout

Passive CLOSE
CLOSE WAIT
CLOSE/FIN
LAST ACK
ACK/-

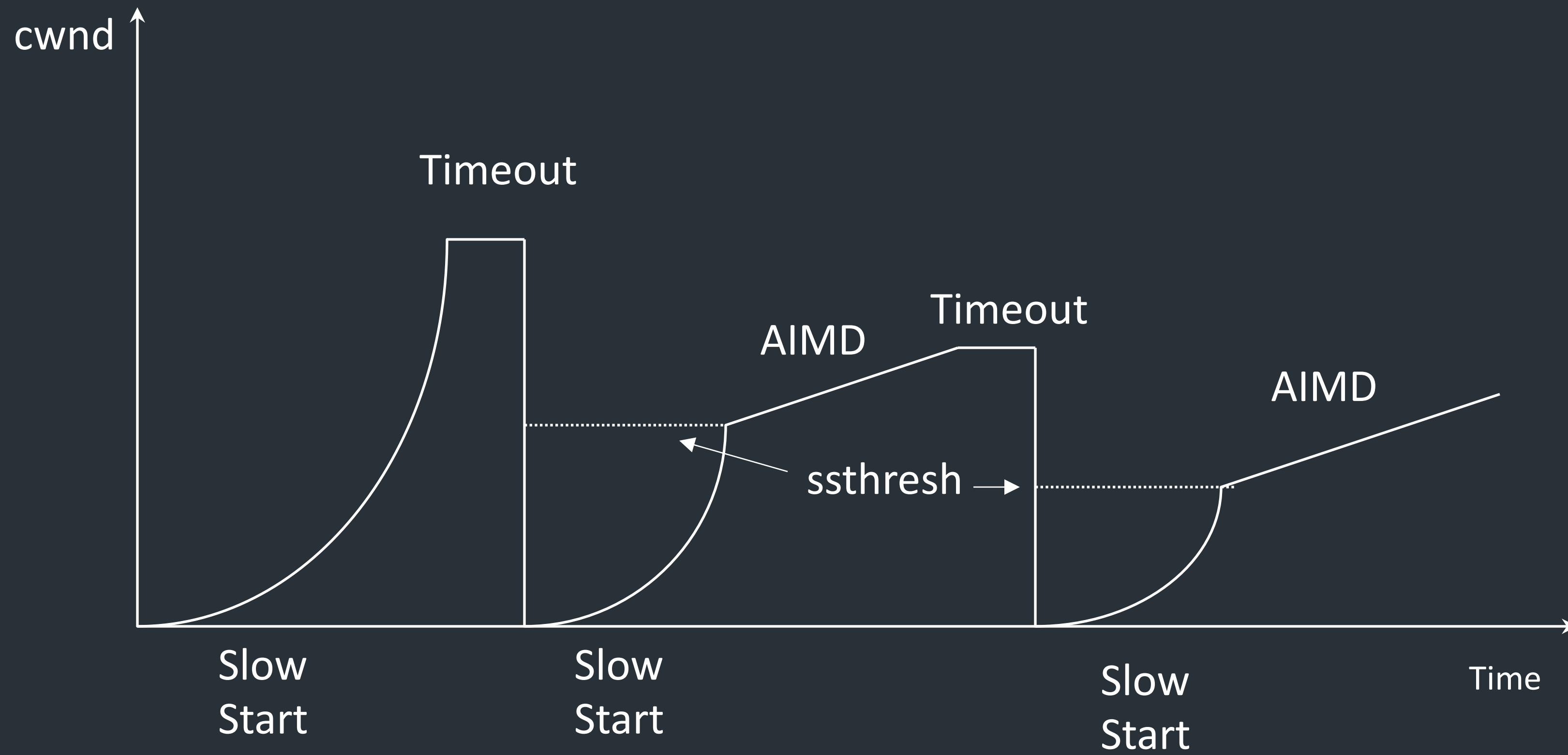
(Go back to start) CLOSED

TCP Header

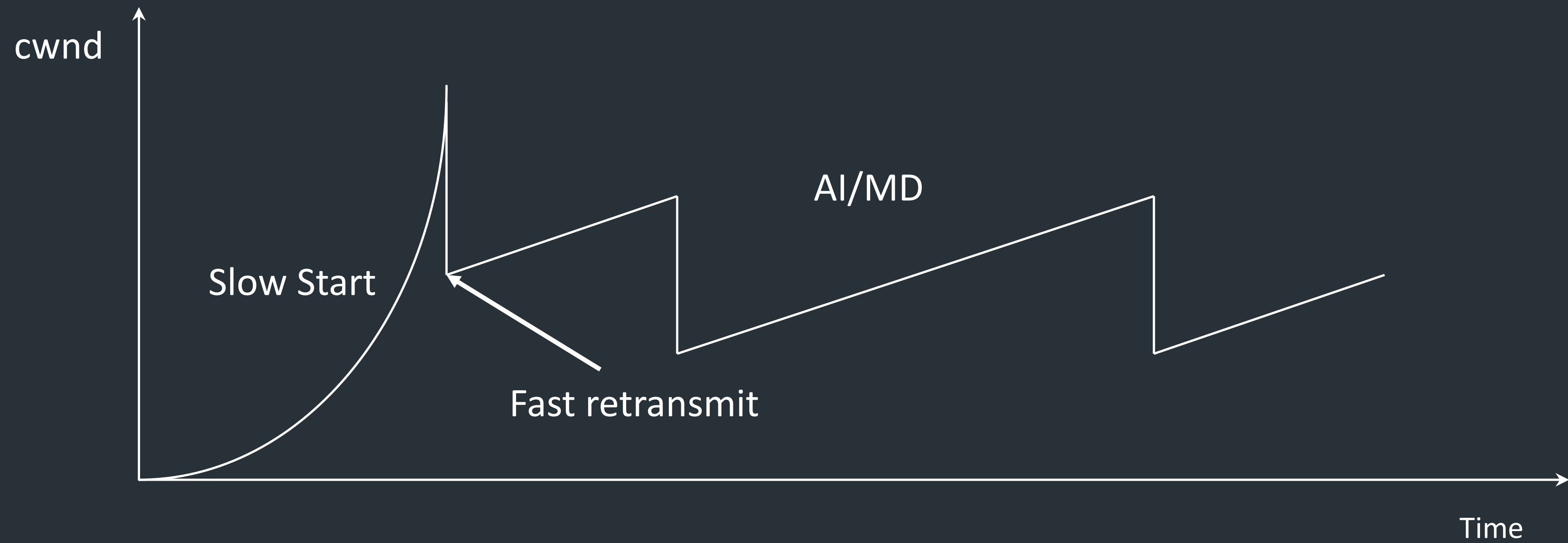
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Extra congestion control content

Putting it all together

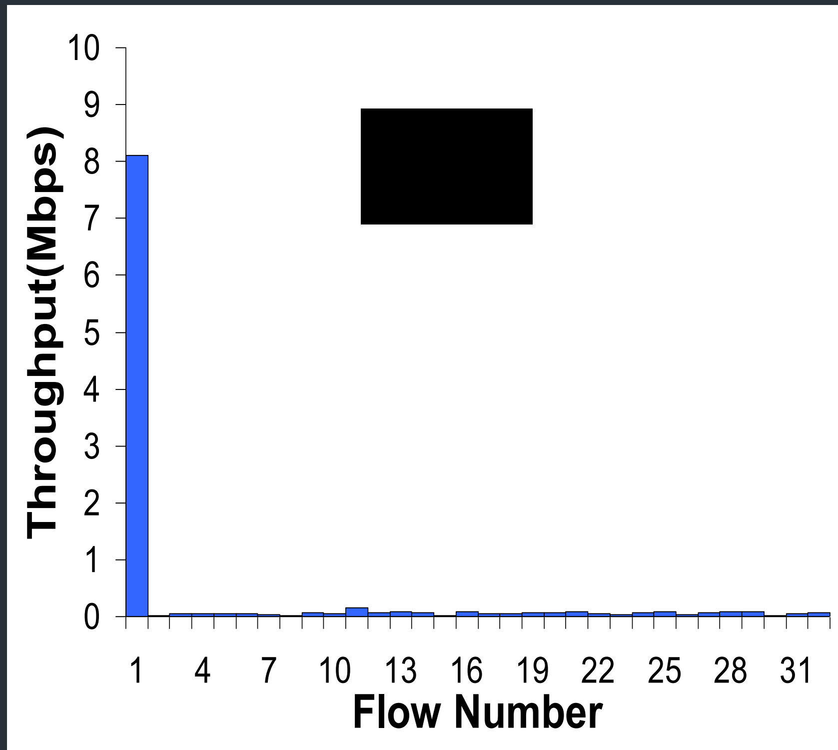


Fast Recovery and Fast Retransmit



TCP Friendliness

- Can other protocols co-exist with TCP?
  - E.g., if you want to write a video streaming app using UDP, how to do congestion control?



1 UDP Flow at 10MBps
31 TCP Flows
Sharing a 10MBps link

TCP Friendliness

- Can other protocols co-exist with TCP?
  - E.g., if you want to write a video streaming app using UDP, how to do congestion control?
- Equation-based Congestion Control
  - Instead of implementing TCP's CC, estimate the rate at which TCP would send. Function of what?
  - RTT, MSS, Loss
- Measure RTT, Loss, send at that rate!

TCP Throughput

- Assume a TCP congestion of window W (segments), round-trip time of RTT, segment size MSS
  - Sending Rate $S = W \times MSS / RTT$ (1)
- Drop: W = W/2
  - grows by *MSS* for *W/2 RTTs*, until another drop at $W \approx W$
- Average window then 0.75x*S*
  - From (1), $S = 0.75\ W\ MSS / RTT$ (2)
- Loss rate is 1 in number of packets between losses:
  - $Loss = 1 / (1 + (W/2 + W/2+1 + W/2 + 2 + \ldots + W)$
    $= 1 / (3/8\ W^2)$ (3)

TCP Throughput (cont)

– Loss = $8/(3W^2)$ (4)

$$\Rightarrow W = \sqrt{\frac{8}{3 \cdot Loss}}$$

– Substituting (4) in (2), $S = 0.75\ W\ MSS\ /\ RTT$,

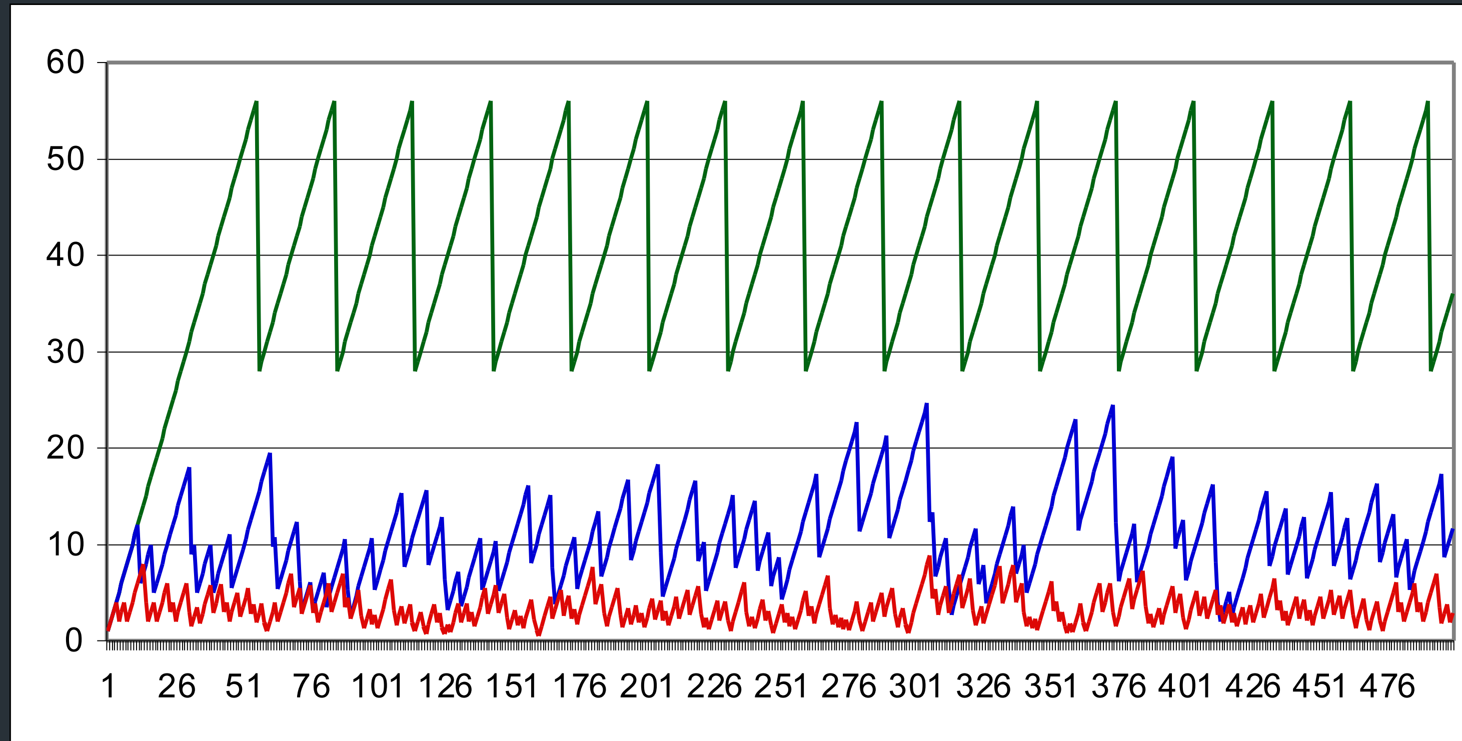
Throughput ≈

$$1.22 \times \frac{MSS}{RTT \cdot \sqrt{Loss}}$$

- Equation-based rate control can be TCP friendly and have better properties, e.g., small jitter, fast ramp-up…

What Happens When Link is Lossy?

- Throughput ≈ 1 / sqrt(Loss)



p = 0

p = 1%

p = 10%

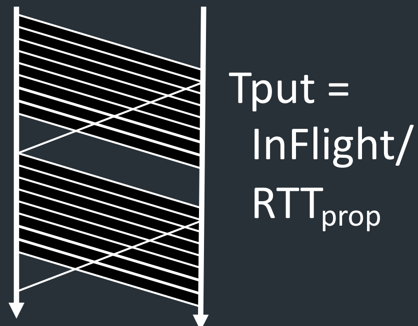
What can we do about it?

- Two types of losses: congestion and corruption
- One option: mask corruption losses from TCP
  - Retransmissions at the link layer
  - E.g. Snoop TCP: intercept duplicate acknowledgments, retransmit locally, filter them from the sender
- Another option:
  - Tell the sender about the cause for the drop
  - Requires modification to the TCP endpoints

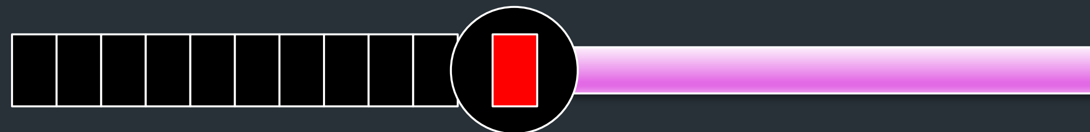
Congestion Avoidance

- TCP creates congestion to then back off
  - Queues at bottleneck link are often full: increased delay
  - Sawtooth pattern: jitter
- Alternative strategy
  - Predict when congestion is about to happen
  - Reduce rate early
- Other approaches
  - Delay Based: TCP Vegas (not covered)
  - Better model of congestion: BBR
  - Router-centric: RED, ECN, DECBit, DCTCP

Another view of Congestion Control

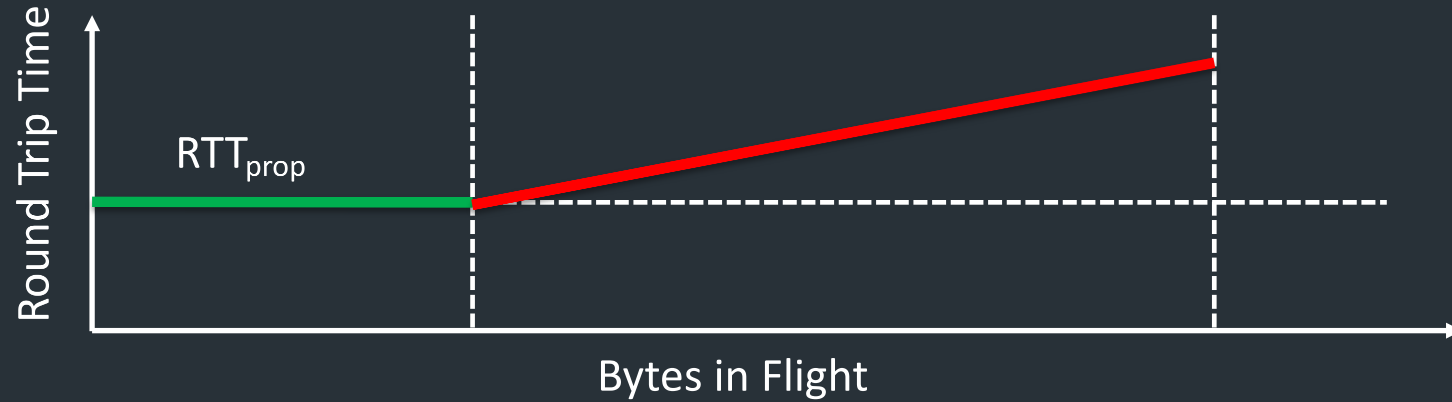


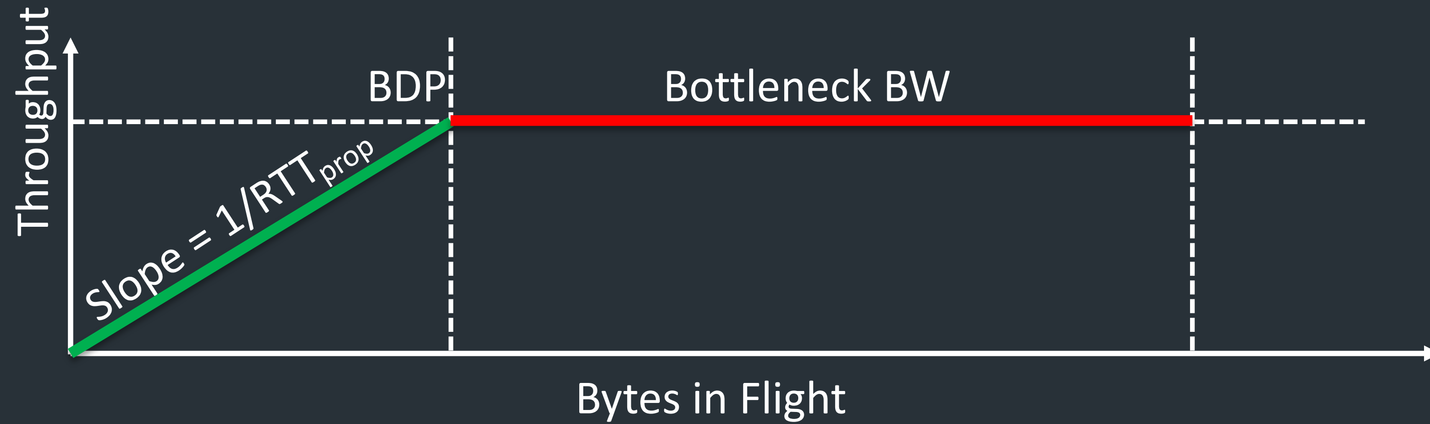
Tput = InFlight/ RTT_{prop}

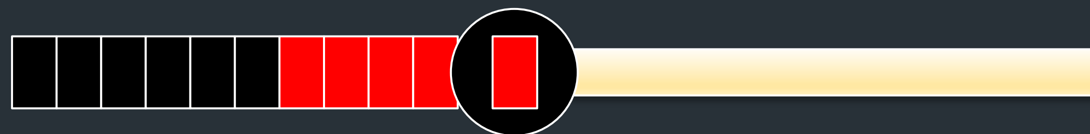


Diagrams based on Cardwell et al., [BBR: Congestion Based Congestion Control](#)," Communications of the ACM, Vol. 60 No. 2, Pages 58-66.

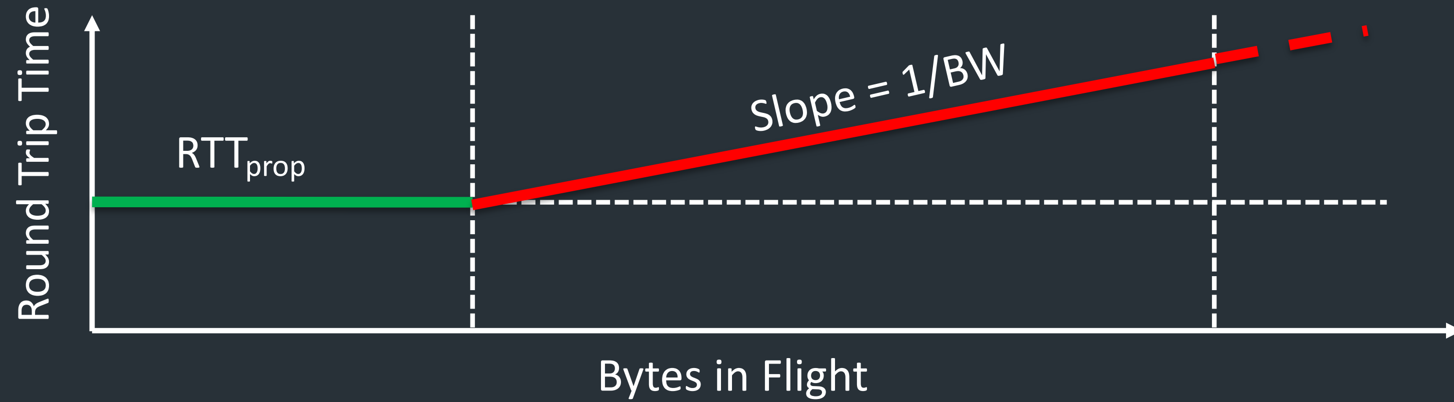
Another view of Congestion Control

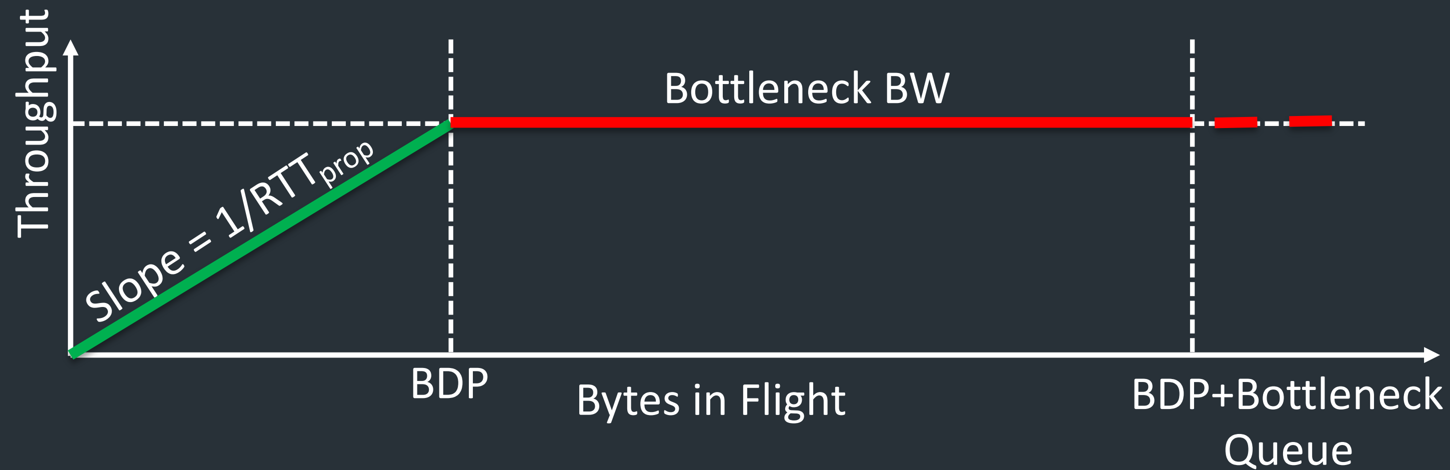


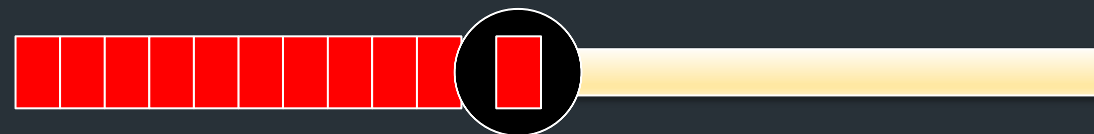




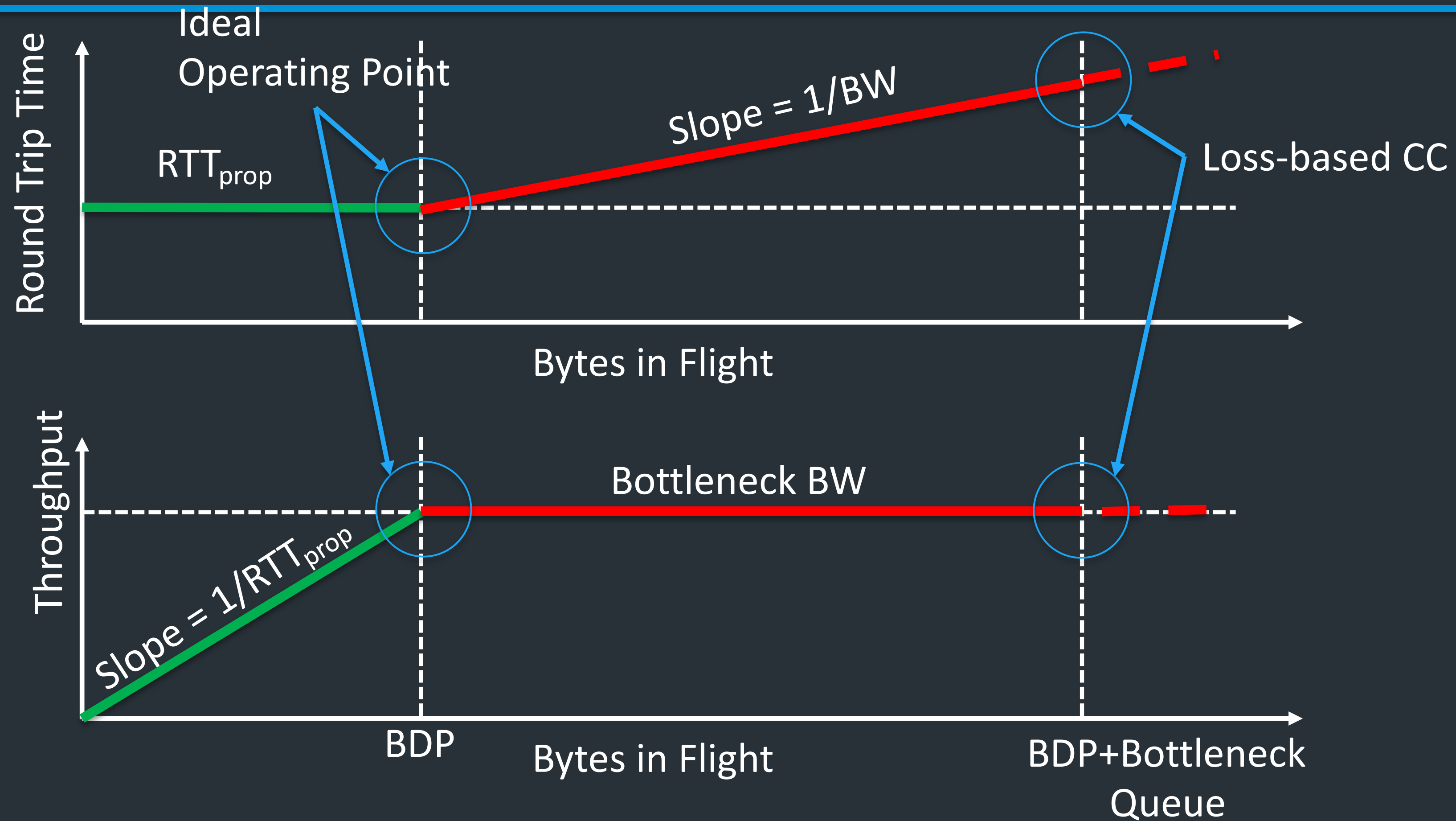
Another view of Congestion Control







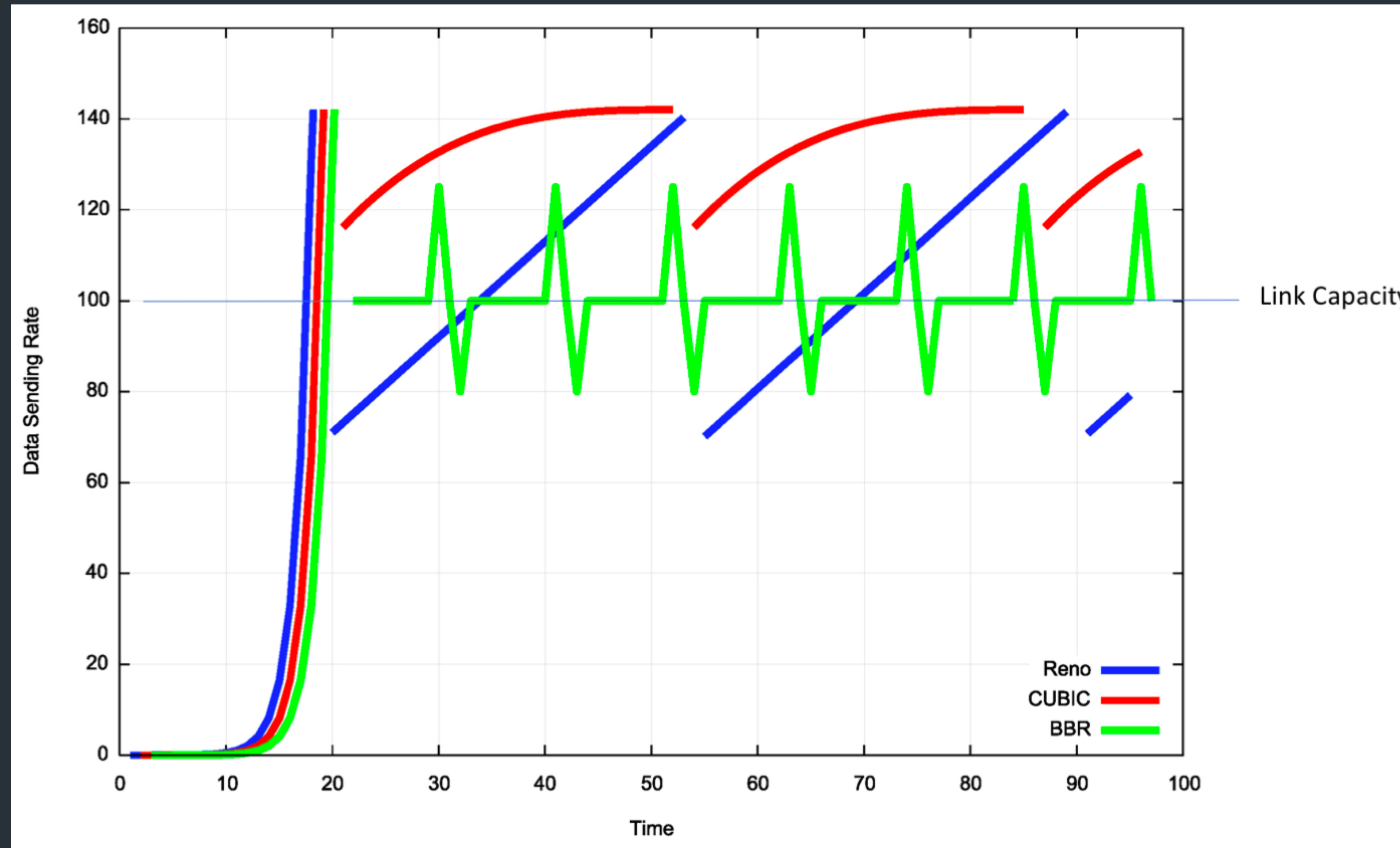
Another view of Congestion Control



BBR

- Problem: can't measure both RTT_{prop} and Bottleneck BW at the same time
- BBR:
  - Slow start
  - Measure throughput when RTT starts to increase
  - Measure RTT when throughput is still increasing
  - Pace packets at the BDP
  - Probe by sending faster for 1RTT, then slower to compensate

BBR



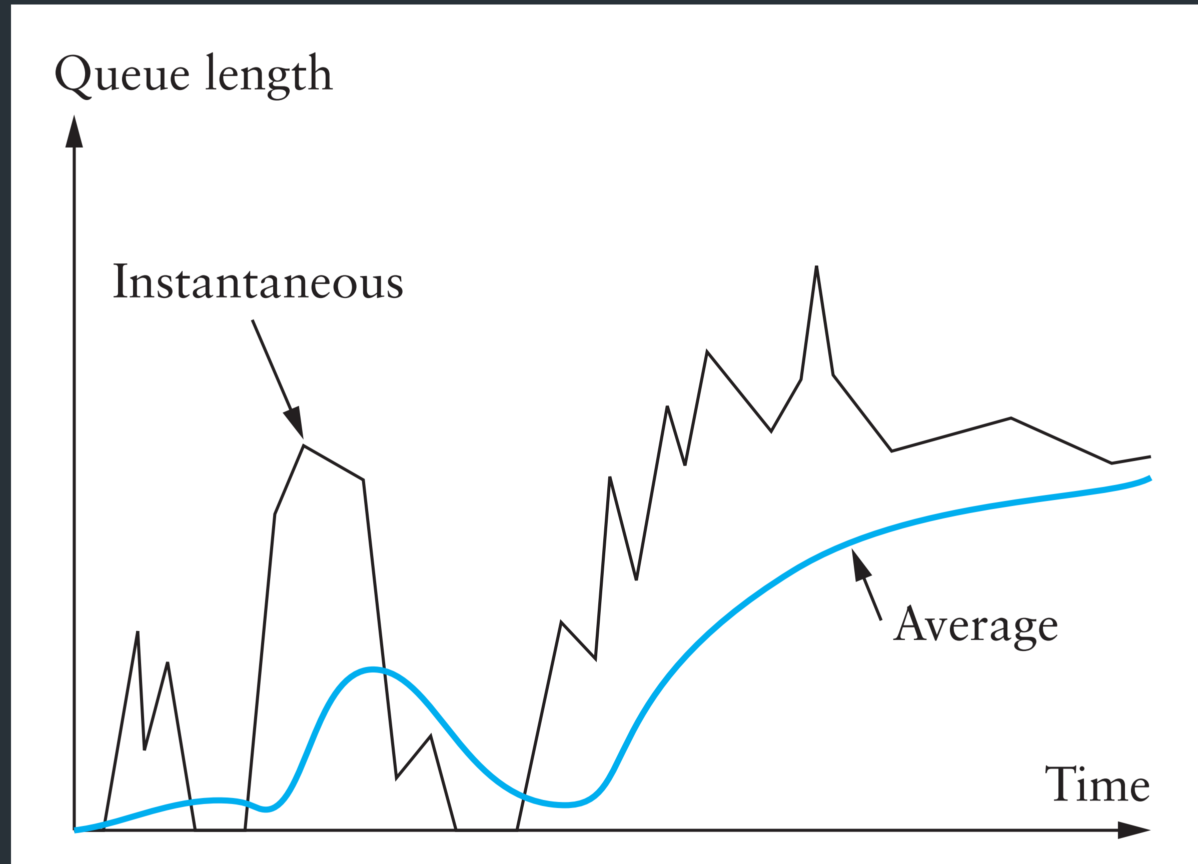
From: https://labs.ripe.net/Members/gih/bbr-tcp

Help from the network

- What if routers could *tell* TCP that congestion is happening?
  - Congestion causes queues to grow: rate mismatch
- TCP responds to drops
- Idea: Random Early Drop (RED)
  - Rather than wait for queue to become full, drop packet with some probability that increases with queue length
  - TCP will react by reducing cwnd
  - Could also mark instead of dropping: ECN

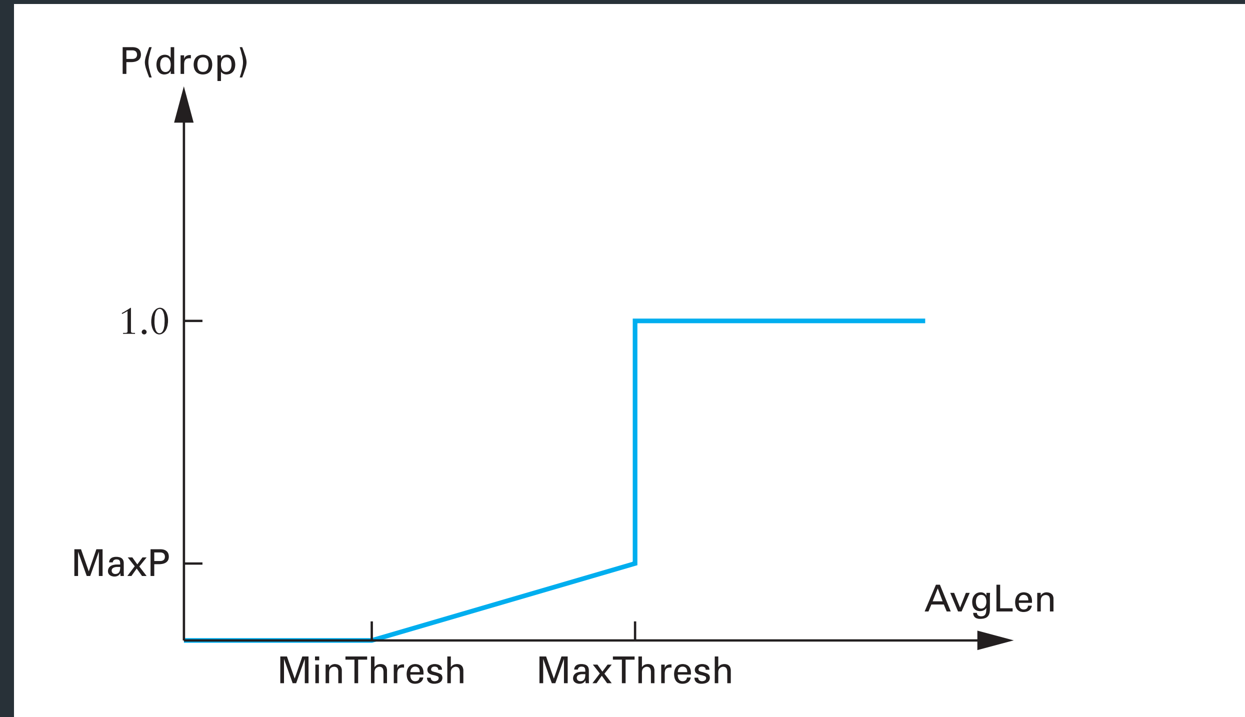
RED Details

- Compute average queue length (EWMA)
  - Don't want to react to very quick fluctuations



RED Drop Probability

- Define two thresholds: MinThresh, MaxThresh
- Drop probability:



- **Improvements to spread drops (see book)**

RED Advantages

- Probability of dropping a packet of a particular flow is roughly proportional to the share of the bandwidth that flow is currently getting
- Higher network utilization with low delays
- Average queue length small, but can absorb bursts
- ECN
  - Similar to RED, but router sets bit in the packet
  - Must be supported by both ends
  - Avoids retransmissions optionally dropped packets

What happens if not everyone cooperates?

- TCP works extremely well when its assumptions are valid
  - All flows correctly implement congestion control
  - Losses are due to congestion

Cheating TCP

- Possible ways to cheat
  - Increasing cwnd faster
  - Large initial cwnd
  - Opening many connections
  - Ack Division Attack

Larger Initial Window

x starts SS with cwnd = 4

y starts SS with cwnd = 1

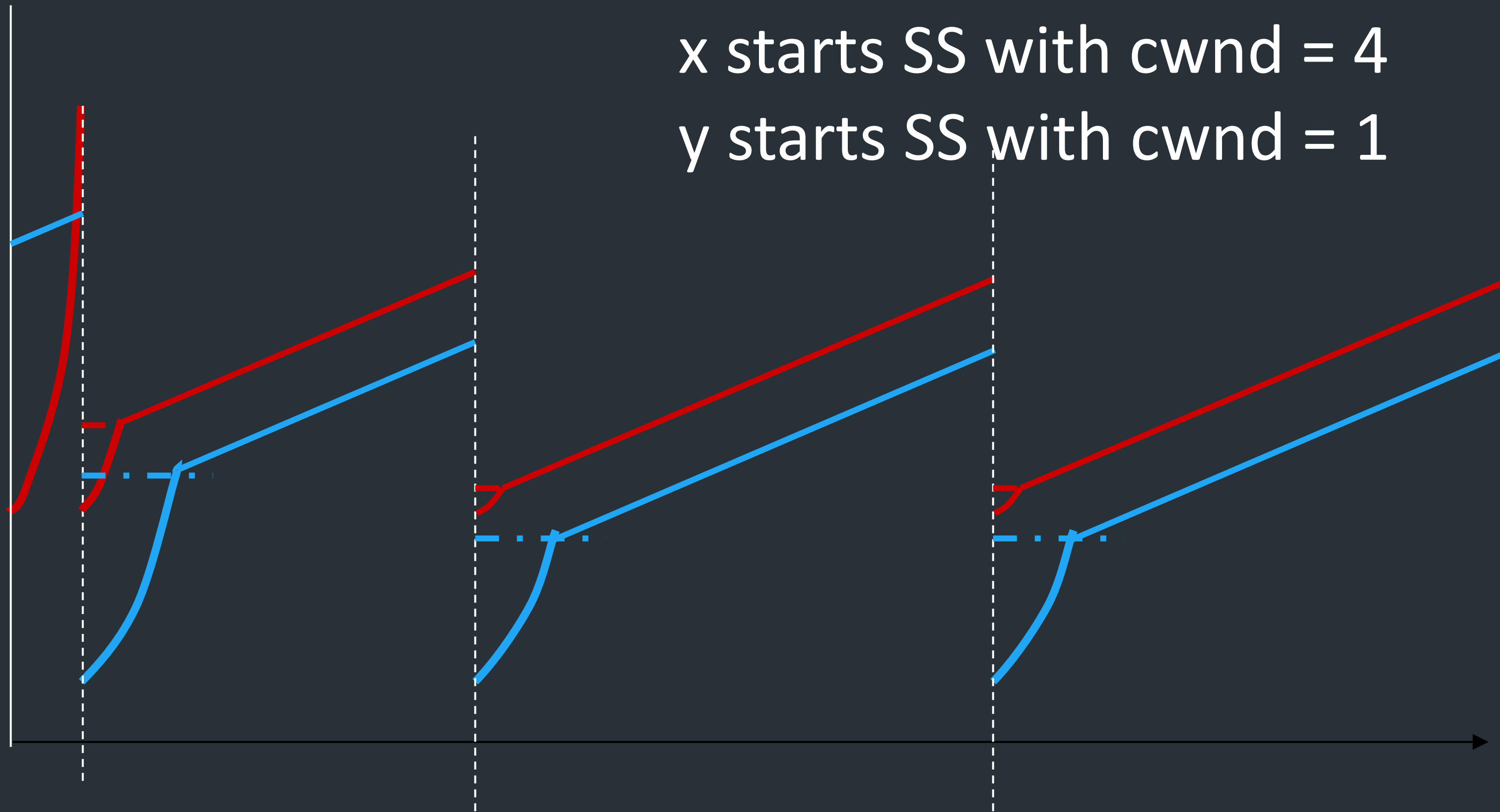


Figure from Walrand, Berkeley EECS 122, 2003

Open Many Connections

- Web Browser: has to download k objects for a page
  - Open many connections or download sequentially?

Assume:

- A opens 10 connections to B
- B opens 1 connection to E

- TCP is fair among connections
  - A gets 10 times more bandwidth than B

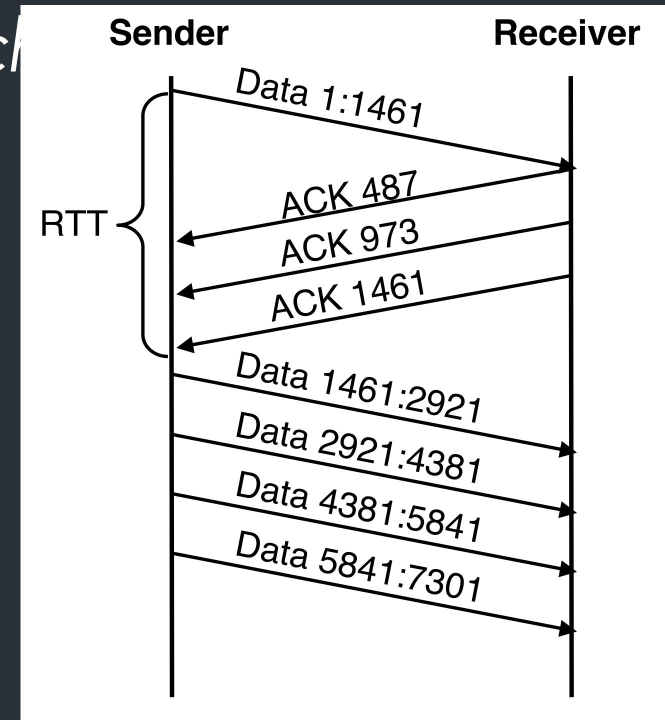
Figure from Walrand, Berkeley EECS 122, 2003

Exploiting Implicit Assumptions

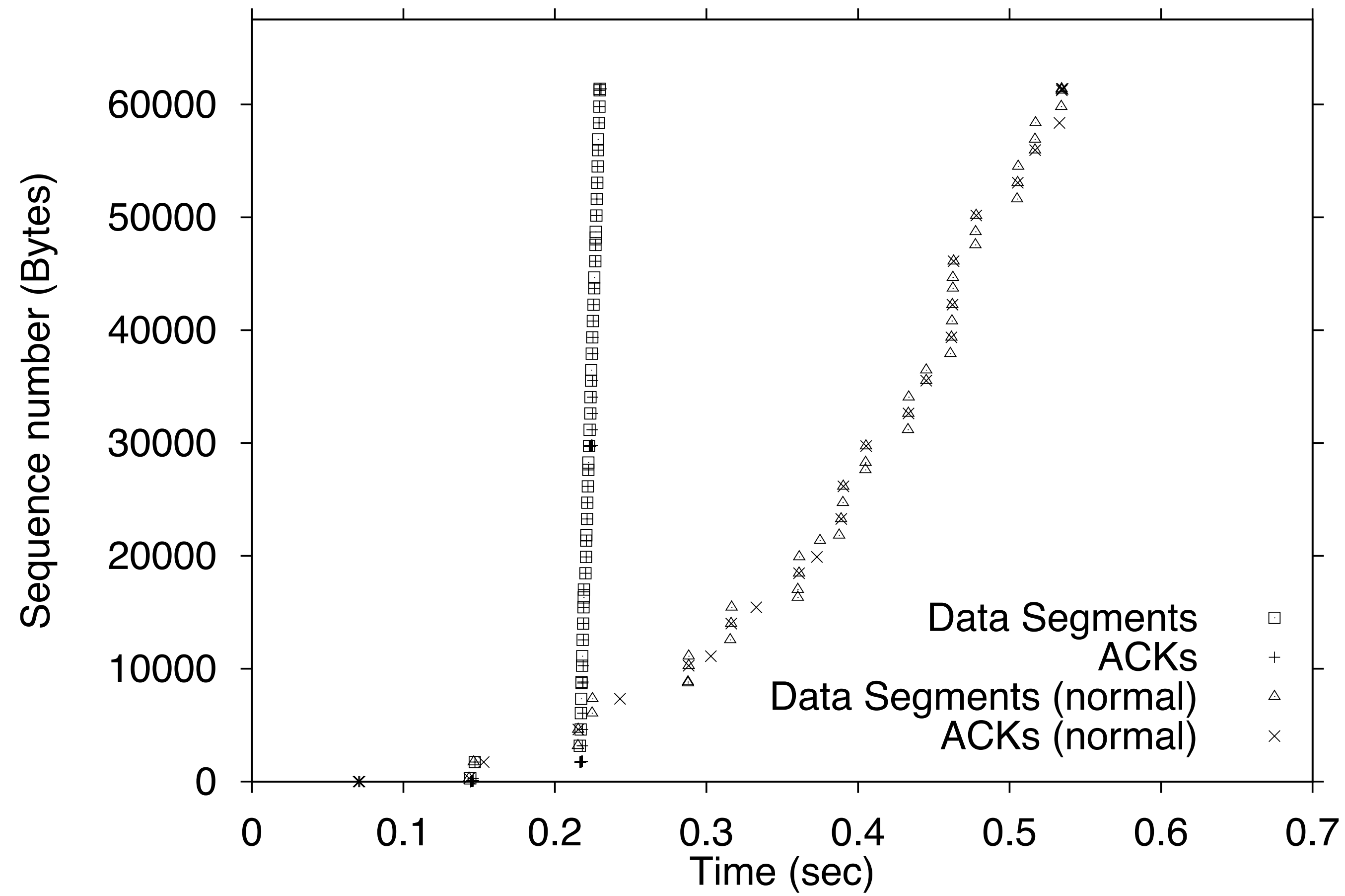
- Savage, et al., CCR 1999:
  - "TCP Congestion Control with a Misbehaving Receiver"
- Exploits ambiguity in meaning of ACK
  - ACKs can specify any byte range for error control
  - Congestion control assumes ACKs cover entire sent segments
- What if you send multiple ACKs per segment?

ACK Division Attack

- Receiver: *"upon receiving a segment with N bytes, divide the bytes in M groups and acknowledge each*
- Sender will grow window M times faster
- Could cause growth to 4GB in 4 RTTs!
  - M = N = 1460



TCP Daytona!



Defense

- Appropriate Byte Counting
  - [RFC3465 (2003), RFC 5681 (2009)]
  - In slow start, cwnd += min (N, MSS)

  where N is the number of newly acknowledged bytes in the received ACK

More help from the network

- Problem: still vulnerable to malicious flows!
  - RED will drop packets from large flows preferentially, but they don't have to respond appropriately
- Idea: Multiple Queues (one per flow)
  - Serve queues in Round-Robin
  - Nagle (1987)
  - Good: protects against misbehaving flows
  - Disadvantage?
  - Flows with larger packets get higher bandwidth

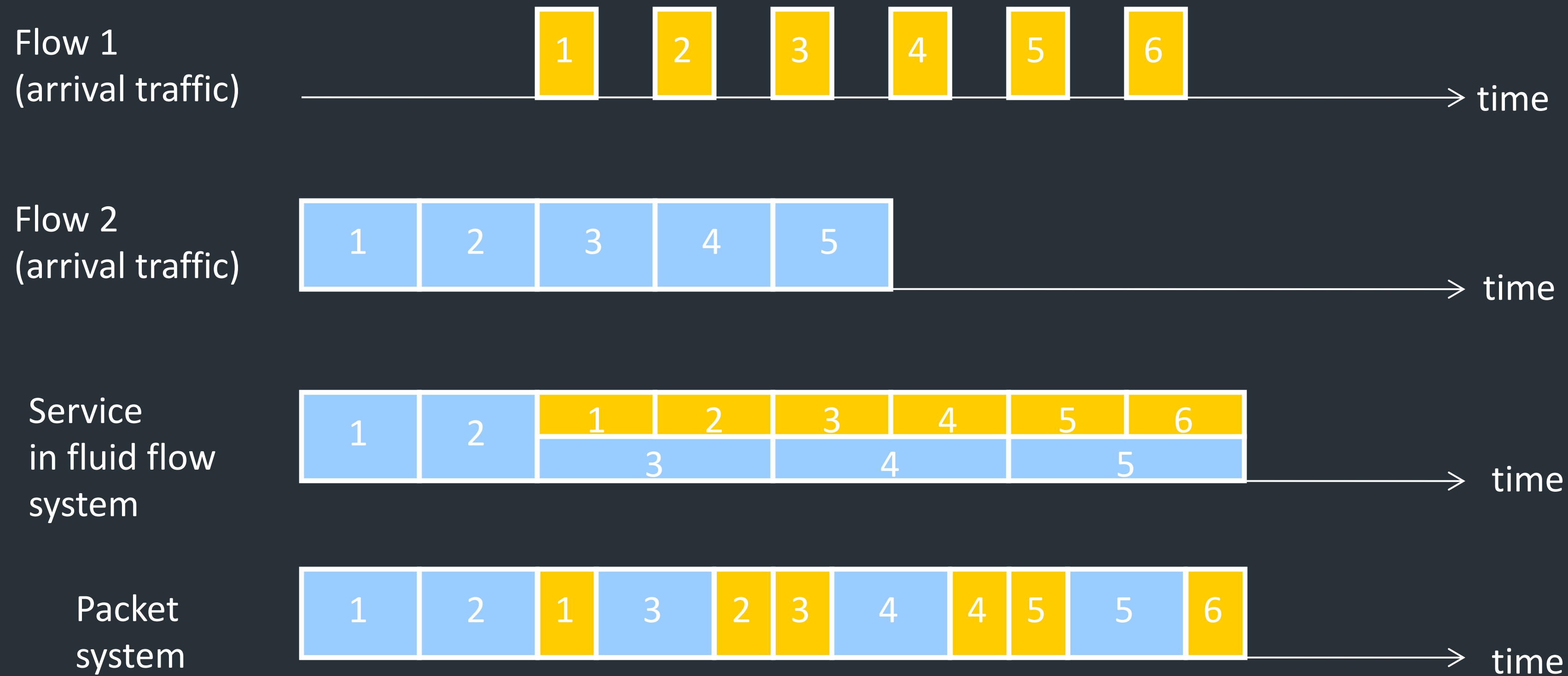
Solution

- Bit-by-bit round robing
- Can we do this?
  - No, packets cannot be preempted!
- We can only approximate it…

Fair Queueing

- Define a *fluid flow* system as one where flows are served bit-by-bit
- Simulate *ff*, and serve packets in the order in which they would finish in the *ff* system
- Each flow will receive exactly its fair share

Example



Implementing FQ

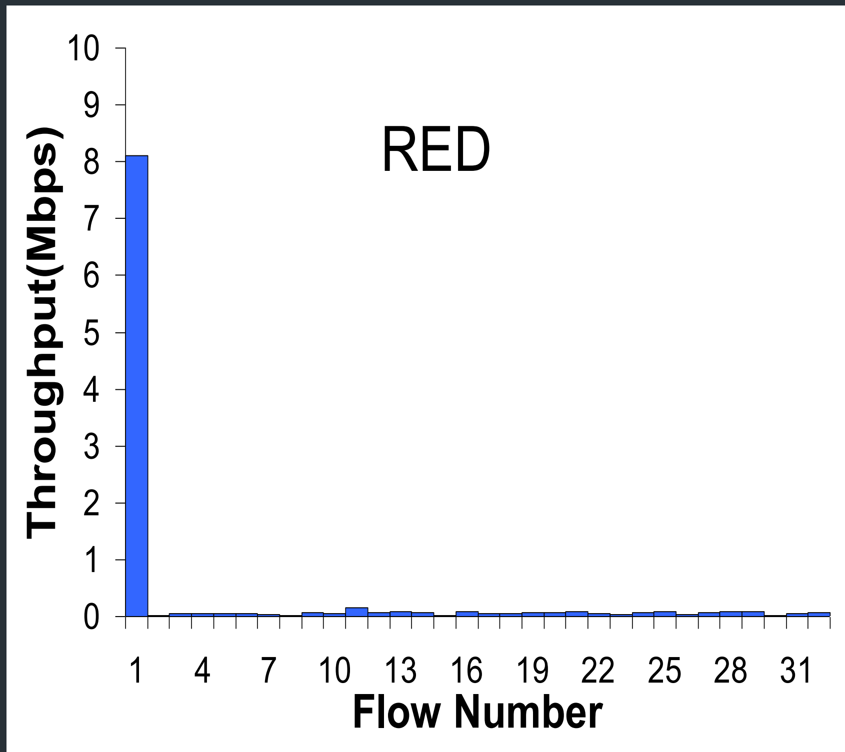
- Suppose clock ticks with each bit transmitted
  - (RR, among all active flows)
- P_i is the length of the packet
- S_i is packet i's start of transmission time
- F_i is packet i's end of transmission time
- $F_i = S_i + P_i$
- When does router start transmitting packet i?
  - If arrived before F_{i-1}, $S_i = F_{i-1}$
  - If no current packet for this flow, start when packet arrives (call this A_i): $S_i = A_i$
- Thus, $F_i = \max(F_{i-1}, A_i) + P_i$

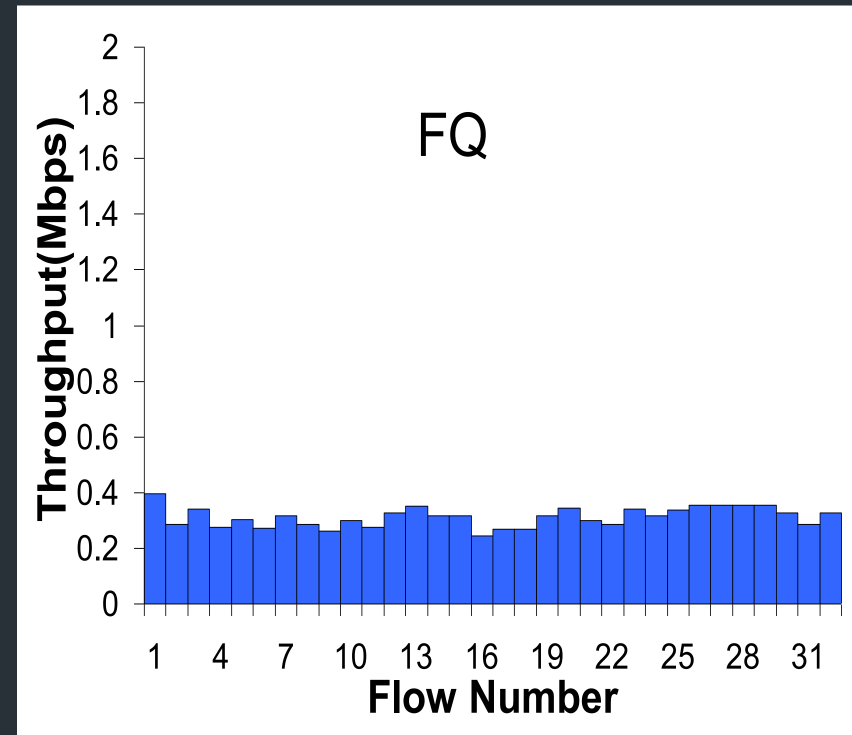
Fair Queueing

- Across all flows
  - Calculate F_i for each packet that arrives on each flow
  - Next packet to transmit is that with the lowest F_i
  - Clock rate depends on the number of flows
- Advantages
  - Achieves max-min fairness, independent of sources
  - Work conserving
- Disadvantages
  - Requires non-trivial support from routers
  - Requires reliable identification of flows
  - Not perfect: can't preempt packets

Fair Queueing Example

- 10Mbps link, 1 10Mbps UDP, 31 TCPs





Big Picture

- Fair Queuing doesn't eliminate congestion: just manages it
- You need both, ideally:
  - End-host congestion control to adapt
  - Router congestion control to provide isolation