Reasoning about Network Performance

COS 316: Principles of Computer System Design
Lecture 9

Wyatt Lloyd

Slides developed by Nicolaas Kaashoek

Review: Performance

Why do we care about performance? Speed matters!

- Cost, competitive advantage, user experience

Goal: Deliver as close to hardware performance as possible

This lecture:

- How to send packets over the internet
- Workloads, deployments, and why they matter
- Compare between TCP Reno* and BBR Congestion control

* Without slowstart

Numbers that Matter

Two metrics we care about: latency and bandwidth

Focus on relative differences, not raw numbers!

Latency (RTT): Physical

Dorm to Cycles: ~1ms

Dorm to NYC: ~10ms

Dorm to VA: ~20ms

Dorm to CA: ~60ms

Dorm to EU: ~100ms

Dorm to JP: ~150ms

Dominated by speed of light

Bandwidth: Hardware

Bluetooth: 3 Mbps

Cellular (5G): ~5 Gbps

Wifi: ~10 Gbps

Ethernet: 100 Gbps

Infiniband: 400 Gbps

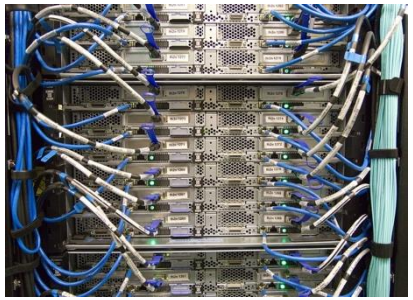
Dominated by technology

Hardware that Matters

Computers/Servers

Senders/Receivers

Many on one network



Switches

Coordinate senders/receivers so they work simultaneously

Routers

Route packets from one network to another

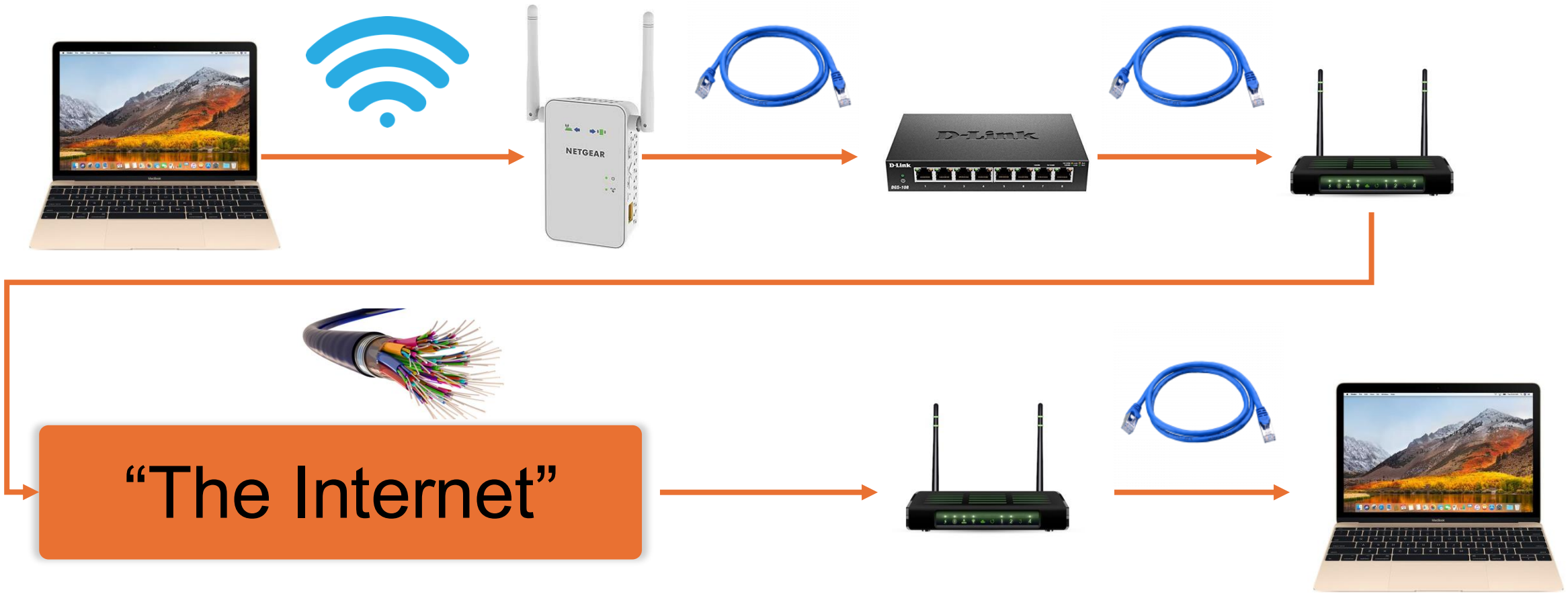
Links

WiFi, Bluetooth, Fiber, Coaxial, etc.

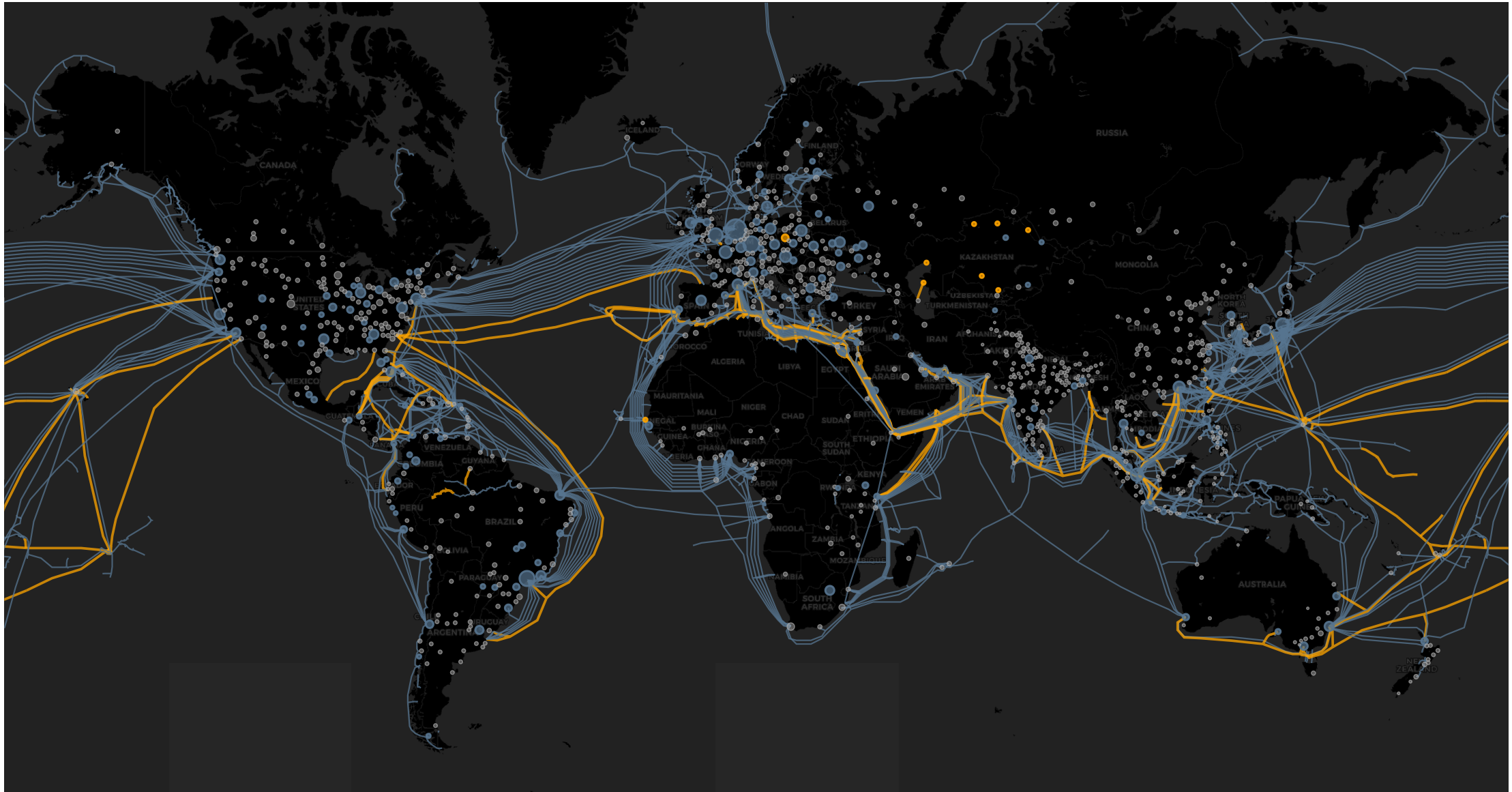
Determines bandwidth

Putting Hardware Together: Internet

How do you send a packet from your computer to someone?



The Internet on Earth



Loss in Networks

Two sources: congestion and error

- Lots of control over congestion, none over error

Wireless errors:

- Bit errors: mostly handle with forward error correction
- Link layer collisions: mostly handle with link layer protocol
- Interference: tell your neighbor to stop their microwave
- Result: variable loss rates from near 0 to quite high

Wired networks:

- Inside datacenter: almost 0
- Across a city: almost 0 (but a bit higher)

Services degrade dramatically at 1% loss. 5% loss is very bad.

Workloads Matter

Scenario 1: Single sender, single receiver

- Relatively easy. No fairness, just need to get our rate right

Scenario 2: Many different senders

- Complex! How do they interact? Fairness?

What metric are we optimizing for? Goodput? Latency?

Takeaway: *Workloads matter!*

Always think about who is using the system and in what situation

Comparing CC Algorithms

What’s the metric that we care about?

- ***End-to-end latency*** of a single task

What are things that contribute to that metric?

- Bandwidth
- Round-trip latency
- Congestion

Scenario 1 – Princeton Campus

Situation: Uploading your homework to TigerFile

How long to get to the TigerFile Server?

- 1 ms

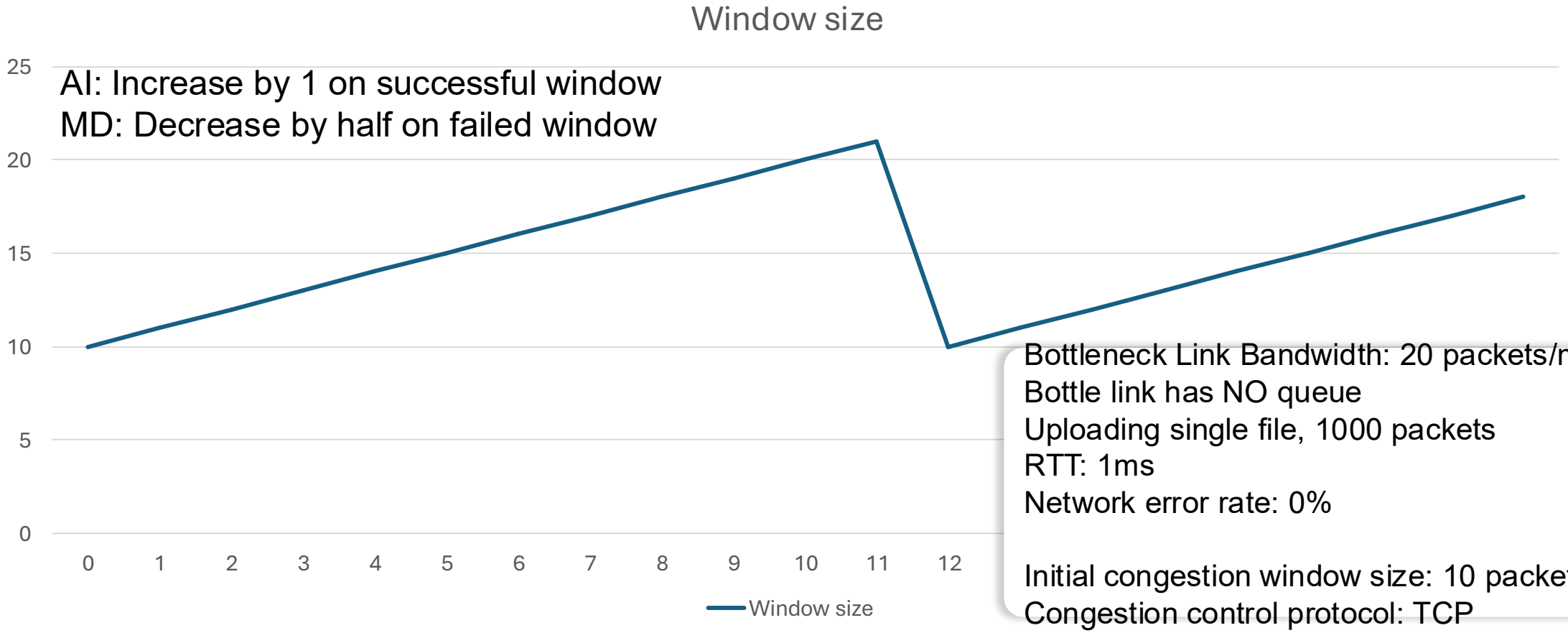
What's our bottleneck?

- 20 packets/ms
- Packet = 1000 bytes

What's our error rate?

- On campus? 0%.

Scenario 1 – TCP Reno, Single Sender



Bottleneck Link Bandwidth: 20 packets/ms
Bottle link has NO queue
Uploading single file, 1000 packets
RTT: 1ms
Network error rate: 0%

Initial congestion window size: 10 packets
Congestion control protocol: TCP

Repeating sawtooth pattern. Send 10, 11, 12, 13, … then cut in half at 21!

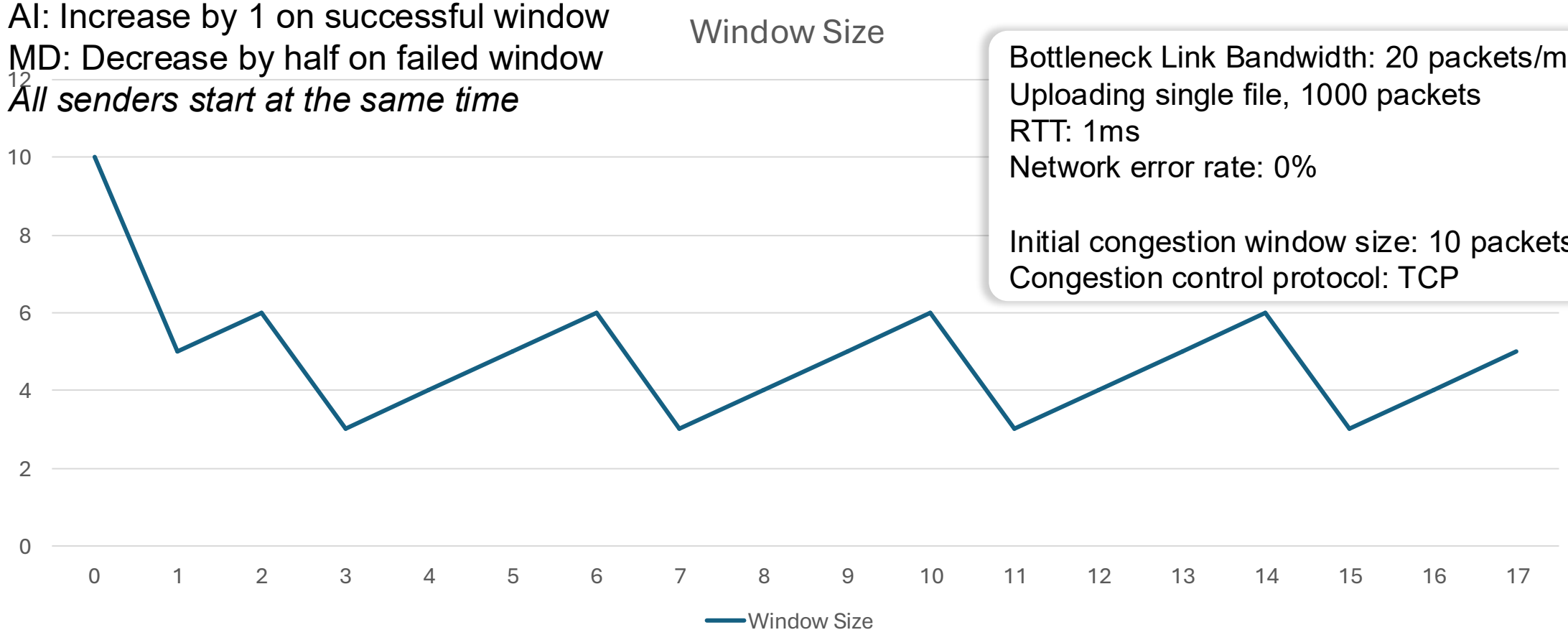
- Every 12ms the pattern repeats. 10 + 11 + 12 + … + 20 + 20 = 185 packets
- 60ms to send 925 packets, 6 to send the remaining 75
- Total time: 66ms

Scenario 1 – TCP Reno, Four Senders

AI: Increase by 1 on successful window
MD: Decrease by half on failed window
All senders start at the same time

Bottleneck Link Bandwidth: 20 packets/ms
Uploading single file, 1000 packets
RTT: 1ms
Network error rate: 0%

Initial congestion window size: 10 packets
Congestion control protocol: TCP



Repeating sawtooth pattern. Send 10, immediate loss! Then 5, 6, overflow again!

- Every 4ms the pattern repeats. 3 + 4 + 5 + 5 + … = 17 packets
- 15 at the start in 3ms. 232ms to send remaining 985 packets
- Total time: 235ms

BBR startup phase

- BBR begins with a startup phase
  - Ramp up: double sending rate until acked data does not increase past a threshold
  - Drain: pace sending at a low rate until data in flight == estimated BDP

- We will use a simple model:
  - Ramp up: $\log_2$(BDP x RTT)
  - Drain: 2*BDP worth of extra data → 2 RTTs to drain
  - Total: $\log_2$(BDP x RTT) + 2 RTTs
  - Assume **maximum** throughput throughout

Scenario 1 – BBR, Single Sender

BBR: First find RTT and Bandwidth estimates

- Model: Takes $\log_2 BDP$ x RTT to find, generates 2 x BDP extra

What's our BDP? RTT x Bottleneck = 20

4.3ms to find estimates! Then we send at 20.

How long does this take in total?

- 6.3 ms to find BDP and drain 40 packets.
- Assumed max tput for 4.3ms = 86 packets
- (1000 – 86)/20 = 45.7 ms to send the rest

Total time: 52ms

Bottleneck Link Bandwidth: 20 packets/ms
Uploading single file, 1000 packets
RTT: 1ms
Network error rate: 0%

Initial congestion window size: 10 packets
Congestion control protocol: BBR

Scenario 1 – BBR, Four Senders

Paper: Takes "a few" cycles to synchronize. Let's say 3.

What's our BDP now?

- BBR is fair, everyone gets equal share of bottleneck link
- BDP = 5

How long will it take us to send all the packets?

Bottleneck Link Bandwidth: 20 packets
Uploading single file, 1000 packets
RTT: 1ms
Network error rate: 0%
Probe time: $\log_2 BDP$ x RTT + 2 RTT for draining
Initial congestion window size: 10 packets

Scenario 1 – BBR, Four Senders

BDP = 5, Probe time = 2.3ms, Overflow packets = 10

3 cycles to sync, for a total of 6.9 ms. That generates 30 packets of overflow, while sending 6.9 * 5 = ~34 packets.

Drain overflow: 30 packets/5 = 6 RTTs = 6ms (wait this out)

Time to send remaining: (1000 – 34)/5 = 193.2ms.

Total time: 193.2 + 6.9 + 6 = ~206ms

Bottleneck Link Bandwidth: 20 packets/ms
Uploading single file, 1000 packets
RTT: 1ms
Network error rate: 0%

Initial congestion window size: 10 packets
Congestion control protocol: BBR

Comparing BBR and TCP

Single sender TCP: 66ms

Four senders TCP: 235ms

Single sender BBR: 52ms

Four senders BBR: 206ms

Bottleneck Link Bandwidth: 20 packets/ms
Uploading single file, 1000 packets
RTT: 1ms
Network error rate: 0%

Initial congestion window size: 10 packets

Key point: Performance isn't magic!

What happens as packets in transmission increases?

Comparing TCP and BBR 2

Let's add some more realistic numbers

- Packet size: 1000 bytes.
- Bottleneck = 100 MB/s, ~100 packets/ms

Bottleneck Link Bandwidth: 20 packets/ms
Uploading single file, 1000 packets
RTT: 1ms
Network error rate: 0%

Initial congestion window size: 10 packets

TCP Sawtooth: 50 + 51 + … + 100 + 100 = 3925 packets/52ms, or ~75 packets/ms

BBR: Sends at a flat 100 packets/ms

How long to send a MB file (1 thousand packets)?

How long to send a GB file (1 million packets)?

Scenario 2 – Sending Across the WAN

New scenario: Studying abroad at Cambridge. Uploading homework to TigerFile.

How long to get from Cambridge to Princeton?

What's our bottleneck link?

- Again, hard to say. Let's use the same: 100 MB/s

What's our loss rate?

- Assume 1% for demonstrative purposes

Scenario 2 – TCP

Bottleneck Link Bandwidth: 100 packets/100ms
RTT: 100ms
Network error rate: every 100th packet

Lossless steady state sending rate: 75 packets/100ms

What does TCP do when it sees a lost packet?

- *Multiplicative decrease*

Contrived loss scenario: every 100th packet is lost

What does the steady state look like?

Prev: 50, 51, …

Now: 50, 51, 25, 26, 27, 28, 14, 15, 16, 17, 18,

Significant decrease in throughput

Scenario 2 – BBR

Bottleneck Link Bandwidth: 100 packets/100ms
RTT: 100ms
Network error rate: 1%

Lossless steady state sending rate: 100 packets/100ms

How about BBR? What does 1% packet loss do?

BBR uses max—min to estimate network properties
- 1 missed packet in every 100 won't change that!

BBR won't lose any throughput under 1% loss
(goodput is 99 packets/100ms due to the error rate)

Tradeoff: Fairness

We looked at TCP compared to BBR

Conclusion: BBR outperforms TCP in nearly every situation

- But there’s a cost – fairness

BBR looks fantastic when everyone’s using BBR. What about when they’re not?

Networks are Decentralized

Fairness is a real tradeoff – know nothing about network

- Congestion control is *decentralized*

BBR works particularly well when everyone uses BBR

- When can we guarantee this? In datacenters

It’s amazing this works at all!

Tradeoffs – The Pantheon

See: pantheon.stanford.edu, a testing ground for *many* congestion control algorithms

Compares many modern congestion control algorithms

Comparison across many different situations/deployments

Pantheon Results

Results taken from 2020 (no longer running)

Date (UTC)	Description	TCP BBR	Copa	TCP Cubic	FillP	FillP-Sheep	Indigo	LEDBAT	PCC-Allegro	PCC-Expr	QUIC Cubic	SCReAM	Sprout	TaoVA-100x	TCP Vegas	Verus	PCC-Vivace	We m
04/17/2020	GCE London to GCE Iowa, Ethernet																	
04/17/2020	GCE Sydney to GCE Tokyo, Ethernet																	
04/17/2020	GCE London to GCE Iowa, Ethernet																	
04/17/2020	GCE Sydney to GCE Tokyo, Ethernet																	
04/16/2020	GCE Sydney to GCE London, Ethernet																	
04/16/2020	GCE Iowa to GCE Tokyo, Ethernet																	
04/16/2020	GCE Iowa to GCE Tokyo, Ethernet																	
04/16/2020	GCE Sydney to GCE London, Ethernet																	
04/16/2020	GCE Tokyo to GCE London, Ethernet																	

Conclusion

Internet infrastructure is complex, multilayered, and interconnected

Networks are complicated: performance reasoning is hard

- Measure, measure, measure!
- Networks have many *emergent properties*

Workloads/deployment scenarios matter.

Always consider the tradeoffs of one system over another.

- Not always directly performance related!