Reasoning about Network Performance

COS 316: Principles of Computer System Design
Lecture 9

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

Numbers that Matter

Two metrics we care about: latency and bandwidth

Focus on relative differences, not raw numbers!

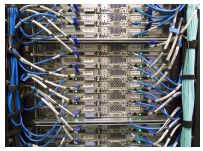
Latency (RTT): Physical	Bandwidth: Hardware
Dorm to Cycles: ~1ms	Cellular (5G): ~3 Gpbs
Dorm to NYC: ~10ms	Wifi: ~3 Gbps
Dorm to VA: ~20ms	Ethernet: 10 Gbps
Dorm to CA: ~60ms	Infiniband: 400 Gpbs
Dorm to EU: ~100ms	
Dorm to JP: ~150ms	
Dominated by speed of light	*Dominated by technology*

Hardware that Matters

Computers/Servers

Senders/Receivers

Many on one network



Switches

Coordinate senders/receivers so they work simultaneously

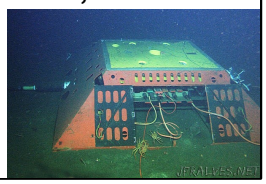
Routers

Route packets from one network to another

Links

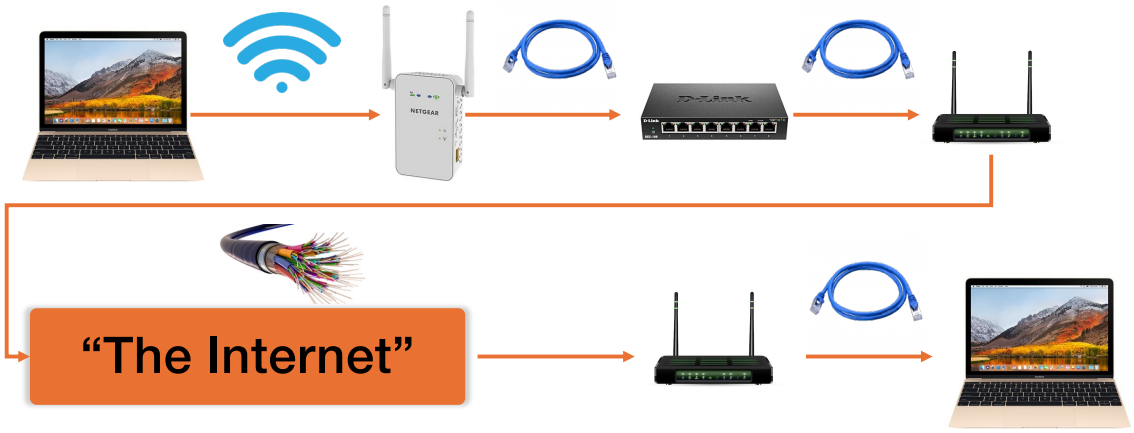
WiFi, Bluetooth, Fiber, Coaxial, etc.

Determines bandwidth

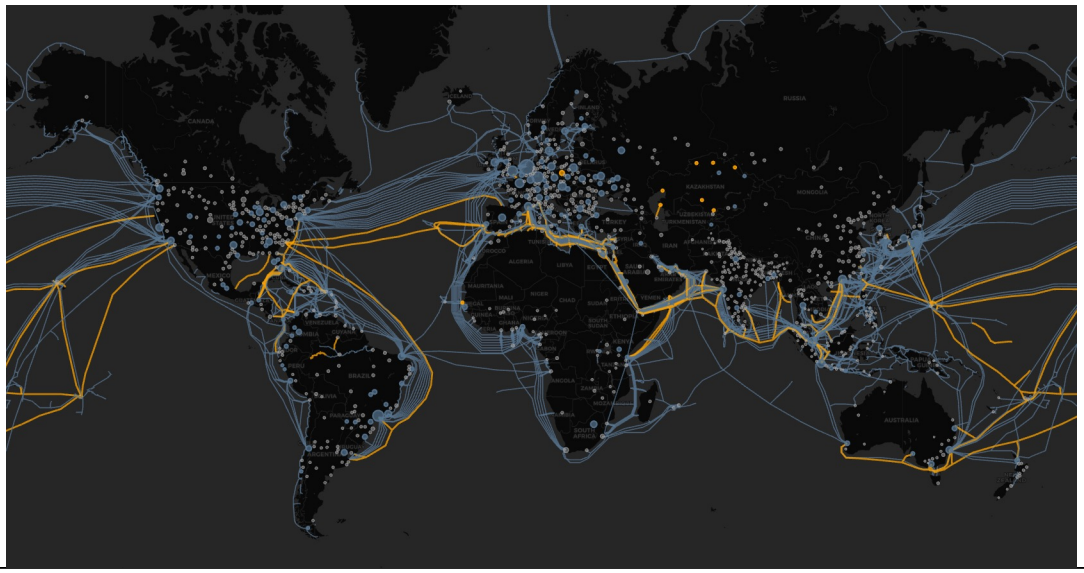


Putting Hardware Together: Internet

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:

- WiFi in dorm: 10^{-8} BER
- 5G in park: 10^{-5} BER

Wired networks:

- Inside datacenter: 0% BER
- Across a city: 10^{-10} BER

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

Someone pointed out after lecture that a BER of 10^-5 would result in nearly every packet being statistically likely to see some corruption. I forgot to mention that networks implement forms of error correction, such as Barker codes in WiFi. This isn't important to know, but it's why things function even with BER, and why those don't correspond to super high rates of packet corruption in the real world. My apologies for skipping over that in lecture!

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? Throughput? 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

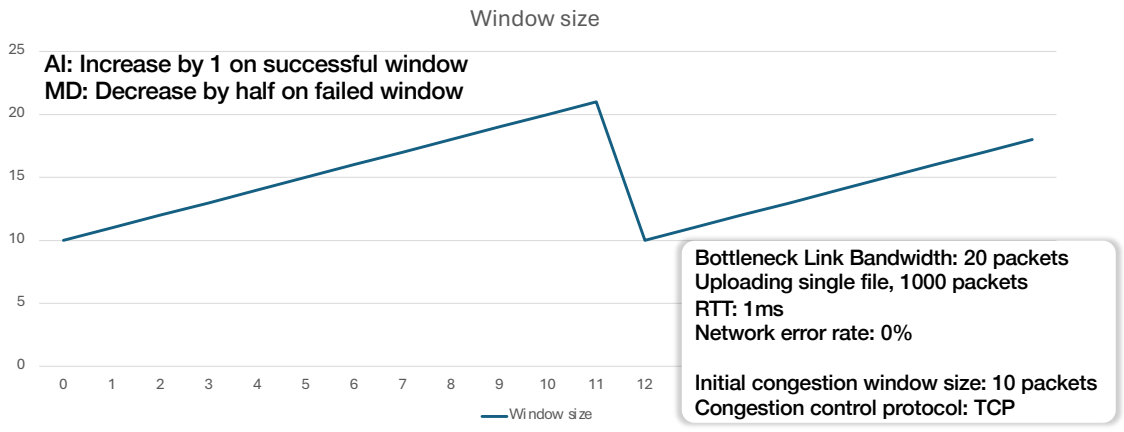
Situation: Uploading your homework to TigerFile

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



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

Here we have one sender and one receiver.

We're trying to get 1000 packets through, and don't start dropping anything until we get a window to size 21.

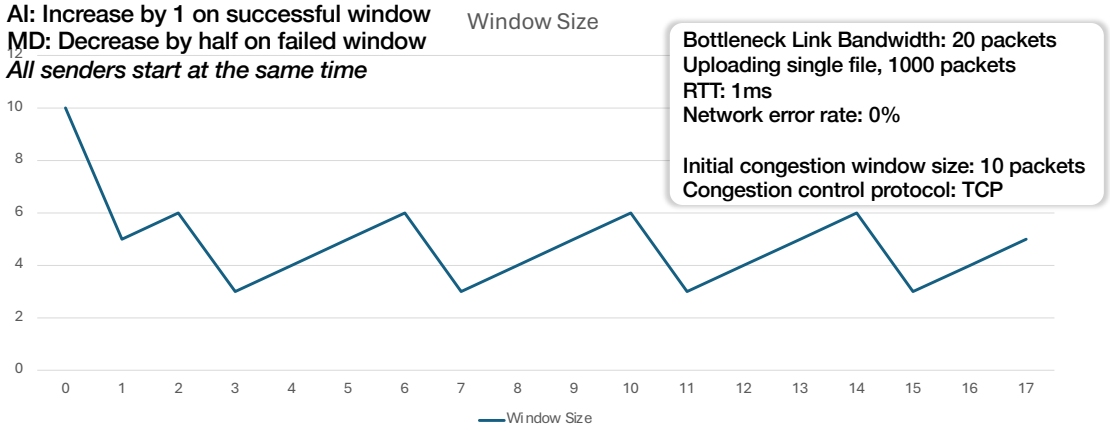
So, we send 10 + 11 + 12 + 13 + 14 + 15 + 16 + 17 + 18 + 19 + 20 + 21, one of which isn't delivered, for a total of 185 packets in 12 ms. This pattern then repeats.

Floor(1000/185) = 5. 5 * 185 = 925 and 5 * 12 = 60, so in 60ms we send 925 packets, which leaves 75. 10 + 11 + 12 + 13 + 14 + 15 = 75, so it takes another 5ms to finish the transmission, for a total of 66ms.

There's probably a fences and posts error here where each cycle actually drops 1 packet which has to be included in the next repetition of the pattern, but this calculation isn't going to be off by more than a ms, and I think it nicely illustrates what happens in TCP without getting too complex.

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
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

Here we treat each sender as identical, and are assuming that packets arrive at the link at the same time. That is to say that if sender 1 sends 5 packets, and sender 2 sends 5 packets, and so on, then our link will receive them in the order: s1_p1, s2_p1, s3_p1, s4_p1, s1_p2, s2_p2, s3_p2, s4_p2, …

Here sX_pY means packet Y from sender X.

We make this assumption to make reasoning easier, as it ensures that all the senders behave in an identical manner. In a real network, it's more likely you'd see more chaotic interleaving of the packets.

The math here is relatively straightforward: we send 10 packets. 5 of those get through. Then 5 more, which all get through, then 6, 5 of which get through. So, in 3ms, we get 15 packets of goodput through the link. That's the 15 in 3ms.
This leaves us with 985 packets left. The repeating pattern of 3+4+5+5=17 packets will get 986 packets through in 232ms (ceiling(985/17) * 4).

So, our total is 235 ms.

Again, there might be a fences and posts error here with some packets rolling over into a new sending window, but for illustrative purposes this is good enough.

Scenario 1 – BBR, Single Sender

BBR: First find RTT and Bandwidth estimates

- Paper: 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.

- (1000 – 126)/20 = 43.7 ms to send the rest

Total time: 50ms

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

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

BBR is a slightly more complex case. The paper gives us two important pieces of information about how long it takes BBR to get a correct estimate of of the RTT and the bottleneck bandwidth.

First, it does a binary search to find the link bandwidth/rtt. This takes log_2 of the actual BDP (which is just RTT * bottleneck bandwidth), multiplied by the RTT to occur. In our situation, that's log_2 20 * 1ms = 4.3ms.

Additionally, during this process, BBR will overflow the link to be sure it finds the max throughput. In a real deployment, this means that it's going to enqueue some packets ahead of the bottleneck. To avoid maintaining that queue during normal operation, BBR needs to wait for it to drain. In our example here, where I've set the situation to have no queues, that means the sender will need to retransmit those packets that were dropped. Luckily, BBR tells us exactly how many of those packets exist: 2 * BDP, or 40 in our case.

With this, we know it's going to take 4.3 ms to get our initial, correct sending rate. We'll also generate 40 extra packets. Sending at 20 packets/ms (our bottleneck throughput) this takes another 2ms to finish, for 6.3ms total to find our

estimates. In those 6.3ms, we'll have sent 6.3 * 20 = 126 packets.

This leaves us with 1000 – 126 = 874 packets left, which we'll send at exactly 20 packets/ms. This takes 874/20 = 43.7ms, for a total time of 50ms.

I'm making a key assumption about BBR here, namely that it's sending goodput when it initially forms its estimates. I believe this to be true, but it may be the case that they're using small packets or something just to get the estimate. If the initial estimation step doesn't use our payload, then it'll take 6.3 + 1000/20 = 56.3ms to finish the transmission instead

Scenario 1 – BBR, Four Senders

Paper: Takes “a few” cycles to synchronize. Let’s say 3.

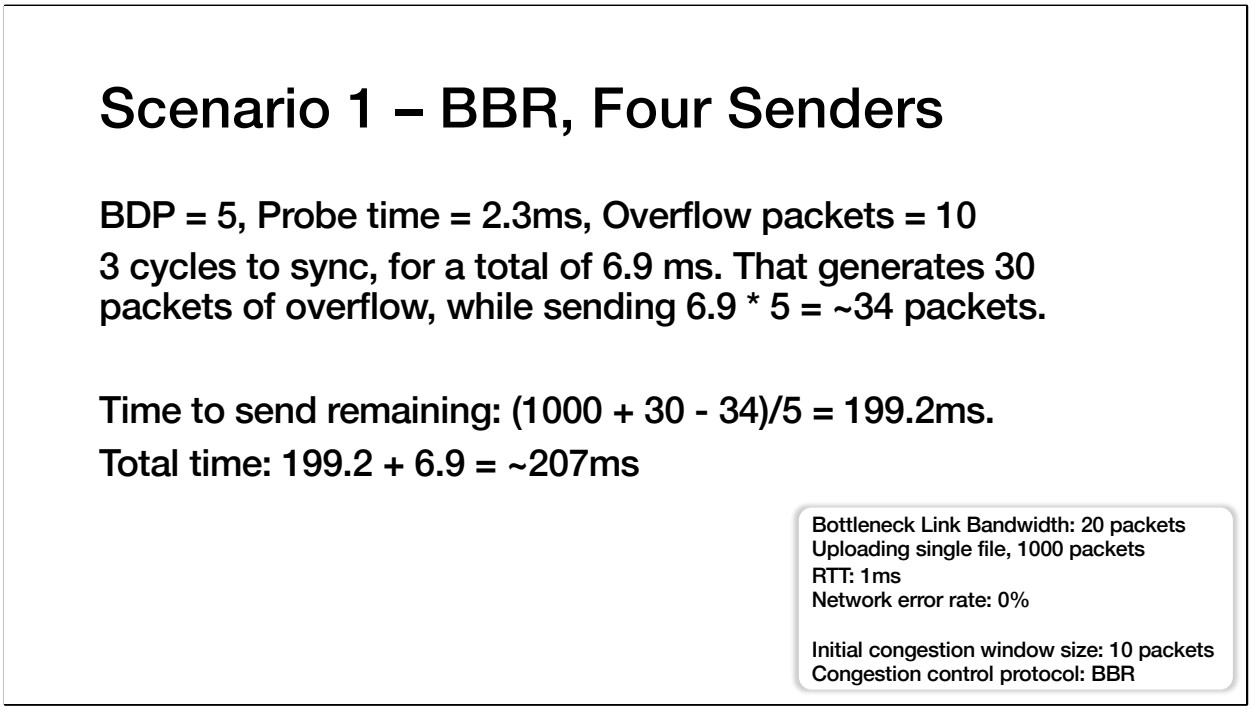
What’s our BDP now?

- 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
Initial congestion window size: 10 packets

See next slide.



I made a mistake during lecture with this slide, and have corrected it now.

Our new BDP here is 5, because we're only getting 5 packets/ms on the link (our fair share of the bandwidth when there's four senders).

I'm assuming *this* is the number BBR uses to get its initial estimates, which should be correct if everyone is estimating at the same time. If not, then we should use the 20 number from the previous slide instead. Let's proceed with 5 as the BDP for now.

There's no way to easily predict how many estimation cycles it'll take for all the senders to get the right estimates. 3 is a number I've picked that seems reasonable, but it'll change depending on when each sender joins the network among other factors.

It now takes us log_2(5) = 2.3ms to find the estimates. 2.3 * 3 = 6.9ms for all the senders to get the right estimates. If we assume that we can actually get goodput during this time, they'll each have sent 6.9 * 5 = 34.5 packets, rounded down to 34. We round down here because it makes the situation worse for BBR, so is

conservative.

During each of these estimation steps, BBR will generate a maximum of 2 x BDP extra packets. Let's assume worst case again and say that we've generated 3 x 2 x BDP = 30 extra packets.

So, after estimating, we need to transmit: 1000 + 30 - 34 = 996 packets, sending at a rate of 5 packets/ms.
996/5 = 199.2ms. Add our estimation time, and we get a total of 206.1ms. Round that up to 207 to be conservative.

Comparing BBR and TCP

Single sender TCP: 66ms
Four senders TCP: 235ms

Single sender BBR: 50ms
Four senders BBR: 207ms

Bottleneck Link Bandwidth: 20 packets
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?
What if we increase link bandwidth?

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
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
RTT: 100ms
Network error rate: 1%

Steady state sending rate: 75 packets/100ms

What does TCP do when it sees a lost packet?

- *Multiplicative decrease*

1 in every 100 packets is lost. For analysis, assume always 100th packet.

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

Here we're simplifying slightly to illustrate a point. As someone noted on Ed, with a 1% packet loss rate, it's unlikely TCP would ever reach a sawtooth that oscillates its sending window from 50 to 100 packets. Rather, I've just constructed a comparison between the steady state performance on a link with no loss (the earlier slides) and compared what would happen if there was suddenly 1% loss on the link.

Scenario 2 – BBR

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

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 goodput under 1% loss.

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	Web
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																	
04/16/2020	GCE Sydney to GCE Iowa, Ethernet																	
04/16/2020	GCE Tokyo to GCE London, Ethernet																	
04/16/2020	GCE Sydney to GCE Iowa, Ethernet																	
02/19/2020	GCE London to GCE Iowa, Ethernet																	
02/19/2020	GCE Sydney to GCE Tokyo, Ethernet																	
02/18/2020	GCE London to GCE Iowa, Ethernet																	
02/18/2020	GCE Sydney to GCE Tokyo, Ethernet																	
02/18/2020	GCE Iowa to GCE Tokyo, Ethernet																	
02/18/2020	GCE Sydney to GCE London, Ethernet																	
02/18/2020	GCE Sydney to GCE London, Ethernet																	
02/18/2020	GCE Iowa to GCE Tokyo, 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!