Intro to Project-3

CSE 461 Computer Networks

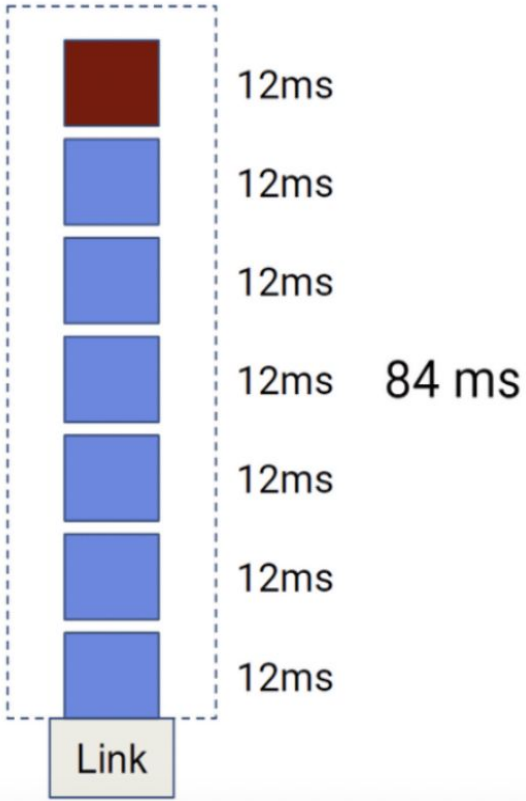
Bufferbloat

“Bufferbloat is a cause of high latency in packet-switched networks caused by excess buffering of packets” – Wikipedia

Bufferbloat can happen anywhere on a network where packets can queue. Every network has a spot in the path that’s naturally slower, and that’s where bufferbloat will rear its ugly head. Typically, this happens at a bottleneck link where many packets are queued up.

Queues and Latency

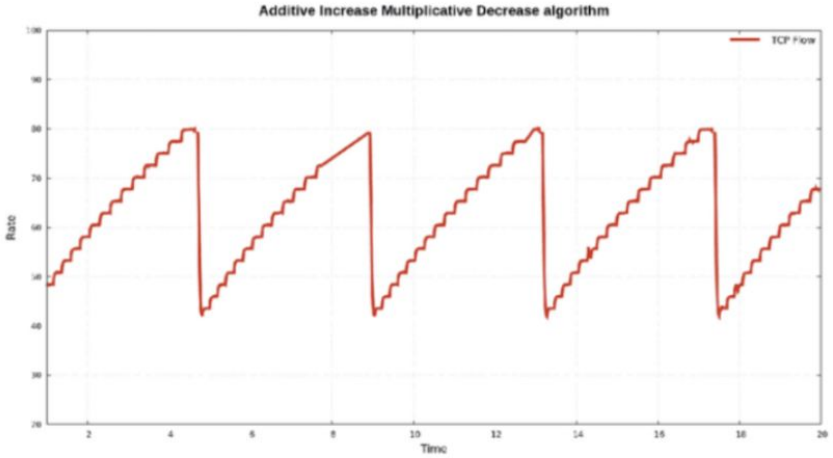
Example
- 1Mbps link
- 1500 byte packets
- Therefore, it takes 12 ms to transmit a packet (write it to the wire)



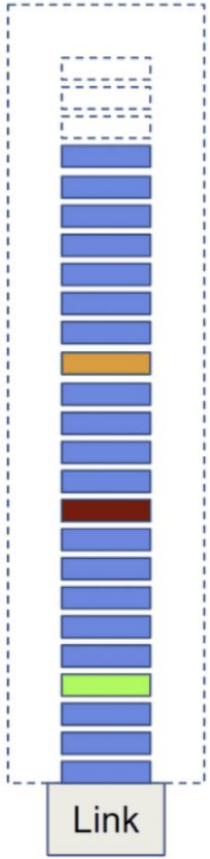
- The last packet in the queue has to wait for all the others to be transmitted
- This is where high latency under load comes from
- Often physical interfaces have queues thousands of packets deep
  - Until recently, Linux defaulted to a 1000 packet queue on each interface
    - In this example that's up to **12 seconds** of latency
  - Router ports often target at least 200 ms of buffering

Consider the user experience when the red packet is important to the the customer experience (eg DNS request)

Big Buffers and TCP



- TCP 'learns' the rate the network can support by probing (increase bandwidth until loss, then back off)
- Deep queues delay the packet loss event causing TCP to push more packets into the network than it can deliver



- The interaction of TCP congestion control and deep buffers can result in a 'standing queue' for any individual TCP flow
- This phenomenon adds latency to every packet that flows through the NIC as long as the TCP connection is active
- In this example, the blue TCP flow's congestion algorithm has settled at a point that has 19 packets in the packet queue at all times

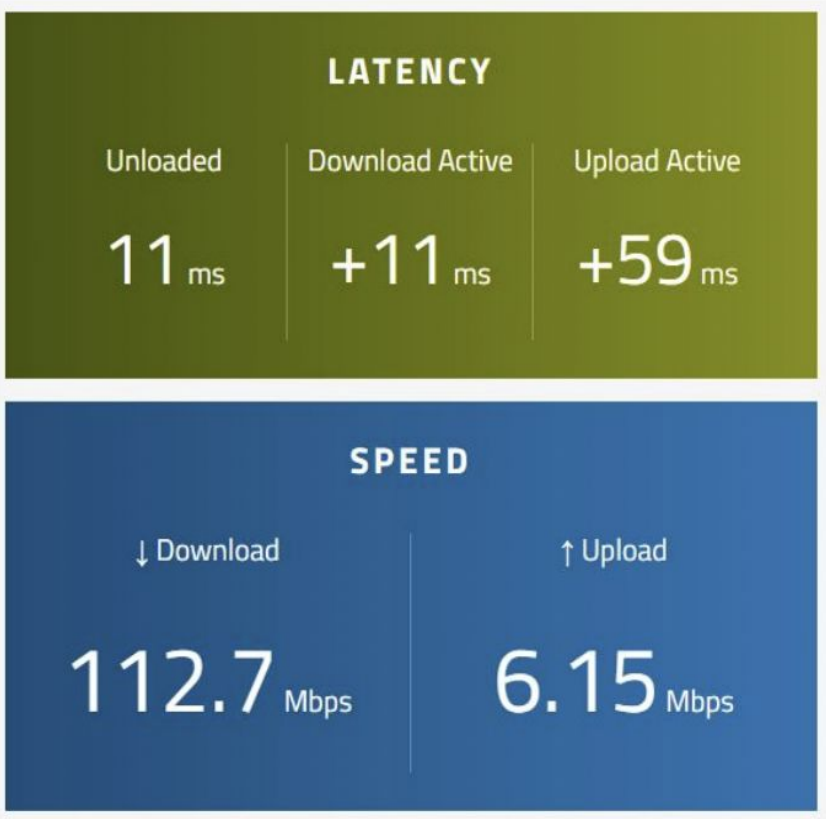
A motivational example...

Not all “speedtests” capture bufferbloat... took a long time for the networking community to realize it was a problem!

- A regular “ping” test, used to measure RTT in practice, won’t fill the buffers!

Let’s as a class try it out:

- https://www.waveform.com/tools/bufferbloat
  - Loaded latency vs. unloaded latency
  - How big is the difference?



Real World Initiatives

Active Queue Management (AQM)

- Goal is to use better queue management techniques.
- Leverage ECN (Explicit Congestion Notification) to give fast feedback without causing loss
    - ECN allows end-to-end notification of network congestion without dropping packets. Unfortunately hard to deploy ECN CC “fairly” with existing CC algos (Reno, Cubic)
    - It works so much more responsively (aka better) it tends to takeover throughput from legacy TCP!

L4S “Low-latency, low-loss, scalable throughput” initiative at IETF

- One solution is to mandate a split at bottlenecks, two queues with independent behavior
- Required in latest cable modem standards

One AQM Technique: FQ_CODEL

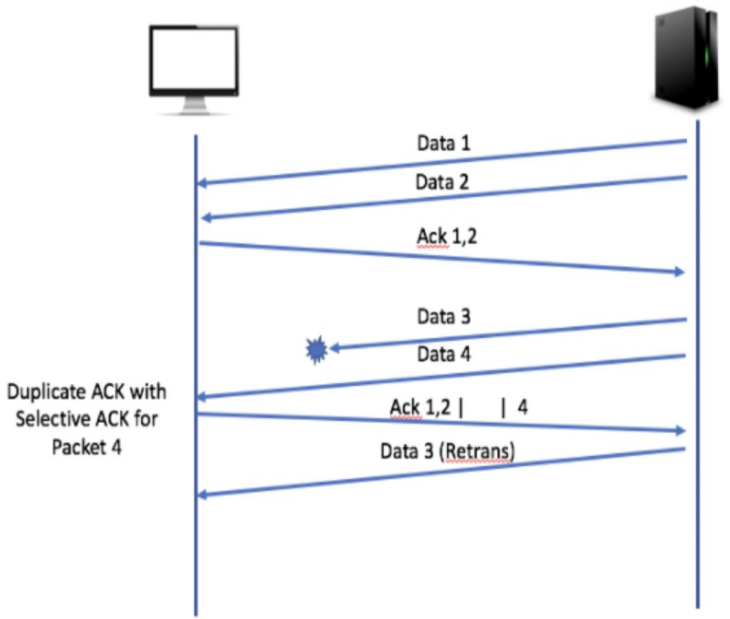
- Initiatives to add flow-independent queues to bottleneck routers…
  - like L4S to an extreme…
  - Each flow gets its own queue, and it’s the router’s job to make them all fair!
  - Attempts to estimate bottleneck and not queue any more than necessary to fill the pipe
- Similar big idea to BBR
- Available in all modern Linux distros (kernel > 3.16)
  - Default in some
- Default in OpenWRT
  - Used as the basis for some commercial routers too (SpaceX Starlink is a prominent example)
- Relatively resource intensive though, so not feasible on “core” routers yet

TCP - detecting, reacting to loss

- Loss indicated by timeout
  - cwnd is set to 1.
  - Window then grows exponentially (as in slow start) to threshold, then grows linearly
- Loss indicated by 3 duplicate ACKs: TCP Reno
  - Dup Acks indicate that network is capable of delivering some segments
  - cwnd is cut in half window and then grows linearly
- TCP Tahoe always set cwnd to 1. (Timeout or 3 duplicate acks)

What is a Duplicate ACK

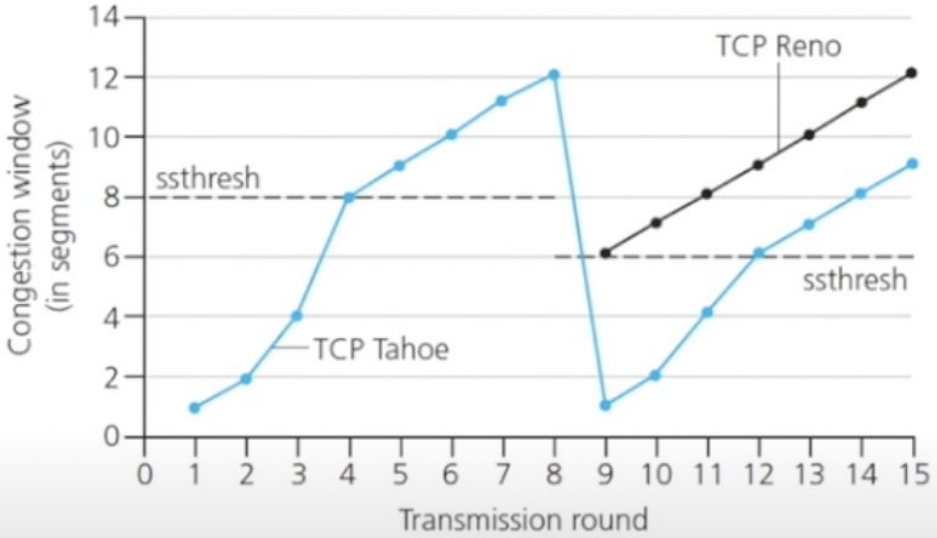
Most packet analyzers will indicate a duplicate acknowledgment condition when two ACK packets are detected with the same ACK numbers.



TCP: switching from slow start to CA

Q: when should the exponential increase switch to linear?

A: when **`cwnd`** gets to 1/2 of its value before timeout.



Implementation:

- variable **`ssthresh`**
- on loss event, **`ssthresh`** is set to 1/2 of **`cwnd`** just before loss event

Bufferbloat aware transport: BBR

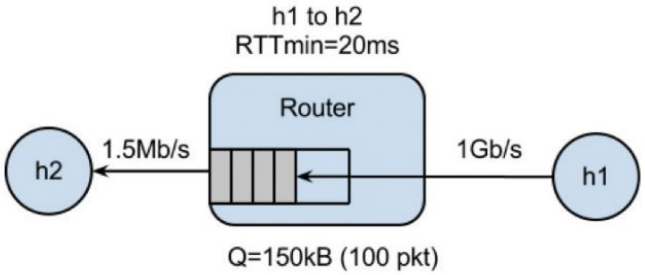
- Different type of solution than AQM
  - Operates only on end hosts
- Developed at Google in 2016 for YouTube traffic.
- Google started from scratch, using a completely new paradigm: to decide how fast to send data over the network, BBR considers how fast the network is delivering data.
- For a given network connection, it uses recent measurements of the network's delivery rate and round-trip time to build an explicit model that includes both the maximum recent bandwidth available to that connection, and its minimum recent round-trip delay. BBR then uses this model to control both how fast it sends data and the maximum amount of data it's willing to allow in the network at any time.

Project 3 – Goal

- Simulate bufferbloat problem.
- See the worse performance when queue size is larger
- See the difference between TCP Reno and TCP BBR.

Experiment Setup

- Long-lived TCP flow from h1 to h2
  - Simulate background traffic
- Back-to-back ping from h1 to h2
  - Measure RTT
- Spawn a webserver on h1 and periodically fetch a page
  - Simulate more important load
  - Measure time
- Plot time series of RTT and number of queued packets.



- Run the experiment with
  - Q=20 and Q=100
  - Reno and BBR
  - 4 experiments total

Setup

- Use Mininet VM (same as Project 2)
- Get the starter code and install dependencies

cd ~

wget

https://courses.cs.washington.edu/courses/cse461/23wi/projects/project3/resources/project3.zip

unzip project3.zip

sudo apt-get update

sudo apt install python3-pip

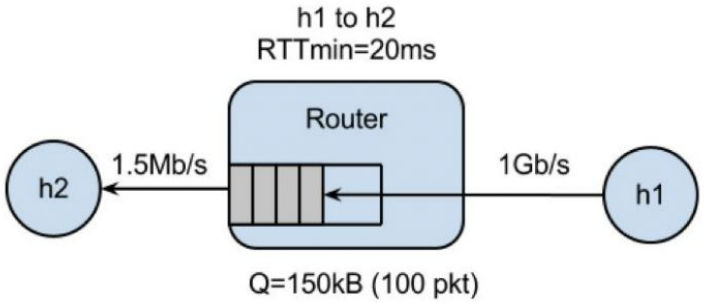
sudo python3 -m pip install mininet matplotlib

Starter Code

- run.sh
  - Run the entire experiment
    - Run bufferbloat.py on q=20 and q=100
    - Generate latency and queue length graphs
- bufferbloat.py
  - Complete the TODOs
    - Setup the mininet topology and the experiment
    - Write shell commands to do the measurements

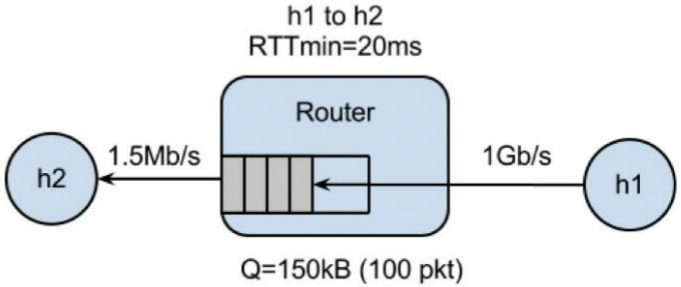
Long-lived TCP Flow

- Starter code sets up iperf server on h2
- Goal: start iperf client on h1, connect to h2
  - Should be “long-lasting”, i.e. for time specified by --time parameter
- How do I connect to a certain IP or make the connection long-lasting?
  - man pages are your friend!
  - type `man iperf` in a Linux terminal



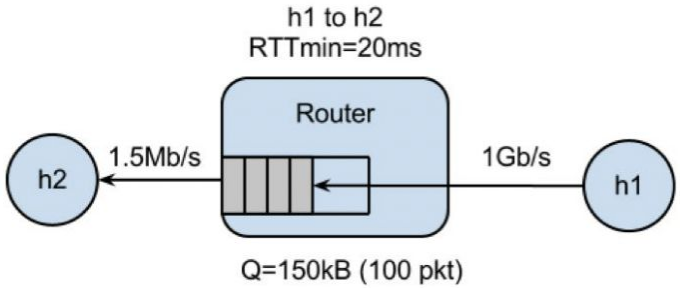
Ping Train

- Goal: Start “ping train” between h1 and h2
  - Pings should occur at 10 per second interval
  - Should run for entire experiment
- How do I specify the ping interval and how long the ping train runs?
  - man pages are your friend!
  - type `man ping` in a Linux terminal
- Write the RTTs recorded from `ping` to {args.dir}/ping.txt
  - See starter code comments for more detail



Download Webpage with curl

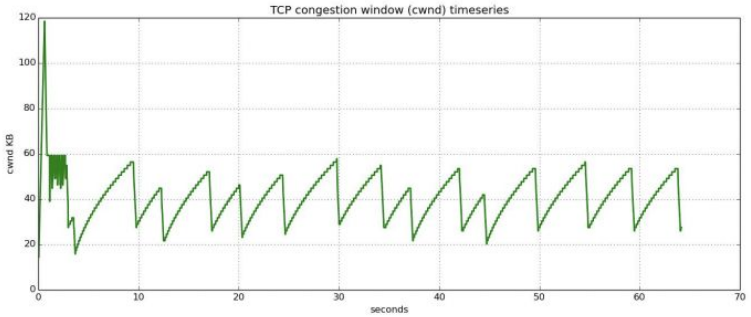
- Starter code spawns webserver on h1
- Goal: Use `curl` to measure fetch time to download webpage from h1
  - Starter code has hint on formatting curl command
  - Make sure `curl` doesn't output an error
    - Errors report very small latency
- No need to plot fetch times; just need to report average fetch time for each experiment.



Plotting

- Starter code contains scripts for plotting, `plot_queue.py`, `plot_ping.py`
  - Expects queue occupancy in $dir/q.txt, ping latency in $dir/ping.txt
  - Plots are useful for debugging!
- Part 3, run same experiments with TCP BBR instead of TCP Reno
  - How do you expect the graph outputs to differ?

Q = 20



Q = 100