Section 7: Project 3 Intro

CSE 461 Computer Networks

Administrivia

- HW5 out, due next Thursday.
- Project 3 released, due on March 13th
  - **This is a hard deadline!**
- Final exam on March 8th!!!

Project 3: Bufferbloat

What is Bufferbloat?

From Wikipedia, “bufferbloat is a cause of high latency in packet-switched networks caused by excess buffering of packets”

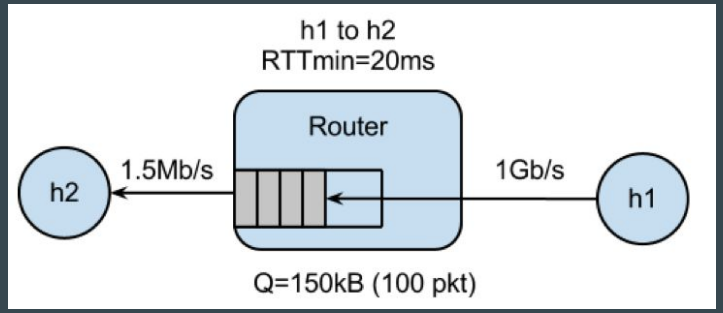
Queue

Project 3

- We will simulate bufferbloat on our mininet network, compare TCP Reno and TCP BBR, and plot the latency and queue length graphs
- The setup is similar to project 2
  - Mininet on the Vagrant VM
  - Python3
  - Given a skeleton code to modify. Don’t forget to check other files which might contain useful helper functions

Project 3: Part 1

- Part 1: Topology Setup
  - Similar to project 2 part 1
  - Except need to specify link characteristics (bandwidth, minimum RTT, max queue size)
  - Look into Mininet documentation!



Project 3: Part 2 & 3

- Part 2: TCP Reno
  - Modify
    - `run.sh`
      A script that runs the experiment with specified parameters
      - Run `bufferbloat.py` on q=20 and q=100
      - Generate latency and queue length graphs
    - `bufferbloat.py`
      Setup the mininet topology and the experiment
      - Write shell commands to do the measurements

- Part 3: TCP BBR
  - Modify Part 2 to run the experiment using BBR

The Experiment

Complete `bufferbloat.py` to run the following in parallel

- Long-lived TCP flow between h1 and h2 (iperf/iperf3)
  - Fills bottleneck router

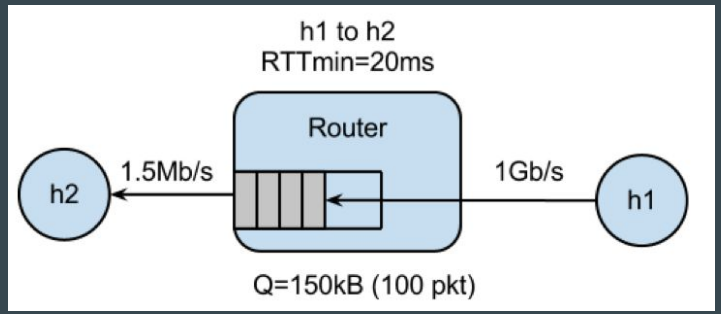
- Ping train between h1 and h2
  - Measure latency between hosts

- Measure time to `curl` down webpage from h1

Goal: See how queue size behaves under congestion, and how that affects latency/download times

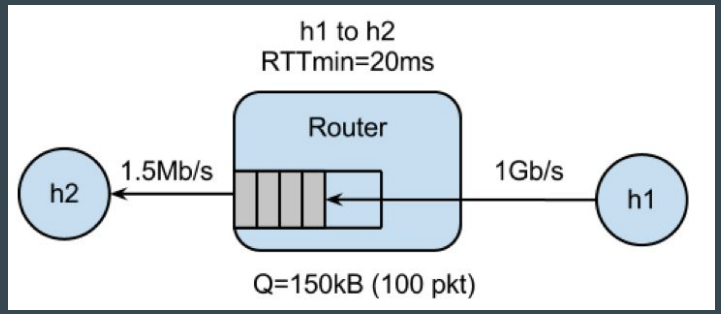
Long-lived TCP Flow

- 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



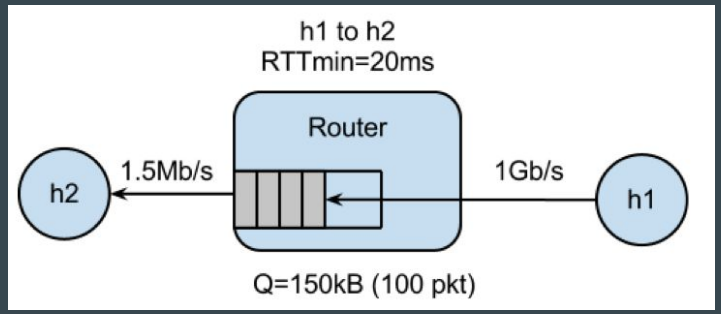
Ping Train

- 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

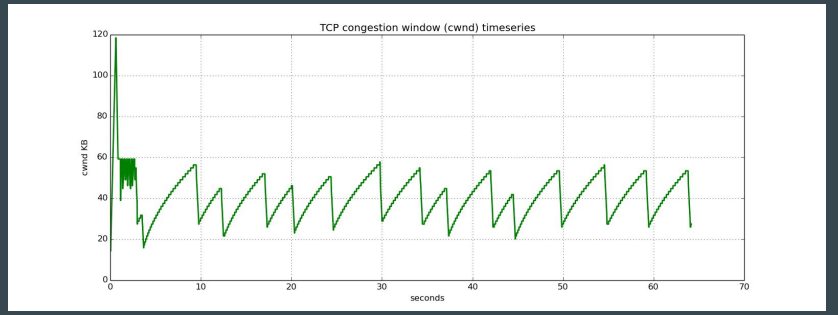


Plotting

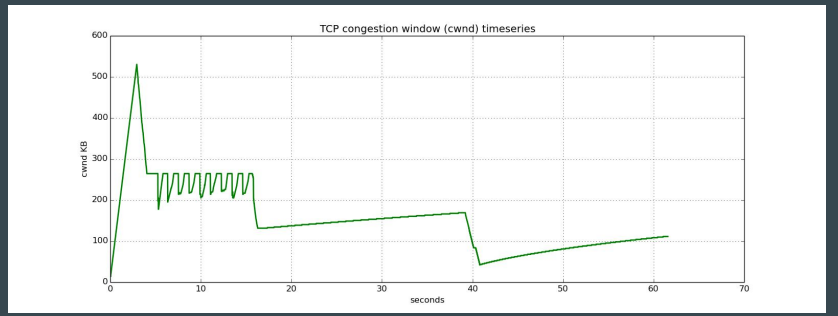
- Starter code contains scripts for plotting, `plot_queue.py`, `plot_ping.py`
  - Expects queue occupancy in $dir/q.txt, ping latency in $dir/ping.txt
  - Plots are useful for debugging!

- 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



Note

- `Sudo mn -c` to restart mininet
- Run `CLI()` in python to enter an interactive shell. This will be useful for debugging/ testing commands to run in h1/h2.
- This is a common mistake in previous quarters! Make sure that your curl command is able to fetch the webpage and receives a valid response from the server before you use its time measurement

Deliverables

- A zip file of
  - Final Code
  - README
  - 8 Plots

Review of BGP

How to figure out what path to take?

- With some hand-waving, we can figure the path based on vibes
  - Usually, the graphs you'll see on quizzes and exams are small enough to intuitively figure out the BGP path
- **But vibes is not good enough!!**
  - How does BGP systematically figure out the path?
- Key technique: path advertisements!
  - For a given destination, an AS will advertise to each of its neighbors exactly ONE path to the destination that includes that AS as a hop along the route
    - Note: the path advertised to two different neighbors may be different (and it is entirely possible that an AS does not advertise a path to a neighbor)

Advertisements are cool, but how do I use them?

- In most BGP questions, you will not need to find the route from every src to every dst – usually just examining a path between two ASes
- In practice, path advertisements are in the opposite direction of the data flow
- So, WORK BACKWARDS!
  - Start at the destination – destination will advertise to any neighbor, it is fine with paying a cost to get messages from the source
  - Then, look at destination's neighbors
    - The neighbor will add themselves to the path, and choose to advertise to its neighbors – note that if the destination was a provider of this neighbor, then this neighbor AS will not advertise the path to any of its provider neighbors!
  - Look at the next set of neighbors
    - At this point, may need to aggregate different advertisements that have been heard by this AS and decide best one to advertise further – hint: look at the relationship between this AS and the neighbors it has heard from

An example...