Programming Assignment 2: Bufferbloat

Introduction tutorial

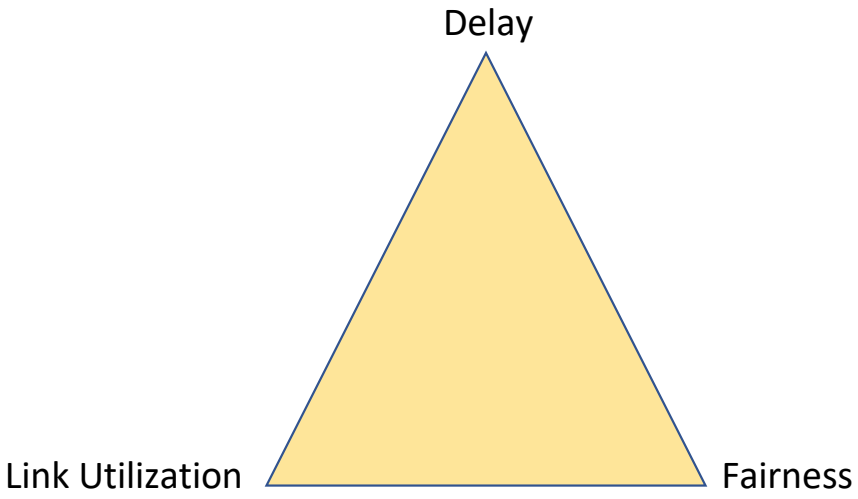
CSC458/2209-Fall 2025

Parsa Pazhooheshy

Computer Science
UNIVERSITY OF TORONTO

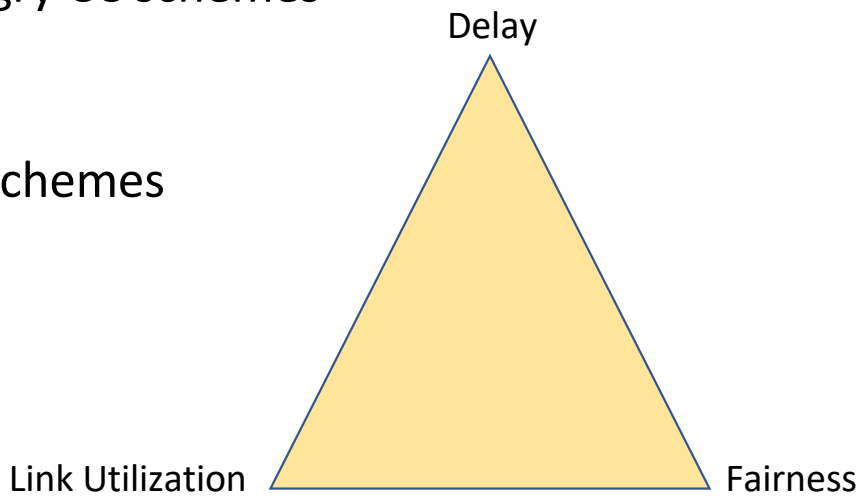
Fundamental Concepts

- Transmission Control Protocol (TCP)
  - Reliability Over a Best-Effort Network
  - One main question:
    - How many packets should be sent in a certain time interval?



Congestion Control (CC)Design

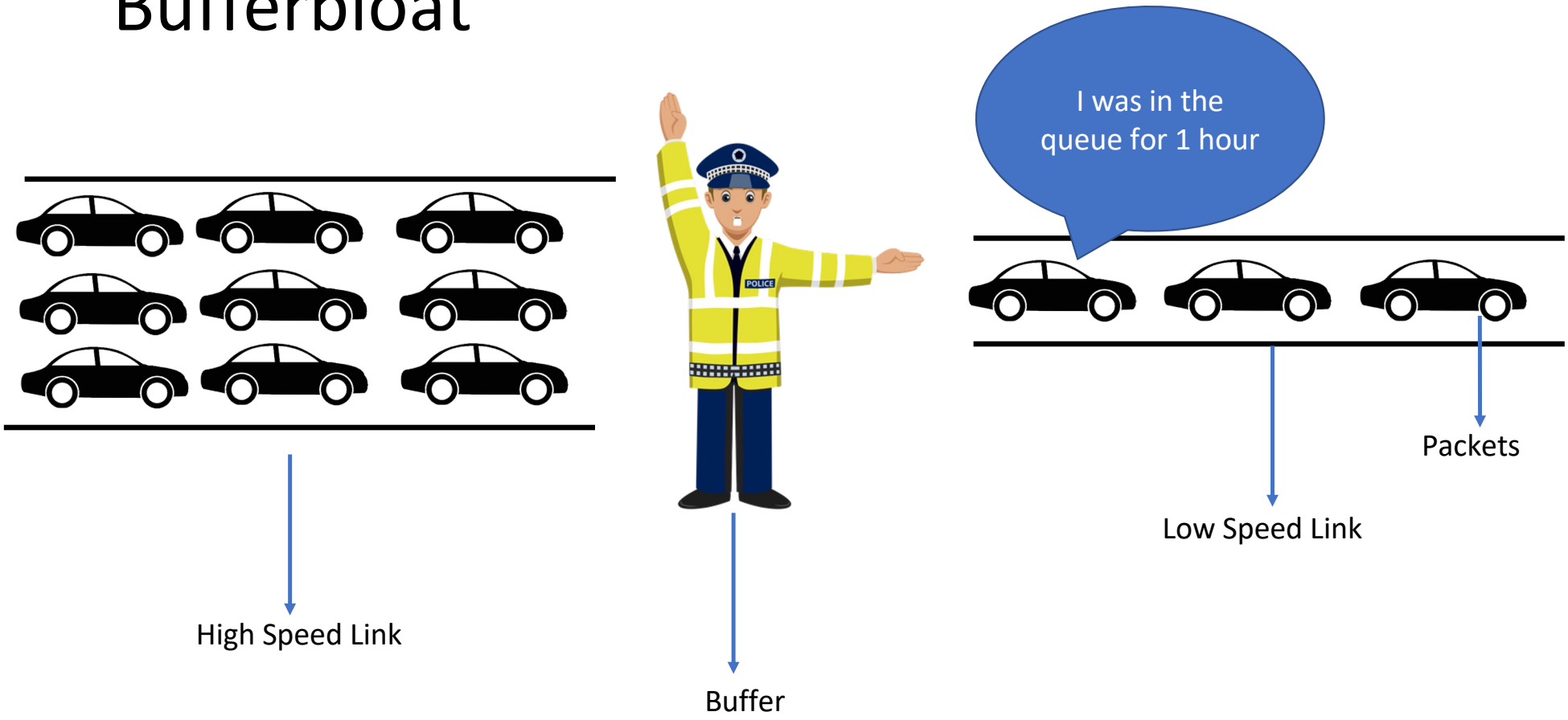
- CC schemes aim to achieve delay, link utilization and fairness at the same time.
- Trade-off
  - Focus on two
- Throughput-hungry CC schemes
  - TCP New-Reno
  - TCP CUBIC
- Delay-based CC schemes
  - TCP Vegas
  - Copa



Congestion window

- Our focus for PA2 is on throughput-hungry schemes
- Congestion Window (CWND):
  - How many in-flight packets a sender has at any given time
    - in-flight: on the link, in the buffers of routers, etc.
- Key question of many CC schemes:
  - How to control CWND?
  - Based on some signals from the network
- Throughput-hungry CC schemes favorite signal: PACKET LOSS
  - Timeout
  - 3 duplicate ACKs
- General behavior of throughput-hungry CC schemes:
  - Increase the CWND if no packet loss detected
  - Reduce the CWND if detected packet loss

Bufferbloat



Why Bufferbloat happens?

- Deep (large) buffer in the bottleneck
  - The buffer only drops the packet when its full
- Sender keeps increasing the CWND as it sees no loss
- More packets get stuck in the buffer

- Packets experience huge delay

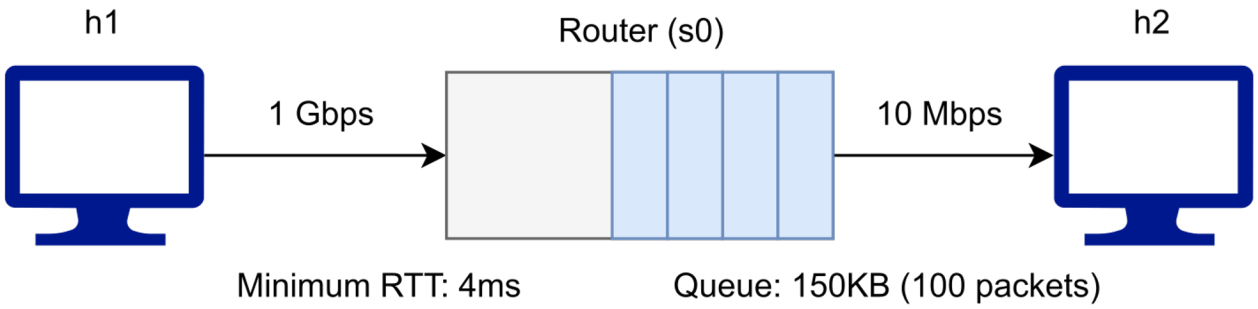
Bufferbloat

PA2 General Goals

- Learn the TCP sawtooth behavior and router buffer occupancy dynamics in a network.
- Learn why large router buffers may lead to poor performance (“bufferbloat.”)
- Learn how to use Mininet to run traffic generators, collect statistics, and plot graphs.
- Learn how to organize your experiments in a reproducible manner.

Methodology

- Implement the following topology in MiniNet
- h1 is a host that has a fast connection (1 Gbps)
- Router (s0) has a slow uplink connection (10 Mbps)
- Minimum RTT between h1 and h2 is 4ms
- The router buffer can hold up to 100 full-sized ethernet frames



Traffic Flows (First Part)

- A long-lived TCP flow, sending data from h1 to h2, using iperf
- Ping from h1 to h2 10 times a second and record the RTTs
- Spawn a webserver on h1
  - Periodically download the index.html web page (three times every five seconds) from h1. (data is sent from h1 to h2)
  - Measure how long it takes to be fetched (on average).
  - Hints in starter code

Traffic Flows

- Keep all the flows alive together

- Plot the following time series:
  - CWND for The long-lived TCP flow (available in h1 iperf)
  - RTT reported by ping (use appropriate ping options. See ping man)
  - Queue size at the bottleneck

- Reduce the router buffer size to 20 packets, repeat the above experiment, and replot the three graphs .

Starting the assignment

- VM
  - You will need the VM loaded with Mininet and the required dependencies of the assignment
    - VM image is provided in course webpage

Assignment files

- Look for TODOs
- **bufferbloat.py**
  - Creates the topology; measures CWND, queue sizes, and RTTs; and spawns a webserver.
- **plot_queue.py**
  - Plots the queue occupancy at the bottleneck router.
- **plot_ping.py**
  - Parses and plots the RTT reported by ping.
- **plot_tcpprobe.py**
  - Plots the cwnd time-series for a flow specified by its destination port
- **run.sh**:
  - Runs the experiment and generates all graphs in one go.

Run.sh

- Your whole code (including running the experiment and then generating the results) should be done by just a “sudo ./run.sh” command

Some hints

- The project is almost complete!
  - You just need to implement few TODOs in bufferbloat.py and run.sh
  - You need to read bufferbloat.py carefully first, it helps you complete TODOs.