CSC458 Programming Assignment 2: Bufferbloat

Introduction

In this exercise we will study the dynamics of TCP in home networks. Figure 1 shows a “typical” home network with a Home Router connected to an end host. The Home Router is connected via Cable or DSL to a Headend router at the Internet access provider’s office. We are going to study what happens when we download data from a remote server to the End Host in this home network.

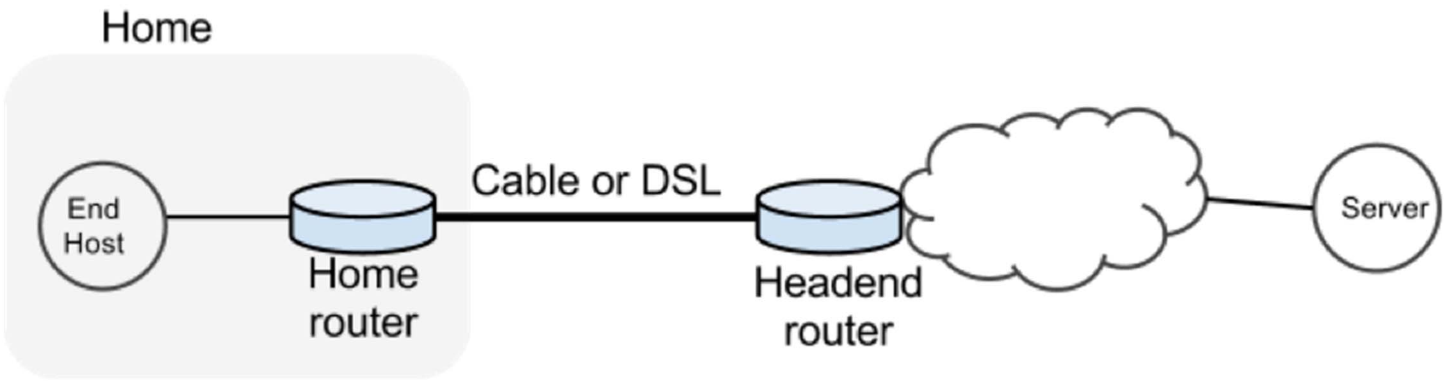

Figure 1. Typical home network setup

In a real network it’s hard to measure TCP’s congestion windows size (i.e., *cwnd*, because it’s private to the Server) and the buffer occupancy (because it’s private to the router). To make our measurement job easier, we are going to emulate the network in Mininet.

Goals

- Learn first-hand the dynamics of TCP sawtooth and router buffer occupancy in a network.
- Learn why large router buffers can lead to poor performance. This problem is often called “*bufferbloat*.”
- Learn how to use Mininet to run traffic generators, collect statistics and plot them.
- Learn how to package your experiments so it’s easy for others to run your code.

Setup

Within Mininet, create the topology that is shown in Figure 2: *h1* represents your home computer that has a fast connection (1Gb/s) to your home router (*s0*) which has a slow uplink connection (10Mb/s). *h2* represents a server that you want to download a file from. The round-trip propagation delay, or the minimum RTT between h1 and h2 is 4ms. The router buffer size can hold 100 full sized ethernet frames (about 150kB with an MTU of 1500 bytes).

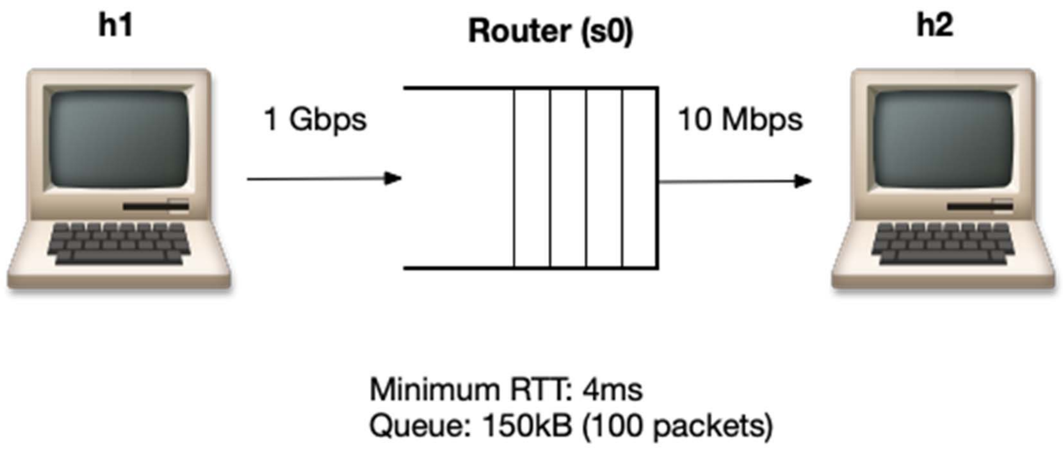

Figure 2. Assignment topology

Tasks

We want to perform the following tasks simultaneously using different queue sizes at the router:

- Start a long-lived TCP flow sending data from *h1* to *h2*. Use *iperf*. We will record the congestion windows size (*cwnd*) of this connection.
- Send 10 pings per second from *h1* to *h2* and record their RTTs.
- Spawn a webserver on *h1*. Download the index.html web page from *h1* repeatedly every two seconds and measure how long it takes to fetch it (on average). The starter code has some hints on how to do this. Make sure that the webpage download data is going in the same direction as the long-lived flow (*h1* to *h2*). You should use the *curl* command to download the page.
- These three tasks (long-lived flow, ping train, and webserver downloads) should all be happening simultaneously.
- Plot the time-series of the following:
    - The long-lived TCP flow's *cwnd*
    - The *RTT* reported by ping
    - Webpage download time
    - Queue size at the router

You should perform this simulation using three different queue sizes: Q=5 packets, Q=20 packets, and Q=100 packets.

Starter Code

To help you get started, we provide a basic skeleton code inside a virtual machine. Once you have your instance running, run a Mininet sanity check (sudo mn --test pingall) and then continue with this assignment. Your instance should have Mininet and most tools pre-installed.

Download Instructions

Download the Mininet VirtualBox VM from the course website (quercus):

http://www.cs.toronto.edu/~yganjali/bb/cs244-vm.ova

After importing it into VirtualBox (or any other VM software that you prefer to use), use the username "cs244" and password "cs244" to sign into the VM. Then clone the starter code for this project from this repository:

https://github.com/Network-Lab/cs244-bufferbloat.git

You can do this by running the following command in the VM:

```
> git clone https://github.com/Network-Lab/cs244-bufferbloat.git
```

This will create a new folder named "cs244-bufferbloat" that contains the starter code. Look for TODOs in the starter code. The following are important files you will find in the repository. Ignore other files.

- ***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.

Note: We recommend you use a version control system to track changes to your assignment and back it up regularly. Both Github and Bitbucket provide private git repositories for free. Be careful to NOT use a public repository.

Submission/Deliverables

This assignment is to be completed ***individually***. Submit your work on MarkUS (https://markus.teach.cs.toronto.edu/csc458-2021-09).

Due date for this assignment is **Sunday, November 28, 2020 at 11:59pm**.

This assignment worth 18% of your total term mark. You should submit three items: a written report (10%), raw generated plots (4%), and your code (4%).

Written Report

You should prepare a 2-5 pages report (with text font size of 12 points). The report should provide a brief description of the bufferbloat problem and its impact on the users. Then it should describe (briefly) the simulation setup that you had. These two parts together should not be more than 50% of the total report. The main part of the report will present your results (graphs) and discuss what it means. It should answer the following questions:

- Why do you see a difference in webpage fetch times with short and large router buffers?
- Bufferbloat can occur in other places such as your network interface card (NIC). Check the output of ifconfig eth0 on your VirtualBox VM. What is the (maximum) transmit queue length on the network interface reported by ifconfig? For this queue size, if you assume the queue drains at 100Mb/s, what is the maximum time a packet might wait in the queue before it leaves the NIC?
- How does the RTT reported by ping vary with the queue size? Write a symbolic equation to describe the relation between the two (ignore computation overheads in ping that might affect the final result).
- Identify and describe two ways to mitigate the bufferbloat problem.

The report should be named "***pa2.pdf***". It should be an electronic and typed document (e.g., not a hand-written and scanned pdf). In addition to be marked for correct technical content (e.g., answering those questions), the presentation is also being marked (e.g., have separate sections for different topics, having a reference section, …). Overall, the report should be similar to a technical reports and conference papers.

Generated Graphs

You should 12 graphs that you code generates. There will be 4 charts for each queue size (rtt, cwnd, queue size, and download time) and 3 different queue sizes (5, 20, 100). Your file names should be:

- RTT graphs: **rtt-5.png**, **rtt-20.png**, **rtt-100.png**
- cwnd size: **cwnd-5.png**, **cwnd-20.png**, **cwnd-100.png**
- Queue size: **q-5.png**, **q-20.png**, **q-100.png**
- Download time: **download-5.png**, **download-20.png**, **download-100.png**

The 5% mark for this part focuses on the correctness of the graphs.

Note that these graphs should be identical to the ones that anyone else (including TAs) can generate by simply running your code (*run.sh*). I.e., these should be the raw output of running your code.

Code

You should upload the 5 main scripts that you need to update to generate the graphs: ***bufferbloat.py***, ***plot_queue.py***, ***plot_ping.py***, ***plot_tcpprobe.py***, and ***run.sh***

This 5% mark of this part will focus on your coding style and documentation, similar to the first programming assignment.

Useful Hints

- To look at your output, you may find it useful to start a web server using python -m SimpleHTTPServer. Start the webserver in the cs244-bufferbloat directory. Then you can point a web browser to :8000 and browse your results.
- If your Mininet script does not exit cleanly due to an error (or if you pressed Control-C), you may want to issue a clean command before you start Mininet again:

```
> sudo mn -c
```

Useful Links

- On the bufferbloat problem"
  - https://netduma.com/blog/beginners-guide-to-bufferbloat/
  - https://www.bufferbloat.net/projects/bloat/wiki/Introduction/
- More about Mininet and how it can be used:
  - http://mininet.org/walkthrough/
  - http://geekstuff.org/2018/09/26/mininet-tutorial/
- Monitoring and Tuning the Linux Networking Stack:
  - https://blog.packagecloud.io/eng/2016/06/22/monitoring-tuning-linux-networking-stack-receiving-data/
  - https://blog.packagecloud.io/eng/2016/10/11/monitoring-tuning-linux-networking-stack-receiving-data-illustrated/