Lab 4: TCP and UDP Congestion

50.012 Networks

Hand-out: October 4
eDimension hand-in: October 11

1 Objectives

- Experiment with buffer sizes, network scheduling, and TCP congestion window
- Goal: Understand why larger buffers can be disadvantageous
- This exercise is based on the Stanford CS244 class lab1

2 Setup

2.1 Lab4 setup

- Download the lab4.zip from eDimension and unpack to some local folder, e.g. ~/lab4/
- Change into the lab4 folder, and open up another terminal tab with CTRL+SHIFT+T

2.2 mininet

- This exercise assumes that you have a running mininet installation
  - It should be installed on the lab machines already
  - On (x)ubuntu, you can install it by running our `install.sh`. Mininet can run on many *nixes, but NOT on Windows or Mac

1. What is Mininet?
   - Mininet is a network simulator (written in Python), that allows you to define topology and actions in python.
   - The simulated hosts run your OS, and can execute your client/server applications easily.
   - You can also easily change link parameters such a loss, bandwidth, etc.
   - Mininet is mainly intended to experiment with *OpenFlow*, but is also convenient for this lab session. We might look at OpenFlow later in the term.
   - More details at `http://mininet.org`, in particular `http://mininet.org/walkthrough/`
   - If you like videos: some basic things of this sheet are also demo'ed in `https://www.youtube.com/watch?v=jmlgXaocwiE`

2. Installation and first test

    - Start up mininet:

```
sudo ./run.sh
```

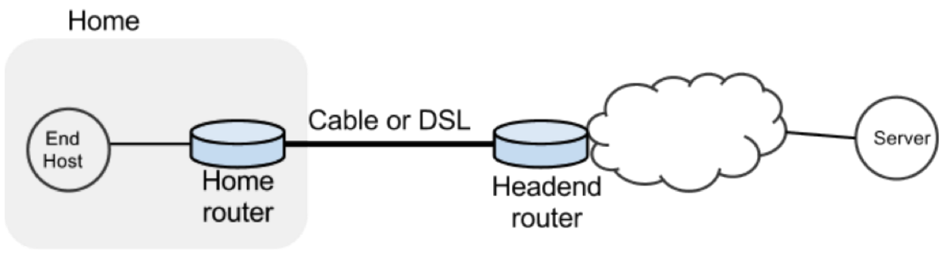
    - Use the following to spawn two terminals, belonging to two nodes in the simulated network.

```
mininet> h1 xterm &
mininet> h2 xterm &
```

    - Using these terminals, you can run applications or perform ping operations on the simulated machines
        - To close running commands like **ping -c 100**, try CTRL-C
    - Find out the IP addresses of h1 and h2
    - To leave/close Mininet, try CTRL-D on the main command line

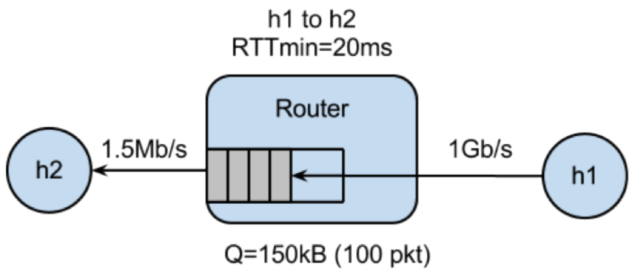
3 Queue Length and TCP

In this exercise we will study the dynamics of TCP in home networks. Take a look at the figure below which 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.



Problem overview

In a real network it’s hard to measure the TCP congestion window `cwnd` (because it’s private to the Server) and the buffer occupancy (because it’s private to the router). Luckily, we have mininet to help allow us to get these values easily.



Topology for Mininet

3.1 Simple TCP connection

- Start up mininet

```
sudo ./run.sh
```

- In the running mininet simulator, test how long it takes to download a webpage on h1 from h2. one second
  - wget's visualization is a bit confusing. You can find the total time by subtracting start time from end time, or by looking at the time at end of last part line.

```
mininet> h2 wget h1
```

- We discussed the TCP congestion window in the last week, what is your expectation how the cwnd changes during this HTTP session?
- How big is the router queue/buffer in your opinion? (RTT*C)/sqrt(N)

3.2 With parallel load (iperf)

- To see how the dynamics of a long flow differs from a short flow (which never leaves slow-start), we are going to repeat Part 2 for a "streaming video flow". Instead of actually watching videos on your machine, we are going to set up a long-lived high speed TCP connection instead, to emulate a long-lived video flow.
- You can generate long flows using the `iperf` command. iperf is a tool for performing network throughput measurements. It can test either TCP or UDP throughput. To perform an iperf test the user must establish both a server (to discard traffic) and a client (to generate traffic). We have wrapped iperf in a script that you can run as follows:

```
mininet> h1 ./iperf.sh
```

- You can see the throughput of TCP flow from h1 to h2 by running:

```
mininet> h2 tail -f ./iperf-recv.txt
```

- You can quit viewing throughput by pressing CTRL-C.
- Think about the congestion window of the TCP stream again. How will it look for a long-established connection? steady?
- You can also use ping to observe RTT while iperf is running

```
mininet> h1 ping -c 100 h2
```
rtt 643ms

The Impact on the Short Flow

- To see how our long-lived iperf flow affects our web page download, download the webpage again - while iperf is running. Observe how long it takes. 3.4s, it takes longer because of traffic congestion which forces us to use the buffer, which in turn introduces delay.

```
mininet> h2 wget h1
```

- Why does the web page take so much longer to download?

3.3 Measuring the real cwnd and buffer occupancy values.

We provided a script that lets you measure cwnd and buffer occupancy values (in terms of values for cwnd and buffer occupancy). We are going to re-run a couple of the experiments and plot the real values.

1. Restart Mininet
   - Stop and restart Mininet and the monitor script, then re-run the above experiment as follows.

```
mininet> exit
sudo ./run.sh
```

2. Monitor TCP CWND and Buffer Occupancy in Mininet
   - In your other terminal tab, type the following giving a name for your experiment.

```
./monitor.sh <EXP_NAME>
```

   - In your first tab with the running mininet, start iperf again

```
mininet> h1 ./iperf.sh
```

   - (wait for 70 seconds . . .)

```
mininet> h2 wget h1
```

   - Wait for the wget to complete, then stop the python monitor script followed by the instructions on the screen. The cwnd values are saved in `<EXP_NAME>_tcpprobe.txt` and the buffer occupancy in `<EXP_NAME>_sw0-qlen.txt`.

3. Plot CWND and Queue Occupancy
   - Plot the TCP cwnd and queue occupancy from the output file

```
./plot_figures.sh <EXP_NAME>
```

   - Adjust command line parameters to generate the figure you want.
   - The script will also host a webserver on the machine and you can use the url the script provided to access to your figures.
   - Note: the figures will sometimes interpolate where it is not appropriate. If you see a slow decrease of cwnd, then that probably is an artefact of such interpolation. Ignore those segments of the figures.
   - If you are unable to see the cwnd, ensure you run wget after you started the monitor.sh script.

3.4 Smaller Buffer: from 100 packets to 20 packets

Restart Mininet with small buffer

- Stop any running Mininet and start Mininet again, but this time we will make the buffers 20 packets long instead:

```
sudo ./run-minq.sh
```

- Repeat the earlier simple tests

```
mininet> h2 wget h1     1s
mininet> h1 ping -c 10 h2 30ms
```

- Did the results change? no

- Close mininet and start monitoring again

```
sudo ./monitor.sh <EXP_NAME>
```

- Restart mininet and re-run the experiment

```
mininet> h1 ./iperf.sh
mininet> h1 ping -c 30 h2  151ms
mininet> h2 wget h1 2.7s
```

- What do you think the cwnd and queue occupancy will be like in this case? this experiment's cwnd will have a higher frequency than the previous one, and the queue occupancy will be more frequently filled up and emptied

- Plot the figure for cwnd and queue occupancy, this time using the script "./plot_figures_minq.sh"

- Then again, use the url to see your figures. Why does reducing the queue size reduce the download time for wget? By reducing the queue size/buffer size, less packets will be in queue and not being sent out before the network detects that there is a congestion which will in turn trigger it to reduce the sending rate, thus help to clear the packets stuck in the queue. This is seen evidently as a sawtooth pattern in the plot which indicates the much faster rate of filling up and clearing of packets in the queue, thus a reduction in delay.

```
./plot_figures_minq.sh
```

4 UDP Experiments

- In this part, you write a simple UDP client/server system
- Then, run the server on `h2` and the client on `h1` in mininet
- Observe what happens if the client is sending too much data in UDP
- A helpful tutorial on a similar application: `https://pymotw.com/2/socket/udp.html`

4.1 Simple UDP client/server application

Your UDP server/client should follow these requirements:

- The server will receive incoming datagrams on port 5555
- The exact message format is up to you, it does not need to be efficient
- The client should send messages with a leading segment ID
- Your client should be configurable to send a fix amount of data per second (e.g. 1MBit/s), including UDP and IP header
- The server should verify if incoming traffic has missing datagrams, and display a warning if that is the case

4.2 Simple UDP connection

- start up mininet

```
sudo ./run.sh
```

- In the running mininet simulator, open a xterm window for h1, and for h2

```
mininet> h1 xterm &
mininet> h2 xterm &
```

- Start your server application on h2, and the client on h1. The client should now be able to send data to the server.
- Set up your client to send approximately 1.5 MBit/s of traffic to h2. Do you see dropped packets at the server?
- Set up your client to send a bit more, e.g. 1.6 MBit/s. Again, do you see dropped packets?

5 What to Hand in

5.1 eDimension submission:

- Submit your server and client python code for the UDP part
- The client should send approximately 1.5MBit/s
  - No packets should be dropped by default
  - If we increase the packet rate by 10%, we should see dropped packets
- The server should report if incoming traffic has dropped datagrams (and, optionally, if a new sequence of data is started)

5.2 Checkoff:

- Demonstrate the TCP queue and UDP part of the exercise to the TA
  - How big is cwnd expected to be for both queue sizes?
  - Why does wget take longer with iperf in parallel?