Logout

My Workspace | CS-6250-O01 FALL14 | CS-6300-O01 FALL14

ASSIGNMENTS

- Home
- Syllabus
- Announcements
- Resources
- Assignments
- Gradebook
- Email Archive
- Roster
- Site Info
- Section Info
- Piazza
- Help

Assignment 5 - Buffer bloat - Returned

Title	Assignment 5 - Buffer bloat
Student	Telfort, Dominique Gayot
Submitted Date	Oct 2, 2014 10:00 pm
Grade	8.0 (max 10.0)

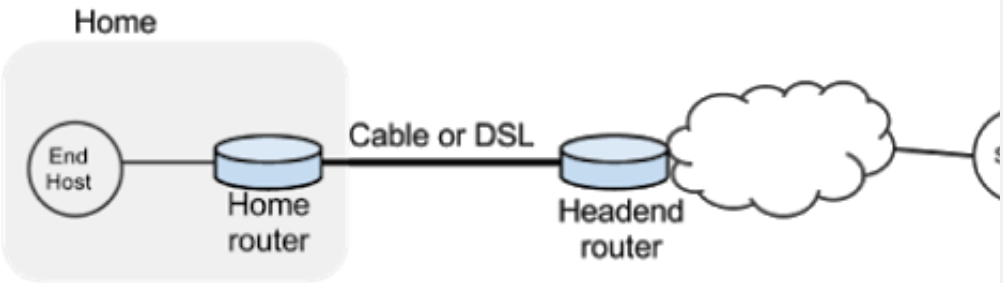
Instructions

Assignment 5 - Buffer Bloat

Goal

In assignment 3 you learned how buffer sizes can be reduced for long lived flows and in assignment 4 you learned how TCP shares bandwidth across multiple flows. This assignment combines buffer sizing and TCP behavior as we study the dynamics of TCP in home networks [1]. This assignment is structured around an interactive exercise and quiz questions you will submit, with the topologies you'll be using provided in the assignment code.

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.



In a real network it’s hard to measure 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

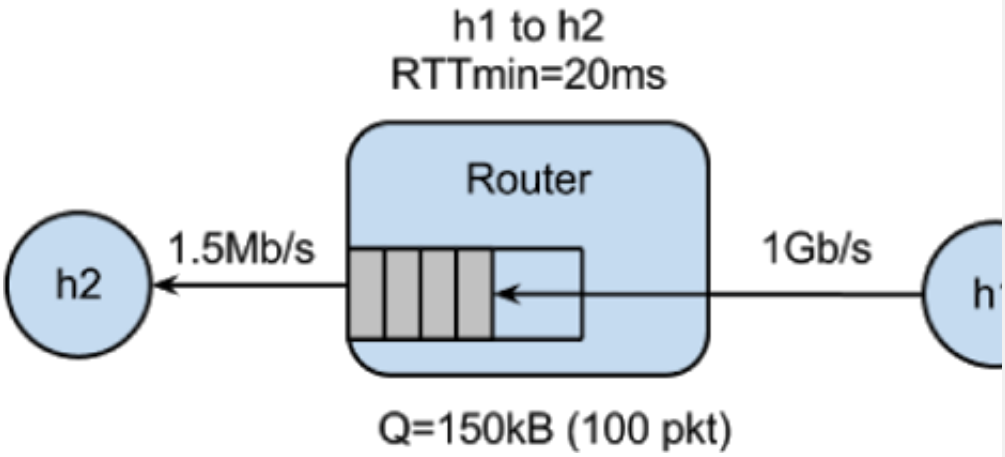
in Mininet.

The goals of the exercise are to:

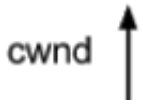
- Review the dynamics of cwnd and buffer occupancy in a “real” network.
- Learn why large router buffers can lead to poor performance in home networks. This problem is often called “Buffer Bloat.”

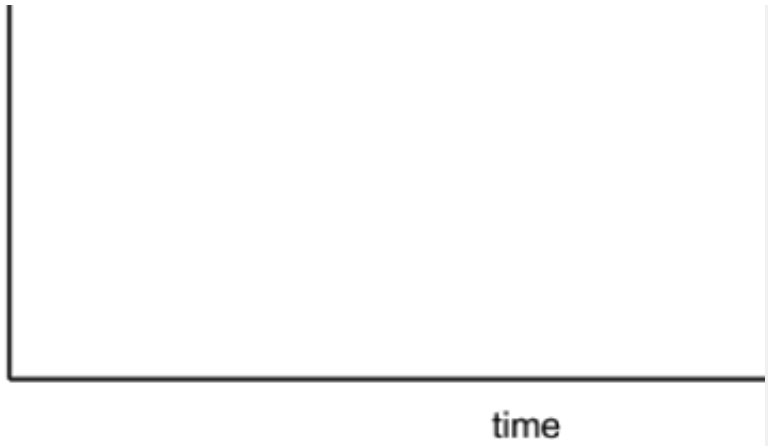
Directions

The topology for this assignment is setup as follows.



1. Update to the latest assignment code: `cd assignment-5/` followed by `git commit -a -m "Saving work"`
   `git pull --rebase`
2. Now run Mininet. This will bring up the Mininet command line interface which you will be using in the assignment.
   `sudo ./run.sh`
3. To begin, you'll measure how long H2, the end host, takes to download a web page from H1 the server. To measure this, run the following command in the Mininet CLI and make a note of the time in seconds (to calculate the time, use the time stamps output by wget at the start and end of the transfer):
   `mininet> h2 wget http://10.0.0.1`
   On scratch paper, sketch how you think `cwnd` evolves over time at H1. Mark multiples of RTT on the x-axis. Don't worry if you are unsure of the behavior, we will be graphing the actual `cwnd` so make your best guess.





4. To see how the dynamics of a long flow (which enters the AIMD phase) differs from a short flow (which never leaves slow-start), we are going to repeat initiate a web request as in step 3 while a “streaming video flow” is running. To simulate a long-lived video flow, we are going to set up a long-lived high speed TCP connection instead. You can generate long flows using the iperf command, and we have wrapped it in a script which you can run as follows:

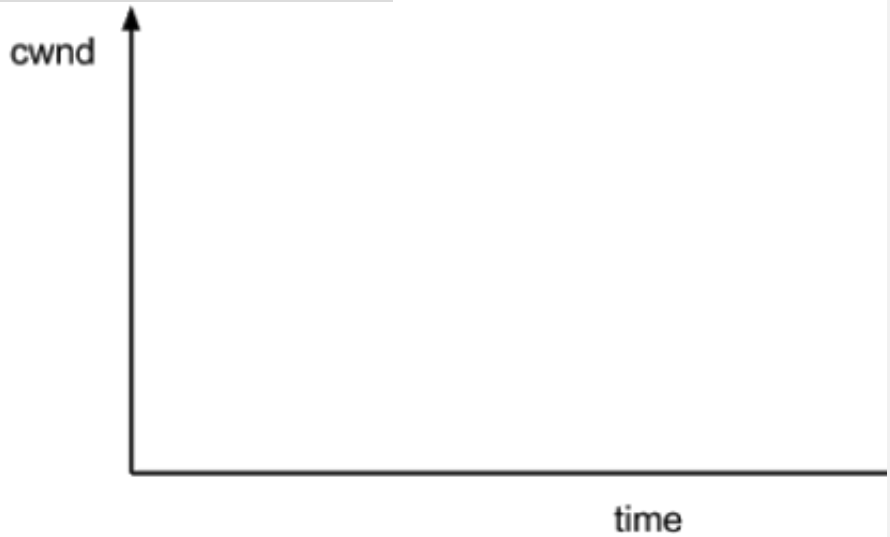
   ```
   mininet> h1 ./iperf.sh
   ```

   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. Before we observe the effect of the long-lived flow on the short flow, sketch how you think `cwnd` evolves over time at H1. Remember the long-lived flow has entered the AIMD phase unlike the slow start phase of the web request. You might find it useful to use ping to measure how the delay evolves over time, after the iperf has started:

   ```
   mininet> h1 ping -c 100 h2
   ```



5. Now to see how our long-lived iperf flow affects our web page download, download the webpage again - while iperf is running.

Make a note of the download time as in step 3.

```
mininet> h2 wget http://10.0.0.1
```

Why does the web page take so much longer to download? You'll submit your answer to this question as part of the assignment.

6. To confirm our understanding in the previous steps, we'll use a provided script to plot the `cwnd` and buffer occupancy values. We're going to re-run a couple of the experiments and plot the real values. Stop and restart Mininet and then start the monitor script. Then re-run the above experiment as follows.

   ```
   mininet> exit
   ```

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

   In another bash terminal, go to `assignment-5` directory and type the following giving a name for your experiment.

   ```
   ./monitor.sh experiment-1
   ```

   `mininet> h1 ./iperf.sh` (wait for 70 seconds such that the iperf stream is in steady state for its congestion window…)

   ```
   mininet> h2 wget http://10.0.0.1
   ```

   Wait for the wget to complete, then stop the python monitor script by following the instructions on the screen (Pressing Enter). The cwnd values are saved in `experiment-1_tcpprobe.txt` and the buffer occupancy in `experiment-1_sw0-qlen.txt`.

7. Plot the TCP cwnd and queue occupancy:

   ```
   sudo ./plot_figures.sh experiment-1
   ```

   The script hosts a webserver on the machine and you can access it in the same way you did the figures in past assignments by navigating to `http://ip_address:8000` where the IP address is the IP address of the virtual machine. You can view sample figures to ensure you generated similar results. If you are unable to see the `cwnd`, ensure you ran wget after you started the monitor.sh script. At this point, you may have realized the buffer in the Headend router is so large that when it fills up with iperf packets, it delays the short wget flow. Next we'll look at two ways to reduce the problem.

8. The first method to speed up the short lived flow in the prescence of the long one follows from realizing the buffer is too large: make the router buffer smaller and reduce it from 100 packets to 20 packets. To do this, stop any running Mininet instances and start Mininet again, but this use the start script will set a 20 packet buffer:

`sudo ./run-minq.sh`

Let's also run the monitor script in another terminal:

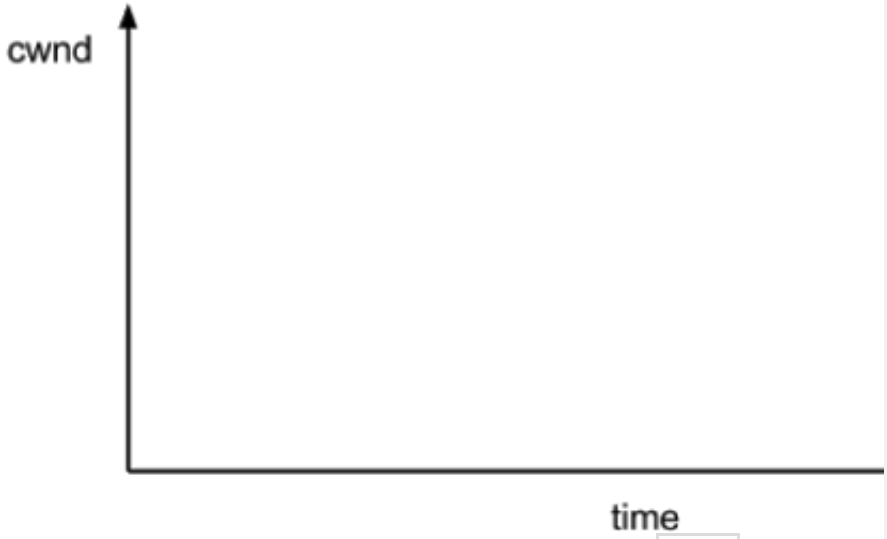
`sudo ./monitor.sh experiment-2`

Then we'll repeat the previous iperf and wget steps:

`mininet> h1 ./iperf.sh` (wait for 70 seconds such that the iperf stream is in steady state for its congestion window…)

`mininet> h2 wget http://10.0.0.1`

Sketch what you think the `cwnd` and queue occupancy will be like in this case.



9. Now confirm your sketch by plotting the figures for `cwnd` and queue occupancy:
   `sudo ./plot_figures_minq.sh experiment-2` Navigate to the web server as in step 7 to see your figures. For comparison you can confirm your figures with thesample figures.
   Why does reducing the queue size reduce the download time for wget? You'll submit your answer to this question as part of the assignment.
10. The buffer bloat problem seems to be that packets from the short flow are stuck behind a lot of packets from the long flow. What if we maintain a separate queue for each flow and then put iperf and wget traffic into different queues? For this experiment, we put the iperf and wget/ping packets into separate queues in the Headend router. The scheduler implements fair queueing so that when both queues are busy, each flow will receive half of the bottleneck link rate.