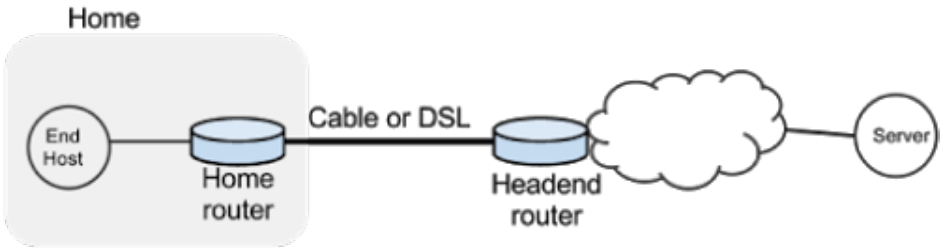
Assignment 4 - Buffer Bloat

Goal

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 questions you will submit, with the topologies you'll be using provided in the assignment code. This assignment does not require you to write code.

You are encouraged to create new topologies (take a look at `run.sh` and try tweaking the `maxq` and `delay` parameters) and discuss your findings on Piazza. For this assignment, posting the images of graphs is acceptable (especially comparisons with tweaked parameters) as you will be turning in the data that generates the images. Please do not post data that generates the images (that will be considered cheating). To summarize: posting graphs - ok; posting raw data - not okay.

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.



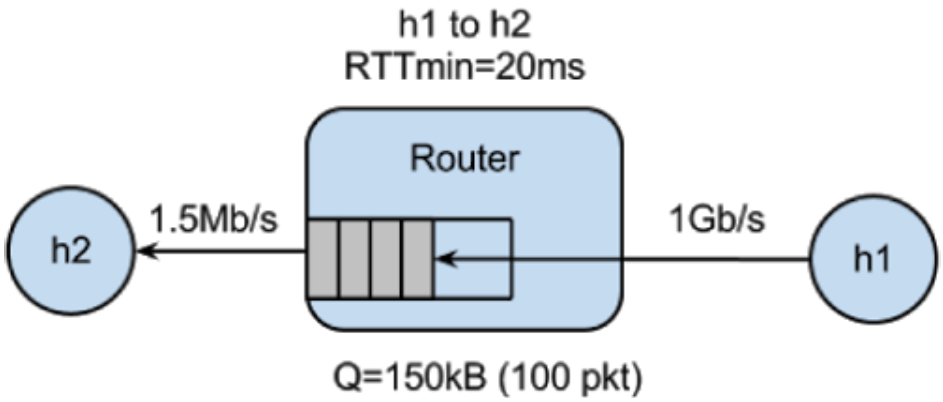
In a real network it's hard to measure cwnd (because it's private to the server) and the buffer occupancy (because it's private to the router). To make our measurement job easier, we are going to emulate the network in Mininet.

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."

Before the Experiment

The topology for this assignment is setup as follows.



We're going to show how different different queue sizes affect how quickly the CWND grows. First, we'll start with a small queue size, then go over with a large queue size. From lecture, you should know that the larger buffers introduce larger delays, which slows down the feedback loop for TCP's congestion control algorithm.

1. Update to the latest assignment code: `git commit -a -m "Saving work"` `git pull --rebase`

2. Change into the assignment-4 directory and run the following command. It will start Mininet, create the appropriate topology, and give you a Mininet command line. `sudo ./run-minq.sh`

3. Now, we have to start the monitor, which will capture information about the queue. In a second terminal window, also in the assignment-4 directory, run the command: `./monitor.sh small-queue`

4. Back on the Mininet terminal, we need to start up some traffic. We'll use iperf, an active measurement tool that tries to shove data through the network as fast as it can. Use the following command: `h1 ./iperf.sh`

5. After waiting 90-120 seconds, on the monitor terminal, press "enter" to stop monitoring. This is enough time to capture slow-start and a few cycles of the sawtooth wave. On the Mininet terminal, you should issue the `exit` command to stop Mininet. Now, we'll use the captured information and plot the data using the following command (on the monitor terminal): `./plot_figures.sh small-queue`

6. Either using a file browser (Files in the Activities side bar), or by installing ImageMagick (`sudo apt-get install imagemagick`) and using the `display` command from the command line, look at the new image files in the assignment-4 directory. Three are created, but we only are interested in the queue occupancy of the switch and the CWND size for iperf. (You can ignore the one that says 'wget' - that will come into play later.) Compare these two graphs (and

discuss on Piazza).

7. Next, we'll rerun these tests for large queues. The commands are slightly tweaked, as are the timing. In the last experiment, we used a queue size of 20, but this time the queue size will be 100 - five times larger. First, start up the switch with a large queue using `sudo ./run.sh` and start the monitor with `./monitor.sh large-queue`. Next, on the Mininet terminal start up iperf again with `h1 ./iperf.sh`

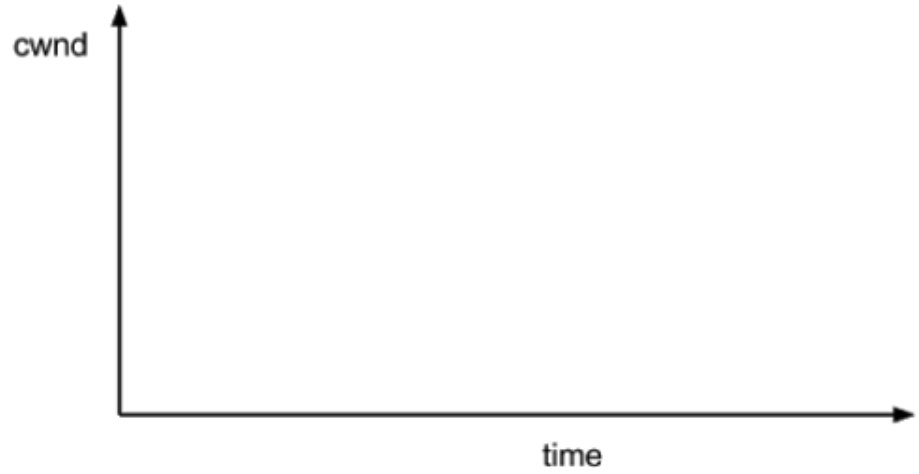
8. Now wait ~10 minutes (yes, this takes much longer than with small queues). When the 10 minutes is up, stop the monitor and Mininet as before (step 5). Next, plot the data that we just collected with `./plot_figures.sh large-queue`

9. Compare these graphs with the graphs we created before. What is different between them? Why do these differences occur? Discuss this on Piazza.

These commands will be used repeatedly below, so feel free to repeat these a few more times until you're comfortable with them and understand what they're doing.

Directions

1. Change into the assignment-4 directory and run the following command. It will start Mininet, create the appropriate topology, and give you a Mininet command line. `sudo ./run.sh`
2. 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.

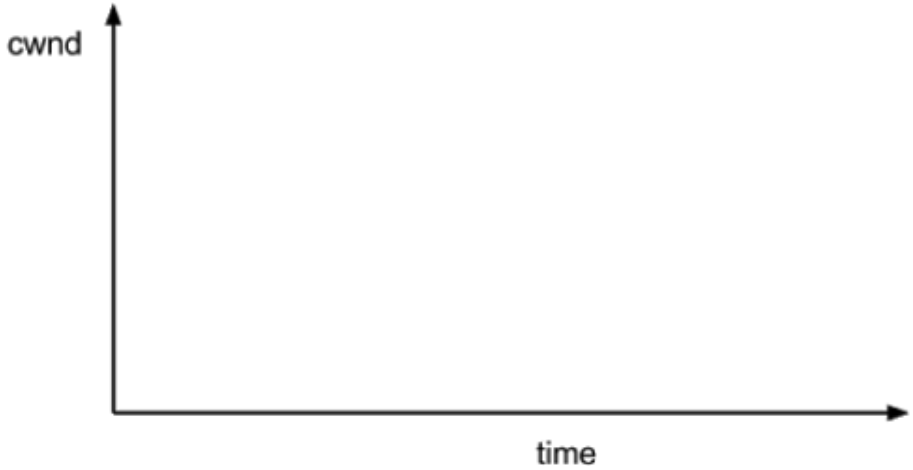


3. To see how the dynamics of a long flow (which enters the AIMD phase) differs from a short flow (which never leaves slow-start), we are going to repeat initiate a web request as in step 3 while a “streaming video flow” is running. You might find it useful to use ping to measure how the delay evolves over time, after the long flow has started. First let's get a baseline and see how the ping is without other traffic:
   `mininet> h1 ping -c 100 h2`

4. 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. (The iperf flow will continue; only your monitoring of it will be stopped.)

5. Now we can run ping again to see how the long flow has affected our network latency and RTT:
   `mininet> h1 ping -c 100 h2`

6. 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.



7. 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? Discuss this on Piazza.

8. 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-4` 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`.

9. Plot the TCP cwnd and queue occupancy:

   `./plot_figures.sh experiment-1`

   You can access the file by navigating to the `assignment-4` directory with a file browser or command line `display`. 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.

10. The first method to speed up the short lived flow in the presence 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 time, the start script will set a 20 packet buffer:

    `sudo ./run-minq.sh`

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

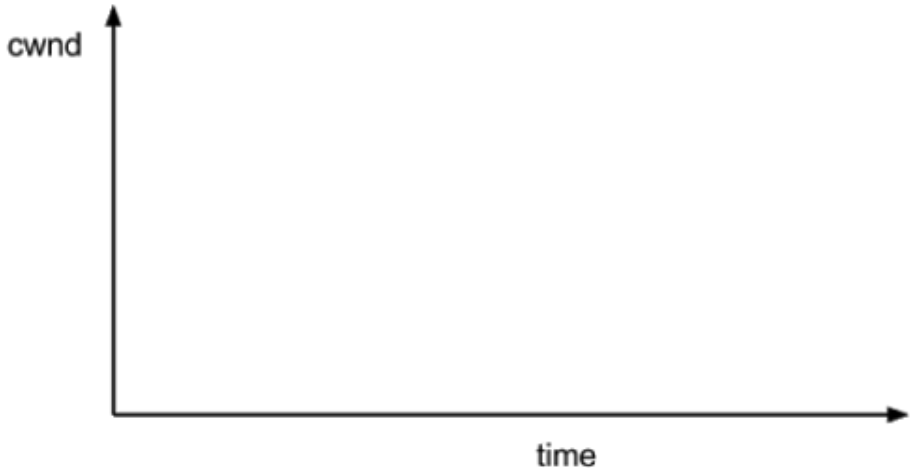
    `./monitor.sh experiment-2`

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

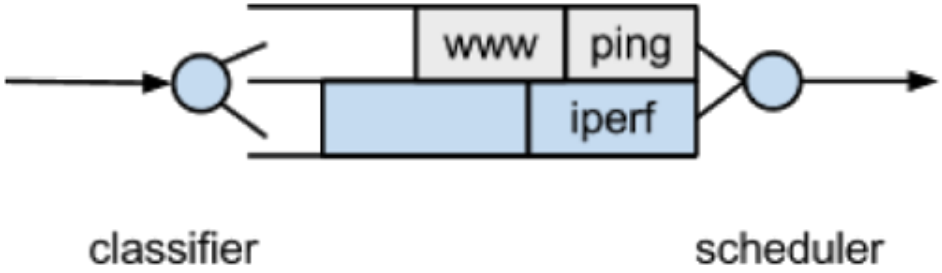
    `mininet> h1 ./iperf.sh` (wait for 70 seconds such that the iperf stream is in steady state for its congestion window...)

    `mininet> h2 wget http://10.0.0.1`

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



11. Now confirm your sketch by plotting the figures for `cwnd` and queue occupancy: `./plot_figures.sh experiment-2` Now view the three figures. Why does reducing the queue size reduce the download time for wget? Discuss this on Piazza.
12. 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.



Start Mininet again, but this time we will create two queues, one for each type of traffic.

```
sudo ./run-diff.sh
```

Then repeat the following steps that have been tweaked slightly for assignment turn-in:

```
mininet> h1 ping -c 10 h2 | tee before-iperf.txt
```

`mininet> h1 ./iperf.sh`

`mininet> h1 ping -c 10 h2 | tee after-iperf.txt`

`mininet> h2 wget http://10.0.0.1`

You should see the ping delay and the wget download time do not change much before and after we start the iperf. The underlying mechanism for these multiple queues is traffic control in Linux. You'll see multiple `tc` commands in `tc_cmd_diff.sh` which setup a filtering mechanism for the iperf traffic. For more information on `tc` take a look at the `man` page and see this and this documentation: `man tc`

13. To complete the assignment submit your queue log file from the small queue experiment, `experiment-2_sw0-qlen.txt`, the files `before-iperf.txt` and `after-iperf.txt` on T-Square.

Notes

[1] Based on Mininet wiki.

This page was last edited on 2015/06/19 00:58:09.