Project 6 - Congestion Control & Buffer Bloat

Goal

The overall goal of this project is to study the dynamics of TCP in home networks [1], specifically congestion control algorithms and buffer occupancy. This project is structured around an interactive experiment utilizing Mininet.

Throughout this project you will be asked to make predictions, record observations, and answer questions about the experiment. To receive credit for this project, you will submit your compiled responses to the items labelled with **Item #X** prompts, as well as your generated experimental data to T-Square after completing the experiment. Please answer each item in 250 words or less. Be concise! Also, please include any relevant graphs / sketches you reference in your answers.

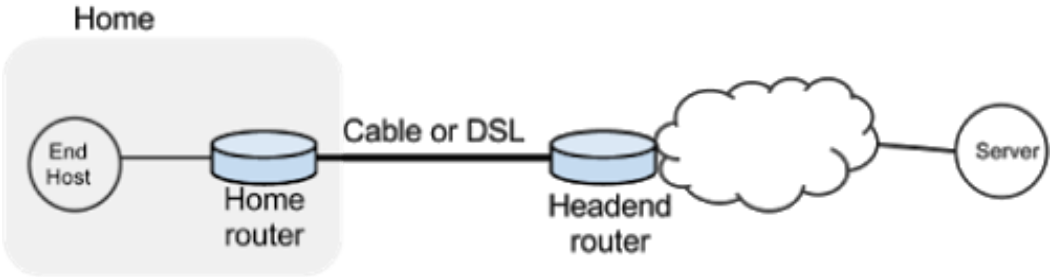
Our study of TCP is focused on two major topics:

1. The dynamics of congestion control algorithms like TCP Reno and TCP CUBIC in a "real" network. Reno is the canonical TCP congestion control algorithm (the one presented in the video lectures), and CUBIC is far more modern and is running in many Linux systems.
2. How excessively large router buffers can lead to poor performance in home networks, commonly known as the "Buffer Bloat" problem. Buffer bloat occurs when large packet buffers cause high latency and jitter (variation in packet delays) due to excess packet buffering.

Before the experiment

First, you will need to read and understand the concepts presented in the CUBIC paper. While a deep understanding of the math in the paper is not required to complete this project, you should make sure you understand the conclusions we can draw as a result of the calculations before starting.

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.



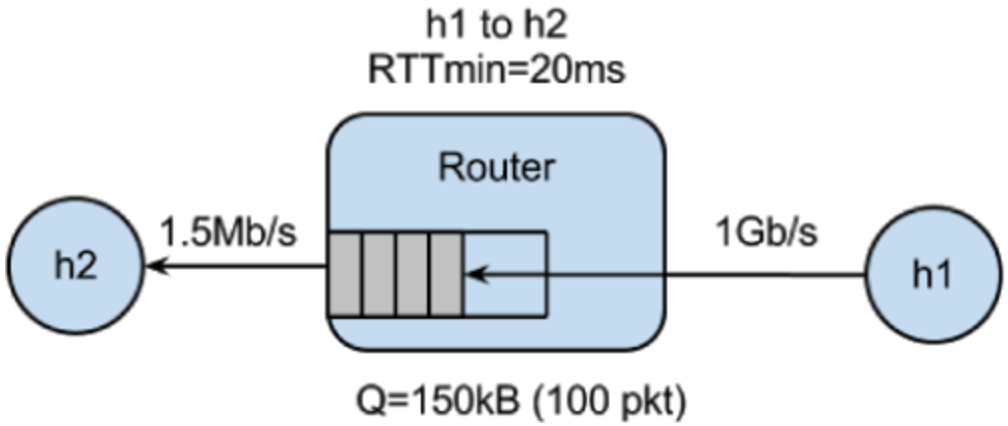
First, record your answers for the following questions after reading the paper (with respect to the diagram above) before you run any experiments:

Item #1: In this network, what do you expect the CWND (congestion window) curve for a long lived TCP flow using TCP CUBIC to look like? Contrast it to what you expect the CWND for TCP Reno will look like.

Item #2: Explain why you expect to see this; specifically relate your answers to details in the paper. (We expect you to demonstrate the ability to apply what you have already learned from the paper in your answer to this question, and that you've furthermore thought about what you read in order to make a prediction. So justifying your prediction is important. — It's okay if your prediction ends up being wrong as long as it was based on an understanding and analysis of the paper!)

The experiment setup

In a real network it's hard to measure the congestion window (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 simulate the network in Mininet so we can capture and measure these values. We will simulate the home network above using the following Mininet topology:



First, download and unzip the Project 6 zip file from T-Square onto the VM. Make sure the files have the right permissions. Go to your parent folder. `chmod -R 777 Project-6/`.

Part 1 - TCP Reno iperf

Let's examine how different different queue sizes affect how quickly the CWND grows in TCP Reno. We'll start with a small queue size, and then try a large queue size. From the lectures, you should know that the larger buffers introduce larger delays, which slows down the feedback loop for TCP's congestion control algorithm.

1. In the `Project-6` directory, you should see several `run*.sh` scripts. When executed, they will start Mininet, create the appropriate topology, and open the Mininet command line interface for you. For the first experiment, execute the following command:
    - `sudo ./run-minq.sh`

2. Next, we need to start the monitor which will capture information about the queue. In a second terminal window, (also in the `Project-6` directory) execute the following command:
    - `./monitor.sh small-queue`
3. Now we need to create some traffic in our topology. We'll use iperf, an active measurement tool that tries to shove data through the network as fast as it can. At the Mininet CLI, execute the following command to start sending data from `h1` to `h2`:
    - `h1 ./iperf.sh`
4. After waiting 90-120 seconds, switch to the monitor terminal and press "enter" to stop monitoring. This is enough time to capture both the slow-start phase of the congestion control algorithm and a few cycles of its steady state behavior.
5. On the Mininet terminal, issue the `exit` command to stop Mininet.
6. Next, we'll use the information captured by the monitor to render the data as a graph. Execute the following command in the monitor terminal:
    - `./plot_figures.sh small-queue`
7. Look at the new image files created in the `Project-6` directory. We are primarily interested in the queue occupancy of the switch and the CWND size for iperf. For now, you can ignore the `wget` graph, it will come into play later. Keep these graphs for later reference.
8. Next, we'll run the same experiment for large queues. The commands are slightly different, and so is the timing. In the last experiment, we used a queue size of 20 packets, but this time the queue size will be 100 packets - five times larger. To run the large queue experiment, execute the following commands:
    - `sudo ./run.sh` in one terminal to start the large queue topology and CLI
    - `./monitor.sh large-queue` in a second terminal to start the monitor
    - `h1 ./iperf.sh` at the Mininet CLI to start sending data
    - Wait ~10 minutes (yes, this takes much longer with large queues)
    - Stop the the monitor and close the Mininet CLI as we did in steps 4 and 5 above
    - `./plot_figures.sh large-queue` to plot the data collected by the monitor as graphs
9. 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.

Next, record your answer to the following question by comparing the large queue graphs we just created with the small queue graphs:

Item #3: With respect to queue occupancy, how are the graphs different? What do you think causes these differences to occur? (Make sure to include the graphs with your answer).

Part 2 - TCP CUBIC iperf

Now we will run the same experiment using a different congestion control algorithm, TCP CUBIC. You will need to run the commands in steps 1-8 above again, with `run-minq-cubic.sh` and `run-cubic.sh`, replacing the the TCP Reno run scripts. Also, when plotting the results, you should pass the names `small-queue-cubic` and `large-queue-cubic` to the plotting script so that you don't overwrite your previous results.

After generating your graphs for TCP CUBIC, record your answers to the following questions:

Item #4: What did you actually see in your CUBIC results? What anomalies or unexpected results did you see, and why do you think these behaviors/anomalies occurred? Be sure to reference the paper you read at the beginning of project the to help explain something that you saw in your results. Your hypothesis doesn't necessarily have to be correct, so long as it demonstrates understanding of the paper.

Item #5: Compare your prediction from **Item #1** with the congestion window graphs you generated for TCP Reno and TCP CUBIC. How well did your predictions match up with the actual results? If they were different, what do you think caused the differences? Which queue size better matched your predictions, small or large?

Part 3 - Resource Contention

So far our observations of TCP have effectively been in a vacuum, that is a single flow across the link. Since we rely on congestion control algorithms to adjust the transmission rate in response to the presence of other flows on the link, it makes more sense to observe how TCP behaves when there is contention for resources.

1. In the `Project-6` directory, start up the TCP Reno large queue topology with the following command:
    - `sudo ./run.sh`
2. First, we need some baseline measurements of performance. We will measure how long `h2` (the end host), takes to download a web page from `h1` (the server). Execute the following command in the Mininet CLI and make a note of the time (in seconds) required to download the file (use the timestamps output by wget at the start and end of the transfer):
    - `mininet> h2 wget http://10.0.0.1`
3. Additionally, let's obtain some baseline measurements of network delay by executing the following command and recording the average RTT:
    - `mininet> h1 ping -c 10 h2`
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 initiate a web request as in step 2 while a streaming video flow is running. We can simulate this TCP flow with iperf by executing:
    - `mininet> h1 ./iperf.sh`
5. 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.)
6. Next, re-run the ping command from step 3 above to see how the long flow has affected our network latency and RTT. Record this value.
7. Let's see how our long-lived iperf flow affects our web page download. Re-run the command from Step 2 above and record the download time.
8. Finally, exit the Mininet CLI:
    - `mininet> exit`

Record your answer to the following question:

Item #6: How does the presence of a long lived flow on the link affect the download time and network latency (RTT)? What factors do you think are at play causing this? Be sure to include the RTT and download times you recorded throughout Part 3.

Part 4 - Resource Contention continued

Let's dig a bit deeper into our experiment from Part 3 by collecting some data about the runs. We'll use a provided monitor and plotting scripts to obtain `cwnd` and buffer occupancy graphs.

1. First, start up the TCP Reno large queue topology with the following command:
    - `sudo ./run.sh`
2. In another terminal, go to the `Project-6` directory and start the monitor script using the following command:
    - `./monitor.sh experiment-1`
3. Next, start the long-lived flow in the Mininet CLI and initiate a web request:
    - `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`
4. Once the download completes, stop the python monitor script by pressing Enter in the monitor terminal, and exit the Mininet CLI in the other terminal. The cwnd values are saved in `experiment-1_tcpprobe.txt` and the buffer occupancy in `experiment-1_sw0-qlen.txt`.
5. Plot the TCP cwnd and queue occupancy with the following command:
    - `./plot_figures.sh experiment-1`
6. Examine the graphs generated. You should be able to see that the buffer in the Headend router is so large that when it fills up with iperf packets, it delays the short wget flow. It seems we have found an instance of Buffer Bloat. Let's look at two ways to reduce the problem.
7. The first method is to make the router buffer smaller, reducing it from 100 packets to 20 packets. To simulate this, we will rerun the experiment with a 20 packet buffer and compare the results:
    - `sudo ./run-minq.sh` - In one terminal to load the topology and bring up the Mininet CLI
    - `./monitor.sh experiment-2` - In another terminal to start the monitoring script (NOTE the change in experiment name)
    - `mininet> h1 ./iperf.sh` - (wait for 70 seconds such that the iperf stream is in steady state for its congestion window…)

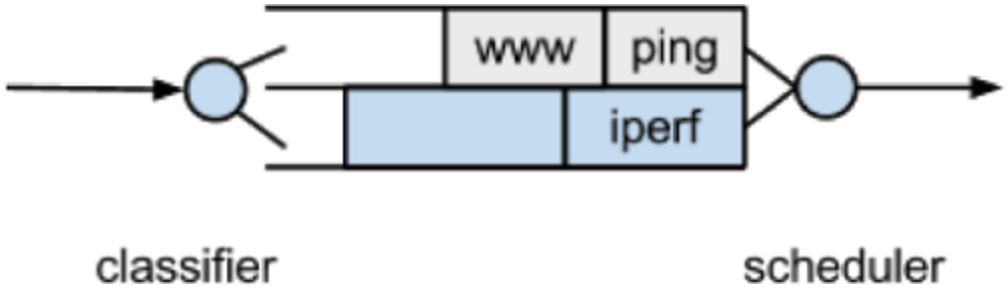
- `mininet> h2 wget http://10.0.0.1` - to run the web page download
- Stop the monitoring script by pressing enter in the monitor terminal and `exit` in the Mininet CLI
- `./plot_figures.sh experiment-2` - to plot the graphs from your collected data

Review your newly generated graphs and record your response to the following question:

Item #7: How does the performance of the download differ with a smaller queue versus a larger queue? Why does reducing the queue size reduce the download time for wget? Be sure to include any graphs that support your reasoning.

Part 5 - Using Traffic Control to prevent Buffer Bloat

When buffer bloat occurs, packets from the short flow are stuck behind a lot of packets from the long flow. Another approach to solving this problem is to maintain a separate queue for each flow and then put 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.



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 link 1 and link 2 for additional documentation.

1. Start Mininet again, but this time we will use a different run script:

- `sudo ./run-diff.sh`

2. Next, we will repeat some of the steps we have used throughout the experiment to analyze the performance of our traffic control technique:
    - `mininet> h1 ping -c 10 h2` - Record the network latency (average RTT) before starting the long lived flow
    - `mininet> h2 wget http://10.0.0.1` - Record the time required (in seconds) to download the file before the long lived flow
    - `mininet> h1 ./iperf.sh` - Start the long lived flow, simulating a streaming video download has begun. Wait ~10 minutes for the flow to stabilize.
    - `mininet> h1 ping -c 10 h2` - Record the network latency (average RTT) after starting the long lived flow
    - `mininet> h2 wget http://10.0.0.1` - Record the time required (in seconds) to download the file in the presence of the long lived flow

Using the information collected in the previous steps, record your answer to the following question:

Item #8: How does the presence of a long lived flow on the link affect the download time and network latency (RTT) when using two queues? Were the results as you expected? Be sure to include the RTT and download times you recorded throughout Part 5.

Part 6 - TCP CUBIC

Finally, you will repeat Parts 3 and 4 of the experiment using TCP CUBIC instead of TCP Reno. Repeat the steps in Parts 3 and 4 above using `run-cubic.sh` and `run-minq-cubic.sh` instead of their counterpart run scripts (`run.sh` and `run-minq.sh`). Also, increment the experiment names to `experiment-3` and `experiment-4` when starting the monitor so that you don't overwrite your data from the TCP Reno version of Part 3 and Part 4.

Compare the data and graphs you collect in these runs with the data and graphs you collected for TCP RENO, and then record your answer to the following question:

Item #9: Did the change in congestion control algorithm result in any significant differences in the results?

- If so, which algorithm performed better? What aspect of that particular algorithm do you think caused it to perform better?
- If there was no change, was this what you expected? Why or why not?

What to turn in

For this project, you must submit the following files generated during the experiment:

- `small-queue_tcpprobe.txt` - generated during Part 1 (be sure this was run with Reno / `run-minq.sh`)
- `large-queue-cubic_tcpprobe.txt` - generated during Part 2 (be sure this was run with CUBIC / `run-cubic.sh`)
- `experiment-2_sw0-qlen.txt` - generated during Part 4 (be sure this was run with Reno / `run-minq.sh`)
- `experiment-4_sw0-qlen.txt` - generated during Part 6 (be sure this was run with CUBIC/`run-minq-cubic.sh`)

You must also turn in a PDF file named `Responses.pdf` containing your responses to the **Items** asked throughout the experiment.

What you can and cannot share

For this project, you can share graphs generated by the experiment, but you are not permitted to share the raw data from which these graphs are generated. This means that you cannot share any experiment files included in the "What to turn in" section, as well as the others like them. Also, do not share any data you record during experiments. This includes the ping times and download times you collect throughout the experiment.

Grading

5 pts	Correct submission	for turning in appropriate and correctly named submission files from the experiment to T-Square .
10 pts	Experimental data	for submitting correct data that was output from your experiments.

35 pts	Response Items 1-9	for answering the items throughout the experiment, referencing the information from the Paper. Fewer points will be given if answers are incomplete or don't show understanding of the paper.

Notes

[1] Based on Mininet wiki.