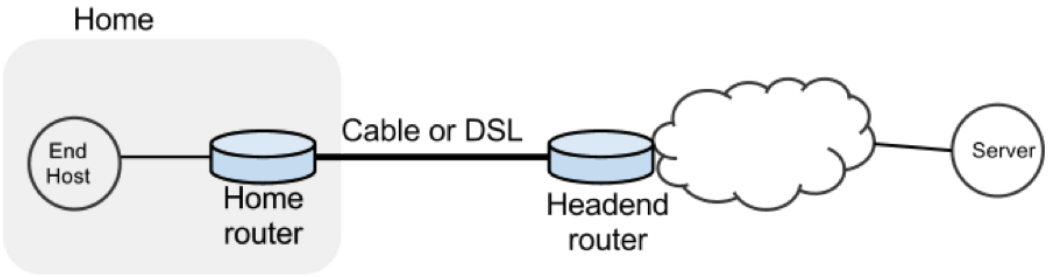
Bufferbloat on Network Simulator (NS-3)

Assignment Due: April, 22nd 2021 11:59 PM (Pacific Time)

This document contains all the instructions necessary for completing the assignment, including setting up the simulation environment. Some sections may be unnecessary depending on students' experience. Yet, it is highly recommended that the document is read in full before attempting anything. This minimizes the chances for missing hints that may be helpful while finding the solutions.

Bufferbloat

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.



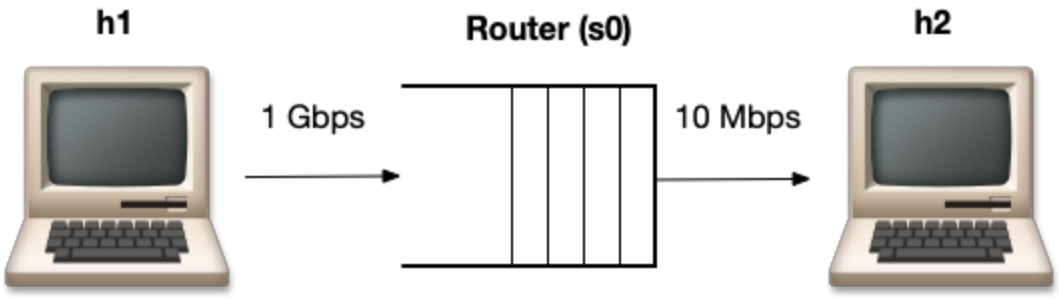
In a real network it's hard to measure cwnd (because it's private to the server) and the buffer occupancy (because it's private to the router). To ease our measurement, we are going to simulate the network in Network Simulator (NS-3).

Goals

- Learn the dynamics of TCP sawtooth and router buffer occupancy in a network.
- Learn why large router buffers can lead to poor performance which is often called "**bufferbloat**."
- Learn how to use NS-3 to create network topologies, run traffic generators, collect statistics and plot them.
- Learn how to package your experiments so it's easy for others to run your code.

Tasks

Within NS-3, create the following topology. Here h1 is a remote server (ie google.com) on the Internet that has a fast connection (1Gb/s) to your home router with a slow uplink connection (10Mb/s). The round-trip propagation delay, or the minimum RTT between h1 and h2 is 40ms. The router buffer size can hold 100 full sized ethernet frames (about 150kB with an MTU of 1500 bytes).



Then do the following:

- Start a long lived TCP flow sending data from h1 to h2. Use the BulkSend application provided with NS-3.
- Trace the congestion window and the delay of the TCP connection.
- Trace the bottleneck queue size throughout the simulation.
- Plot the time series of the following:
    - The long lived TCP flow's cwnd vs time
    - The RTT experienced vs time
    - Queue size at the bottleneck vs time

Repeat the above experiment and replot all three graphs with a smaller router buffer size (Q=20 packets). Make sure to run the simulations long enough so that the connection can stabilize.

Setting up the Simulation Environment

NS-3 is supported and currently tested on the following primary platforms:

1. Linux (x86 and x86_64): gcc/g++ versions 4.9 and above

    Note: If you are using RHEL or Centos, you will likely need to install a more up-to-date compiler than the default; search for how to enable 'software collections' or 'devtoolset' on these distributions. Other Linux distributions typically have a suitable default compiler (at least version 4.9).

2. MacOS Apple LLVM: version 8.0.0 and above (version 7.0.0 may work)
3. FreeBSD and Linux (x86_64): clang/LLVM version 3.9 and above (older versions down to 3.3 may work)

The minimum Python version supported is currently version 3.5 or greater (major version 3).

If your own computer meets the requirements above, you can safely skip the VM setup section below and simply run your simulations on your computer. If this is not the case, don't worry! Next, we describe how to set up a VM that would run the simulations. If you have already set up a VM in previous assignments, you can skip this section.

Setting up a VM

As you've read above, NS-3 works pretty much any recent Linux version, so feel free to simply use a generic Linux VM to run your simulations. In order to give you a more comprehensive "Computer Networks Researcher" VM (because you are one), we will describe setting up a VM with other useful networking research tools installed.

- Visit http://mininet.org/download/ and download Mininet VM (mininet-2.3.0-210211-ubuntu-16.04.7-server-amd64-ovf.zip)
- Note that we are downloading Ubuntu 16.04.7 (not the most recent one)
- VM is about 1 GB, so it might take a moment to download
- Follow VM Setup Notes (http://mininet.org/vm-setup-notes/) to import the downloaded VM with default settings
- Before starting the VM, Settings > Network > Adapter 1 then select "Bridged Adapter". This will allow you to ssh into the VM once you turn it on.
- If you encounter a "Kernel driver not installed" error, you can follow the steps at https://medium.com/@Aenon/mac-virtualbox-kernel-driver-error-df39e7e10cd8 to solve it
- Both the username and password for the VM are `mininet`.
- Once you logged into the VM run `ifconfig` to figure out the IP address of your VM. It should look like the following:

```
mininet@mininet-vm:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:12:21:f8
          inet addr:192.168.1.181  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:151 errors:0 dropped:0 overruns:0 frame:0
          TX packets:143 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:24084 (24.0 KB)  TX bytes:13772 (13.7 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

- In our case the address of the VM is 192.168.1.181, so we will run `ssh -L 8000:localhost:8000 mininet@192.168.1.181` when logging into it from our terminal.
- There is a step in Mininet Walkthrough where Wireshark is tested. If it doesn't work for you don't worry, we will not use Wireshark for our assignments.
- Since the Linux (and Python) versions of the VM are a little bit old we will need to upgrade some modules for our assignments. Run the following commands:
  - `pip install --upgrade pip==9.0.1`
  - `pip install termcolor --user`
  - `pip install matplotlib --user`
  - `sudo apt-get install curl`
- Now the VM is ready for running assignment programs.

Setting up the NS-3 Environment

NS-3 is one of the most widely used discrete event network simulators in the community. It allows researchers to simulate large scale networks without actually having access to a large lab. Many influential papers in the field tend to include NS-3 simulations to convince their readers. Therefore you are highly encouraged to learn how to use it. It will probably be useful for your final project as well. :)

To help you get started, we provide a basic skeleton code. In order to fetch the starter code, run the following:

```
git clone --single-branch --branch cs244-simulations https://github.com/serhatarslan-hub/ns-3-dev-git.git

cd ns-3-dev-git
```

This clones a custom NS-3 repository that includes the starter code for the assignment. For your future research projects, we highly recommend that you fork the original NS-3 repository (https://github.com/nsnam/ns-3-dev-git) and work on your own fork.

In order to build the new NS-3 environment you downloaded, run the following:

```
./waf configure --enable-examples
```

This may show some settings as not supported, but don't worry! It has all the modules we need. Now run the following to actually build the simulator. It normally takes a couple of minutes, but in our case it will not compile because the starter code has a bunch of TODOs. Come back to this step once you complete the starter code. Or you can simply move scratch/bufferbloat.cc to some other location and complete this step to make sure the simulator works fine. Then remember to bring back this file to its original location.

```
./waf
```

In order to check everything works as expected, you can run `./waf --run scratch/scratch-simulator`. If this prints "Scratch Simulator" on the terminal, you are good to go!

The Assignment Code

NS-3 has many files for its own infrastructure. However we will not get into details of how NS-3 modules are written in this assignment. We will only look into a couple of simulation files that have the starter code. Look for TODOs in the starter code. The following are important files you will find in the repository. Ignore other files.

File	**Purpose**
scratch/bufferbloat.cc	Creates the topology, runs the simulations and measures cwnd, queue sizes and RTTs.
scratch/plot_bufferbloat_figures.py	Plots all the required figures for this assignment. There are no TODOs in this script, but feel free to take a look if you are interested in learning more about how the trace files are converted into figures.
run.sh	Runs the experiment and generates all graphs in one go. It starts up a simple http server which allows you to see the created figures at once easily.
index.html	The simple html file to show the created figures.

Note: We recommend you using a version control system to track changes to your assignment and back it up regularly on the "cloud." Both Github and Bitbucket host private git repositories for free. Do NOT use a public repo; you will run the risk of violating the Honor Code. This applies for all the assignments; feel free to use a public repo for the final project.

Submission/Deliverables

Assignment Due: April, 22nd 2021 11:59 PM (Pacific Time)

Please upload all of the following to gradescope:

- **Final Code**: Remember one of the goals of this assignment is for you to build a system that is easy to rerun to reproduce results. Therefore, your final code MUST be runnable

as a single shell command ("`sudo ./run.sh`"). We will test your code in the same setup.

- **README**: Create a file named `outputs/README.md` with instructions to reproduce the results as well as the answers to the below questions. Please identify your answers with the question number, and please keep your answers brief. Please list installation instructions for any dependencies required to run your code.
- **Plots**: There should be 6 plots in total, 3 each for router buffer sizes 100 and 20 packets. They MUST have the following names and be present in your submission folder
  - `outputs/bb-q100/cwnd.png, outputs/bb-q100/q.png, outputs/bb-q100/rtt.png.`
  - `outputs/bb-q20/cwnd.png, outputs/bb-q20/q.png, outputs/bb-q20/rtt.png.`

Questions

Include your answers to the following questions in your README file. Remember to keep answers brief.

1. Why do you see a difference in cwnd sizes with short and large router buffers? What is the relationship between these cwnd values and the bottleneck buffer occupancy? What are the consequences of having such queue occupancy trends in a network?

2. How does the RTT vary with the queue size? Describe the relation between the two.

3. Bufferbloat can occur in other places such as your network interface card (NIC). Check the output of `ifconfig eth0` on your VirtualBox VM. (On MAC-OS, you can run `sysctl net.link.generic.system.sndq_maxlen`) 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, and all the packets are MTU size, what is the maximum time a packet might wait in the queue before it leaves the NIC?

4. Identify and describe two ways to mitigate the bufferbloat problem.

5. Compare your experience with simulator vs emulator. Describe scenarios where one would be more useful than the other.

Grading

Total: 10 pts

- 3 points: Producing working code that generates the graphs
- 5 points: Answering the questions
- 1 point: Producing correct graphs
- 1 point: Design and code quality

NS-3 Tutorial

The NS-3 tutorial (https://www.nsnam.org/docs/tutorial/html/) has a pretty comprehensive curriculum to teach details about the simulator. Feel free to read modules from the tutorial to learn more about particular topics.

Very Useful Hint

The whole codebase of NS-3 is documented via Doxygen (https://www.nsnam.org/docs/release/3.33/doxygen/index.html) which is the key to using this massive simulator. You will definitely need to search the name of the class that you are interested in the search bar at this web site to learn more about the APIs available for it. In fact, most of the TODOs in this assignment are just "go read about this class and find the correct function" type of instructions.

Tracing is one of the most valuable features of a simulation environment. It means we can get to see evolution of any value / state we are interested in throughout the simulation. Basically, in NS-3, you set some tracing options for pre-defined TraceSources and provide a function that defines what to do when the traced value changes. You can learn more about tracing at https://www.nsnam.org/docs/tutorial/html/tracing.html If you are going to work with NS-3 in the future, you will definitely need to read this page. We highly recommend reading it before working on the starter code.