CS6250 Project 1 – Virtual Machine (VM) Setup, Defining Topologies and Simulating Networks

This document covers the following:

- Project Goals
- VM Setup Directions
- Defining Topologies
- Network Simulation
- Data Center Topology
- What to Turn In
- What You Can and Cannot Share
- Additional Resources
- Project Notes
- Project Grading

Project Goals

This project has three goals: to set up the virtual machine (VM) that we will be using for the projects in this course; to learn how represent network topologies in Mininet; and to practice how to simulate basic network commands on these topologies from the Mininet command prompt. Mininet is a network simulator. It runs multiple Linux containers for individual hosts and uses Open vSwitch for network device emulation.

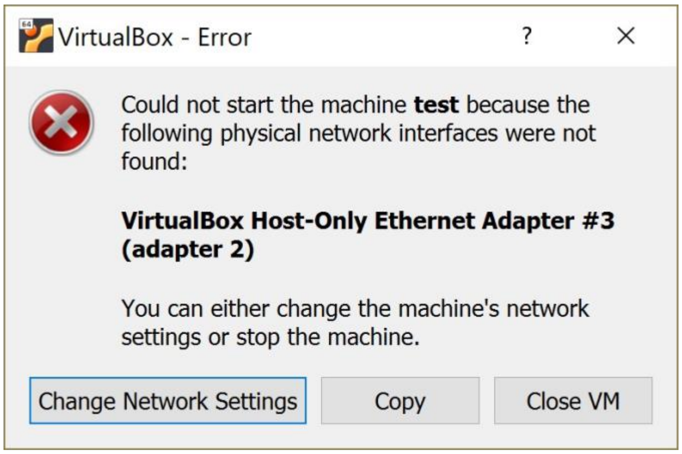
This project is split into four parts: Setup, Static Topologies, Simulation, and Dynamic Topologies. The Setup stage is reasonably straightforward. You must download and setup a VM in VirtualBox. In the second part, you will learn how to represent static network topologies in Mininet. Third, you will learn how to run basic network commands on these topologies using the Mininet command line interface. Finally, you will use what you have learned to create a dynamic datacenter topology that can be defined at runtime using command line parameters, and to verify that the network simulation works properly.

VM Setup Directions

1. Download and install the latest VirtualBox for your platform. You can find VirtualBox here.
2. Download the CS6250 virtual machine image (please see links in Piazza post “Course Virtual

Machine (VM) Tips and Tricks”). The download is ~2.2 GB in size so be patient with the download and, if possible, connect your computer to the Internet via a wired connection. If the download is especially slow, setup your computer to download the image overnight.

3. In VirtualBox select File -> Import Appliance and select the .ova you just downloaded. Virtualbox will show you the VM settings and you can then click Import.
4. Start the VM by clicking Start.
   a) If on your first start of the VM you get an error like the following:



   Then try: Change Network Settings and set Adapter 1 to Bridged Adapter selecting an adapter specific to your host machine and continue.

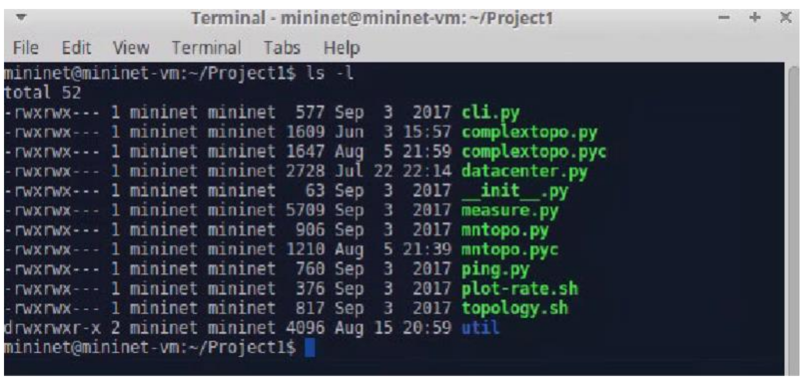
   b) If on your first start of the VM you get an error for the second Network Adapter, try selecting the Host-only adapter option for the second network adapter and continue.

   c) At this point it may be easier on your eyes to select (from the VirtualBox menu): View -> Virtual Screen 1 -> Scale to 250%
5. Log in to the VM using mininet for the username and password.
6. Open up a terminal in the virtual machine.
7. Now we will run a test to ensure Mininet is working correctly. Type `$ sudo mn --test pingpair` (you will have to enter the password mininet again, as running Mininet requires root privileges.
8. The output should look like Results: 0% dropped (2/2 received).

Issues with VM - If there are any questions about the VM, please post them to Piazza. But expect that your environment may require some customization.

Defining Topologies

1. The starter code required for this project is available on Canvas as Project1.zip. Download this directly on the VM.
2. Use the following command to unzip the files
   - `$ unzip Project1.zip`
3. The above command should preserve original file permissions, but if you run into permission errors while working on the project, ensure the permissions match the following:
   - In the new Project1 folder (`$ cd Project1`), use the `$ ls -l` command to view permissions and use `$ sudo chmod -R 777 .` command to change them if required (**note period . at end of command**).



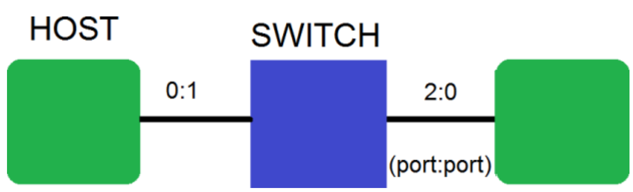
4. You can now run the example topology provided to simulate a host communicating with another host. Change into the project directory and run the topology using the following commands:
   - `$ cd Project1`
   - `$ sudo ./topology.sh`. (This step may take a minute.)
5. The script produces a time-stamped results folder that contains some raw data as well as a couple of graphs. One is the TCP congestion window (`cwnd.png`) and another is the bandwidth in megabits per second (`rate.png`). To view the graphs, use the command:
   - `$ display {image file name}`

The bandwidth graph should show a constant rate of about 10 Mbps, and the congestion window graph should show a familiar pattern if you've had an earlier networking course or are otherwise familiar with TCP (but if you aren't, don't worry - we'll learn about this pattern later in the class!)

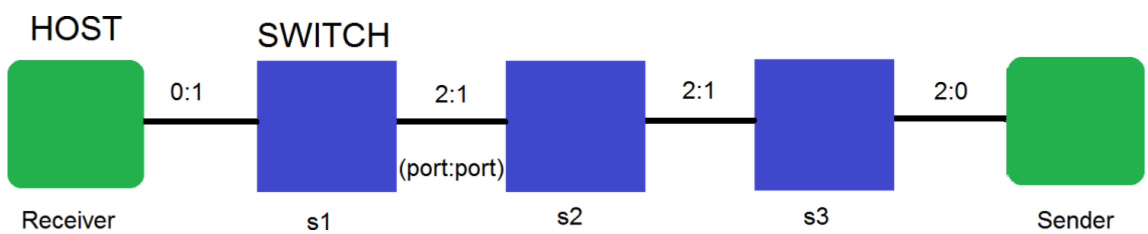
6. Now you will modify the Mininet topology to add more switches. The current topology is setup as shown in the first image below (2 hosts, 1 switch, 2 links). You will modify the topology to the second topology shown below (2 hosts, 3 switches, 4 links). To modify the topology, you should edit the Mininet topology file `mntopo.py`. (See if you can understand how this code is creating the hosts,

switches, and links in the topology. Refer to the Mininet documentation to help you piece apart the topology file). You should add two new switches and two new links to the topology. When you have modified the topology, re-run the topology test script (`$ sudo ./topology.sh`). The graphs should be similar to the graphs produced in your earlier test run. The similarity should come as no surprise because the new switch and links in the topology are adding a slight amount of total latency but have the same bandwidth properties as the other links. NOTE: Be sure not to add or leave extra links in the topology!

Topology provided:



Topology you will create:



7. The next two steps involve tweaking topology parameters and observing their outputs. First we will modify the latency of the topology. Before we modify the latency, we will test the current latency using the `ping` command. Run the following from the project folder:
   - `$ sudo python ./ping.py`

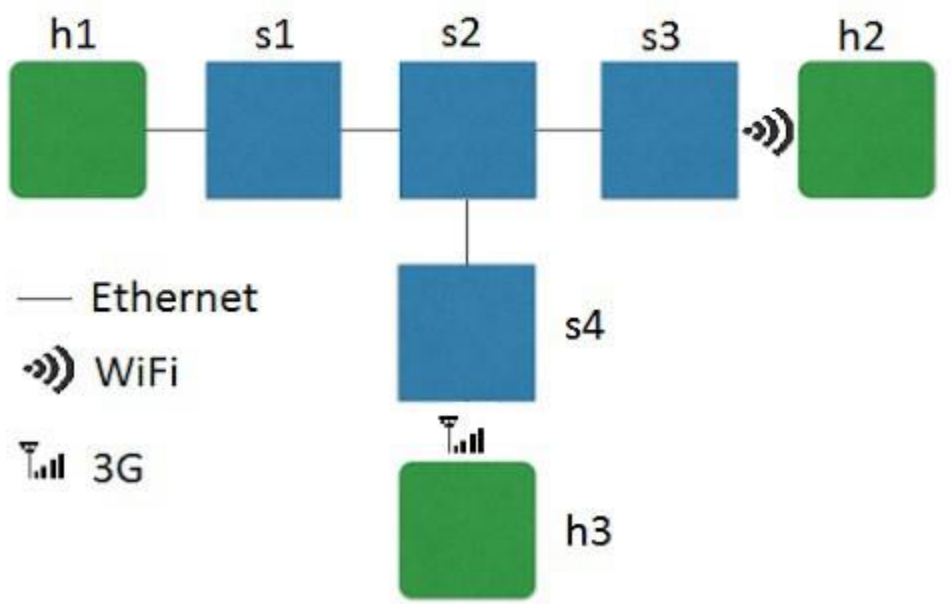
8. You should see results around 8-10 ms. (If the first one is a bit longer, that's normal. It's likely due to time required for ARP to run - if you don't know about ARP yet, that's okay; we'll learn about it later in this course!) To modify the latency we will adjust the delay on the links in the `mntopo.py` file. Adjust the `delay` parameter in the `linkConfig` dictionary to 10ms. Then run ping script again (`$ sudo python ./ping.py`). This time you should see pings just a bit over 80 ms. This is the time for one packet to traverse four links to the receiver, and the ping reply to traverse the same four links back to the sender. The beauty of Mininet is these configuration parameters allow us to emulate real network events without modifying common network tools like `ping`.

9. Now we will modify the bandwidth and observe the change in the topology. Adjust the `bw` in the `linkConfig` dictionary to 50 which will adjust the bandwidth along each link to 50 Mbits per second (Mbps). To confirm Mininet emulates this correctly, re-run the topology test script (`$ sudo`

`./topology.sh`) and then view the `rate.png` output graph using `display` as in step 5. Does the graph match what you expected to see after you changed the `bw` parameter?

> **NOTE:** Make sure you save your output as well as `mntopo.py` after this step, it is a deliverable for this project.

10. To exercise what we have just learned, we will create a new topology representing a more complicated network topology:



11. To create your topology, edit the starter file `complextopo.py` and create three hosts, `h1`, `h2`, and `h3`. Next create four switches, `s1`, `s2`, `s3`, and `s4`. It is important that your hosts and switches use these names for grading purposes. Then add the links between these hosts and switches as depicted above. The properties for each link type are provided below:
    - Ethernet: Bandwidth 25 Mbps, Delay 2 ms, and loss rate 0%
    - WiFi: Bandwidth 10 Mbps, Delay 6 ms, and loss rate 3%
    - 3G: Bandwidth 3 Mbps, Delay 10 ms, and loss rate 8%

NOTE: We recommend that you not specifying port numbers when configuring your hosts and switches. What port numbers does mininet use when you do not specify them? Note that the that default starting port number for hosts is different than for switches. We will interact with this topology in the next section!

Network Simulation

1. Using our complex topology, let's explore how to simulate basic network commands over our

topology. To do this, we will launch Mininet's command line interface, or CLI for short. To launch the simulation, run the following command:

- `$ sudo python ./cli.py`
- After Mininet loads the complex topology, you should see the Mininet command prompt: `mininet>`

2. Now let's run some commands. To execute a command on one host, type the host name followed by the command. For example, let's test the connection between h1 and h2 by typing the following at the Mininet prompt:
    - `$ h1 ping h2 -c 10`

This will cause h1 to ping h2 with 10 packets of data and print out results like those in steps 7 and 8 above. Due to the loss rates on the wireless links, you may see some packet loss occur during the ping command execution.

3. Let's perform a casual experiment. Issue a 100-packet ping command from h1 to h2, and then a 100-packet ping command from h1 to h3. How do the reported statistics differ across the two different wireless links?
4. Another useful command provided by Mininet is `pingall`. This command issues ping commands between all hosts on the topology and can be useful to verify that your topology is connected. Issue this command at the Mininet prompt. A failed ping between two hosts is indicated by an `X`. You may see an X in the results of your `pingall` due to the loss rates on the wireless links, but you can run the command again to confirm the topology is behaving as you intended.
5. We will explore more complex experiments as the course continues, but for now we are finished! To close the Mininet simulation, type `exit` at the Mininet prompt.

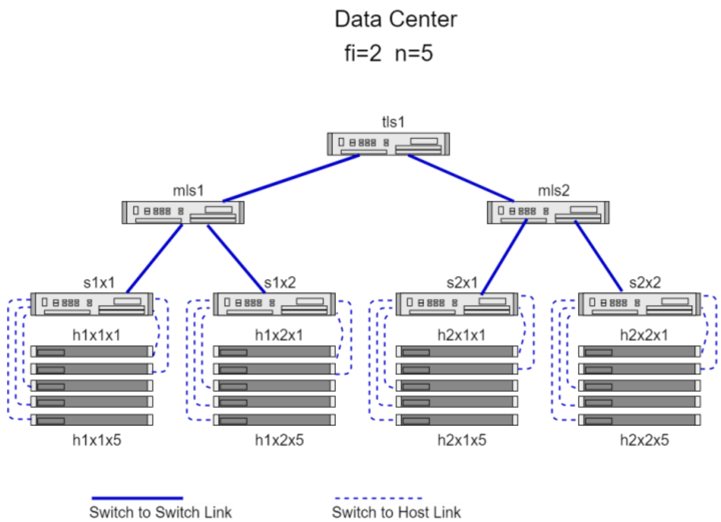
Datacenter Topology

Since we are defining our Mininet topologies in a programming language, it may have occurred to you that we can define topologies dynamically prior to the initialization of the network. This is particularly useful for creating large scale topologies and automating network testing. In this last section of the project, you will create a datacenter topology that builds a custom topology and launches the Mininet CLI.

Our custom datacenter topology will emulate a **fan in** type topology where there will be a top-level switch (tls) connected to a number of mid-level switches (mls) which are connected to rack switches with a number of hosts connected to them. The ratio of rack switches to mid-level switches will be the same as the number of mid-level switches connected to the top-level switch.

Our custom topology is defined by 2 parameters:

- ***fi***: Fan-In rate. The number of mid-level switches connected to the top-level switch. This will also be the same number that represents the number of rack switches connected to the mid-level switches.
- ***n***: The number of hosts connected to each rack switch



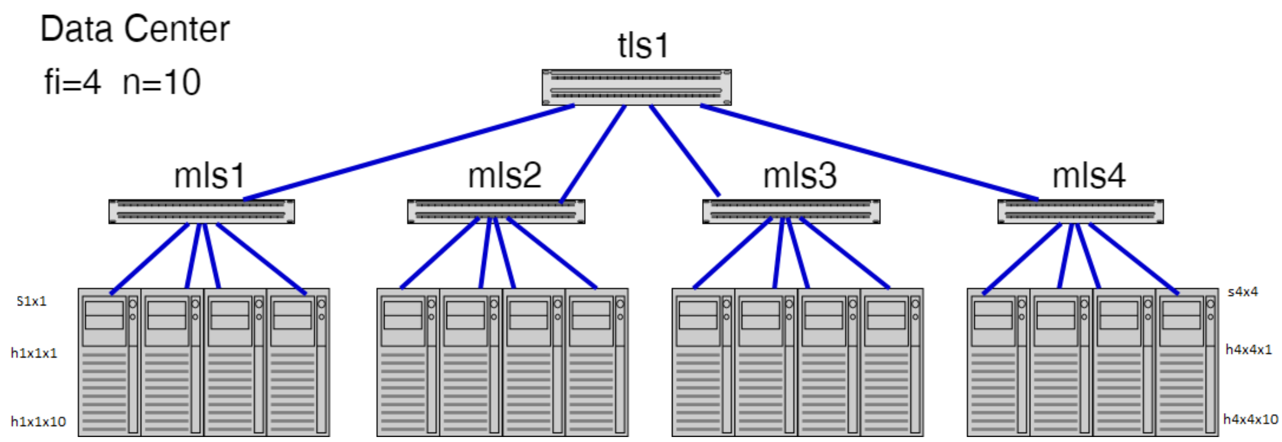
It is important for grading purposes that the mid-level switches are named mls1, mls2, to mls*fi*. Lower level rack switches should be named s1x1, s1x2... to s*fi*x*fi* and hosts are named h1x1x1, h1x1x2, to h*fi*x*fi*x*n*. **Your code must use this naming convention to receive full credit**. To implement your datacenter topology, complete the TODO sections marked in datacenter.py.

Once complete, you should be able to launch your topology and enter the Mininet CLI with the following command: `$ sudo python ./datacenter.py`.

For example: `$ sudo python datacenter.py --fi 2 --n 2`

You can verify your topology is properly connected using the command line interface. You can print out the network configuration using the **dump** command, as well as using the **pingall** command to verify all hosts can reach each other. You can verify all the nodes were created as expected with the **nodes** command.

Below is a graphic example of a larger datacenter topology. In the past some student host machines were not able to handle simulating a topology of this size. If that is the case on your host machine don't worry about it – thoroughly test your code using smaller topologies built with smaller values of fi, n.



Note the args variable used in the template datacenter.py file. This variable is used to run your DataCenter from the command line. **Do not use this args variable in your Datacenter implementation.** When grading your submission, the autograder will create an instance of your DataCenter class and run tests on that object. It will not have access to the args variable so any references to it at runtime, outside of the command line interface, will fail, and you will not receive full credit. In other words, **just simply use the variables "fi" and "n" in your code, do not use "args.fi" or "args.n"** in your code.

What to Turn In

To complete this project, submit your `mntopo.py` file, `bwm.txt` raw data and `rate.png` image files generated in ***Defining Topologies: Step 9***, your `complextopo.py` file from ***Defining Topologies: Step 11***, and your `datacenter.py` file created in ***DataCenter Topologies*** to Canvas as five separate files in a zip file named **GTLogin_p1.zip** where GTLogin should be replaced with your ID you use to log into Canvas (e.g., smith7 in smith7_p1.zip). Do ***not*** modify the names of those files or else grading will be affected and you may receive a zero. The file names must be ***exactly*** as stated here, and the directory scheme must be the files at the top level when extracted from the **GTLogin_p1.zip** file. When extracted your top level folder should contain:

- `mntopo.py`
- `bwm.txt`
- `rate.png`
- `complextopo.py`
- `datacenter.py`

What You Can and Cannot Share

For this project, you are encouraged to share your experience/assistance in setting up the course VM on Piazza. Due to variations in computing platforms, virtualization software, and operating systems, VM setup can be painful for some students. This portion of the project is ungraded, so please don't hesitate to discuss / troubleshoot on Piazza.

You are *not* permitted to share code from `mntopo.py`, `complextopo.py`, or `datacenter.py` on Piazza or other platforms. Additionally, you are not permitted to share the contents of your experiment data (`bwm.txt` files) on Piazza or other platforms. You *are* permitted (and encouraged!) to share `rate.png` and `cwnd.png` files on Piazza with other students, and discuss how these simple experiments lined up with your expectations (or didn't!).

Additional Resources

Project Descriptions will frequently provide additional resources towards the end. These are not required reading so as to not cause the project to become overwhelming but will often contain helpful tutorials or additional information for completing the project or taking the project one step further.

This Mininet walkthrough may be helpful for this project.

Project Notes

The course VM is this course's common operating platform. It is designed specifically for compatibility with our project code, and as a result uses some packages considered to be legacy, this is an intentional design. Therefore, students are ***highly*** encouraged to complete and turn in all projects via the course VM (using the provided browser and GUI). All projects are developed and graded in this exact same VM, to ensure consistency and eliminate platform dependency issues. In the interests of maintaining this consistency, students ***should not*** perform any of the actions throughout the duration of the course:

- Install new software or apply package or OS updates (unless instructed to by the professor or a TA). Students in past semesters have installed IDEs such as PyCharm without issue so you can do that if you prefer.
- Alter the VM's hardware virtualization settings unless necessary. Some changes may improve

the VM performance on your system, like increasing available memory, video memory etc. and shouldn't affect the projects, however some changes can affect projects, like adding CPUs for example. If in doubt, undo hardware virtualization setting changes and ensure your project still runs as expected prior to turn in.

- Using shared/mounted folders with your host systems. These are unnecessary and sometimes cause issues, particularly with projects involving mininet.
- Transfer files to be submitted to Canvas to a different platform (i.e Windows OS) before turn-in. Specifically, do not open code in Windows because this alters the line endings and will cause the files to not run in the VM or to fail the auto-grader.
- Using tabs instead of spaces in Python files. **Using tabs frequently causes the autograders to fail**.
- Not removing print statements. **Leaving unnecessary print statements in your files may affect the autograders.**
- Specifically, for this project you may lose points if:
  - You don't follow the proper naming convention for switches and hosts
  - You don't use correct link configurations
  - You use tabs instead of spaces
  - You leave print statements in the final submitted program
  - Your files zip file contains a folder and does not have the submission files at the top level.

Project Grading

5 pts	Correct Submission	For turning in all project files with the correct names, and significant effort ha been made in each file towards completing the project.
10 pts	Simple Topology	The topology in `mntopo.py` is correctly implemented, the output in `bwm.txt` i correct, and the `rate.png` file is consistent with the experiment conducted.
10 pts	Complex Topology	The topology in `complextopo.py` is correctly implemented, and commands issued against the topology run without error and produce the expected output.
25 pts	Data Center Topology	The script created in `datacenter.py` is correctly implemented, and correctly generates topologies according to specified parameters. We will test your script against several different test cases to determine if it correct.