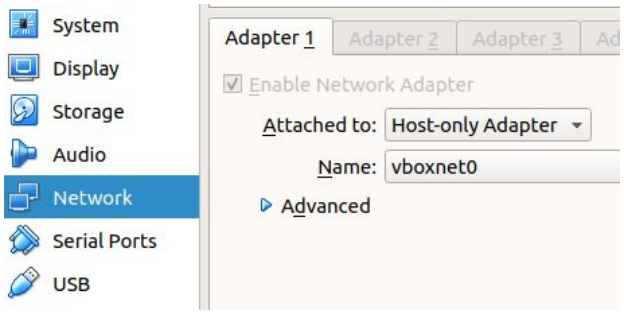
Virtual networks provide a good framework for testing and experimenting with networking environments in a safe and configurable way. Additionally, SDNs (Software Defined Networks) provide an added level of ease of network management for both real and simulated environments through centralized configuration and monitoring.

Your Assignment:

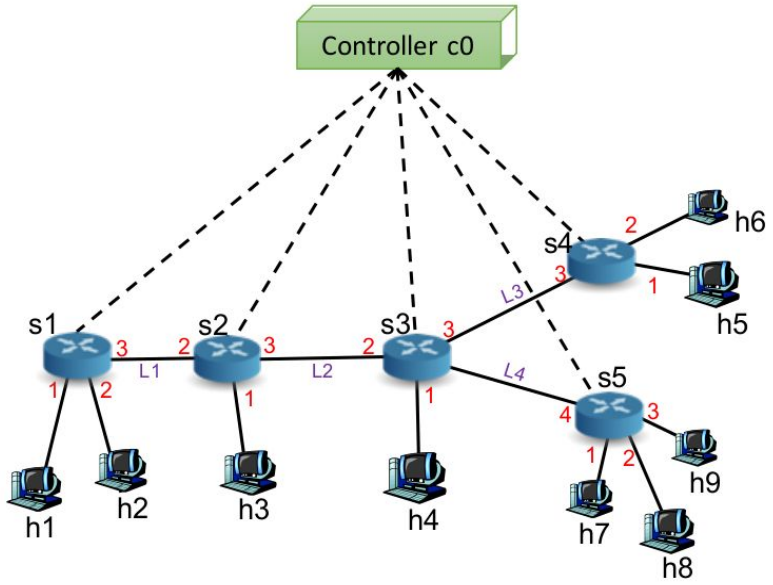
- You will create virtual networks with Mininet, evaluate them, and then build SDN controller logic for them
- Setup the Testing Environment
    - Download the VM containing Mininet
        - (download the i386.zip one, and if it doesn't work later try the amd64.zip)
    - Install a Virtual Machine Manager (VMM)
        - You can use any virtualization system of your choice, but we recommend installing Virtualbox (and the follow instructions use it). It is free and runs on Windows, Linux, and macOS.
    - Start your VMM (Oracle Virtualbox here), and in the **"File"** menu, select **"import Appliance"**.
    - Click **"Choose"** (a folder icon may also be **"Choose"**), browse to the location containing Mininet-VM (the .ovf file you downloaded), and click **"Open"**, and click **"Continue"**.
    - Click **"Import"** and then select the imported virtual machine
    - Select **"Host Network Manager"** from the **"File"** menu
    - Click **"Create"**, you should see a new network named like **"vboxnet0"** or similar
    - Click **"Enable DHCP"** in the **"vboxnet0"** entry
    - Now go back to main menu, click on the Mininet-VM, click on **"Settings"**
        - Select **"Network"** and then set **"Attached to"** to **"Host-only Adapter"** then click **"OK"**



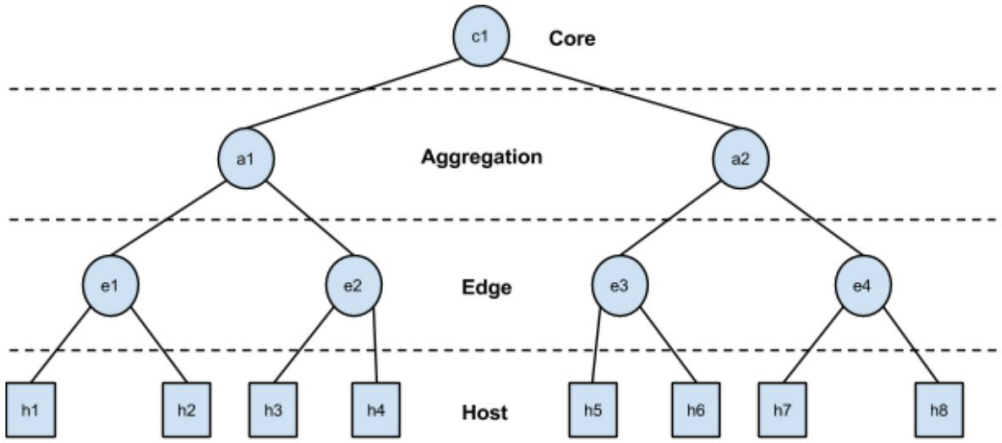
    - Select the Mininet-VM and click the **"Start"**
    - Login (or SSH) with
        - mininet-vm login: mininet
        - Password: mininet
    - Additionally, if your VM won't boot to login, check your BIOS/UEFI settings to ensure virtualization is enabled. To do this you will need to look up how to access your bios for your machine. For example, a lenovo laptop can access its bios by

restarting and hitting enter as soon as the initial boot screen appears and from there the bios configuration can be accessed.

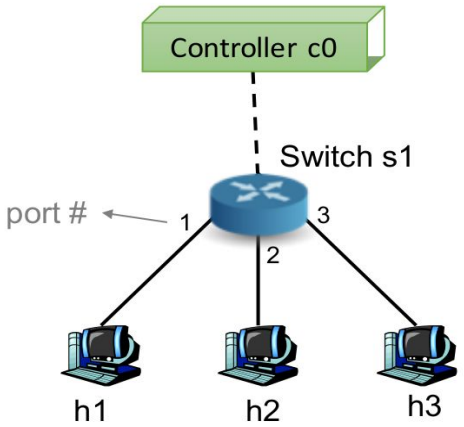
- Consider Topology A:



  - Is it possible to estimate the throughput and latency between the switches for the links L1, L2, L3, and L4? If yes, provide the estimated throughput and latency for each of the links and explain the process used to estimate them in detail. If no, justify your response with an explanation. Write your answers in a file named (**topology-a-report.txt**)
    - Note: To run topology A's mininet script use the command "sudo python topology-a.py"
    - Note: The hosts (h1 - h9) are assigned IP addresses 10.0.0.1 through 10.0.0.9 (h1 is 10.0.0.1, h9 is 10.0.0.9).
    - Note: You may use Iperf and Ping to answer these questions
- Create Topology B with Mininet's Python API (using the skeleton code **topology-b.py**)



    - Note: Each level (i.e., core, aggregation, edge and host) is composed of a single layer of switches or hosts.
    - Note: All links between hosts and edge switches will have the same predefined performance parameter. Here there are 3 types of links, the links between core and aggregation switches, the links between aggregation and edge switches, links between edge switches and host. You will see these specific parameters in the skeleton code (**topology-b.py**)
- Evaluate Topology B similarly to Topology A (measurements of throughput and latency for links between switches (example: e1 to a1, a1 to c1)
    - Write your answers in a file named (**topology-b-report.txt**)
- Create an Ethernet-based self learning algorithm
    - Consider the Topology C:



        - **Switch s1** maintains a flow table which contains match-action rules that are added/removed/updated by the controller. At the beginning, there are no rules installed in the flow table. When packets are sent by hosts to one another, the controller extracts information from these packets and builds certain internal data structures (such as a hashmap, etc.) to represent the topology. The controller will use these internal data structures to decide what rules should be installed in the switch's flow table.
        - As an example, consider **host h1** wants to send a packet to **h2**. When **switch s1** receives this packet from **h1**, it checks if there is any corresponding match-action rule (or flow entry) for this packet in its flow table. If there is no entry, the switch will encapsulate the packet and send it inside as an **OFPacketIn** message to the **controller c0**. On the other hand, if there is a flow entry, **switch s1** will execute the action associated with the rule. However, as discussed earlier, the flow table is initially empty. Therefore, the switch will forward the packet to the controller. When the controller receives the **OFPacketIn** message from **switch s1**, a packet-in event is triggered. Two tasks are performed due to a packet-in event: 1) using the information from packet headers/event, the controller may update its internal state (e.g. it learns something new about the

topology), 2) using the internal state, it makes a decision on what action(s) **s1** should take to send this packet to **h2**. If the controller knows what interface **s1** should forward the packet to, it will inform **s1** and also install a flow entry which could be used for future packet transmissions of similar kind. If the controller does not know what interface **s1** should forward to, then it will instruct the switch to flood the packet to all the interfaces except the one. This flooding instruction is a one-time action without installing any rules in the switch.

- Try to come up with an Ethernet-based self learning algorithm at the controller that does not unnecessarily flood packets while learning the topology.
  - *Hint: Try to test your algorithm under multiple scenarios such that the controller is able to learn the whole topology quickly without unnecessarily flooding packets. For example, one such scenario could be **h1** sends a packet to **h2**, then **h2** sends a packet to **h1**, then **h3** sends a packet to **h1**. Another direction to think about is if your algorithm should be source-based, destination-based, or a combination of multiple packet header fields.*
  - Write this in your submission following the **"What to Submit"** guidelines
- Implement your algorithm (using the skeleton code **ethernet-learning.py** or **EthernetLearning.java** you decide which)
- Test your algorithm using POX (Python) or Floodlight (Java):
  - Against Topology A
    - **Comment out** the line "net = Mininet(topo=topo, link=TCLink, autoSetMacs=True,autoStaticArp=True)" and **uncomment** the line "#net = Mininet(controller=RemoteController, topo=topo, link=TCLink, autoSetMacs=True,autoStaticArp=True)". This will allow the topology to connect to your controller.
    - Ping and Iperf (you should see latencies and throughputs similar to what is defined for each link)
    - Make sure that each host can reach each other
  - Against Topology B
    - **Comment out** the line "net = Mininet(topo=topo, link=TCLink, autoSetMacs=True,autoStaticArp=True)" and **uncomment** the line "#net = Mininet(controller=RemoteController, topo=topo, link=TCLink, autoSetMacs=True,autoStaticArp=True)". This will allow the topology to connect to your controller.
    - Ping and Iperf (you should see latencies and throughputs similar to what is defined for each link)
    - Make sure that each host can reach each other
  - Note: When you start your topology (using the python) it will connect to this controller, so run your controller first, then run the target topology
  - Note: **To run your controller with POX:**

- Put ethernet-learning.py file in the directory pox/pox/samples. You will then run the controller and your module using the following commands:
  - you@yourmachine$ cd pox
  - you@yourmachine$ ./pox.py samples.ethernet-learning

- Note: **To run your controller with Floodlight:**
  - **This is a more involved process:**
    - You may have to follow pre-req instructions for modifying the VM
  - Put EthernetLearning.java file in src/main/java/net/floodlightcontroller/ethernetlearning/ .
  - We need to tell Floodlight to load the module on startup. First we have to tell the loader that the module exists. This is done by adding the fully qualified module name on it's own line in src/main/resources/META-INF/services/net.floodlightcontroller.core.module.IFloodlightModule. We open that file and append this line to the file: net.floodlightcontroller.ethernetlearning.EthernetLearning Then we tell the module to be loaded. We modify the Floodlight module configuration file to append the EthernetLearning module. The default one is src/main/resources/floodlightdefault.properties. The key is floodlight.modules and the value is a comma separated list of fully qualified module names.floodlight.modules = <leave the default list of modules in place>, net.floodlightcontroller.ethernetlearning.EthernetLearning
  - Proceed with re-building the controller using:
    - you@yourmachine$ cd floodlight/target
    - you@yourmachine$ ant
  - Finally, run the controller by using the floodlight.jar file produced by ant from within the floodlight directory:
    - you@yourmachine$ java -jar target/floodlight.jar
  - Floodlight will start running and print log and debug output to your console
- Note: Your code will be executed to show it can properly map out the topology, eg: all hosts reachable to each other via appropriate paths

What To Submit:

- A zip file (.tar.xz or .zip formats), named as <FirstAndLastName>-project3.zip (or .tar.xz) (example: **AliceBoberson-project3.tar.xz**), to the class canvas site containing:
  - The evaluation report for Topology A (as **topology-a-report.txt**)
  - Your Topology B code (as **topology-b.py**)
  - The evaluation report for Topology B (as **topology-b-report.txt**)

- A readme file (as **README.md** using Markdown syntax) with sections:
  - A header
    - <Your name>,Zhang CSCI4211,<current date (day/month/year)>
    - Ex: Marky Markdowns,Zhang CSCI4211,31/12/1969
  - An “Ethernet-Algorithm” section that describes at a high level your Ethernet-based self learning algorithm
    - Including detailed explanations of your data structures and their purposes
  - A “Pseudocode” section describing your algorithm following Pseudocode guidelines
  - (optional) A(n) comments/assumption section
- Your Ethernet-based self-learning algorithm code (**ethernet-learning.py** or **EthernetLearning.java**)