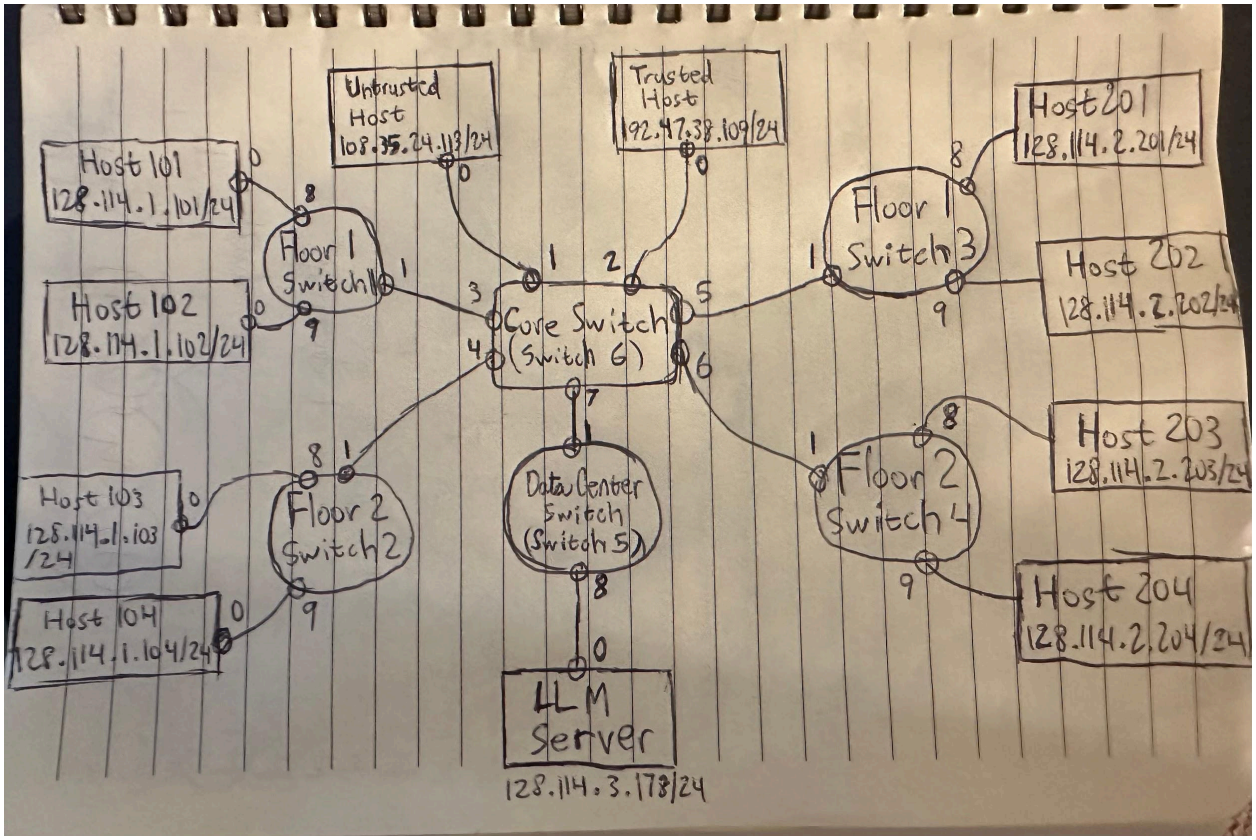
CSE150: Final Project

Anish Talluri




```python
class final_topo(Topo):
  def build(self):
    # Examples!
    # Create a host with a default route of the ethernet interface. You'll need to set the
    # default gateway like this for every host you make on this assignment to make sure all
    # packets are sent out that port. Make sure to change the h# in the defaultRoute area
    # and the MAC address when you add more hosts!

    # Outside Hosts
    h_untrust = self.addHost('h_untrust',mac='66:66:66:66:66:66',ip='108.35.24.113/24', defaultRoute="h_untrust-eth0")
    h_trust = self.addHost('h_trust',mac='77:77:77:77:77:77',ip='192.47.38.109/24', defaultRoute="h_trust-eth0")

    # Floor 1 Hosts
    h101 = self.addHost('h101',mac='00:00:00:00:00:01',ip='128.114.1.101/24', defaultRoute="h101-eth0")
    h102 = self.addHost('h102',mac='00:00:00:00:00:02',ip='128.114.1.102/24', defaultRoute="h102-eth0")
    h103 = self.addHost('h103',mac='00:00:00:00:00:03',ip='128.114.1.103/24', defaultRoute="h103-eth0")
    h104 = self.addHost('h104',mac='00:00:00:00:00:04',ip='128.114.1.104/24', defaultRoute="h104-eth0")

    # Floor 2 Hosts
    h201 = self.addHost('h201',mac='00:00:00:00:00:05',ip='128.114.2.201/24', defaultRoute="h201-eth0")
    h202 = self.addHost('h202',mac='00:00:00:00:00:06',ip='128.114.2.202/24', defaultRoute="h202-eth0")
    h203 = self.addHost('h203',mac='00:00:00:00:00:07',ip='128.114.2.203/24', defaultRoute="h203-eth0")
    h204 = self.addHost('h204',mac='00:00:00:00:00:08',ip='128.114.2.204/24', defaultRoute="h204-eth0")

    # LLM Server Host
    h_llm = self.addHost('h_llm',mac='99:99:99:99:99:99',ip='128.114.3.178/24', defaultRoute="h_llm-eth0")
```

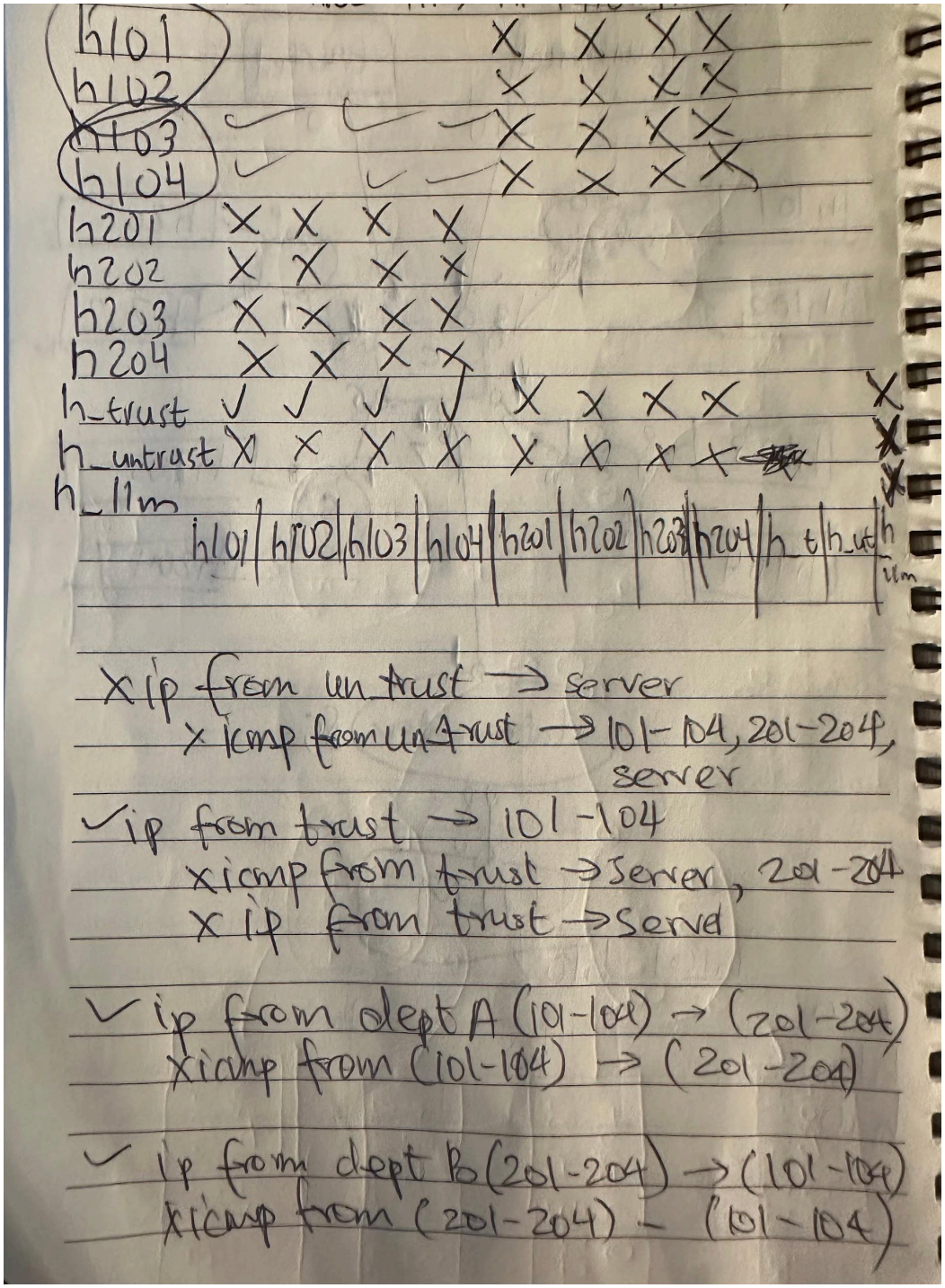
```python
class final_topo(Topo):
    def build(self):

        # Create a switch. No changes here from Lab 1.

        # Core Switch
        s0 = self.addSwitch('s0')

        # Floor 1 Switches
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')

        # Floor 2 Switches
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')

        # Data Center Switch
        s5 = self.addSwitch('s5')

        # Connect Port 8 on the Switch to Port 0 on Host 1 and Port 9 on the Switch to Port 0 on
        # Host 2. This is representing the physical port on the switch or host that you are
        # connecting to.

        # IMPORTANT NOTES:
        # - On a single device, you can only use each port once! So, on s1, only 1 device can be
        #   plugged in to port 1, only one device can be plugged in to port 2, etc.
        # - On the "host" side of connections, you must make sure to always match the port you
        #   set as the default route when you created the device above. Usually, this means you
        #   should plug in to port 0 (since you set the default route to h#-eth0).

        # Floor 1 Switch 1
        self.addLink(s1,h101, port1=8, port2=0)
        self.addLink(s1,h102, port1=9, port2=0)

        # Floor 2 Switch 1
        self.addLink(s3,h201, port1=8, port2=0)
        self.addLink(s3,h202, port1=9, port2=0)

        # Floor 1 Switch 2
        self.addLink(s2,h103, port1=8, port2=0)
        self.addLink(s2,h104, port1=9, port2=0)

        # Floor 2 Switch 2
        self.addLink(s4,h203, port1=8, port2=0)
        self.addLink(s4,h204, port1=9, port2=0)

        # Datacenter Link to Server
        self.addLink(s5,h_llm, port1=8, port2=0)

```

```
# Core Switch Links
self.addLink(s0,h_untrust, port1=1, port2=0)
self.addLink(s0,h_trust, port1=2, port2=0)
self.addLink(s0,s1, port1=3, port2=1)
self.addLink(s0,s2, port1=4, port2=1)
self.addLink(s0,s3, port1=5, port2=1)
self.addLink(s0,s4, port1=6, port2=1)
self.addLink(s0,s5, port1=7, port2=1)
```

The pictures shown above depict the specific topology I created and the code for it. The diagram depicts the specific ports I used in order to understand how to properly link all the hosts and switches together. While I was doing this, I ran into a bug where I could not connect any of the switches with each other (via the core switch). It turns out that you cannot link switches with other switches using port 0, so I had to change it all to a non-zero port and it ended up linking (as shown in the picture below when I ran “links” in mininet)

```
mininet> links
s1-eth8<->h101-eth0 (OK OK)
s1-eth9<->h102-eth0 (OK OK)
s2-eth8<->h103-eth0 (OK OK)
s2-eth9<->h104-eth0 (OK OK)
s3-eth8<->h201-eth0 (OK OK)
s3-eth9<->h202-eth0 (OK OK)
s4-eth8<->h203-eth0 (OK OK)
s4-eth9<->h204-eth0 (OK OK)
s5-eth8<->h_llm-eth0 (OK OK)
s6-eth2<->h_trust-eth0 (OK OK)
s6-eth1<->h_untrust-eth0 (OK OK)
s6-eth3<->s1-eth1 (OK OK)
s6-eth4<->s2-eth1 (OK OK)
s6-eth5<->s3-eth1 (OK OK)
s6-eth6<->s4-eth1 (OK OK)
s6-eth7<->s5-eth1 (OK OK)
```

I mostly struggled with getting my controller to run. I had issues with just getting it to connect with all the other hosts, as everytime I ran pingall it would drop all the packets. However, I was able to determine the cause for this issue. My firewall implementation was fine, as I did filter for just ARP, both ICMP and IPv4, then just IPv4. I ended up repeating a lot of the same code (hence why there are so many lines of similar code), but it was a measure I was okay with because it allowed me to track all the rules I needed to implement per each type of package.



The image above shows the conditions that I had to implement. Notice at the very top that there is this table-like structure. This was used to help me visualize the output I should receive when running “pingall”, with blanks/checkmarks being connections (where floods through specific ports should happen) and the Xs representing dropped packets. As shown in

the picture below, it outputs something almost exact in mininet, which tells me that my firewall seems to have done its job.