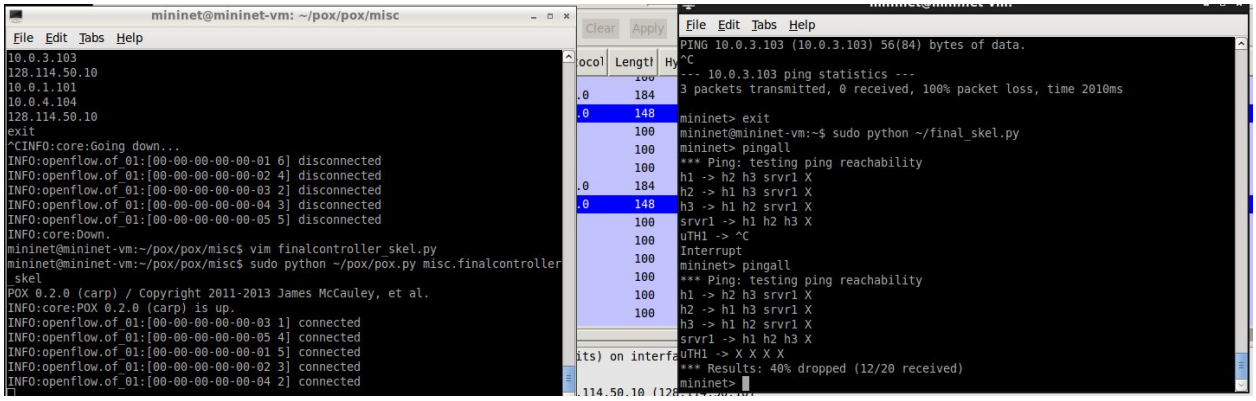
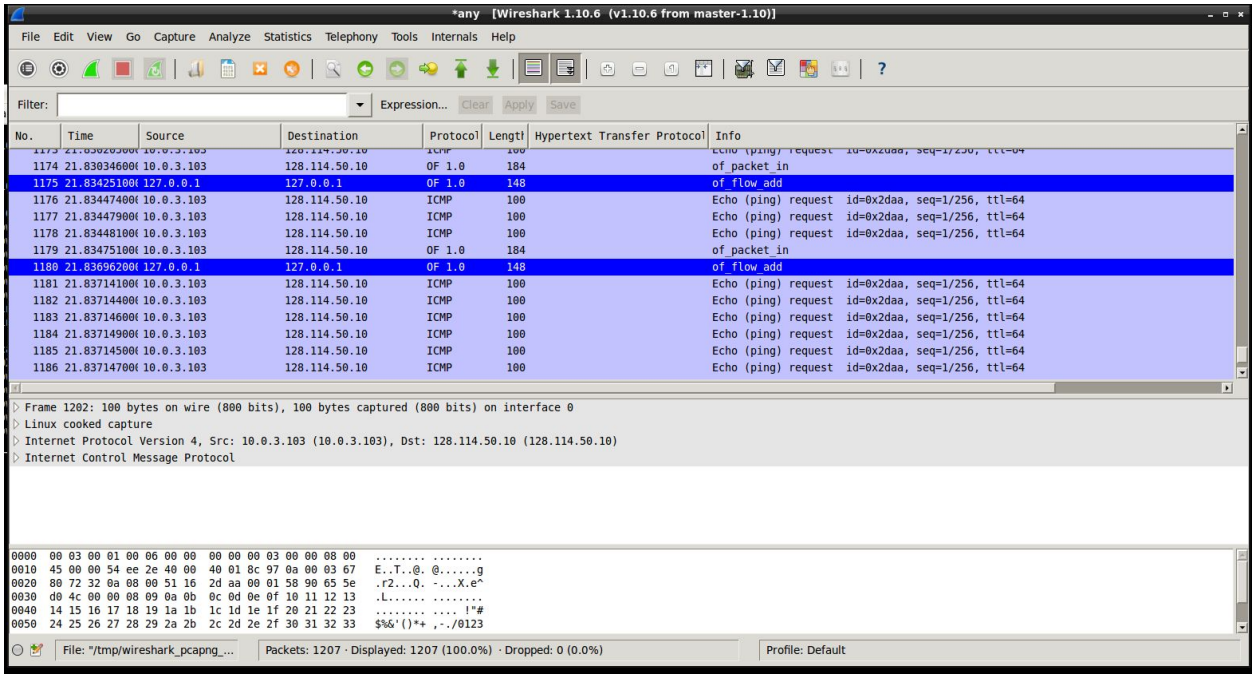
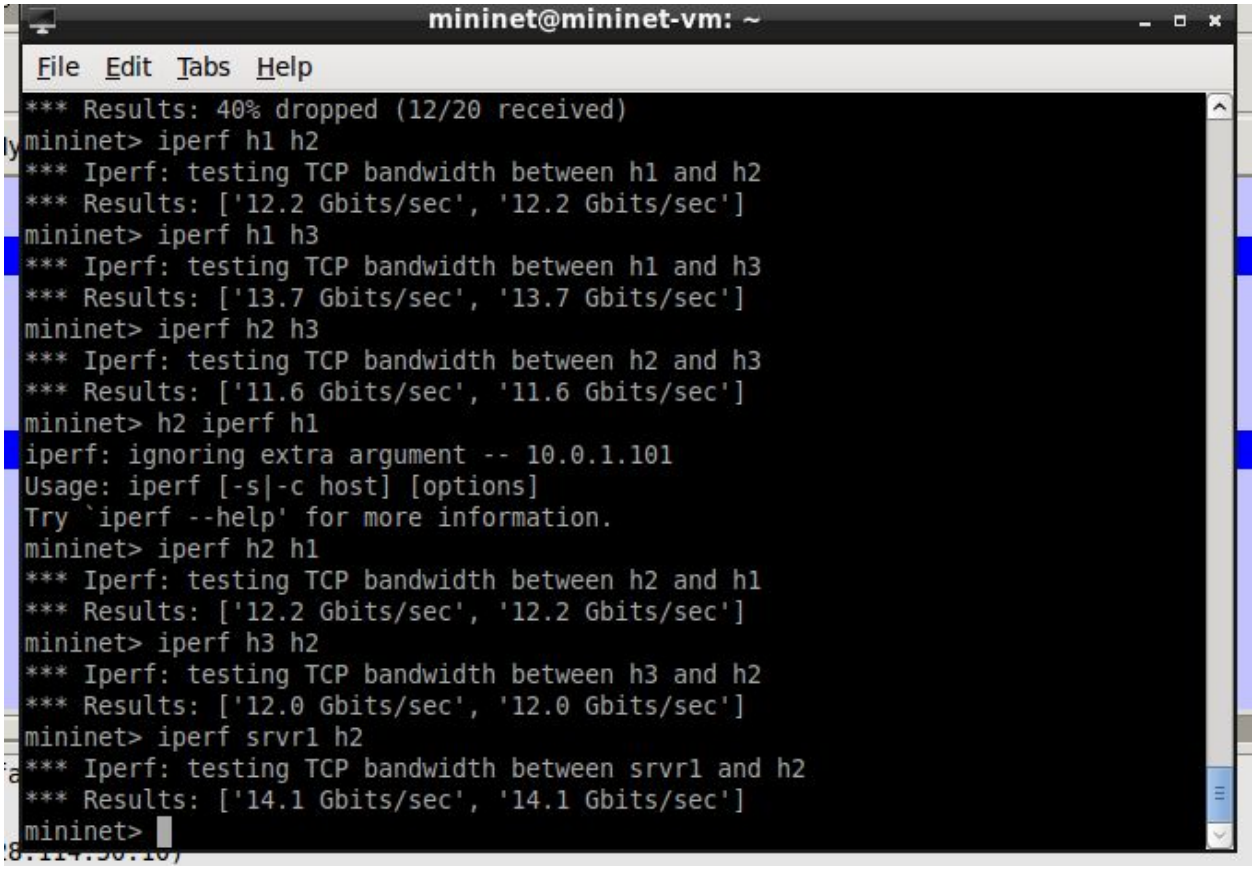
The final topology was built by instantiating the IP addresses and setting the port numbers to for the hosts and adding links to the switches within the topology. Within the finalcontroller.py file, the do_final function contains a series of nested if-else statements that check either if the packet is an ARP packet or if it is in IP packet. If the packet is of type ARP, the packet is flooded into all open ports in the network. If the packet is an IP packet, then the program checks the each switch_id that the packet is currently visiting and checks whether the switch is connected to the destination host IP address; otherwise, forwards the packet back to the core switch to forward somewhere else.
At each check for switch_id, the program also checks if the source or destination IP address comes from the Untrusted Host, which automatically drops the packet if the packet had originated from the Untrusted Host address. Also, at each check for switch_id, the program checks if the packet's destination address is an address that is connected to the current switch which then forwards the packet to the given port address for the host.




*any [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: Expression... Clear Apply Save
No. Time Source Destination Protocol Length Hypertext Transfer Protocol Info
1174 21.83034600( 10.0.3.103 128.114.50.10 OF 1.0 184 of_packet_in
1175 21.83425100( 127.0.0.1 127.0.0.1 OF 1.0 148 of_flow_add
1176 21.83447400( 10.0.3.103 128.114.50.10 ICMP 100 Echo (ping) request id=0x2daa, seq=1/256, ttl=64
1177 21.83447900( 10.0.3.103 128.114.50.10 ICMP 100 Echo (ping) request id=0x2daa, seq=1/256, ttl=64
1178 21.83448100( 10.0.3.103 128.114.50.10 ICMP 100 Echo (ping) request id=0x2daa, seq=1/256, ttl=64
1179 21.83475100( 10.0.3.103 128.114.50.10 OF 1.0 184 of_packet_in
1180 21.83696200( 127.0.0.1 127.0.0.1 OF 1.0 148 of_flow_add
1181 21.83714100( 10.0.3.103 128.114.50.10 ICMP 100 Echo (ping) request id=0x2daa, seq=1/256, ttl=64
1182 21.83714400( 10.0.3.103 128.114.50.10 ICMP 100 Echo (ping) request id=0x2daa, seq=1/256, ttl=64
1183 21.83714600( 10.0.3.103 128.114.50.10 ICMP 100 Echo (ping) request id=0x2daa, seq=1/256, ttl=64
1184 21.83714900( 10.0.3.103 128.114.50.10 ICMP 100 Echo (ping) request id=0x2daa, seq=1/256, ttl=64
1185 21.83714500( 10.0.3.103 128.114.50.10 ICMP 100 Echo (ping) request id=0x2daa, seq=1/256, ttl=64
1186 21.83714700( 10.0.3.103 128.114.50.10 ICMP 100 Echo (ping) request id=0x2daa, seq=1/256, ttl=64
Frame 1202: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.3.103 (10.0.3.103), Dst: 128.114.50.10 (128.114.50.10)
Internet Control Message Protocol
0000 00 03 00 01 00 06 00 00 00 00 00 03 00 00 08 00 ........ ........
0010 45 00 00 54 ee 2e 40 00 40 01 8c 97 0a 00 03 67 E..T..@. @......g
0020 80 72 32 0a 08 00 51 16 2d aa 00 01 58 90 65 5e .r2...Q. -...X.e^
0030 d0 4c 00 00 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 .L...... ........
0040 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 ........ .... !"#
0050 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 $%&'()*+ ,-./0123
File: "/tmp/wireshark_pcapng_... Packets: 1207 · Displayed: 1207 (100.0%) · Dropped: 0 (0.0%) Profile: Default


mininet@mininet-vm: ~
File Edit Tabs Help
*** Results: 40% dropped (12/20 received)
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['12.2 Gbits/sec', '12.2 Gbits/sec']
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['13.7 Gbits/sec', '13.7 Gbits/sec']
mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['11.6 Gbits/sec', '11.6 Gbits/sec']
mininet> h2 iperf h1
iperf: ignoring extra argument -- 10.0.1.101
Usage: iperf [-s|-c host] [options]
Try `iperf --help' for more information.
mininet> iperf h2 h1
*** Iperf: testing TCP bandwidth between h2 and h1
*** Results: ['12.2 Gbits/sec', '12.2 Gbits/sec']
mininet> iperf h3 h2
*** Iperf: testing TCP bandwidth between h3 and h2
*** Results: ['12.0 Gbits/sec', '12.0 Gbits/sec']
mininet> iperf srvr1 h2
*** Iperf: testing TCP bandwidth between srvr1 and h2
*** Results: ['14.1 Gbits/sec', '14.1 Gbits/sec']
mininet>