

Henry Nguyen

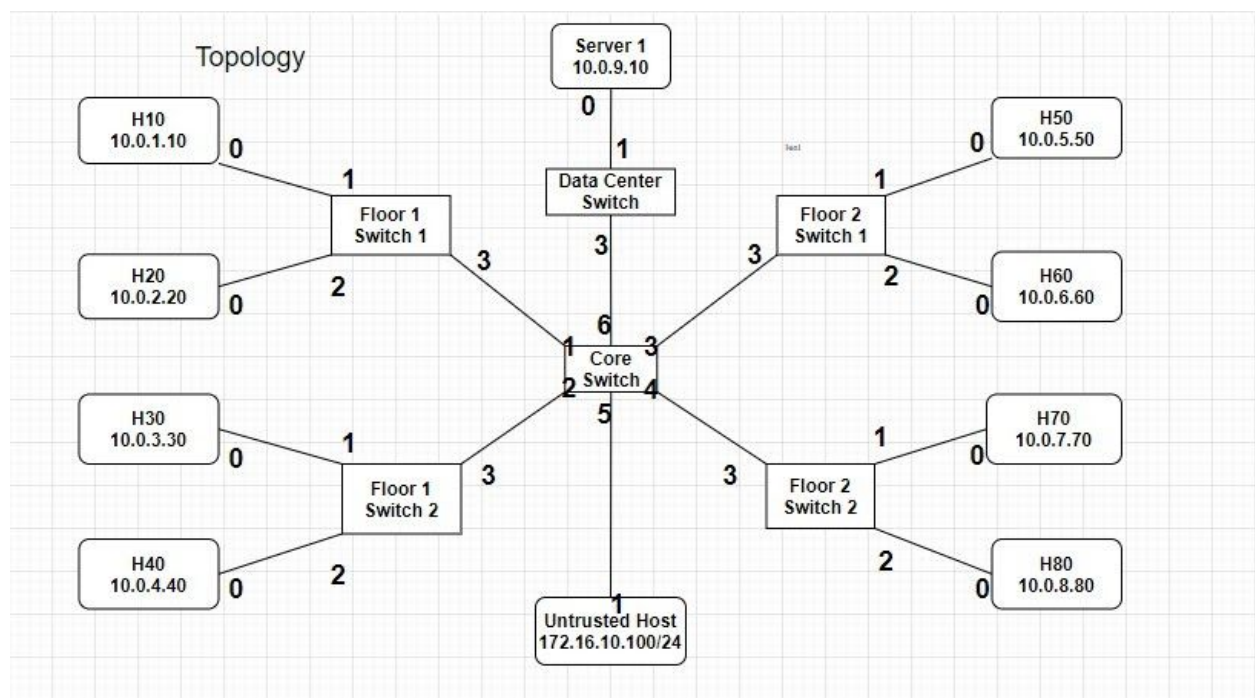
hnguye87

Final Project: Implementing a Simple Router

Logic of Controller:

The logic of the controller is broken into three parts. The first part is if the packet is not an IP packet. This can be done by checking if the packet is of type IPV4 or not. In the case it is not IPV4, the controller simply floods all traffic using of.OFPP_FLOOD as the destination port since it is not an IP packet.

The second part is if the packet is an IP packet. In this case, we specify the ports instead of flooding. This was the topology and port numbers used when specifying.



Knowing it is an IP packet, the second part now must first check if the packet is ok to send under the firewall rules, so there are two flags that must be checked before sending it to the destination port. The first flag checks if the packet is of type ICMP and the source is not from the untrusted host. This is the case since the untrusted host is not allowed to send ICMP packets whatsoever. The second flag is if the packet is of type TCP, and if [the source is the untrusted host while the destination is server] is False. This first flag exists because the untrusted host can not send any IP traffic to the server; since the first flag covers ICMP packets being sent from the untrusted host to the server, the second flag has to check for the case of TCP packets.

If either of these flags are true then the packet is deemed either a safe ICMP or TCP packet respectively. Checking these two flags saves a lot of trouble from having to make special cases for the rest of the firewall code. The source and destination can be found in the IPV4 attributes.

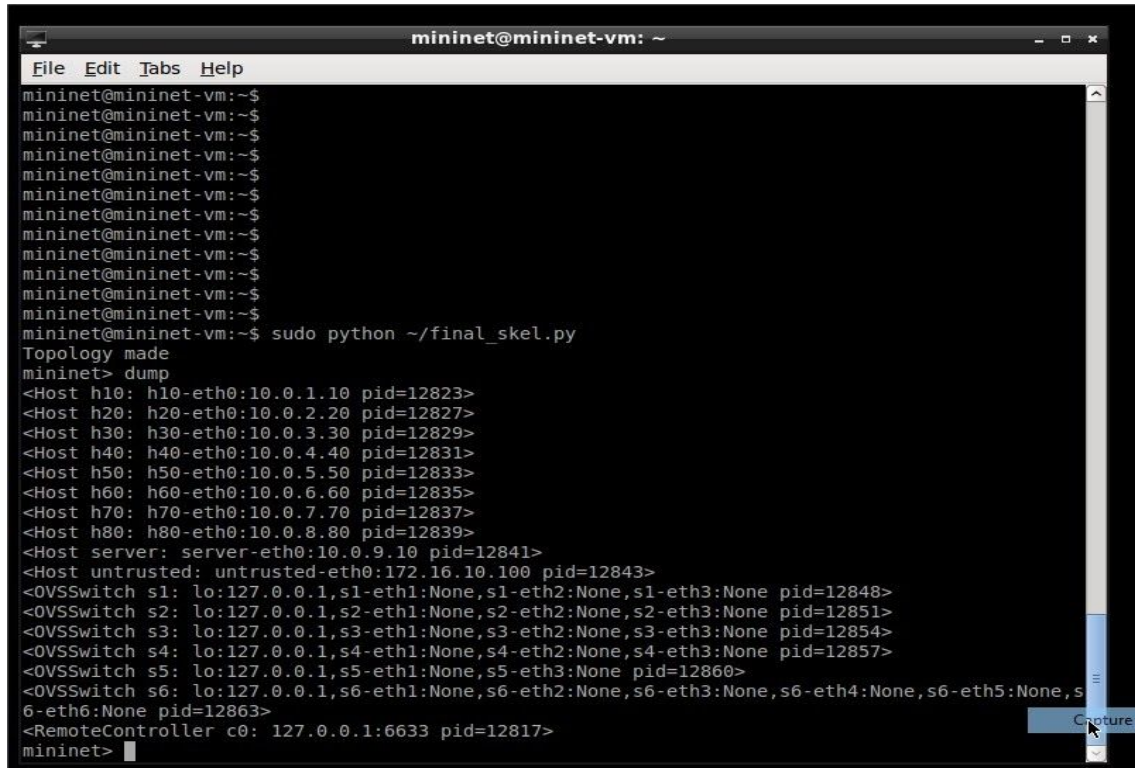
Now that the packet is confirmed safe to send, it is time to send it to its destination IP. To do this, we first have to check what switch the packet is currently in. After figuring that out, the packet is sent out to the correct port given the topology defined in the topology file, `final.py`, until it finally reaches the destination IP.

The third part of the code is if the both flags were false and the packet is not safe to send. In that case the packet is dropped by not adding the action so it is not sent out any ports.

Test Evidence:

dump:

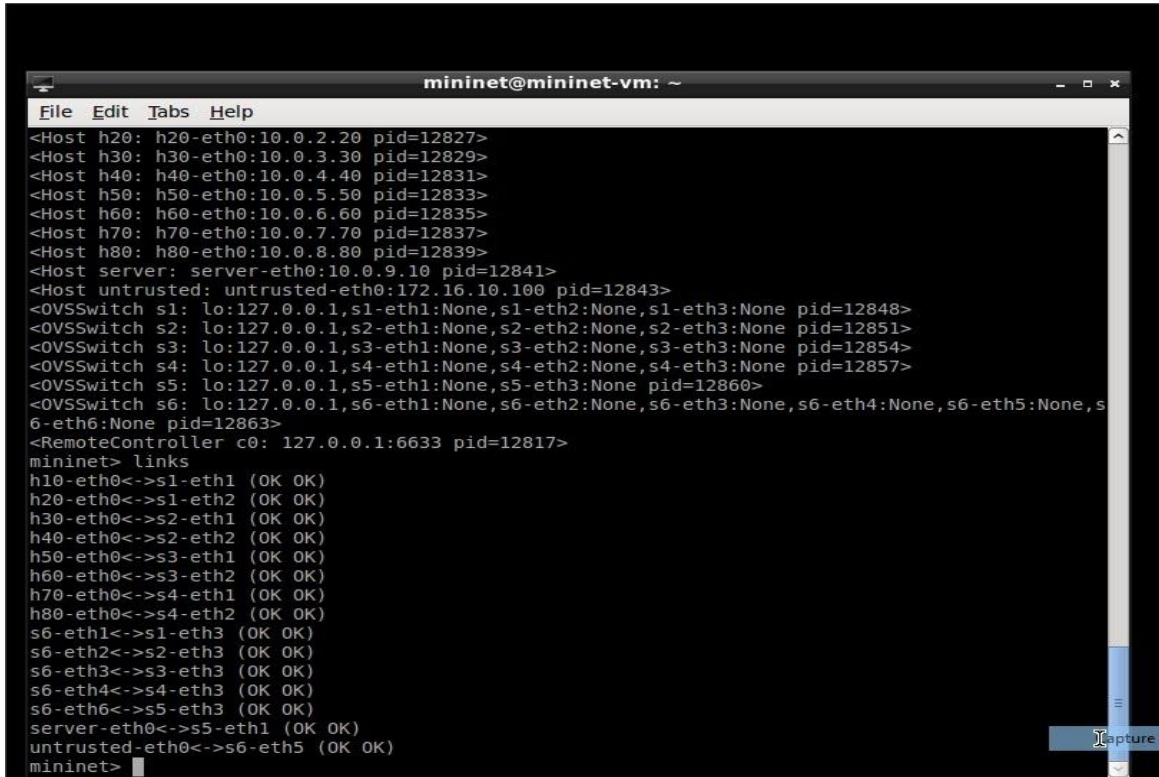
The command *dump* shows that all the devices were successfully created, and the IP addresses on all the hosts are correct.



```
mininet@mininet-vm: ~  
File Edit Tabs Help  
mininet@mininet-vm:~$  
mininet@mininet-vm:~$  
mininet@mininet-vm:~$  
mininet@mininet-vm:~$  
mininet@mininet-vm:~$  
mininet@mininet-vm:~$  
mininet@mininet-vm:~$  
mininet@mininet-vm:~$  
mininet@mininet-vm:~$  
mininet@mininet-vm:~$  
mininet@mininet-vm:~$  
mininet@mininet-vm:~$ sudo python ~/final_skel.py  
Topology made  
mininet> dump  
<Host h10: h10-eth0:10.0.1.10 pid=12823>  
<Host h20: h20-eth0:10.0.2.20 pid=12827>  
<Host h30: h30-eth0:10.0.3.30 pid=12829>  
<Host h40: h40-eth0:10.0.4.40 pid=12831>  
<Host h50: h50-eth0:10.0.5.50 pid=12833>  
<Host h60: h60-eth0:10.0.6.60 pid=12835>  
<Host h70: h70-eth0:10.0.7.70 pid=12837>  
<Host h80: h80-eth0:10.0.8.80 pid=12839>  
<Host server: server-eth0:10.0.9.10 pid=12841>  
<Host untrusted: untrusted-eth0:172.16.10.100 pid=12843>  
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=12848>  
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=12851>  
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=12854>  
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None pid=12857>  
<OVSSwitch s5: lo:127.0.0.1,s5-eth1:None,s5-eth3:None pid=12860>  
<OVSSwitch s6: lo:127.0.0.1,s6-eth1:None,s6-eth2:None,s6-eth3:None,s6-eth4:None,s6-eth5:None,s6-eth6:None pid=12863>  
<RemoteController c0: 127.0.0.1:6633 pid=12817>  
mininet>
```

links:

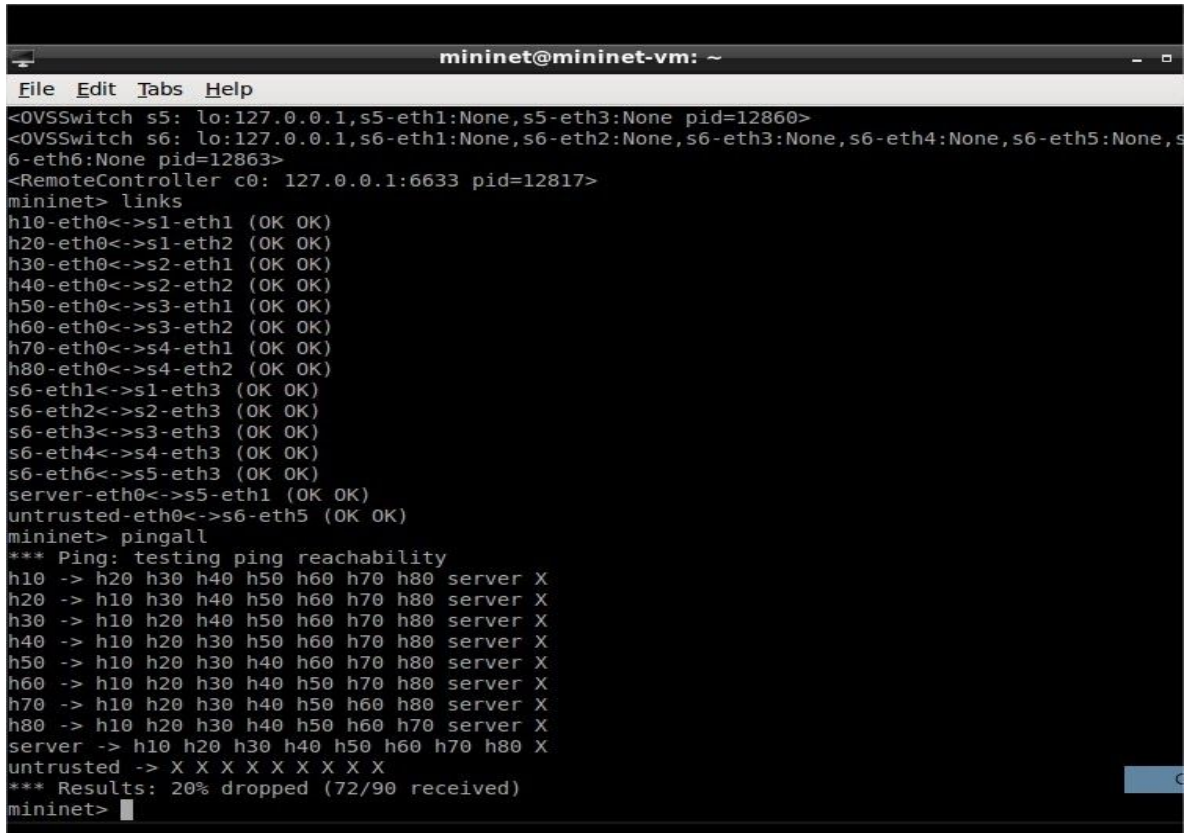
The *links* command shows all links were successfully made, it also shows that the topology is correctly made also.



```
mininet@mininet-vm: ~  
File Edit Tabs Help  
<Host h20: h20-eth0:10.0.2.20 pid=12827>  
<Host h30: h30-eth0:10.0.3.30 pid=12829>  
<Host h40: h40-eth0:10.0.4.40 pid=12831>  
<Host h50: h50-eth0:10.0.5.50 pid=12833>  
<Host h60: h60-eth0:10.0.6.60 pid=12835>  
<Host h70: h70-eth0:10.0.7.70 pid=12837>  
<Host h80: h80-eth0:10.0.8.80 pid=12839>  
<Host server: server-eth0:10.0.9.10 pid=12841>  
<Host untrusted: untrusted-eth0:172.16.10.100 pid=12843>  
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=12848>  
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=12851>  
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=12854>  
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None pid=12857>  
<OVSSwitch s5: lo:127.0.0.1,s5-eth1:None,s5-eth3:None pid=12860>  
<OVSSwitch s6: lo:127.0.0.1,s6-eth1:None,s6-eth2:None,s6-eth3:None,s6-eth4:None,s6-eth5:None,s6-eth6:None pid=12863>  
<RemoteController c0: 127.0.0.1:6633 pid=12817>  
mininet> links  
h10-eth0<->s1-eth1 (OK OK)  
h20-eth0<->s1-eth2 (OK OK)  
h30-eth0<->s2-eth1 (OK OK)  
h40-eth0<->s2-eth2 (OK OK)  
h50-eth0<->s3-eth1 (OK OK)  
h60-eth0<->s3-eth2 (OK OK)  
h70-eth0<->s4-eth1 (OK OK)  
h80-eth0<->s4-eth2 (OK OK)  
s6-eth1<->s1-eth3 (OK OK)  
s6-eth2<->s2-eth3 (OK OK)  
s6-eth3<->s3-eth3 (OK OK)  
s6-eth4<->s4-eth3 (OK OK)  
s6-eth6<->s5-eth3 (OK OK)  
server-eth0<->s5-eth1 (OK OK)  
untrusted-eth0<->s6-eth5 (OK OK)  
mininet>
```

pingall:

The *pingall* command shows that all hosts and the server can ping one another with the exception of the untrusted host. Since the untrusted host is not trusted to send ICMP packets to anyone and the *pingall* command uses ICMP packets, the output is correct in showing that the pings using the untrusted host fails.



```
mininet@mininet-vm: ~  
File Edit Tabs Help  
<OVSSwitch s5: lo:127.0.0.1,s5-eth1:None,s5-eth3:None pid=12860>  
<OVSSwitch s6: lo:127.0.0.1,s6-eth1:None,s6-eth2:None,s6-eth3:None,s6-eth4:None,s6-eth5:None,s6-eth6:None pid=12863>  
<RemoteController c0: 127.0.0.1:6633 pid=12817>  
mininet> links  
h10-eth0<->s1-eth1 (OK OK)  
h20-eth0<->s1-eth2 (OK OK)  
h30-eth0<->s2-eth1 (OK OK)  
h40-eth0<->s2-eth2 (OK OK)  
h50-eth0<->s3-eth1 (OK OK)  
h60-eth0<->s3-eth2 (OK OK)  
h70-eth0<->s4-eth1 (OK OK)  
h80-eth0<->s4-eth2 (OK OK)  
s6-eth1<->s1-eth3 (OK OK)  
s6-eth2<->s2-eth3 (OK OK)  
s6-eth3<->s3-eth3 (OK OK)  
s6-eth4<->s4-eth3 (OK OK)  
s6-eth6<->s5-eth3 (OK OK)  
server-eth0<->s5-eth1 (OK OK)  
untrusted-eth0<->s6-eth5 (OK OK)  
mininet> pingall  
*** Ping: testing ping reachability  
h10 -> h20 h30 h40 h50 h60 h70 h80 server X  
h20 -> h10 h30 h40 h50 h60 h70 h80 server X  
h30 -> h10 h20 h40 h50 h60 h70 h80 server X  
h40 -> h10 h20 h30 h50 h60 h70 h80 server X  
h50 -> h10 h20 h30 h40 h60 h70 h80 server X  
h60 -> h10 h20 h30 h40 h50 h70 h80 server X  
h70 -> h10 h20 h30 h40 h50 h60 h80 server X  
h80 -> h10 h20 h30 h40 h50 h60 h70 server X  
server -> h10 h20 h30 h40 h50 h60 h70 h80 X  
untrusted -> X X X X X X X X  
*** Results: 20% dropped (72/90 received)  
mininet>
```

iperf:

The screenshot shows the result of three different *iperf* uses. The first one is between two hosts, h10 and h20. Both these hosts are trusted so the iperf command, using TCP packets, are successful.

The second *iperf* command is between h10 and untrusted. Since The untrusted host is not allowed to send TCP packets to only the server, the command is successful as seen below.

The third iperf command is between the untrusted host and the server. The untrusted host can't send any IP traffic to the server, so the packet is dropped. The command is stuck in a loop of retransmission due to timeout and the program had to be manually killed.

The image displays two terminal windows side-by-side, both running the 'mininet@mininet-vm: ~' shell. The left window shows a packet capture with the following details:

```

File Edit Tabs Help
Destination IP: 172.16.10.100
Type: TCP packet
valid IP packet
Source IP: 10.0.1.10
Destination IP: 172.16.10.100
Type: TCP packet
To untrusted
valid IP packet
Source IP: 172.16.10.100
Destination IP: 10.0.1.10
Type: TCP packet
valid IP packet
Source IP: 172.16.10.100
Destination IP: 10.0.1.10
Type: TCP packet
Dropped Packet
Source IP: 172.16.10.100
Destination IP: 10.0.9.10
Type: TCP packet
non-IP packet
non-IP packet
non-IP packet
non-IP packet
non-IP packet
non-IP packet
non-IP packet
non-IP packet
non-IP packet
non-IP packet
non-IP packet

```

The right window shows the same capture with additional details and commands:

```

File Edit Tabs Help
h60-eth0<->s3-eth2 (OK OK)
h70-eth0<->s4-eth1 (OK OK)
h80-eth0<->s4-eth2 (OK OK)
s6-eth1<->s1-eth3 (OK OK)
s6-eth2<->s2-eth3 (OK OK)
s6-eth3<->s3-eth3 (OK OK)
s6-eth4<->s4-eth3 (OK OK)
s6-eth6<->s5-eth3 (OK OK)
server-eth0<->s5-eth1 (OK OK)
untrusted-eth0<->s6-eth5 (OK OK)
mininet> pingall
*** Ping: testing ping reachability
h10 -> h20 h30 h40 h50 h60 h70 h80 server X
h20 -> h10 h30 h40 h50 h60 h70 h80 server X
h30 -> h10 h20 h40 h50 h60 h70 h80 server X
h40 -> h10 h20 h30 h50 h60 h70 h80 server X
h50 -> h10 h20 h30 h40 h60 h70 h80 server X
h60 -> h10 h20 h30 h40 h50 h70 h80 server X
h70 -> h10 h20 h30 h40 h50 h60 h80 server X
h80 -> h10 h20 h30 h40 h50 h60 h70 server X
server -> h10 h20 h30 h40 h50 h60 h70 h80 X
untrusted -> X X X X X X X X
*** Results: 20% dropped (72/90 received)
mininet> iperf h10 h20
*** Iperf: testing TCP bandwidth between h10 and h20
*** Results: ['45.1 Gbits/sec', '45.2 Gbits/sec']
mininet> iperf h10 untrusted
*** Iperf: testing TCP bandwidth between h10 and untrusted
*** Results: ['43.4 Gbits/sec', '43.5 Gbits/sec']
mininet> iperf untrusted server
*** Iperf: testing TCP bandwidth between untrusted and server
^C
Interrupt
mininet>

```

dpctl dump-flows:

The command *dpctl dump-flows* outputs all the flow entries in the flow table.

```
mininet@mininet-vm: ~  
File Edit Tabs Help  
h20 -> h10 h30 h40 h50 h60 h70 h80 server X  
h30 -> h10 h20 h40 h50 h60 h70 h80 server X  
h40 -> h10 h20 h30 h50 h60 h70 h80 server X  
h50 -> h10 h20 h30 h40 h60 h70 h80 server X  
h60 -> h10 h20 h30 h40 h50 h70 h80 server X  
h70 -> h10 h20 h30 h40 h50 h60 h80 server X  
h80 -> h10 h20 h30 h40 h50 h60 h70 server X  
server -> h10 h20 h30 h40 h50 h60 h70 h80 X  
untrusted -> X X X X X X X X  
*** Results: 20% dropped (72/90 received)  
mininet> iperf h10 h20  
*** Iperf: testing TCP bandwidth between h10 and h20  
*** Results: ['45.1 Gbits/sec', '45.2 Gbits/sec']  
mininet> iperf h10 untrusted  
*** Iperf: testing TCP bandwidth between h10 and untrusted  
*** Results: ['43.4 Gbits/sec', '43.5 Gbits/sec']  
mininet> iperf untrusted server  
*** Iperf: testing TCP bandwidth between untrusted and server  
^C  
Interrupt  
mininet> dpctl dump-flows  
*** s1 -----  
NXST_FLOW reply (xid=0x4):  
*** s2 -----  
NXST_FLOW reply (xid=0x4):  
*** s3 -----  
NXST_FLOW reply (xid=0x4):  
*** s4 -----  
NXST_FLOW reply (xid=0x4):  
*** s5 -----  
NXST_FLOW reply (xid=0x4):  
*** s6 -----  
NXST_FLOW reply (xid=0x4):  
mininet> 
```