

EE555 Final Project Report

Yong Wang (yongw@usc.edu)

Chunyang Wu (chunyanw@usc.edu)

Part 1: Learn Development Tools

Start network with kernel switch:

```
sudo mn --topo single,3 --switch ovsk --controller remote
```

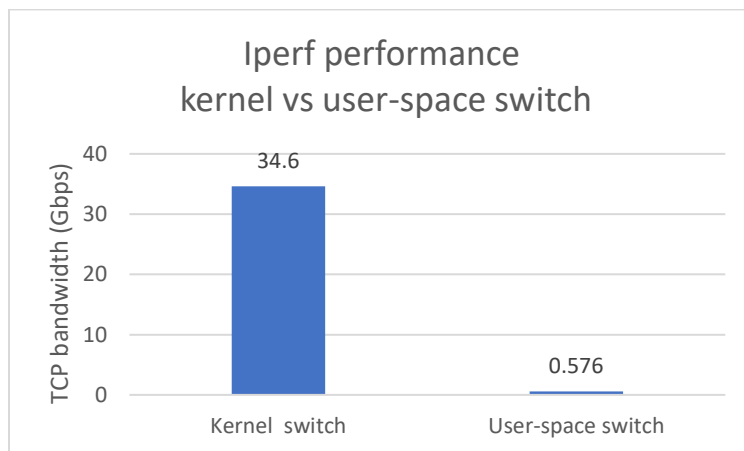
In another ssh terminal, start controller:

```
$sudo controller ptcp:
```

And then run with user space switch

```
sudo mn --topo single,3 --switch user --controller remote
```

Iperf performance comparison.



Part 1: Create a Learning Switch

Our Switch Implementation:

Use mac_to_port as mac address table which maps mac to port.

When controller receives a packet, add src_mac to mac_to_port table, and then look dst_mac in mac_to_port table, if found create a new flow and forward that packet to specified port. If dst_mac not found in table, flood to all ports.

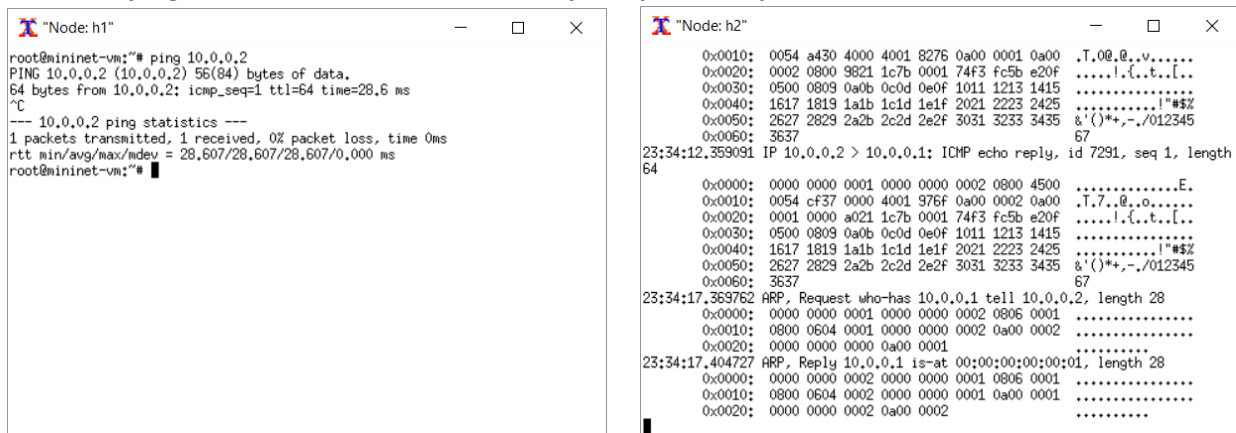
Run hub controller

```
$. /pox.py log.level --DEBUG misc.of_tutorial
```

Create virtual net

```
$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

Next, use pingall to test connection and use tcpdump to view packets.



```
"Node: h1"
root@mininet-vml:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=28.6 ms
^C
--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 28.607/28.607/28.607/0.000 ms
root@mininet-vml:~#

"Node: h2"
0x0010: 0054 a430 4000 4001 8276 0a00 0001 0a00 .T.00.@..v.....
0x0020: 0002 0800 9821 1c7b 0001 74f3 fc5b e20f .....!.t..[..
0x0030: 0500 0809 0a0b 0c0d 0e0f 1011 1213 1415 .....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!""$%
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 67
23:34:12.359091 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7291, seq 1, length
64
0x0000: 0000 0000 0001 0000 0000 0002 0800 4500 .....E.
0x0010: 0054 cf37 0000 4001 976f 0a00 0002 0a00 .T.7..@..o.....
0x0020: 0001 0000 a021 1c7b 0001 74f3 fc5b e20f .....!.t..[..
0x0030: 0500 0809 0a0b 0c0d 0e0f 1011 1213 1415 .....
0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425 .....!""$%
0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435 &'()*+,-./012345
0x0060: 3637 67
23:34:17.369762 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
0x0000: 0000 0000 0001 0000 0000 0002 0806 0001 .....
0x0010: 0800 0604 0001 0000 0000 0002 0a00 0002 .....
0x0020: 0000 0000 0000 0a00 0001 .....
23:34:17.404727 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
0x0000: 0000 0000 0002 0000 0000 0001 0806 0001 .....
0x0010: 0800 0604 0002 0000 0000 0001 0a00 0001 .....
0x0020: 0000 0000 0002 0a00 0002 .....
█
```

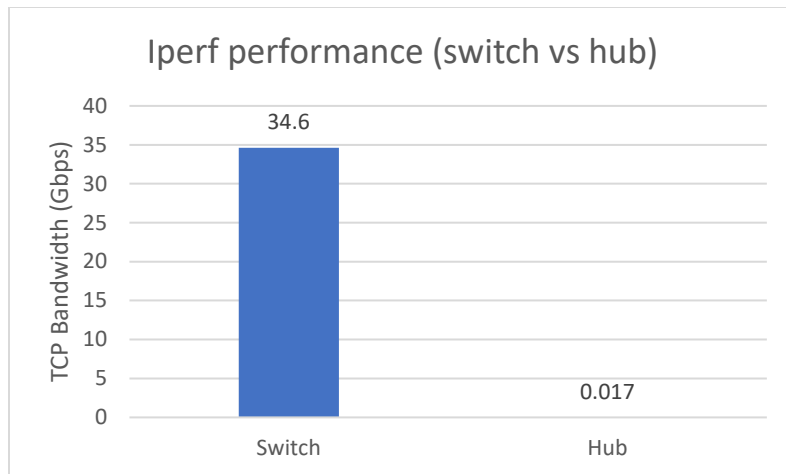
Then, run switch controller and virtual net

(first copy switch.py from part1 to ~/pox/pox/misc directory)

```
$. /pox.py log.level --DEBUG misc.switch
```

```
$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

Iperf performance comparison of hub and switch



Part 1: Router Exercise

Our Router implementation:

IP packets process:

If dst_ip is router, pass it to icmp handler.

If dst ip is not router, need to check arp table and route table.

If dst ip doesn't match route table, then reply ICMP unreachable packet to src ip.

If dst ip match route table but not in arp table, router need to send arp request to find where dst ip is, at the same time put the original ICMP packet into a buffer. When receive arp reply, then send the buffered ICMP packet to dst ip.

ARP process:

When receive a ARP request, add an entry into arp table, then add a flow rule (match dst_ip = protosrc in arp packet) with action (modify dst_mac and output port).

When receive a ARP reply, add arp entry into arp table, then process arp message queue (buffered packets).

ICMP process:

If dst_ip is router, reply ICMP echo directly.

Run router controller and topology:

(copy router.py from part1 to ~/pox/pox/misc directory)

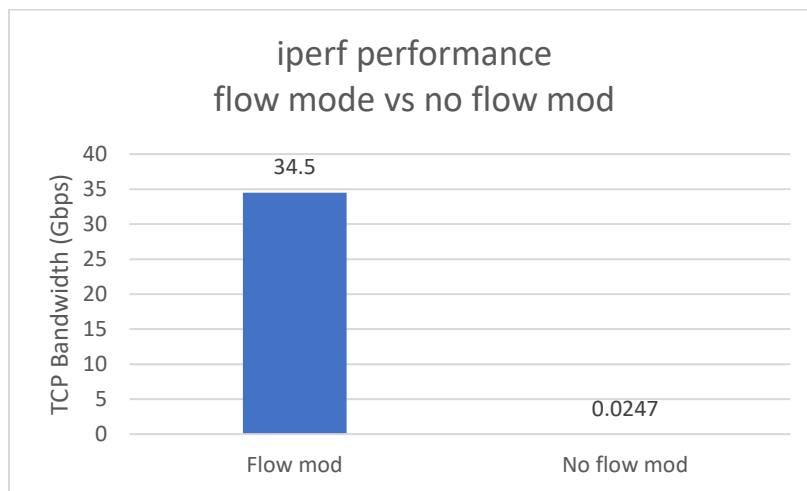
(copy mytopo.py from part1 to current directory)

```
./pox.py log.level --DEBUG misc.router  
$ sudo mn --custom mytopo.py --topo mytopo -mac --controller remote
```

Next, test ICMP (ping unreachable host, ping h3, ping router)

```
Node: h1  
root@mininet-vm:~# ping 10.99.0.1  
PING 10.99.0.1 (10.99.0.1) 56(84) bytes of data.  
From 10.0.1.1 icmp_seq=1 Destination Net Unreachable  
From 10.0.1.1 icmp_seq=2 Destination Net Unreachable  
^C  
--- 10.99.0.1 ping statistics ---  
2 packets transmitted, 0 received, 100% packet loss, time 1001ms  
  
root@mininet-vm:~# ping 10.0.3.100  
PING 10.0.3.100 (10.0.3.100) 56(84) bytes of data.  
64 bytes from 10.0.3.100: icmp_seq=1 ttl=64 time=73.7 ms  
64 bytes from 10.0.3.100: icmp_seq=2 ttl=64 time=0.110 ms  
64 bytes from 10.0.3.100: icmp_seq=3 ttl=64 time=0.032 ms  
^C  
--- 10.0.3.100 ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 2001ms  
rtt min/avg/max/mdev = 0.032/24.614/73.700/34.709 ms  
root@mininet-vm:~# ping 10.0.1.1  
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.  
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=35.8 ms  
^C  
--- 10.0.1.1 ping statistics ---  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 35.812/35.812/35.812/0.000 ms  
root@mininet-vm:~#
```

Then, test Iperf performance between h1 and h3



Part 2: Advanced router

In this part, controller need to support both switching and routing.

Implementation:

We use dpid as switch identification to support multi-switch controller, separating route table, arp table, mac table according to dpid. Each switch has its own tables.

Then based on part1, we add switching function to router like this:

When receiving packet, first handle it using act_like_switch function which use mac table to mod flow, and then pass packet to act_like_router function which use arp and route table to mod flow.

Ping test (on h3 ping h4, ping sw1, ping sw2, ping unreachable host)

```

Node: h3
root@mininet-vm:~# ping 10.0.1.3
PING 10.0.1.3 (10.0.1.3) 56(84) bytes of data.
64 bytes from 10.0.1.3: icmp_seq=1 ttl=64 time=0.401 ms
64 bytes from 10.0.1.3: icmp_seq=2 ttl=64 time=0.083 ms
^C
--- 10.0.1.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.083/0.242/0.401/0.159 ms
root@mininet-vm:~# ping 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=26.6 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=18.3 ms
^C
--- 10.0.1.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 18.301/22.485/26.670/4.187 ms
root@mininet-vm:~# ping 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
64 bytes from 10.0.2.1: icmp_seq=1 ttl=64 time=3.21 ms
^C
--- 10.0.2.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 3.213/3.213/3.213/0.000 ms
root@mininet-vm:~# ping 10.0.5.1
PING 10.0.5.1 (10.0.5.1) 56(84) bytes of data.
From 10.0.1.1 icmp_seq=1 Destination Net Unreachable
From 10.0.1.1 icmp_seq=2 Destination Net Unreachable
^C
--- 10.0.5.1 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1001ms

root@mininet-vm:~# █
```

Iperf test between h3 and h5

Result: TCP bandwidth 30.6 Gbps

```
"Node: h3"
root@mininet-vm:~# iperf -c 10.0.2.2
-----
Client connecting to 10.0.2.2, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 17] local 10.0.1.2 port 60706 connected with 10.0.2.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.0 sec  35.6 GBytes 30.6 Gbits/sec
root@mininet-vm:~#
```

Flow table of s1.

From this table, we can see the last two entries are for L2 switching using mac matching. The others are L3 switching entries which use ip matching.

00:00:11:00:00:11 is mac address of s1

00:00:11:00:00:12 is mac address of s2

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=8.973s, table=0, n_packets=2, n_bytes=196, idle_timeout=240,
  idle_age=8, priority=200,ip,nw_dst=10.0.1.2
  actions=mod_dl_dst:00:00:00:00:00:01,mod_dl_src:00:00:11:00:00:11,output:1
  cookie=0x0, duration=8.813s, table=0, n_packets=2, n_bytes=196, idle_timeout=240,
  idle_age=8, priority=200,ip,nw_dst=10.0.1.3
  actions=mod_dl_dst:00:00:00:00:00:02,mod_dl_src:00:00:11:00:00:11,output:2
  cookie=0x0, duration=12.62s, table=0, n_packets=4, n_bytes=392, idle_age=8,
  ip,nw_dst=10.0.2.0/24
  actions=mod_dl_dst:00:00:11:00:00:12,mod_dl_src:00:00:11:00:00:11,output:3
  cookie=0x0, duration=8.852s, table=0, n_packets=1, n_bytes=42, idle_age=3,
  dl_dst=00:00:00:00:00:02 actions=output:2
  cookie=0x0, duration=8.861s, table=0, n_packets=2, n_bytes=140, idle_age=3,
  dl_dst=00:00:00:00:00:01 actions=output:1
```

Flow table of s2

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=839.203s, table=0, n_packets=235402, n_bytes=15536668,
  idle_age=511, ip,nw_dst=10.0.1.0/24
  actions=mod_dl_dst:00:00:11:00:00:11,mod_dl_src:00:00:11:00:00:12,output:2
  cookie=0x0, duration=321.184s, table=0, n_packets=0, n_bytes=0, idle_age=321,
  dl_dst=00:00:11:00:00:11 actions=output:2
  cookie=0x0, duration=521.56s, table=0, n_packets=0, n_bytes=0, idle_age=521,
  dl_dst=00:00:00:00:00:03 actions=output:1
```

Part 3: Firewall

We use the same topology as part2 but add a firewall rule to s1 dropping all http packet from s3 to s5.

Implementation:

Use higher priority to add firewall rule. In this case, we use `OFP_DEFAULT_PRIORITY + 100`, which is the highest in the flow table. Every incoming packet will first try to match this firewall rule. It would be dropped if matched, otherwise go to next rule.

Ping test (pingall)

```
mininet> pingall
*** Ping: testing ping reachability
h3 -> h4 h5
h4 -> h3 h5
h5 -> h3 h4
*** Results: 0% dropped (6/6 received)
```

Flow table of s1 (the red one is the firewall rule)

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):

  cookie=0x0, duration=76.466s, table=0, n_packets=162451, n_bytes=10721774,
  idle_timeout=240, idle_age=53, ip,nw_dst=10.0.1.2
  actions=mod_dl_dst:00:00:00:00:00:01,mod_dl_src:00:00:11:00:00:11,output:1

  cookie=0x0, duration=265.173s, table=0, n_packets=705847, n_bytes=40264194478,
  idle_age=53, ip,nw_dst=10.0.2.0/24
  actions=mod_dl_dst:00:00:11:00:00:12,mod_dl_src:00:00:11:00:00:11,output:3

  cookie=0x0, duration=265.135s, table=0, n_packets=7, n_bytes=518, idle_age=162,
  priority=32868,tcp,nw_src=10.0.1.2,nw_dst=10.0.2.2,tp_dst=80 actions=drop

  cookie=0x0, duration=76.471s, table=0, n_packets=0, n_bytes=0, idle_age=76,
  dl_dst=00:00:00:00:00:01 actions=output:1
```

Iperf test

```

Node: h3
root@mininet-vm:~# iperf -c 10.0.2.2
-----
Client connecting to 10.0.2.2, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 17] local 10.0.1.2 port 33904 connected with 10.0.2.2 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 17] 0.0-10.0 sec  37.3 GBytes  32.0 Gbits/sec
root@mininet-vm:~#
root@mininet-vm:~#
root@mininet-vm:~#
root@mininet-vm:~# iperf -c 10.0.2.2 -p 80
connect failed: Connection timed out
root@mininet-vm:~# █

```

```

Node: h5
root@mininet-vm:~# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 18] local 10.0.2.2 port 5001 connected with 10.0.1.2 port 33904
[ ID] Interval      Transfer    Bandwidth
[ 18] 0.0-10.0 sec  37.3 GBytes  32.0 Gbits/sec
^Croot@mininet-vm:~#
root@mininet-vm:~# iperf -s -p 80
-----
Server listening on TCP port 80
TCP window size: 85.3 KByte (default)
-----
█

```

This result shows h3 is able to connect h5's 5001 port and transfer data but cannot connect to port 80.