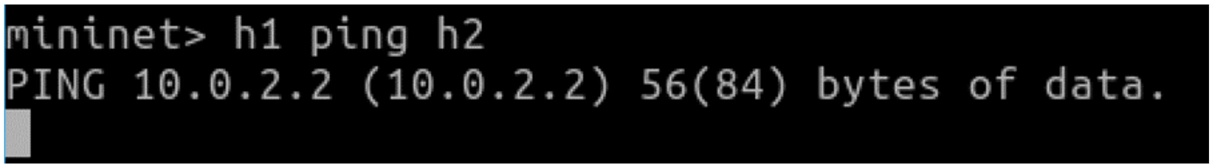
# Networked Systems Coursework 2: Dataplane Programming in P4

Viyan Raj

## 1 Deliverable 1: Basic Forwarding

### 1.1 Mininet instance with pod topology

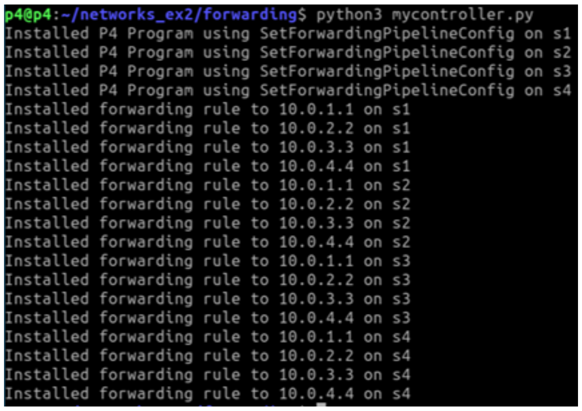
The file 'forwarding.p4' was compiled, and a mininet instance with pod topology was started.



```
mininet> h1 ping h2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
```

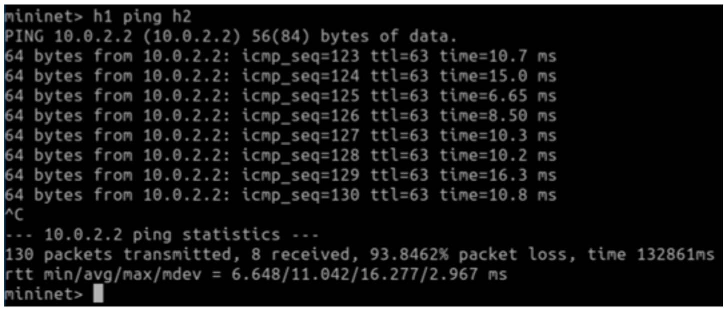
Figure 1: Running 'h1 ping h2'

This shows that h1 cannot ping h2 yet, since the controller rules haven't been installed.



```
p4@p4:~/networks_ex2/forwarding$ python3 mycontroller.py
Installed P4 Program using SetForwardingPipelineConfig on s1
Installed P4 Program using SetForwardingPipelineConfig on s2
Installed P4 Program using SetForwardingPipelineConfig on s3
Installed P4 Program using SetForwardingPipelineConfig on s4
Installed forwarding rule to 10.0.1.1 on s1
Installed forwarding rule to 10.0.2.2 on s1
Installed forwarding rule to 10.0.3.3 on s1
Installed forwarding rule to 10.0.4.4 on s1
Installed forwarding rule to 10.0.1.1 on s2
Installed forwarding rule to 10.0.2.2 on s2
Installed forwarding rule to 10.0.3.3 on s2
Installed forwarding rule to 10.0.4.4 on s2
Installed forwarding rule to 10.0.1.1 on s3
Installed forwarding rule to 10.0.2.2 on s3
Installed forwarding rule to 10.0.3.3 on s3
Installed forwarding rule to 10.0.4.4 on s3
Installed forwarding rule to 10.0.1.1 on s4
Installed forwarding rule to 10.0.2.2 on s4
Installed forwarding rule to 10.0.3.3 on s4
Installed forwarding rule to 10.0.4.4 on s4
```

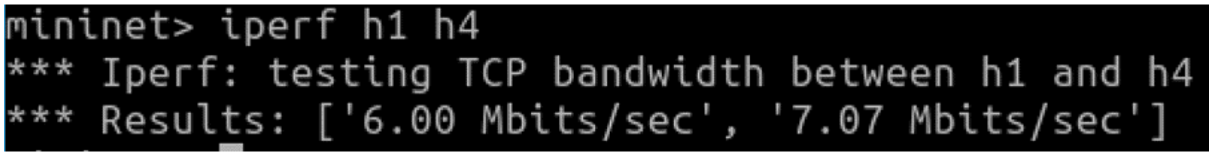
Figure 2: Running 'python3 mycontroller.py'



```
mininet> h1 ping h2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=123 ttl=63 time=10.7 ms
64 bytes from 10.0.2.2: icmp_seq=124 ttl=63 time=15.0 ms
64 bytes from 10.0.2.2: icmp_seq=125 ttl=63 time=6.65 ms
64 bytes from 10.0.2.2: icmp_seq=126 ttl=63 time=8.50 ms
64 bytes from 10.0.2.2: icmp_seq=127 ttl=63 time=10.3 ms
64 bytes from 10.0.2.2: icmp_seq=128 ttl=63 time=10.2 ms
64 bytes from 10.0.2.2: icmp_seq=129 ttl=63 time=16.3 ms
64 bytes from 10.0.2.2: icmp_seq=130 ttl=63 time=10.8 ms
^C
--- 10.0.2.2 ping statistics ---
130 packets transmitted, 8 received, 93.8462% packet loss, time 132861ms
rtt min/avg/max/mdev = 6.648/11.042/16.277/2.967 ms
mininet>
```

Figure 3: Running 'h1 ping h2' again

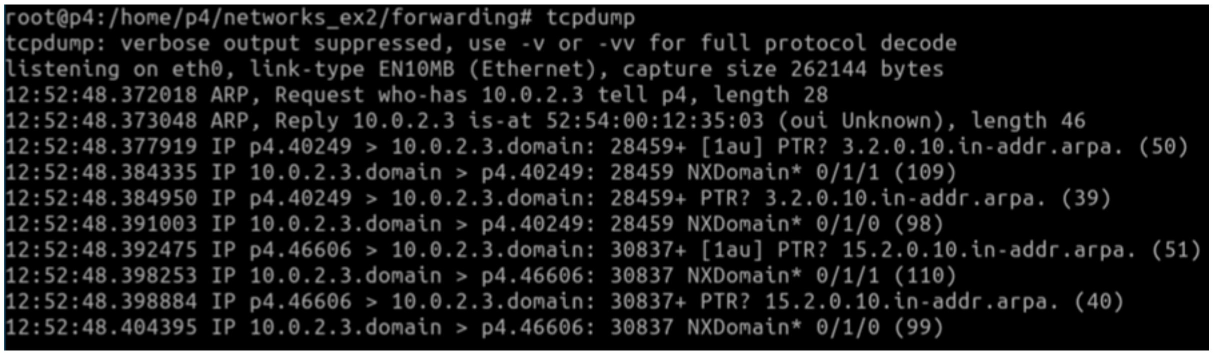
h1 can now ping h2 since the controller rules have been installed.



```
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['6.00 Mbits/sec', '7.07 Mbits/sec']
```

Figure 4: Running 'iperf h1 h4'

This shows the TCP bandwidth between h1 and h4.



```
root@p4:/home/p4/networks_ex2/forwarding# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:52:48.372018 ARP, Request who-has 10.0.2.3 tell p4, length 28
12:52:48.373048 ARP, Reply 10.0.2.3 is-at 52:54:00:12:35:03 (oui Unknown), length 46
12:52:48.377919 IP p4.40249 > 10.0.2.3.domain: 28459+ [1au] PTR? 3.2.0.10.in-addr.arpa. (50)
12:52:48.384335 IP 10.0.2.3.domain > p4.40249: 28459 NXDomain* 0/1/1 (109)
12:52:48.384950 IP p4.40249 > 10.0.2.3.domain: 28459+ PTR? 3.2.0.10.in-addr.arpa. (39)
12:52:48.391003 IP 10.0.2.3.domain > p4.40249: 28459 NXDomain* 0/1/0 (98)
12:52:48.392475 IP p4.46606 > 10.0.2.3.domain: 30837+ [1au] PTR? 15.2.0.10.in-addr.arpa. (51)
12:52:48.398253 IP 10.0.2.3.domain > p4.46606: 30837 NXDomain* 0/1/1 (110)
12:52:48.398884 IP p4.46606 > 10.0.2.3.domain: 30837+ PTR? 15.2.0.10.in-addr.arpa. (40)
12:52:48.404395 IP 10.0.2.3.domain > p4.46606: 30837 NXDomain* 0/1/0 (99)
```

Figure 5: Running 'tcpdump'

### 1.1.1 h1 to h4 path

The packet first goes out from h1 to s1 (in through port 1, out through port 4). It then goes to s4 (in through port 2, out through port 1). It then goes to s2 (in through port 3, out through port 2). Finally, it is forwarded to h4.

### 1.1.2 h4 to h1 path

The packet first goes out from h4 to s2 (in through port 2, out through port 4). It then goes to s3 (in through port 2, out through port 1). It then goes to s1 (in through port 3, out through port 1). Finally, it is forwarded to h1.

## 1.2 Inspecting mycontroller.py script

### 1.2.1 How traffic is routed between machines

Traffic is routed using a P4Runtime controller which implements forwarding rules in each switch. The controller script (controller.py) manages the setup of the switches and specifies how packets are to be forwarded.

It does this by extracting their destination IP address from the packet header. The controller establishes connections to all switches (s1, s2, s3, s4) and installs a P4 program onto each switch, defining the rules for packet processing.

For example, the controller configures forwarding rules in s1 to forward traffic destined for IP address 10.0.1.1 to port 1 with the corresponding Ethernet address 08:00:00:00:01:11. Other rules are defined for other IP addresses on s1 as well as the other switches (s2, s3, and s4).

As a packet traverses the network, each switch it encounters applies the specified rules to determine the next hop. For example, if a packet from h1 with destination IP 10.0.2.2 arrives at s1, the forwarding rule instructs s1 to forward the packet to port 2 with the destination Ethernet address 08:00:00:00:02:22. At each switch the packet reaches, this process will be repeated until the packet reaches its destination.

The controller establishes and manages the logical pathways through the network by configuring rules that guide how packets are directed based on their destination IP addresses.

#### 1.2.2 Problems with Routing Logic

There are two main problems with this routing logic.

1. Static Routing: The routing logic is static and hardcoded. It assumes that the network topology will stay constant and IP addresses assigned to hosts/switches won't change. In a dynamic network environment where devices may be added or removed, this is impractical. A more scalable solution would use a dynamic routing protocol which adapts to changes in the network topology.

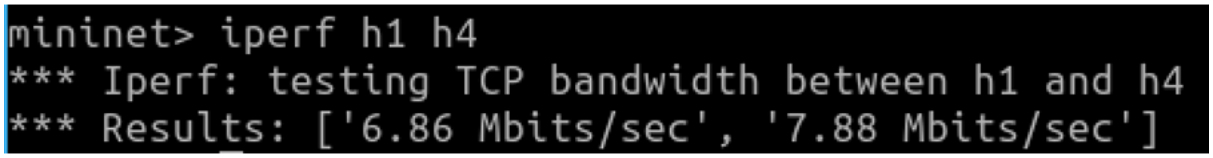
2. Lack of Redundancy: The routing logic doesn't account for fault tolerance. There are redundant links in the network topology which aren't being made use of. If a link or switch fails, there are no alternative paths defined in the routing rules. Implementing redundancy mechanisms would improve the network's reliability and resilience.

## 2 Deliverable 2: Equal Cost Multi Path

### 2.1 ECMP Implementation

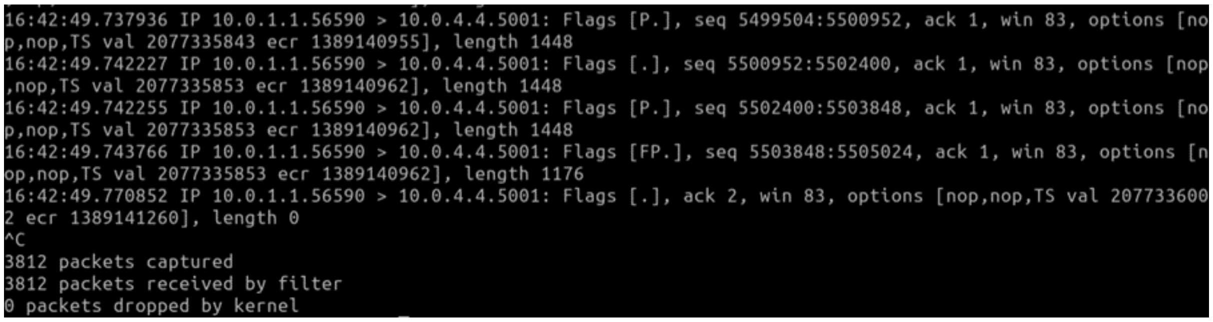
The dataplane (ecmp.p4) and control plane (mycontroller.py) were extended, following the specification, to implement ECMP functionality. Below are tcpdump captures from both s1-eth3 and s1-eth4 when running 'iperf h1 h4' on two separate instances (without making any changes to the code).

### 2.2 First instance of running 'iperf h1 h4'



```
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['6.86 Mbits/sec', '7.88 Mbits/sec']
```

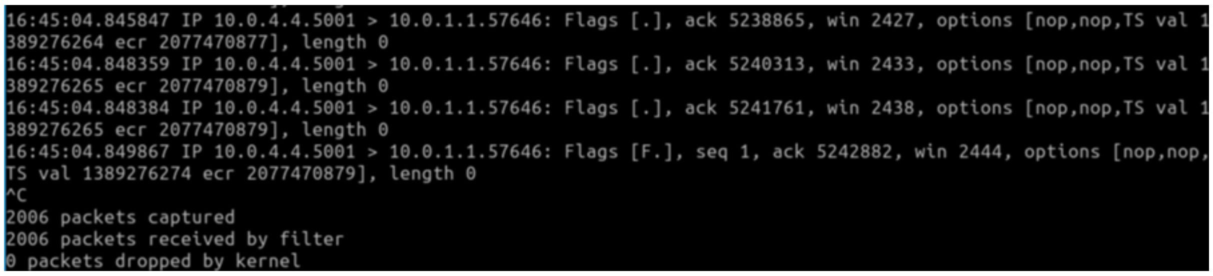
Figure 6: Running 'iperf h1 h4'



```
16:42:49.737936 IP 10.0.1.1.56590 > 10.0.4.4.5001: Flags [P.], seq 5499504:5500952, ack 1, win 83, options [no
p,nop,TS val 2077335843 ecr 1389140955], length 1448
16:42:49.742227 IP 10.0.1.1.56590 > 10.0.4.4.5001: Flags [.], seq 5500952:5502400, ack 1, win 83, options [nop
,nop,TS val 2077335853 ecr 1389140962], length 1448
16:42:49.742255 IP 10.0.1.1.56590 > 10.0.4.4.5001: Flags [P.], seq 5502400:5503848, ack 1, win 83, options [no
p,nop,TS val 2077335853 ecr 1389140962], length 1448
16:42:49.743766 IP 10.0.1.1.56590 > 10.0.4.4.5001: Flags [FP.], seq 5503848:5505024, ack 1, win 83, options [n
op,nop,TS val 2077335853 ecr 1389140962], length 1176
16:42:49.770852 IP 10.0.1.1.56590 > 10.0.4.4.5001: Flags [.], ack 2, win 83, options [nop,nop,TS val 207733600
2 ecr 1389141260], length 0
^C
3812 packets captured
3812 packets received by filter
0 packets dropped by kernel
```

Figure 7: tcpdump capture on 's1-eth3' interface

All transmit traffic is being sent via this s1-eth3 port.

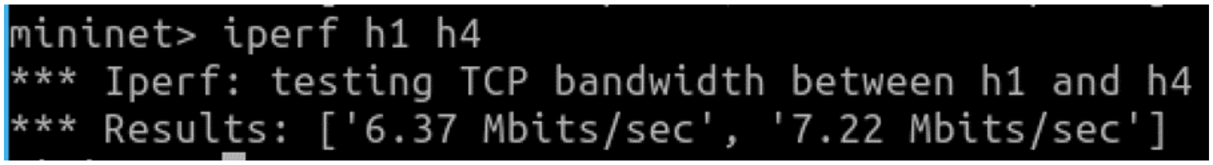


```
16:45:04.845847 IP 10.0.4.4.5001 > 10.0.1.1.57646: Flags [.], ack 5238865, win 2427, options [nop,nop,TS val 1
389276264 ecr 2077470877], length 0
16:45:04.848359 IP 10.0.4.4.5001 > 10.0.1.1.57646: Flags [.], ack 5240313, win 2433, options [nop,nop,TS val 1
389276265 ecr 2077470879], length 0
16:45:04.848384 IP 10.0.4.4.5001 > 10.0.1.1.57646: Flags [.], ack 5241761, win 2438, options [nop,nop,TS val 1
389276265 ecr 2077470879], length 0
16:45:04.849867 IP 10.0.4.4.5001 > 10.0.1.1.57646: Flags [F.], seq 1, ack 5242882, win 2444, options [nop,nop,
TS val 1389276274 ecr 2077470879], length 0
^C
2006 packets captured
2006 packets received by filter
0 packets dropped by kernel
```

Figure 8: tcpdump capture on 's1-eth4' interface

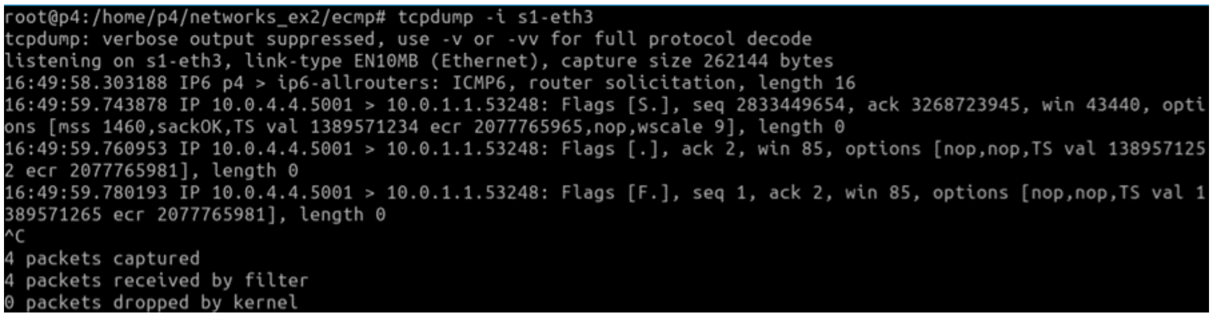
All receive traffic (acknowledgement) is sent via this s1-eth4 port.

## 2.3 Subsequent instance of running 'iperf h1 h4'



```
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['6.37 Mbits/sec', '7.22 Mbits/sec']
```

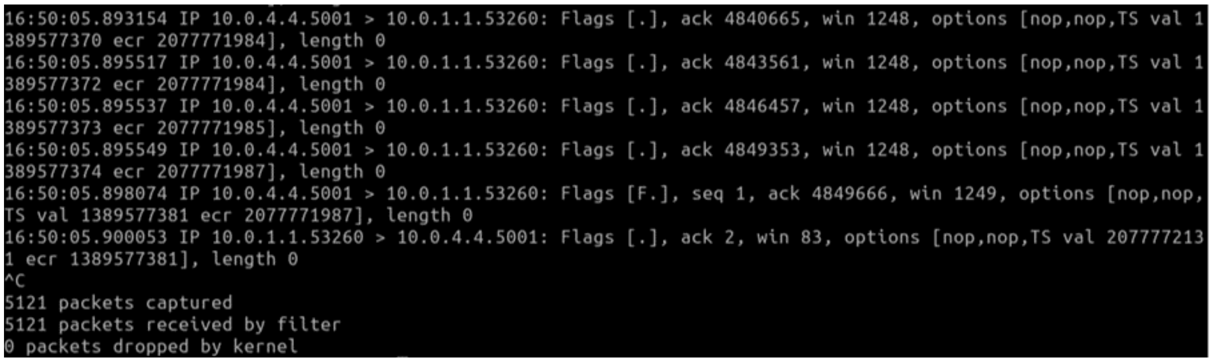
Figure 9: Running 'iperf h1 h4'



```
root@p4:/home/p4/networks_ex2/ecmp# tcpdump -i s1-eth3
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s1-eth3, link-type EN10MB (Ethernet), capture size 262144 bytes
16:49:58.303188 IP6 p4 > ip6-allrouters: ICMP6, router solicitation, length 16
16:49:59.743878 IP 10.0.4.4.5001 > 10.0.1.1.53248: Flags [S.], seq 2833449654, ack 3268723945, win 43440, opti
ons [mss 1460,sackOK,TS val 1389571234 ecr 2077765965,nop,wscale 9], length 0
16:49:59.760953 IP 10.0.4.4.5001 > 10.0.1.1.53248: Flags [.], ack 2, win 85, options [nop,nop,TS val 138957125
2 ecr 2077765981], length 0
16:49:59.780193 IP 10.0.4.4.5001 > 10.0.1.1.53248: Flags [F.], seq 1, ack 2, win 85, options [nop,nop,TS val 1
389571265 ecr 2077765981], length 0
^C
4 packets captured
4 packets received by filter
0 packets dropped by kernel
```

Figure 10: tcpdump capture on 's1-eth3' interface

All receive traffic (acknowledgement) is being sent via this s1-eth3 port.



```
16:50:05.893154 IP 10.0.4.4.5001 > 10.0.1.1.53260: Flags [.], ack 4840665, win 1248, options [nop,nop,TS val 1
389577370 ecr 2077771984], length 0
16:50:05.895517 IP 10.0.4.4.5001 > 10.0.1.1.53260: Flags [.], ack 4843561, win 1248, options [nop,nop,TS val 1
389577372 ecr 2077771984], length 0
16:50:05.895537 IP 10.0.4.4.5001 > 10.0.1.1.53260: Flags [.], ack 4846457, win 1248, options [nop,nop,TS val 1
389577373 ecr 2077771985], length 0
16:50:05.895549 IP 10.0.4.4.5001 > 10.0.1.1.53260: Flags [.], ack 4849353, win 1248, options [nop,nop,TS val 1
389577374 ecr 2077771987], length 0
16:50:05.898074 IP 10.0.4.4.5001 > 10.0.1.1.53260: Flags [F.], seq 1, ack 4849666, win 1249, options [nop,nop,
TS val 1389577381 ecr 2077771987], length 0
16:50:05.900053 IP 10.0.1.1.53260 > 10.0.4.4.5001: Flags [.], ack 2, win 83, options [nop,nop,TS val 207777213
1 ecr 1389577381], length 0
^C
5121 packets captured
5121 packets received by filter
0 packets dropped by kernel
```

Figure 11: tcpdump capture on 's1-eth4' interface

All transmit traffic is sent via this s1-eth4 port.

## 2.4 Bidirectional TCP traffic

As shown in the screenshots above, transmit and receive traffic follows different (asymmetric) paths.