Waterford Institute of Technology

Cloud Computing

Bachelor of Science (Hons) Applied Computing

SDN OpenFlow Firewall

Ciaran Roche - 20037160

April 16, 2019

Plagiarism Declaration

Unless otherwise stated all the work contained in this report is my own. I also state that I have not submitted this work for any other course of study leading to an academic award.

Contents

1 Introduction

This paper discusses the work carried out during the Openflow Firewall Assignment, some background information is also given to demonstrate overall understanding of Software Defined Networks.

The practical element looked at building on a Learning Switch to add Firewall rules to create block flows in the Switch. These rules where read from a CSV file and both MAC address filtering and also IP address filtering are implemented. Finally we look at the Transport Layer by blocking port numbers.

2 OpenFlow

'OpenFlow is a communications protocol that gives access to the forwarding plane of a network switch or router over the network.' (*OpenFlow*, 2018) This definition is the result of Google, coming directly from Wikipedia. By the end of this section I will demonstrate an understanding of this definition and provide some context to it.

An understanding of what a Software-Defined Network (SDN) is, is needed before discussing OpenFlow. We should consider an SDN as a paradigm for the decoupling of forwarding hardward and the control decisions. In other words, it is the physical separation of the network control plane from the forwarding plane. It allows for the control plane to be extensible in that it can control multiple devices (*Software-Defined Networking (SDN) Definition*, n.d.).

If we think of the control plane as the state of our routers and switches, in that the control plane worries about how and where packets are forwarded, it deals with routing, traffic engineering, firewall and all that good stuff. This is where OpenFlow fits in. As it is a communications protocol that enables the SDN Controller to interact with the control plane of a networks devices. We think of a controller as the essential brain behind an SDN network as it relays information around the network to switches/routers via southbound APIs (*What is OpenFlow? Definition and How it Relates to SDN*, n.d.).

Through this information, we are able to push changes to our network nodes, in the form of flows. We can insert flows into our nodes flow-tables which allow us to partition traffic, control flows and tweak for optimal performance. Through the use of flows we are able to leverage our own firewall within a switch. And this will be demonstrated within Section 3.

3 Practical

3.1 Prerequisites

To start this practical section I must mention a few prerequisites and define some terminology which is used in this practical. The practical itself builds upon the tutorial linked in Appendix B. To build our development environment we use an OpenFlowTutorial VM. Within this Virtual Machine we get a number of utilities out of the box:
OpenFlow Controller – this acts as an Ethernet learning switch when combined to an OpenFlow switch. This is used in the early parts of the practical as I became familiar with the workings of OpenFlow.
OpenFlow Switch – Like the Controller above, this is used in the early stages of the practical gaining experince of the new technologies.
ovs-ofctl – is a command line tool used for debugging and testing throughout the practical. As it allows for the quick sending of OpenFlow messages. As well as manually setting flows within the switches. Combined with the two mentioned above, they where used to further explore the environment and capabilities of OpenFlow and SDN.
WireShark – This graphical utility provided an interface to easily view the OpenFlow message patterns. Using the filter *of* I was easily abple to parse and view OpenFlow messages throughout the pracitical.
iperf – is a cli that allows for the testing of speed withing a TCP connection. While included in the VM I did not use this during the practical.
Mininet – creates a virtual OpenFlow network. Through the use of a single command I was easily able to spin up an OpenFlow network which will be showing in Section
cbench – a utility tool for testing the flow setup rate of OpenFlow controllers.

3.2 The Network

The following command was used to generate the topology seen in Figure 1 – *$ sudo mn –topo single,8 –mac –switch ovsk –controller remote.* This could of been scripted, but looking at the number of lines of code needed to generate the topology I decided on using the one-liner instead.

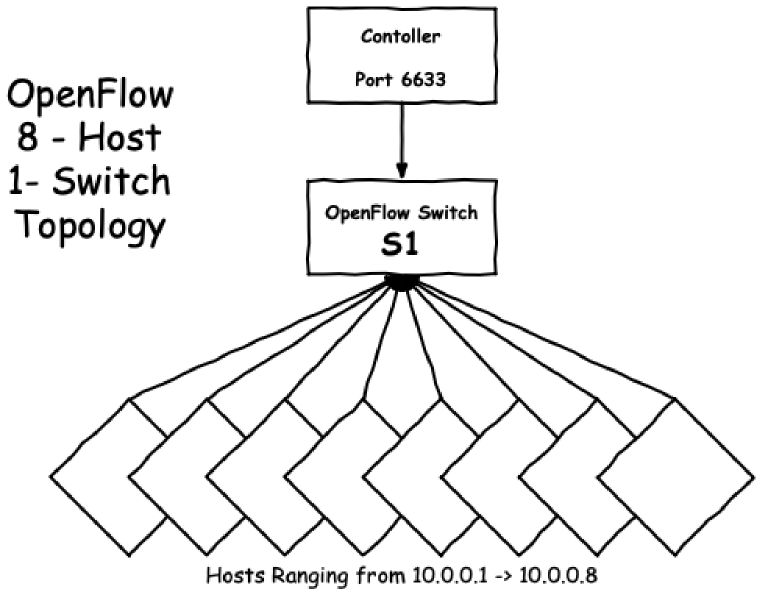
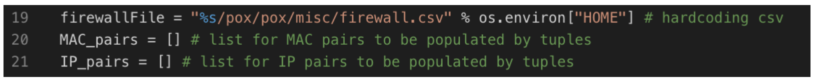


Figure 1: *OpenFlow Topology*

3.3 Steps Taken

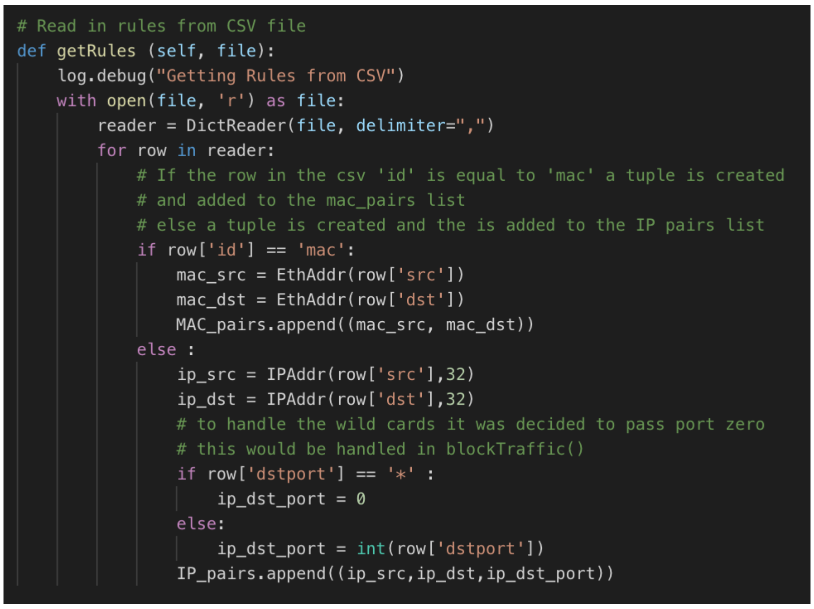
Before talking through the steps I need to point out there is two kind of flows, a proactive and a reactive. Reactive flows happen on when a message is received the controller looks for a match and creates a flow. Where as a proactive flow is where flows are inserted ahead of time. For the purpose of this practical I focused on proactive flows mainly due to my own skill level in python, as I do not care to admit the time it took me to read in and parse the CSV file.
To begin as our CSV file had both IP and MAC address to deal with, I decided to split the list into two. The initial lists can be seen in Figure 2 along with the hardcoded CSV location. If we look at Figure 3, I am parsing the CSV file, for every line in the file I am checking the key, if it is equal to *MAC* I am adding creating a tuple containing the *srcMac* and *dstMac* and adding it to my mac list. If it doesnt have the key *MAC* I am creating a tuple containing *srcIP destIP* and *destPort*. This tuple was added to my ip list.



```python
firewallFile = "%s/pox/pox/misc/firewall.csv" % os.environ["HOME"] # hardcoding csv
MAC_pairs = [] # list for MAC pairs to be populated by tuples
IP_pairs = [] # list for IP pairs to be populated by tuples
```

Figure 2: *Declaring Lists*

Just to note, the csv contained a wildcard for the destPort, later on when creating the flow I had no way to handle the wildcard, so for that I added a check here. And let any wild cards equal to zero, I picked zero as this is a port that is not used. Another thing worth mentioning, as I was creating the tuples, I cast the following to the values, EthAddr to mac addresses, IpAddr to IP addresses and a regular int to port numbers, this is required when creating the flow that the inputs are of correct value. Using the casts I was able to add the subnet mask to the IP addresses Figure 4 shows



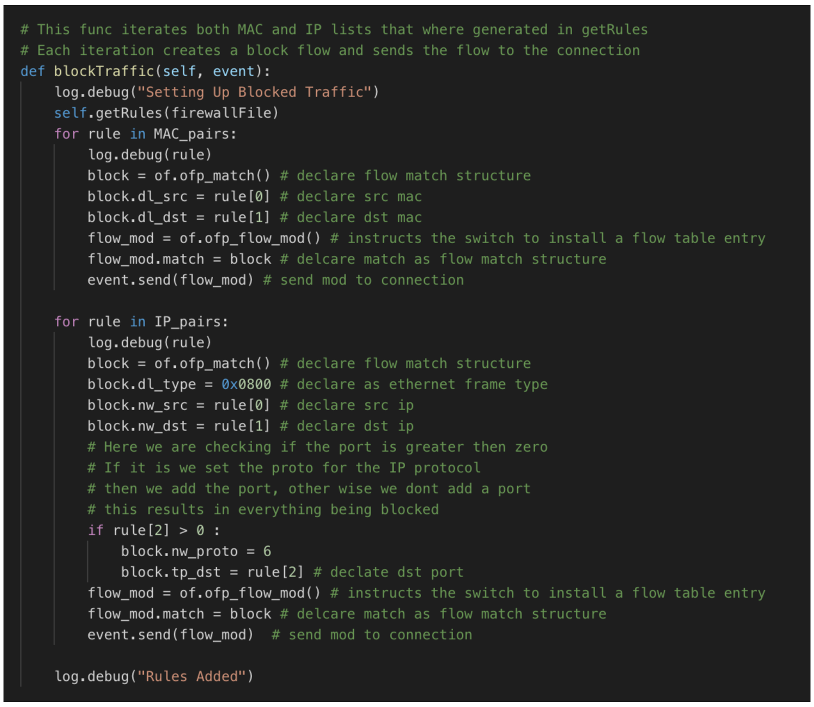
```python
# Read in rules from CSV file
def getRules (self, file):
    log.debug("Getting Rules from CSV")
    with open(file, 'r') as file:
        reader = DictReader(file, delimiter=",")
        for row in reader:
            # If the row in the csv 'id' is equal to 'mac' a tuple is created
            # and added to the mac_pairs list
            # else a tuple is created and the is added to the IP pairs list
            if row['id'] == 'mac':
                mac_src = EthAddr(row['src'])
                mac_dst = EthAddr(row['dst'])
                MAC_pairs.append((mac_src, mac_dst))
            else :
                ip_src = IPAddr(row['src'],32)
                ip_dst = IPAddr(row['dst'],32)
                # to handle the wild cards it was decided to pass port zero
                # this would be handled in blockTraffic()
                if row['dstport'] == '*' :
                    ip_dst_port = 0
                else:
                    ip_dst_port = int(row['dstport'])
                IP_pairs.append((ip_src,ip_dst,ip_dst_port))
```

Figure 3: *Parsing CSV File*

my function for adding flows. It consists of two for loops that iterates my IP and MAC lists. For every tuple it creates a rule. It does this through the use of openflow match function. With the flow created I declare an openflow mod type which I set my match to block before sending it to my connection.

Worth pointing out in the case of the port on the IP address, I add a check so that if the port is greater then 0 then set the proto 6, and the port to that specified in the tuple else dont add a port to the flow. In this case with no port added it blocks all communication between the two IP's. Proto 6 sets the type to TCP. Another thing worth mentioning I needed to set the type to 0x0800 which set the flow to match on Ethernet frames.

This function was executed within the *init* function belonging to the ActLikeSwitch solution.

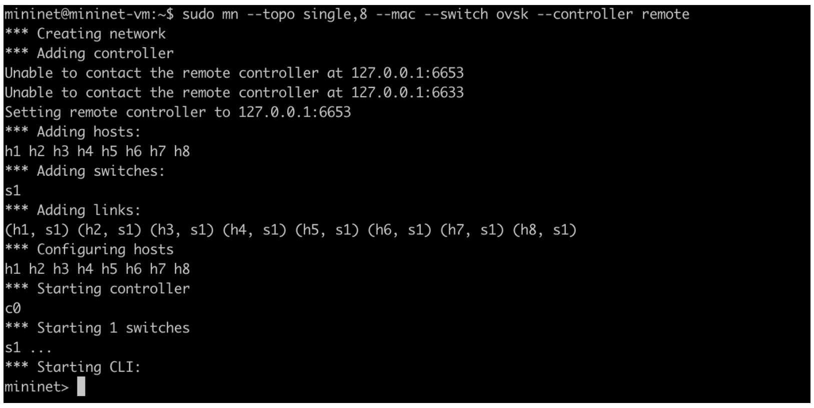


```python
# This func iterates both MAC and IP lists that where generated in getRules
# Each iteration creates a block flow and sends the flow to the connection
def blockTraffic(self, event):
    log.debug("Setting Up Blocked Traffic")
    self.getRules(firewallFile)
    for rule in MAC_pairs:
        log.debug(rule)
        block = of.ofp_match() # declare flow match structure
        block.dl_src = rule[0] # declare src mac
        block.dl_dst = rule[1] # declare dst mac
        flow_mod = of.ofp_flow_mod() # instructs the switch to install a flow table entry
        flow_mod.match = block # delcare match as flow match structure
        event.send(flow_mod) # send mod to connection

    for rule in IP_pairs:
        log.debug(rule)
        block = of.ofp_match() # declare flow match structure
        block.dl_type = 0x0800 # declare as ethernet frame type
        block.nw_src = rule[0] # declare src ip
        block.nw_dst = rule[1] # declare dst ip
        # Here we are checking if the port is greater then zero
        # If it is we set the proto for the IP protocol
        # then we add the port, other wise we dont add a port
        # this results in everything being blocked
        if rule[2] > 0 :
            block.nw_proto = 6
            block.tp_dst = rule[2] # declate dst port
        flow_mod = of.ofp_flow_mod() # instructs the switch to install a flow table entry
        flow_mod.match = block # delcare match as flow match structure
        event.send(flow_mod)  # send mod to connection

    log.debug("Rules Added")
```

Figure 4: *Adding Flows*

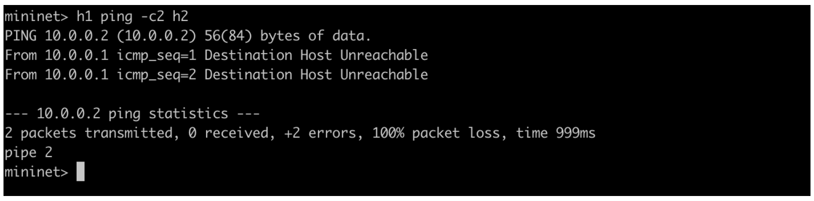
3.4 Verification

The following section shows images demonstrating the verification steps. For brevity I have added the descriptions to the image tags explaining the steps.



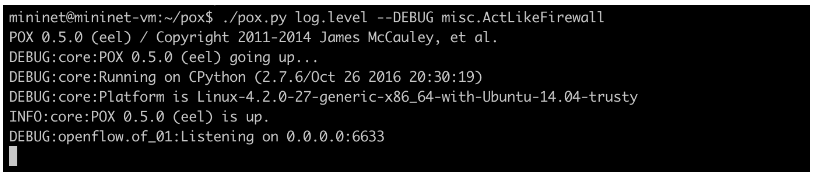
```
mininet@mininet-vm:~$ sudo mn --topo single,8 --mac --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1) (h7, s1) (h8, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> 
```

Figure 5: *Creating the Topology*



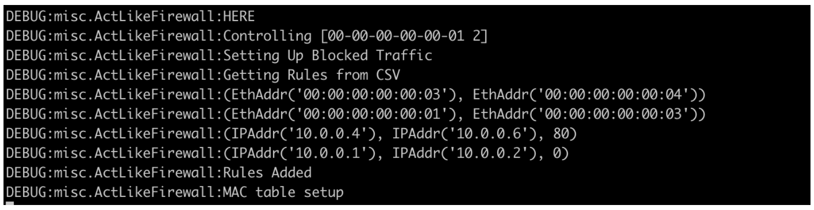
```
mininet> h1 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 999ms
pipe 2
mininet>
```

Figure 6: *Showing there is no connectivity due to their being no controller*



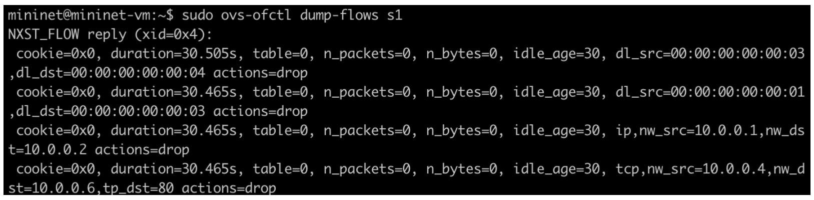
```
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG misc.ActLikeFirewall
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.6/Oct 26 2016 20:30:19)
DEBUG:core:Platform is Linux-4.2.0-27-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
```

Figure 7: *Creating the controller, ActLikeFirewall*



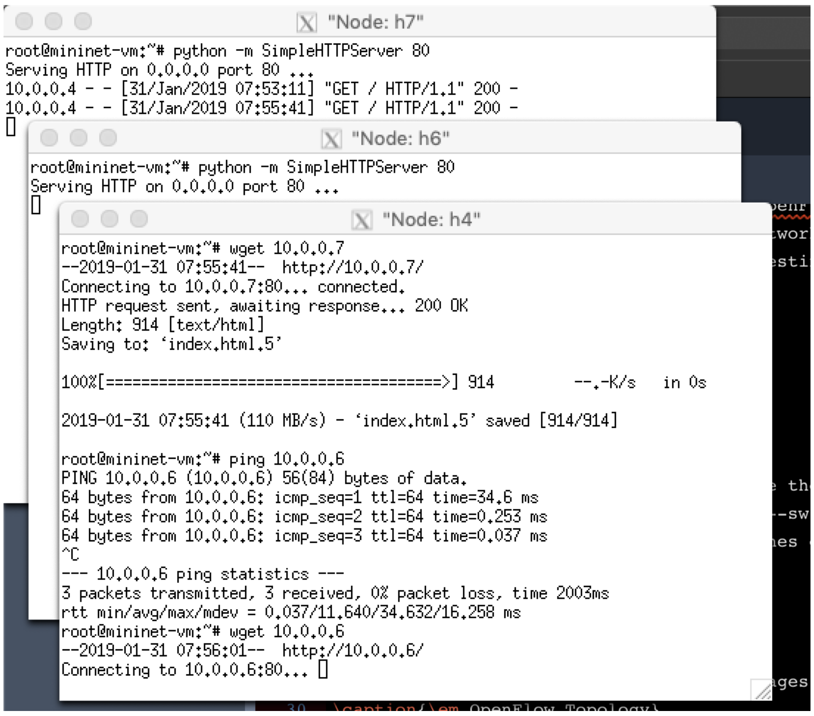
```
DEBUG:misc.ActLikeFirewall:HERE
DEBUG:misc.ActLikeFirewall:Controlling [00-00-00-00-00-01 2]
DEBUG:misc.ActLikeFirewall:Setting Up Blocked Traffic
DEBUG:misc.ActLikeFirewall:Getting Rules from CSV
DEBUG:misc.ActLikeFirewall:(EthAddr('00:00:00:00:00:03'), EthAddr('00:00:00:00:00:04'))
DEBUG:misc.ActLikeFirewall:(EthAddr('00:00:00:00:00:01'), EthAddr('00:00:00:00:00:03'))
DEBUG:misc.ActLikeFirewall:(IPAddr('10.0.0.4'), IPAddr('10.0.0.6'), 80)
DEBUG:misc.ActLikeFirewall:(IPAddr('10.0.0.1'), IPAddr('10.0.0.2'), 0)
DEBUG:misc.ActLikeFirewall:Rules Added
DEBUG:misc.ActLikeFirewall:MAC table setup
```

Figure 8: *Flows being added to table*



```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=30.505s, table=0, n_packets=0, n_bytes=0, idle_age=30, dl_src=00:00:00:00:00:03
,dl_dst=00:00:00:00:00:04 actions=drop
 cookie=0x0, duration=30.465s, table=0, n_packets=0, n_bytes=0, idle_age=30, dl_src=00:00:00:00:00:01
,dl_dst=00:00:00:00:00:03 actions=drop
 cookie=0x0, duration=30.465s, table=0, n_packets=0, n_bytes=0, idle_age=30, ip,nw_src=10.0.0.1,nw_ds
t=10.0.0.2 actions=drop
 cookie=0x0, duration=30.465s, table=0, n_packets=0, n_bytes=0, idle_age=30, tcp,nw_src=10.0.0.4,nw_d
st=10.0.0.6,tp_dst=80 actions=drop
```

Figure 9: *The flow table showing newly created flows, based on input from csv*

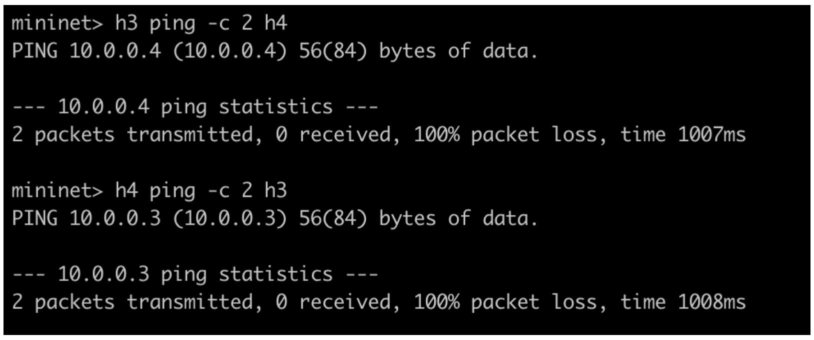


```
root@mininet-vm:~# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.4 - - [31/Jan/2019 07:53:11] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [31/Jan/2019 07:55:41] "GET / HTTP/1.1" 200 -
```

```
root@mininet-vm:~# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

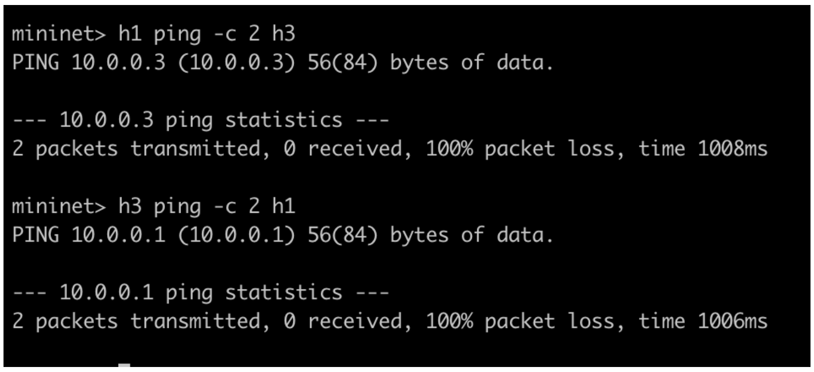
```
root@mininet-vm:~# wget 10.0.0.7
--2019-01-31 07:55:41--  http://10.0.0.7/
Connecting to 10.0.0.7:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 914 [text/html]
Saving to: ‘index.html.5’

100%[=====================================>] 914         --.-K/s   in 0s

2019-01-31 07:55:41 (110 MB/s) - ‘index.html.5’ saved [914/914]

root@mininet-vm:~# ping 10.0.0.6
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.
64 bytes from 10.0.0.6: icmp_seq=1 ttl=64 time=34.6 ms
64 bytes from 10.0.0.6: icmp_seq=2 ttl=64 time=0.253 ms
64 bytes from 10.0.0.6: icmp_seq=3 ttl=64 time=0.037 ms
^C
--- 10.0.0.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.037/11.640/34.632/16.258 ms
root@mininet-vm:~# wget 10.0.0.6
--2019-01-31 07:56:01--  http://10.0.0.6/
Connecting to 10.0.0.6:80...
```

Figure 10: *3 windows showing a http server running on h7 and h6, h4 is allowed to communicate with h7 but is blocked from h6*



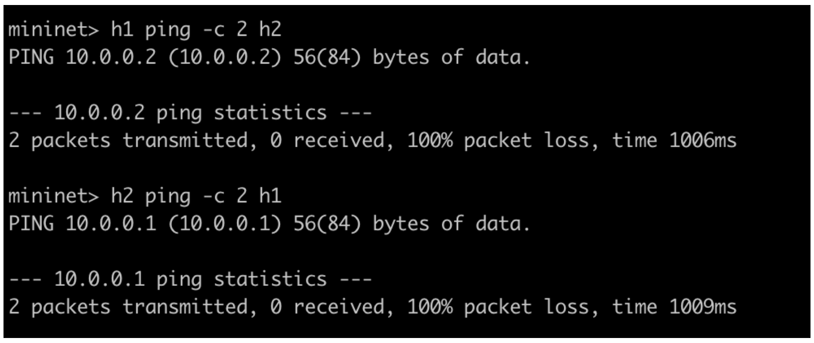
```
mininet> h3 ping -c 2 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.

--- 10.0.0.4 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1007ms

mininet> h4 ping -c 2 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.

--- 10.0.0.3 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1008ms
```

Figure 11: *Ping showing the rules imposed on communication between h3 and h4*



```
mininet> h1 ping -c 2 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.

--- 10.0.0.3 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1008ms

mininet> h3 ping -c 2 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.

--- 10.0.0.1 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1006ms
```

Figure 12: *Ping showing the rules imposed on communication between h1 and h3*



```
mininet> h1 ping -c 2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1006ms

mininet> h2 ping -c 2 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.

--- 10.0.0.1 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1009ms
```

Figure 13: *Ping showing the rules imposed on communication between h1 and h2*

4 Summary

What can I say? If I am honest this practical was not a nice dev experience. The first and it being the least relevant to SDN and Openflow as a technology, was my own lack of knowledge in Python. Small little nuances where tripping me up in this practical and meant more time was spent upskilling in python trying to get a grasp on the paradigm before I could tackle Openflow.
Away from that, I found the documentation poor, and also hard to find. Googling Openflow lead to an abundance of blog posts on the protocol and nothing on official documentation, which when found was poor. Mixed with my lack of python knowledge lead to a poor dev experience. This meant I leaned toward, going to blog posts and github for help, but due to different naming conventions between the versions often meant time was spent doing something that was incompatible with my version.
At face value my solution outlined in the report appears quite simple, but in reality to get to that simple solution took quite an amount of time to achieve. This leads me to think a greater level of abstraction is needed to provide a dev friendly experience. As this was a very basic example, I could see that it would become quite complex quite quickly depending on the use case.
There was also a couple of weird behaviors which I noted, when dumping the flows from the switch sometimes would not reflect the order in which they where inserted. This lead to some head scratching as I am used to order being pedantic within ACL's etc. On the topic of the likes of ACL's, this may be due to the example topology I used, I kept asking myself what value is Openflow adding, in that I am creating OpenFlow traffic on top of the regular network traffic and basically achieving the same functionality as the network would have without Openflow. That said looking at the bigger picture having a central control over the network would add a lot of value.
To sum up, a view of Openflow in a bigger environment is needed to gain a greater appreciation of the protocol. Also a greater level of abstraction to provide a cleaner developer experience.

Bibliography

OpenFlow (2018).
URL: *https://en.wikipedia.org/wiki/OpenFlow*

Software-Defined Networking (SDN) Definition (n.d.).
URL: *https://www.opennetworking.org/sdn-definition/*

What is OpenFlow? Definition and How it Relates to SDN (n.d.).
URL: *https://www.sdxcentral.com/sdn/definitions/what-is-openflow/*

Appendices

A MiniNet OpenFlow Tutorial

https://github.com/mininet/openflow-tutorial/wiki/Learn-Development-Tools

B OpenFlow Documentation

https://ryu.readthedocs.io/en/latest/ofproto_v1_0_ref.html