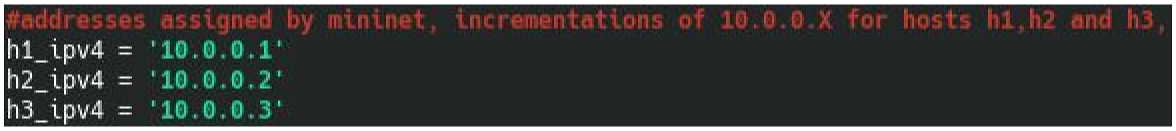
NWEN302 Lab 3 Report - David Burrell:300209541

Tasks implemented and achieved:

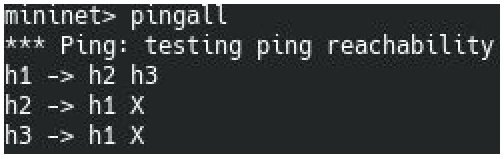
Key Task 1

Modify `simple_switch_13.py` to include logic to block traffic between host 2 and host 3.
This task was achieved by implementing methods that check packet information and then send a blocking flow if conditions have been met. Firstly the ip addresses of the hosts were needed, these were supplied by the mininet interface and hardcoded into my simple_switch file.



```
#addresses assigned by mininet, incrementations of 10.0.0.X for hosts h1,h2 and h3,
h1_ipv4 = '10.0.0.1'
h2_ipv4 = '10.0.0.2'
h3_ipv4 = '10.0.0.3'
```

Beyond this a checker method was implemented to look at the ipv4 data on an incoming packet, and if the check passes then a flow is passed on in which acts as a block of traffic between the chosen hosts. Resulting in pingall failing on h2 -> h3 and h3 -> h2

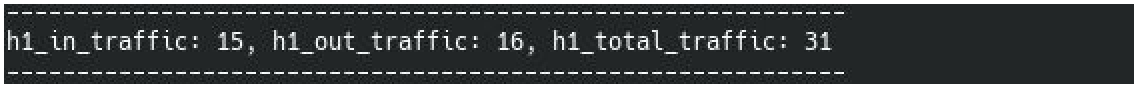


```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 X
h3 -> h1 X
```

The flows allow traffic to be passed without needing to be checked up in tables, this allows a better stream (or ‘flow’) of data. In the ryu implementation this means that any incoming packet that has a flow already set up will not need to go into the _in_packet_handler.
As stated: https://stackoverflow.com/questions/41023354/ryu-controller-drop-packet
The default handling of a packet in OpenFlow is to drop it, and thus installing an instruction to set the flow to default handling and matching it to the correct connection provides the dropping of traffic.

Key Task 2

Extend `simple_switch_13.py` to count all traffic going to and originating from host 1.
This was simpler task to implement, as most of the code I used was available at:
https://osrg.github.io/ryu-book/en/html/traffic_monitor.html?fbclid=IwAR0HQ6nrK7mbXzANQeIvzMhSFbnhWrG_QvT6vrzCbnMDiLTv6fl9ep7h1Zk
And the task was to simply identify the correct information to store in some global variables, and print an output of these values



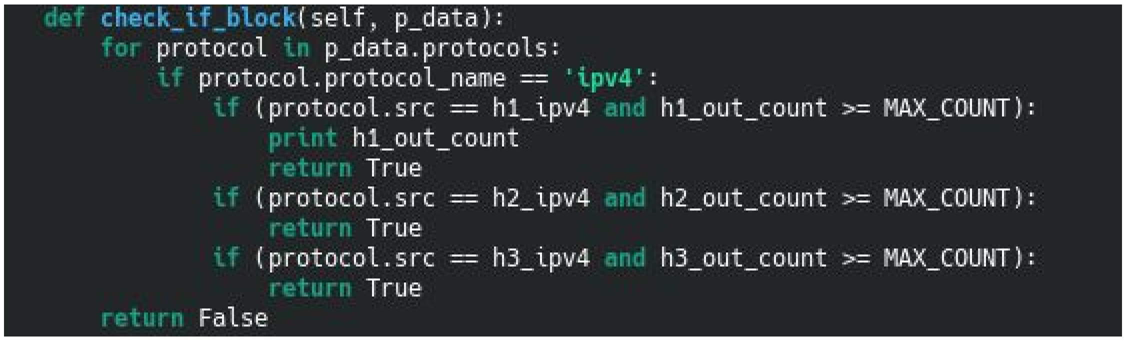
```
------------------------------------------------------------
h1_in_traffic: 15, h1_out_traffic: 16, h1_total_traffic: 31
------------------------------------------------------------
```

The values in question were within the port stats reply handler rx_packets for in traffic, tx_packets for sent and add them both together to get a total of the traffic.

Key Task 3

Extend `simple_switch_13.py` to combine Task 1 and Task 2 functionalities. Keep track of all traffic (count the number of packets) originating from each host. If the counter exceeds a specific number, block all the traffic originating from this host for 24 hours. The maximum packet count number should be configured through `MAX_COUNT` variable.

This task was attempted but fell short of successful implementation. The idea is the same as mentioned, combining task one and task two, using the port stats to calculate total traffic out of each host. Comparing against the global variable MAX COUNT and dropping the flow from the flow table if the criteria are met. The problems arise because of how flows work making intercepting the packet event a difficult task, as the use of *in packet handler* was bypassed.
I was unable to find any useful information about the types of events I could declare a method to use, as such my plan to find the event that triggers when flow packets are passed and check if count was exceeded and drop the flow there was moot.
Another idea was to use the *flow stats reply handler* to get the correct information needed to pass into the *drop_flow* method used in task one and drop the flow the same way, with the added caveat of using the *hard_timeout* flag in the msg mod to block for 24 hours.



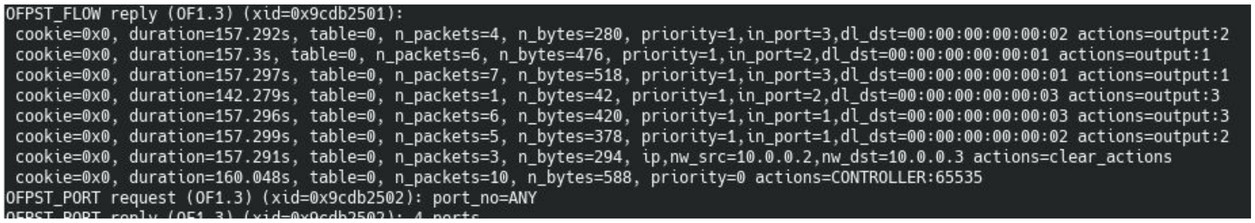
```
def check_if_block(self, p_data):
    for protocol in p_data.protocols:
        if protocol.protocol_name == 'ipv4':
            if (protocol.src == h1_ipv4 and h1_out_count >= MAX_COUNT):
                print h1_out_count
                return True
            if (protocol.src == h2_ipv4 and h2_out_count >= MAX_COUNT):
                return True
            if (protocol.src == h3_ipv4 and h3_out_count >= MAX_COUNT):
                return True
    return False
```

Modified task one checking code for task three

Reflection:
The main issues encountered were due to lack of information around implementations of the specific requested blocking processes, added to some less than ideal documentation provided much frustration. A lot of the early progress made was simply from looking at the various files found within the /ryu directory, mainly those in the /app and /packet directories. The webpage specifying the Traffic Monitor code was extremely useful for Task two.
And
https://inside-openflow.com/2016/07/21/ryu-api-dissecting-simple-switch/?fbclid=IwAR0K_snb1KHVMRmg7TAcOGq4jshApQAdomsBsBiTURAvUy84t1b9Un1EyiE
For a detailed breakdown of *simple_switch_13.py* that we built upon in this lab.

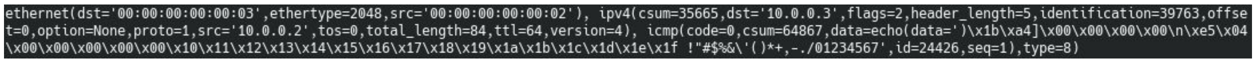
The level of abstraction we were interacting with was the source of most of the issues, some of this could be mitigated by the use of a proper IDE, simply having the ablity to see possible methods and information about various objects based on imports and such would have streamlined the information seeking process a bit, rather than the next to nothing supplied by a text editor. Python itself became an annoyance too because of this, and that can come down to my lack of experience with it.

Testing Methodologies:
The main testing method relied upon was simply pings, pingall to check task one, pings between other hosts to fill out the stats, which were being printed out by the monitor.
Python issues were dealt with using the python debugger, mainly indentation problems, and a few issues with global variable declarations.
The provided command line methods for fewing the flow-dumps



```
OFPST_FLOW reply (OF1.3) (xid=0x9cdb2501):
 cookie=0x0, duration=157.292s, table=0, n_packets=4, n_bytes=280, priority=1,in_port=3,dl_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x0, duration=157.3s, table=0, n_packets=6, n_bytes=476, priority=1,in_port=2,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=157.297s, table=0, n_packets=7, n_bytes=518, priority=1,in_port=3,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=142.279s, table=0, n_packets=1, n_bytes=42, priority=1,in_port=2,dl_dst=00:00:00:00:00:03 actions=output:3
 cookie=0x0, duration=157.296s, table=0, n_packets=6, n_bytes=420, priority=1,in_port=1,dl_dst=00:00:00:00:00:03 actions=output:3
 cookie=0x0, duration=157.299s, table=0, n_packets=5, n_bytes=378, priority=1,in_port=1,dl_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x0, duration=157.291s, table=0, n_packets=3, n_bytes=294, ip,nw_src=10.0.0.2,nw_dst=10.0.0.3 actions=clear_actions
 cookie=0x0, duration=160.048s, table=0, n_packets=10, n_bytes=588, priority=0 actions=CONTROLLER:65535
OFPST_PORT request (OF1.3) (xid=0x9cdb2502): port_no=ANY
OFPST_PORT reply (OF1.3) (xid=0x9cdb2502): 4 ports
```

Used to see my changes in the flow table.
I used the ever reliable print statements to do the usual check if loops were being entered and executed, as well as providing useful information regarding the contents of certain attributes such as this packet containing an ipv4 protocol header



```
ethernet(dst='00:00:00:00:00:03',ethertype=2048,src='00:00:00:00:00:02'), ipv4(csum=35665,dst='10.0.0.3',flags=2,header_length=5,identification=39763,offse
t=0,option=None,proto=1,src='10.0.0.2',tos=0,total_length=84,ttl=64,version=4), icmp(code=0,csum=64867,data=echo(data=')\x1b\xa4]\x00\x00\x00\x00\n\xe5\x04
\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567',id=24426,seq=1),type=8)
```

Observations:
Some time consuming issues arose as usual when dealing with these virtual machines, not many on this lab though. More time consuming was the constant needing to clear the mininet and reset everything between running of the ryu controller, but this was streamlined through repetition. Another annoyance is the lack of permissions in being able to write to the /usr/local/lib/python2.7/dist-packages/ryu/app folder.