**Johnny Sandiford 300365133**

**Lab 3 Report**

Before starting with task 1 the SimpleSwitch class needed to know the IP addresses and mac addresses of each host. As we are not implementing discover this was simple hardcoded in has value pairs, the host number paired with its associated information [ipv4\_addr, ipv4\_addr\_asInt, mac\_addr].

Please not the commented areas for each task as the code implemented is different in each task.

*Key Task 1*

*Modify simple\_switch\_13.py to include logic to block traffic between host 2 and host 3.*

The first task requires the blocking of traffic between host 2 and 3. Before approaching this task the packet\_in\_handler and flows must be understood. The handling of a packet is expressed through the event handler method packet\_in\_handler, it has event ev as a parameter which contains the msg data.

The skeletal code contains code for the flooding and forwarding of packets based on mac\_to\_port entries, however it currently allows the installation of flows between hosts 2 and 3.

Flows are a way to rout packets without the added overhead of doing lookups and handling packets individually for forwarding by the switch. It is commonly used when handling streaming data to ensure smoothness and prevent bugs and jitteriness. RYU allows the installation of flows between end points to by pass the packet\_in\_handler method. Therefore, the flow must be deleted.

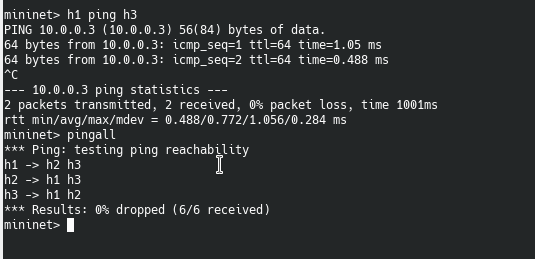
The way the flow is deleted follows the convention outlines in <https://github.com/osrg/ryu/blob/master/ryu/app/simple_switch_stp.py> under the delete\_flow method. Not the following code from this documentation:

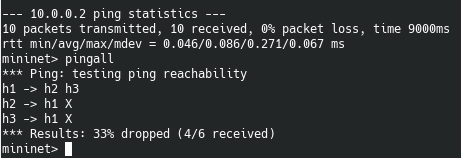
The match structure basically represents a wrapped package specifying the input port the packet came in on. This combined with the MAC of the destination expresses a basic flow of data through the switch. The main way the flow is removed, is actually not a removal but an edit. The flow(s) between host 2 and 3 are updated in such a way that caused the flow data to be reset. From here the packets enter the flow but they are simple dropped in transit.

It is important to not here the data path is class object describing the openflowswitch.

*Key Task 1: Testing*

To test packets were being blocked, pingall was used before the new code was added and after. The results are the following



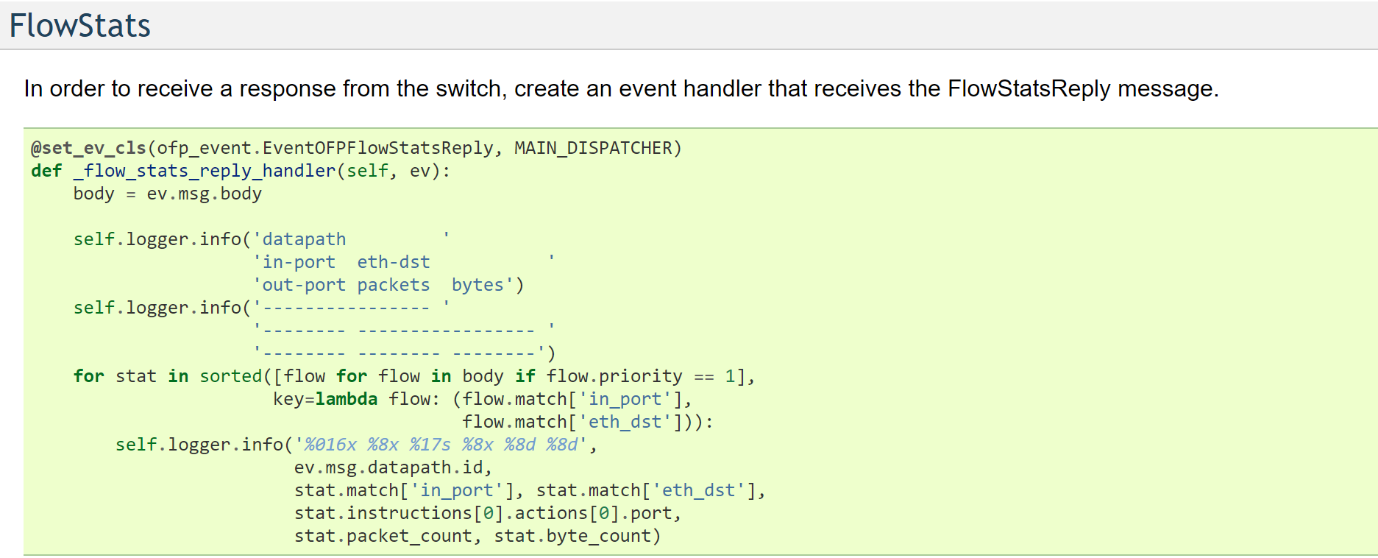


|  |
| --- |
|  |

*Key Task 2*

*Extend simple\_switch\_13.py to count all traffic going to and originating from host1*

We could implement a simple counter that counted the number of packets coming from host 1. This would involve looking into the packet data and doing a match for host1 eth. Instead we can use EventOFPFlowStatsReply in the SimpleMonitor13 class.



This will involve defining a method to make a OFFlowStatsequest to receive the data for the flow originating at host1 and the flow where host1 is the destination. The reason for doing it this way is than rather than just get the count, we can get the number of bytes passing between hosts in the flow.

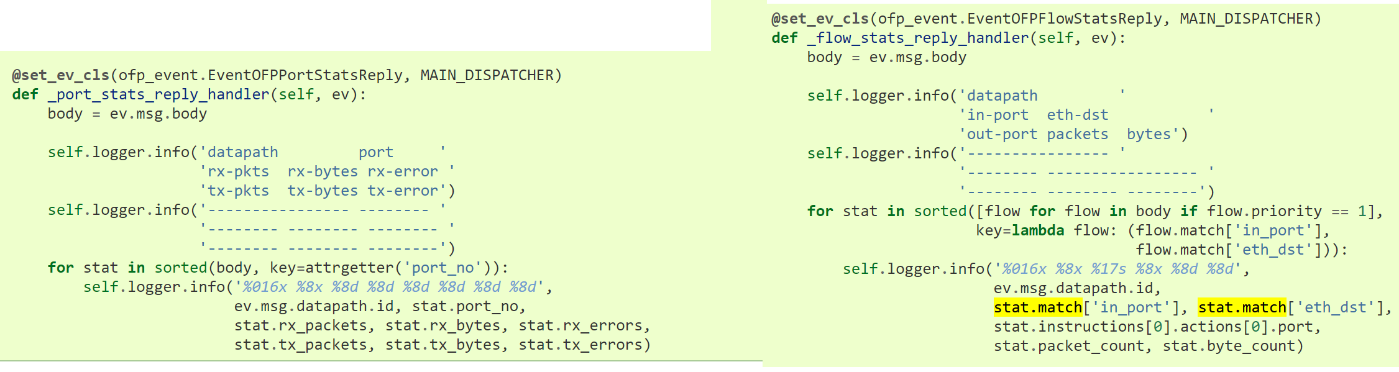
It is important here to discuss the @set\_ev\_cls class decorator, this decorator is used to get stats information from the Datapath. This method must also be modified to ensure we are only getting the stats where host1 is either the source or the destination.

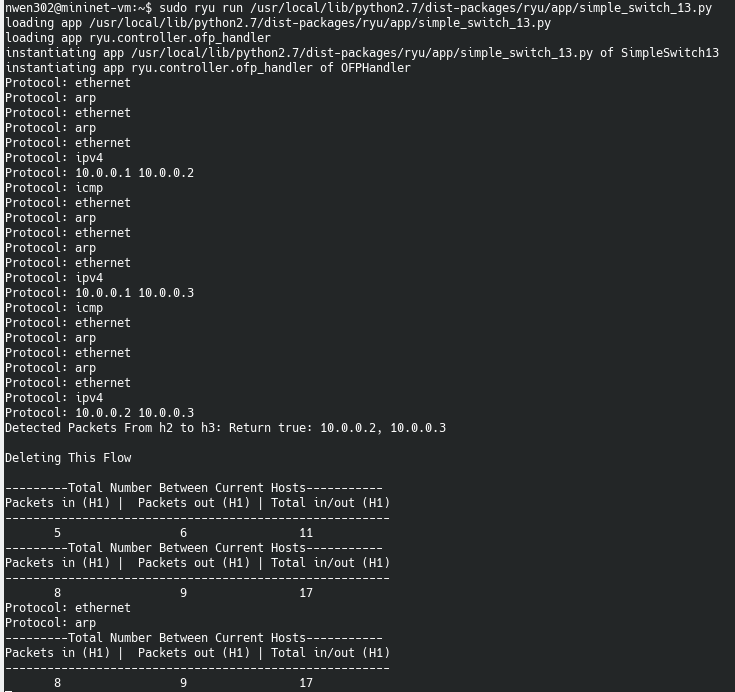
*Key Task 2: Testing*

The following screenshot shows the order in which protocol headers are displayed on the packet. This is done for 10.0.0.1 to 10.0.0.2, and then again for 10.0.0.1 to 10.0.0.3. Note and the end where it is from 10.0.0.2 to 10.0.0.3, this is caught by our code and returns true for a packet that must be blocked.

Below these values is the number of packets table. It uses the port\_stats\_reply\_handler from the EventOFPortStatsReply to grab the stats from the monitor we implemented. You will see the simple\_monitor\_switch has been integrated with the code to give accurate output.

When testing task 2 it was discovered that the Port Monitor was required over the FlowMonitor when it came to eiciting count data for the packets.



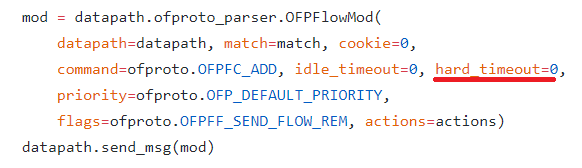


*Key Task 3*

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.

To achieve Task3 the data from PortMonitor had to be combined with the function that changes the action on the flows. Before Task3 this was only implemented for flows between host2 and host3. Currently the Code for task 3 has been commented out as it was not quite working correctly (see check\_if\_block).

The original train of though was that when a packet come in on packet\_in, it brings with its data that allows us to identify its flow. Once its flow has been identified, we need to retrieve its stats from the monitor. If the number of packets exceeds the value set in MAX\_COUNT then a call to alter its flow will be made. A flow has a flag called Hard\_Timeout, this flow needs to be updated to set the Hard\_Timeout to 24 hours.



**Reflection**

Overall the amount of code added was short and this made the general direction of the assignment easy. I also found the documentation relatively easy to ready, however the real problem was that certain important pieces of information were missing, or were not linked on that page. It was difficult to know where methods were being called from and asides from the specific file we were working on, program flow was difficult to follow.

One of the most difficult issues with this assignment was differentiating between flows and ports. This was especially frustrating when dealing with the second and third tasks, through eliciting the data with the simple\_switch\_monitor. Each flow had different value, and we could only receive data for the specific ev object we had in simple\_switch\_13.

To overcome these issues, it was realized we needed to be focusing on the monitor data from PortMonitor rather than the flow monitor. However, with the flow monitor we could show statistics based on flow. When a flow was setup packet\_in is completely ignored as the pathway can be taken through the flow object. This was difficult to see in practice as we only have access to switch file that dealt with streams that had no flow setup for them yet.

Overall Task1, Task2 and some of Task3 was accomplished.