Friday 7 November 2025

SDN – CA1

Question 1

Oisin Gibson - STUDENT

Contents

Introduction

This report documents the design, implementation, and testing of a SDN environment using Mininet and the Ryu and POX controllers.

Phase 1 focused on building a custom multi-department topology and testing controller connectivity.

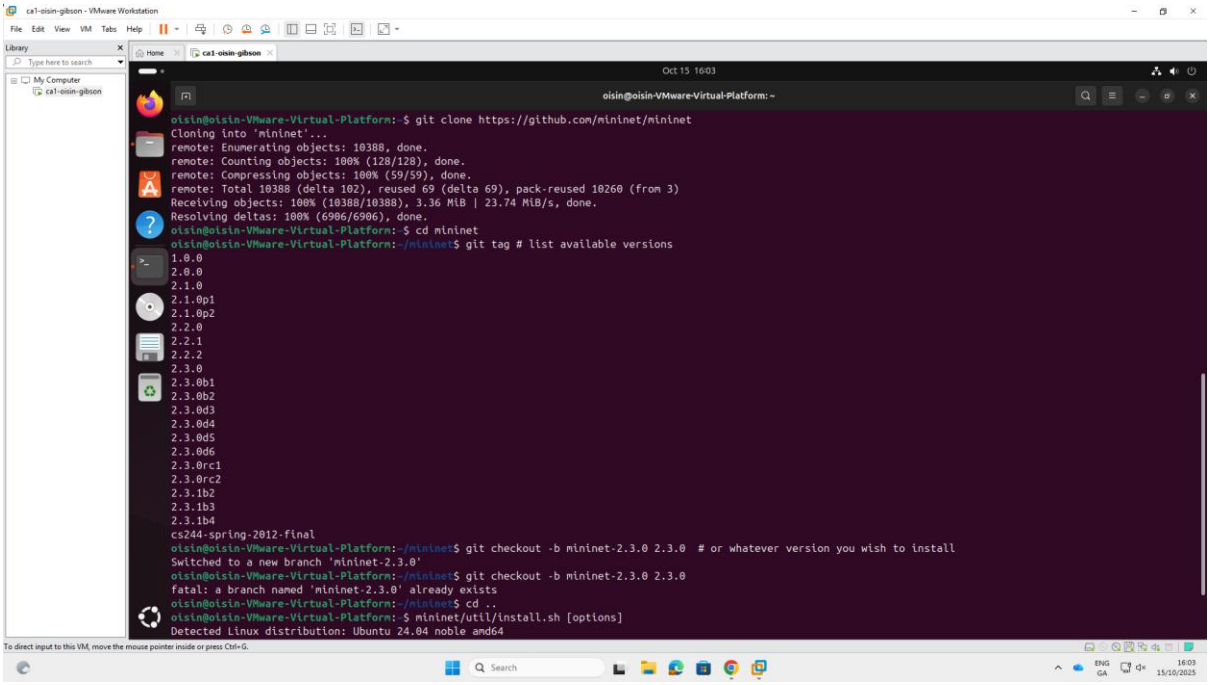
Phase 2 implemented security features including firewall rules, ACLs, and DDoS mitigation.

Task 1

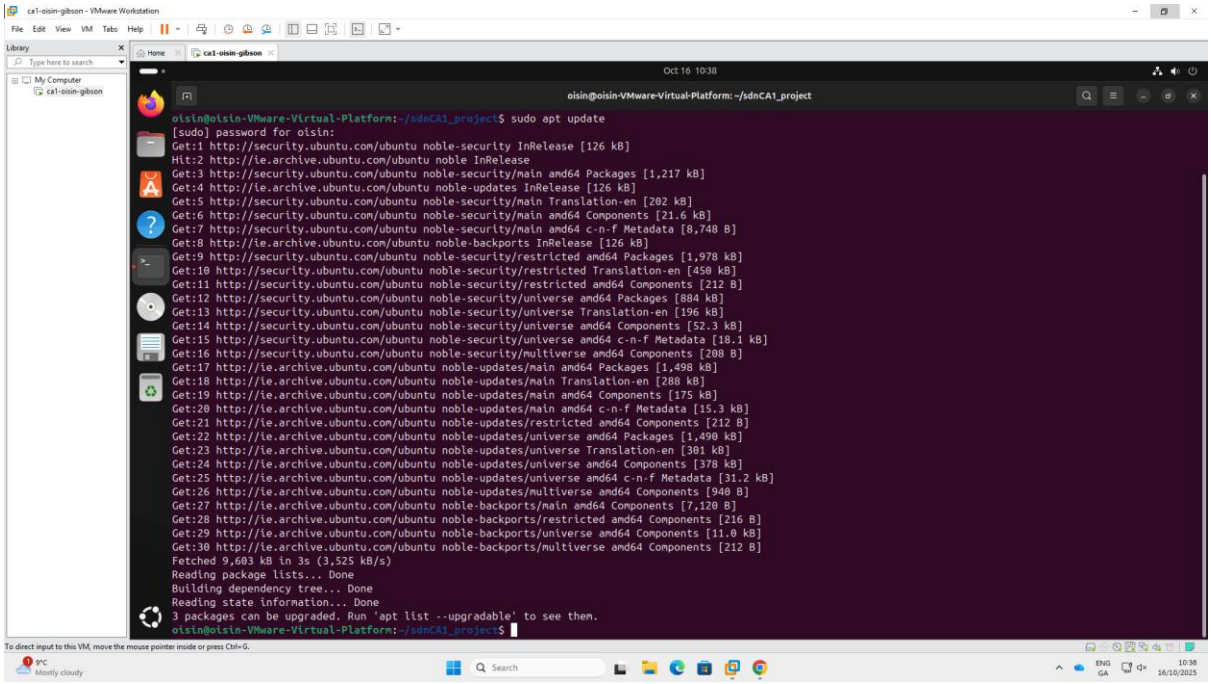
Both Ryu and POX were tested as external SDN controllers connected to the network. The aim of this phase is to serve as a foundation for demonstrating traffic segmentation and controller logic in SDN.

Implementation

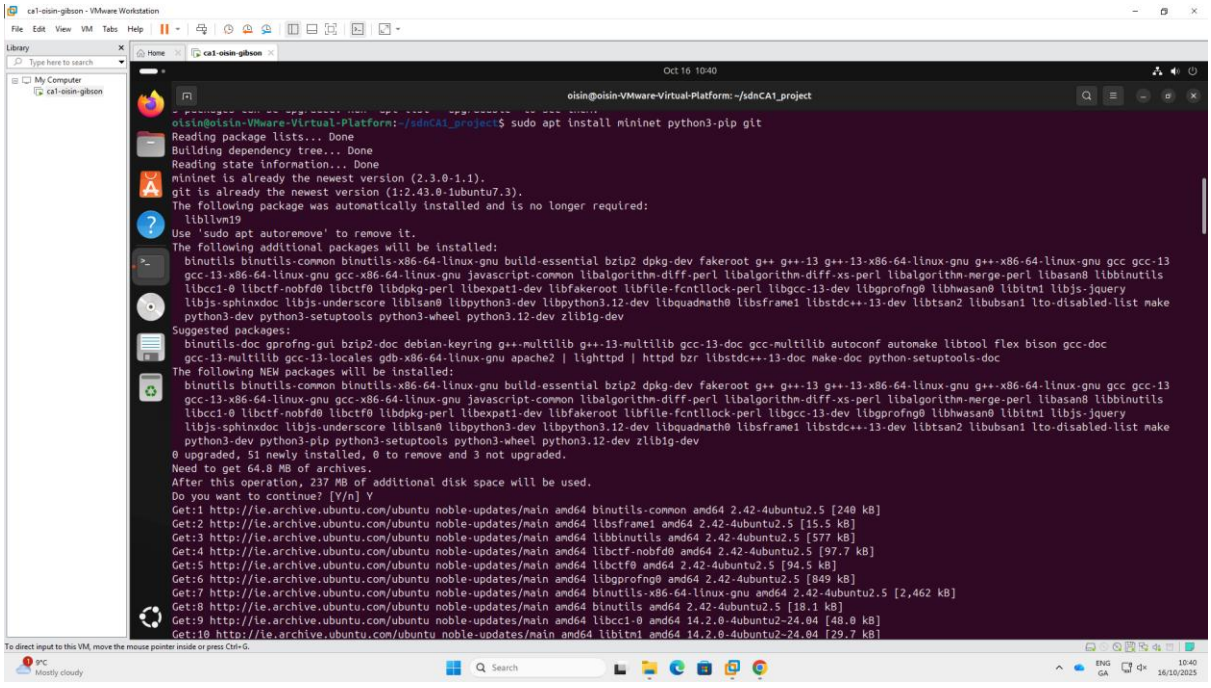
Step one was to install Mininet. This was done during a lab so will not be included in this report.

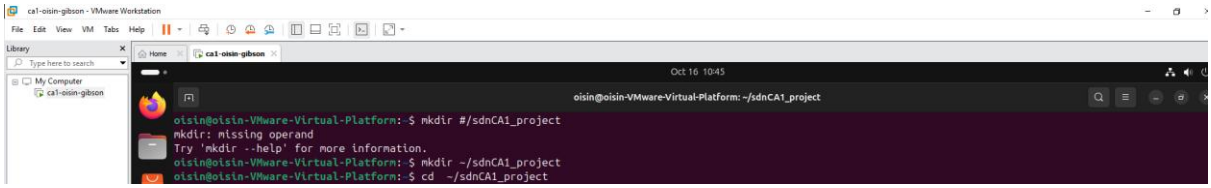


After Mininet was installed, updates were needed.

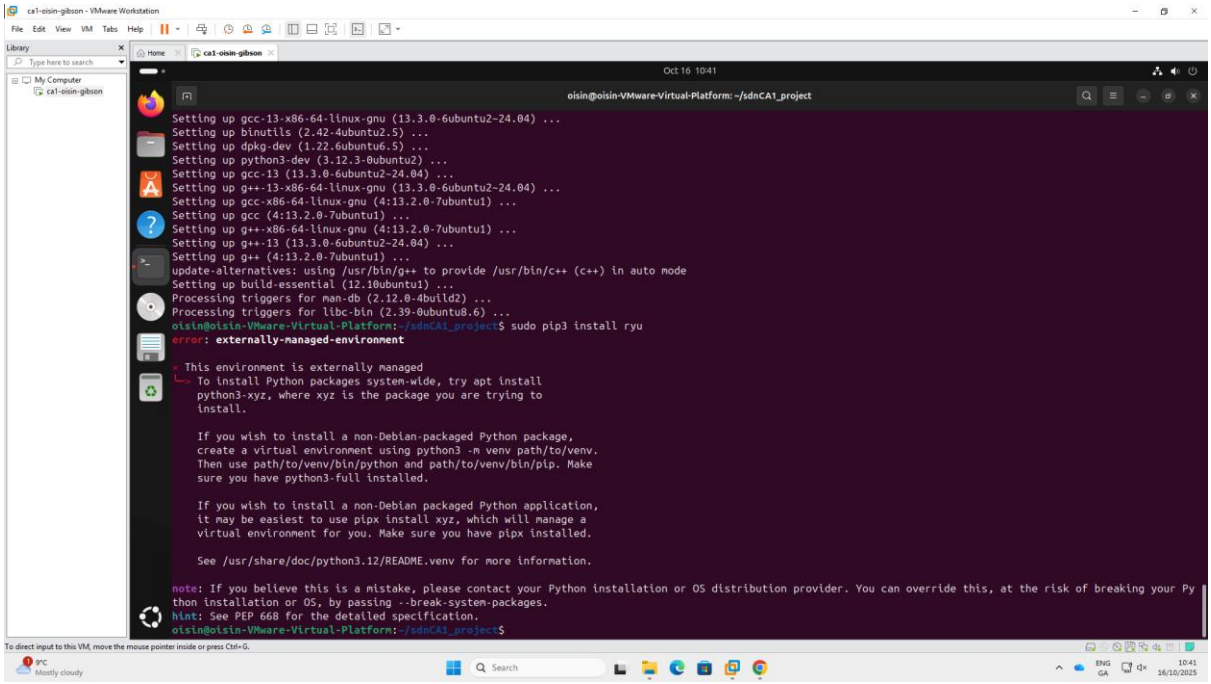


Following that packages had to be installed.

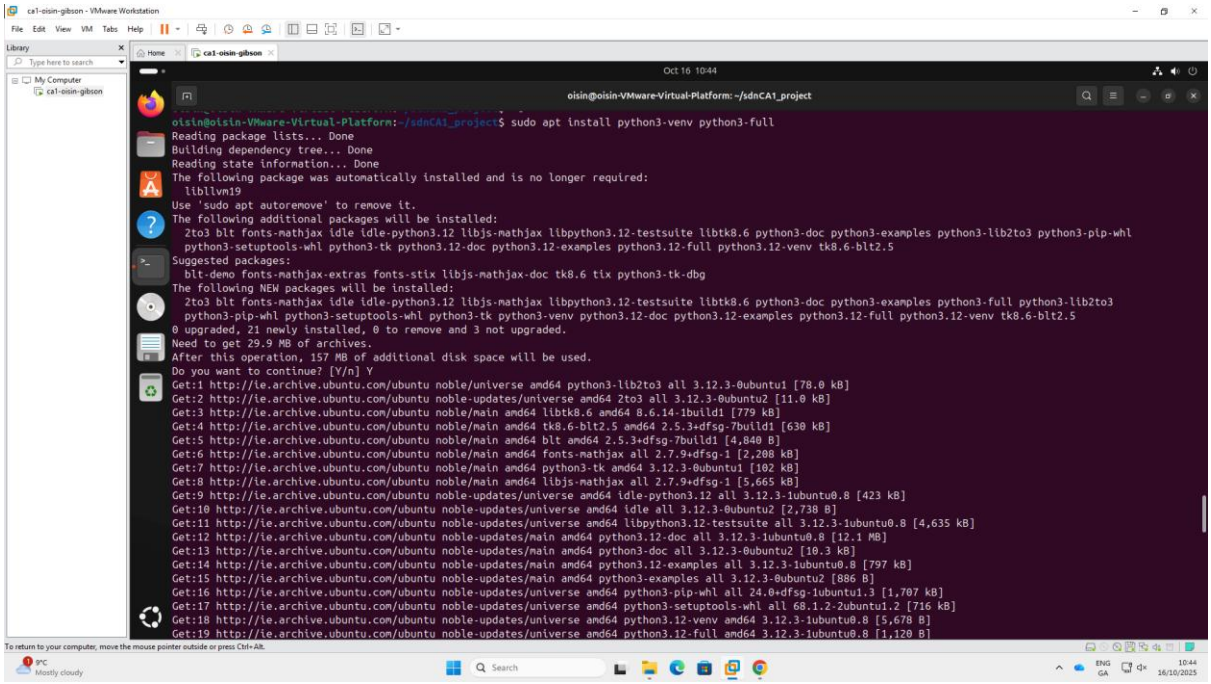




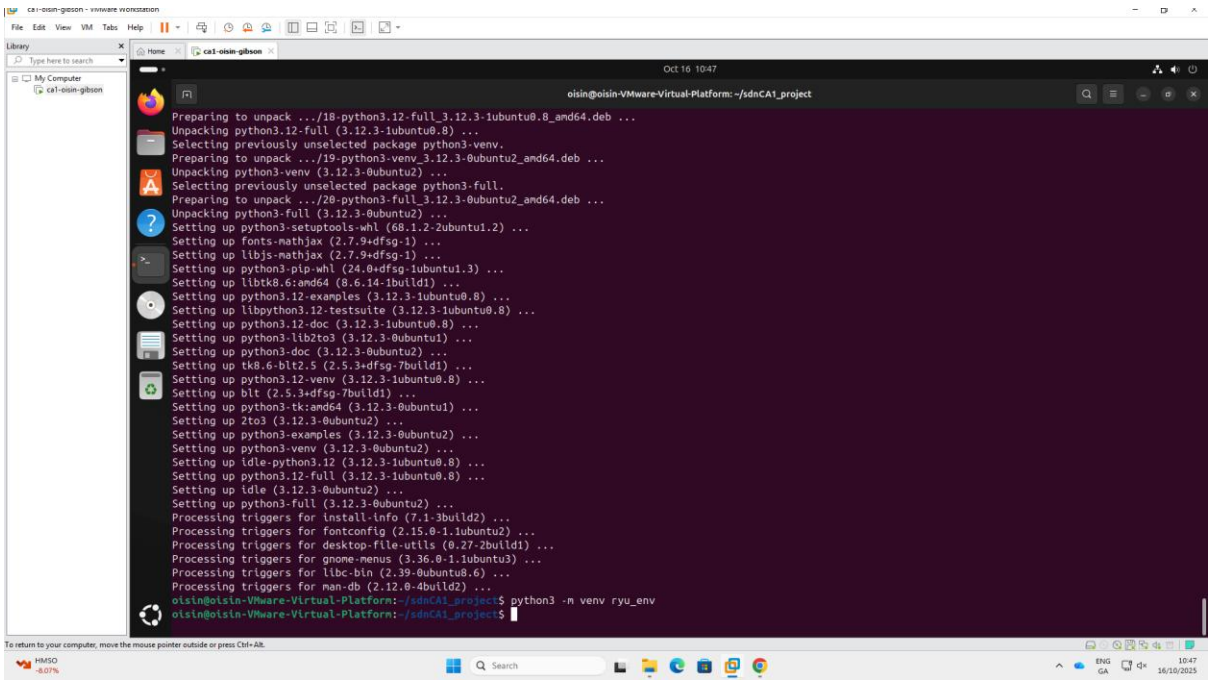
For the first attempt, Ryu was installed and used as the project controller. This was another reason to install python as Ryu is written in python.



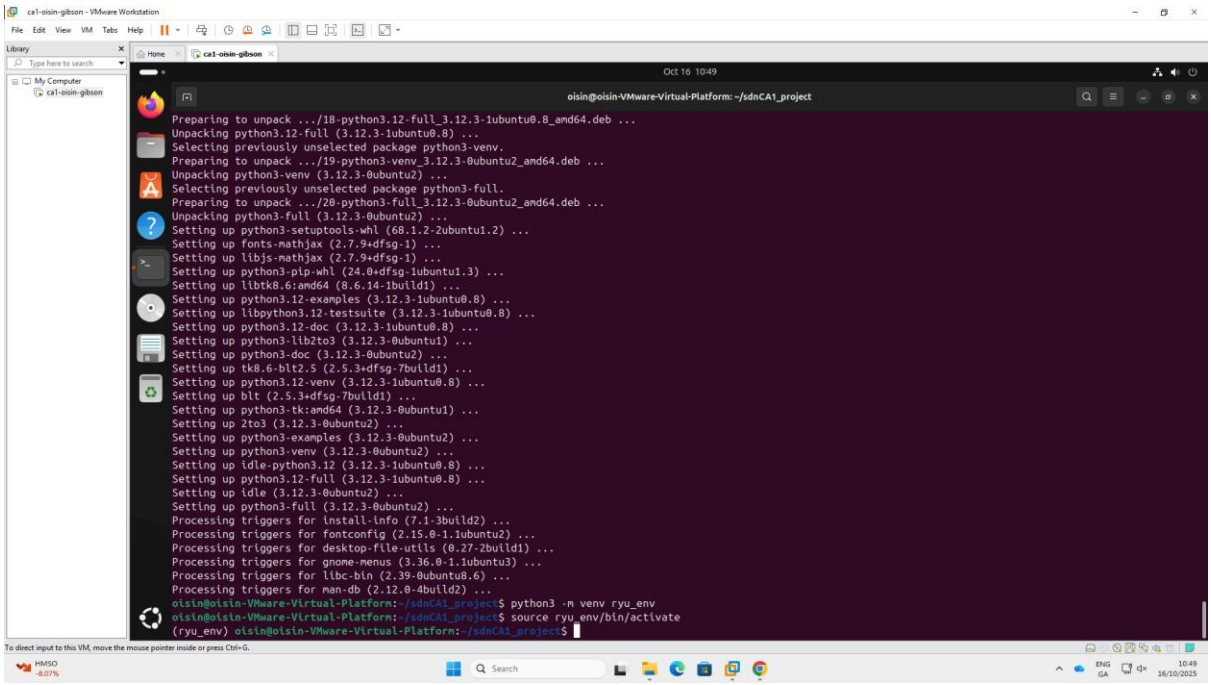
There were issues when installing Ryu. The error showed the command was blocked to avoid overwriting or conflicting with system packages. It should also have been installed in a user-space or virtual environment, not system wide.



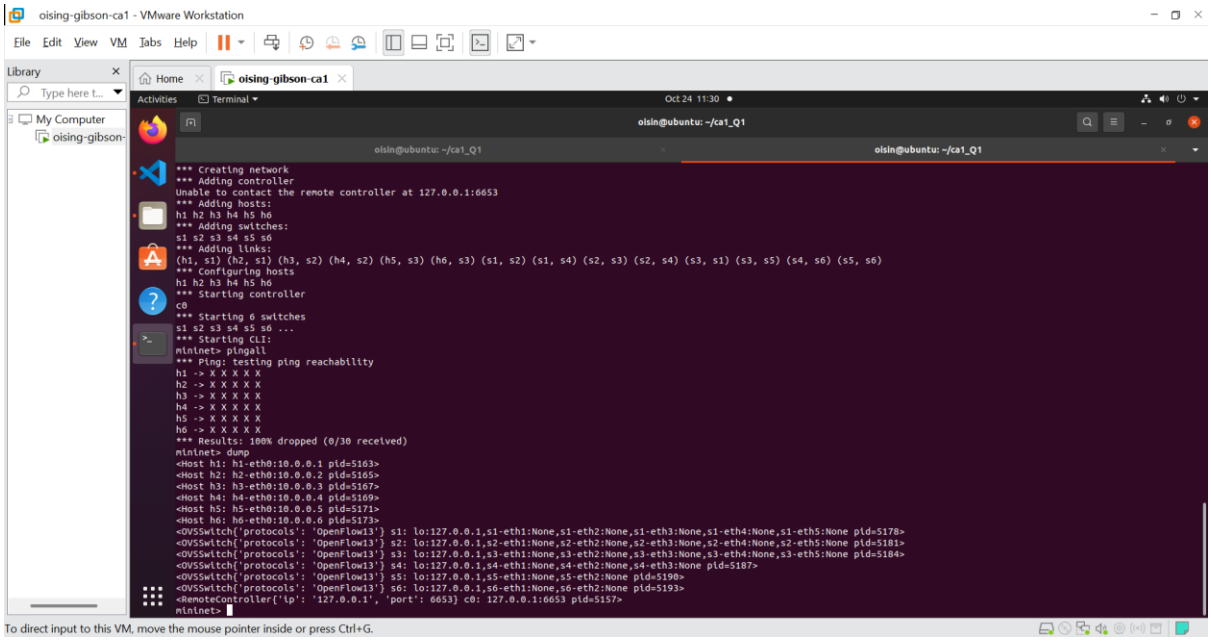
Using a fix allowed the creation of isolated environments. Once run, Ryu was able to be installed.



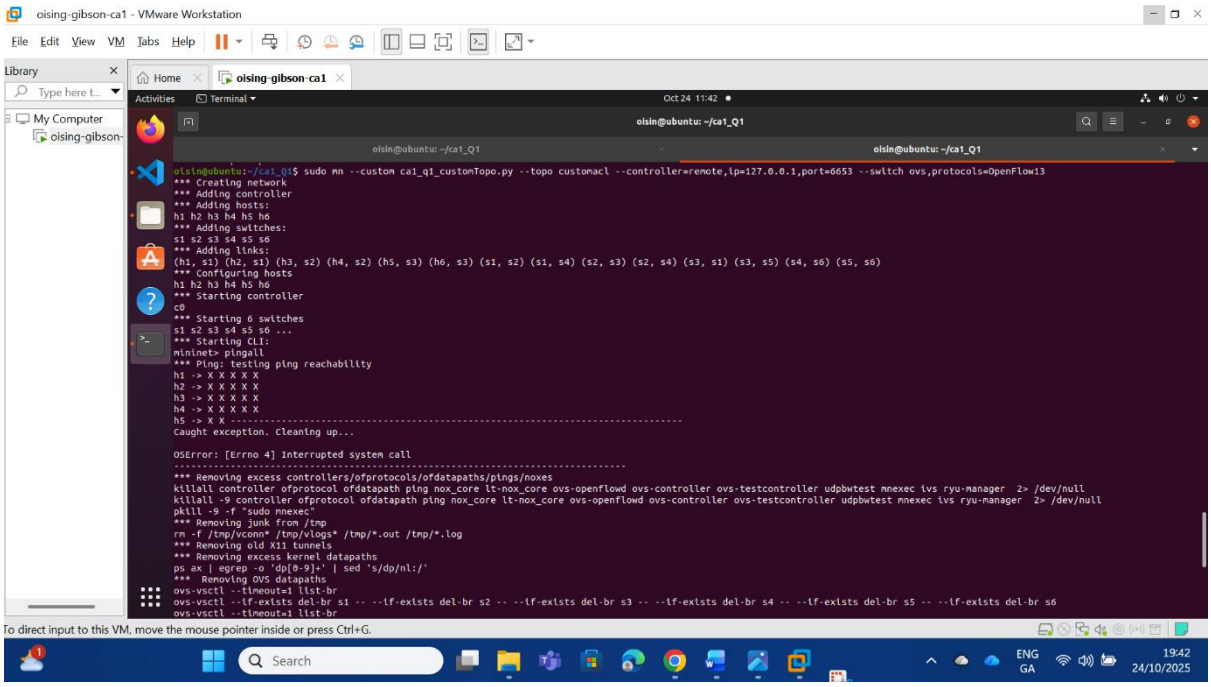
Once Ryu was created and installed, it was time to activate it.



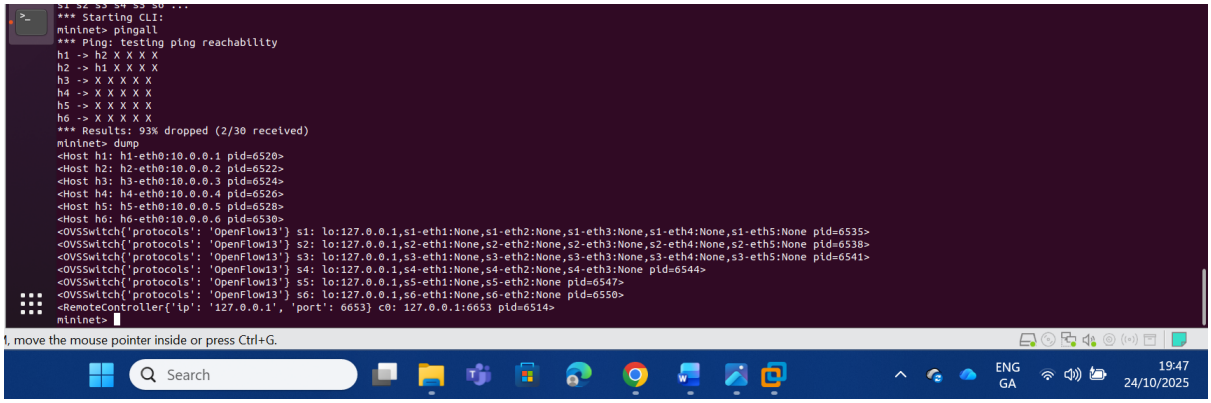
Ryu didn't end up working out.



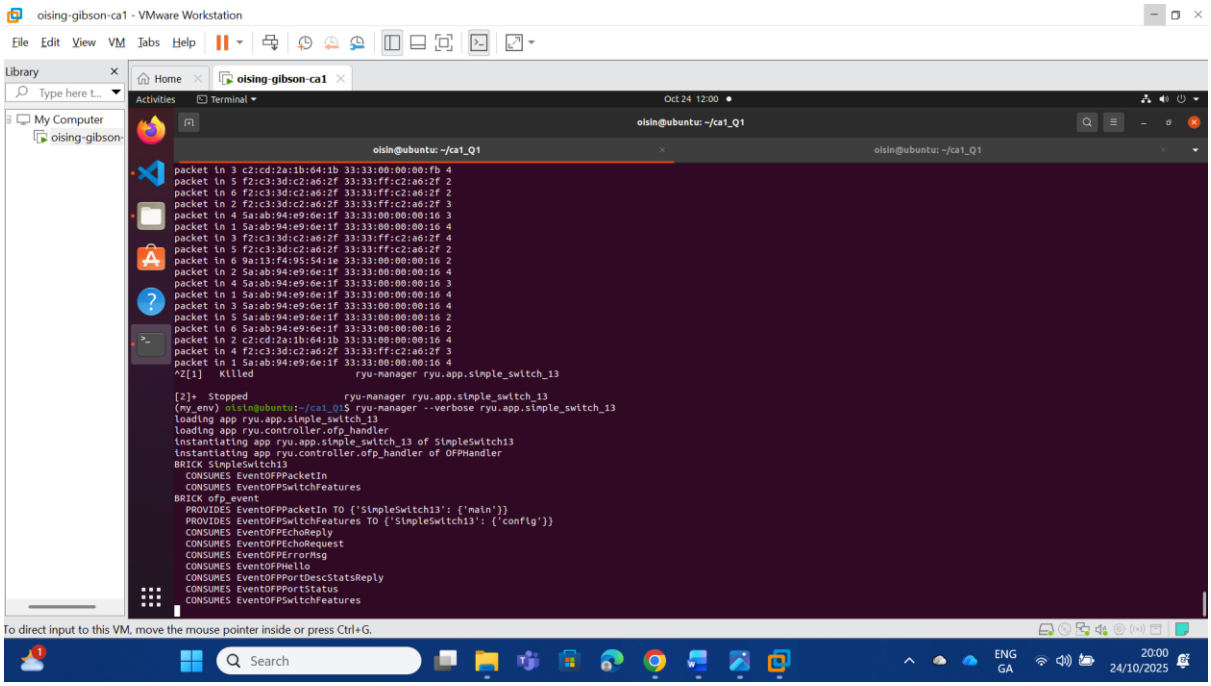
The Mininet was unable to connect to the remote controller. A solution was found. The following command was used to install missing dependencies.



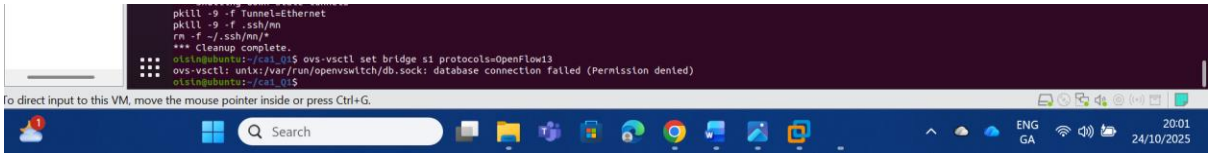
This resolved the issue. The pingall caught an exception and was interrupted resulting in the process being terminated. Trying the process again, the result improved slightly.



A 93% drop rate was achieved, with only h1 and h2 connecting. Despite both the Ryu controller and the Mininet activating as expected, Mininet couldn't connect to the controller.

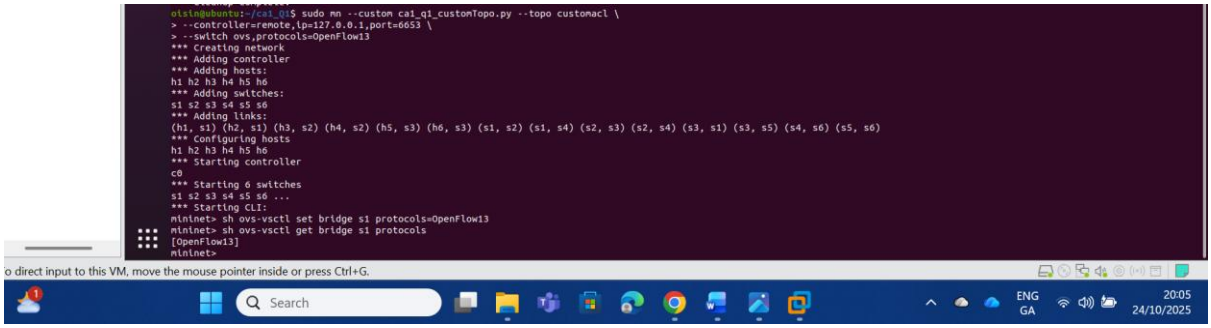


Research suggested that ryu-manager may be a better option. Even then the issues persisted. After an attempted fix with the flooding controller, the pingall still wouldn't work.

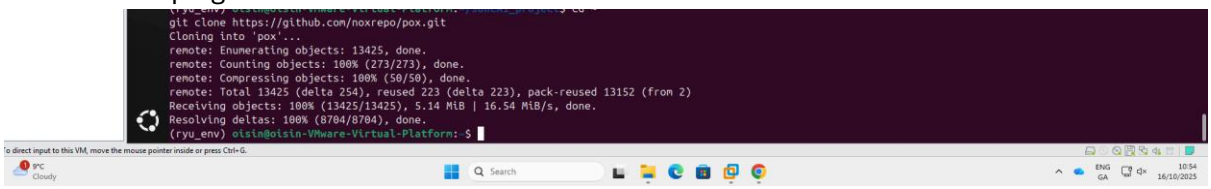


It was essential to ensure that all switches were operating with the correct OpenFlow version. This was achieved by explicitly setting the protocol during Mininet launch and verifying it within the CLI. Ensuring protocol compatibility was critical for successful communication between the

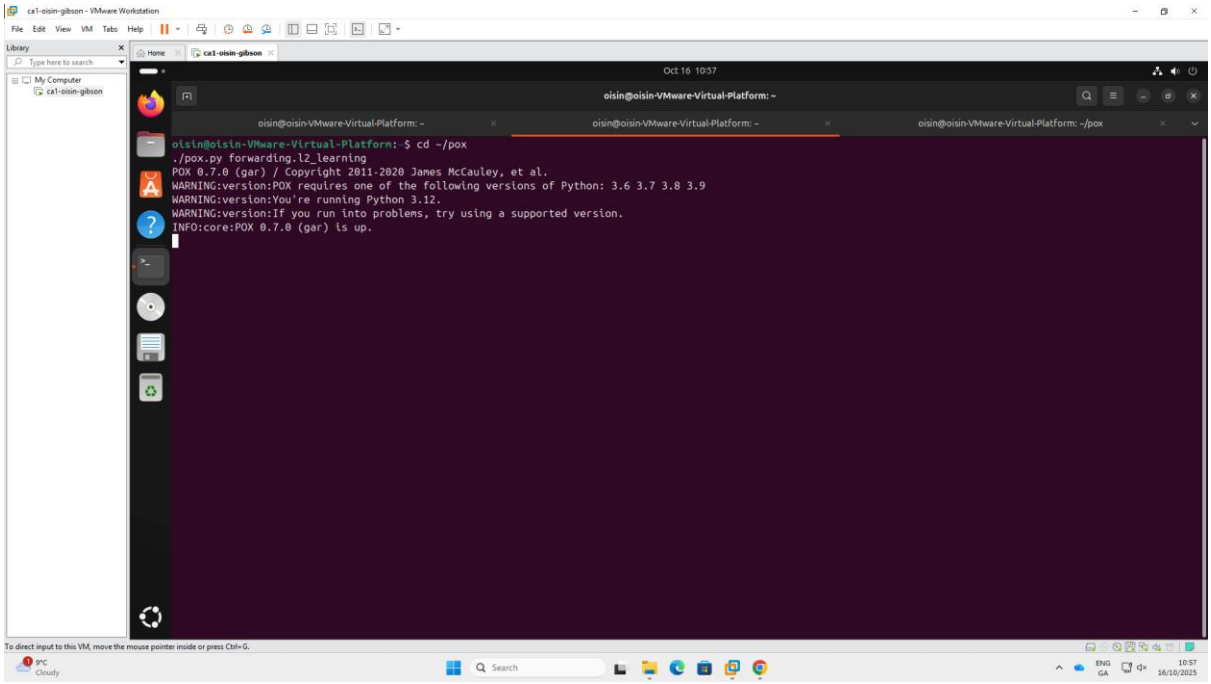
switches and the Ryu controller.



After this the pingall still would not work. The decision was made to move onto Pox.

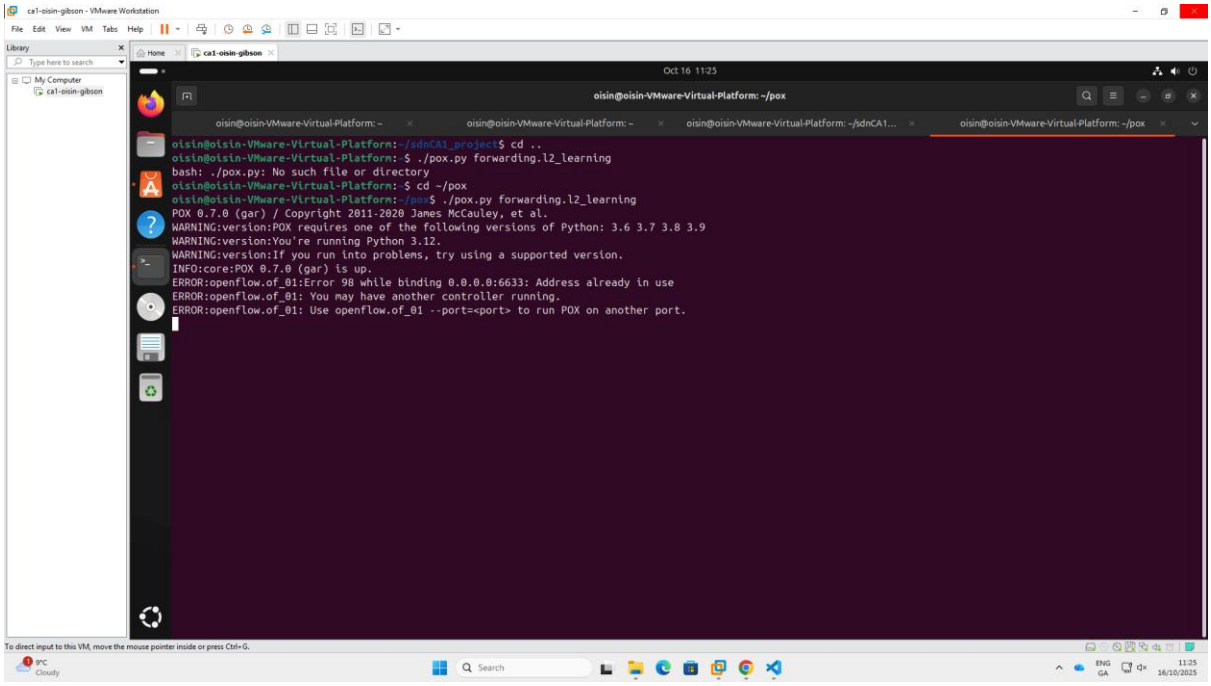


Once POX was successfully installed, it was necessary to activate the controller.

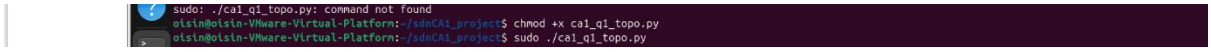


The same Python topology was reused for the POX controller trial. Using this topology proved to be a mistake as will later be explained. POX was successfully activated allowing it to run and

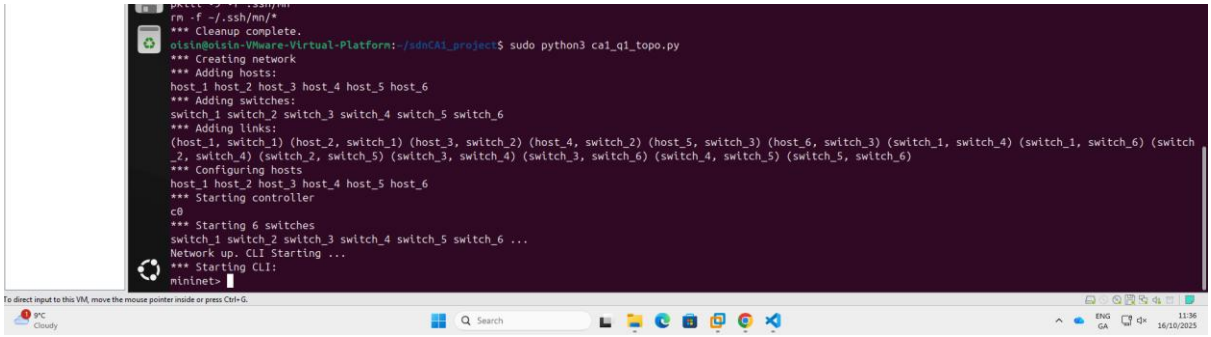
interact with the network as intended.

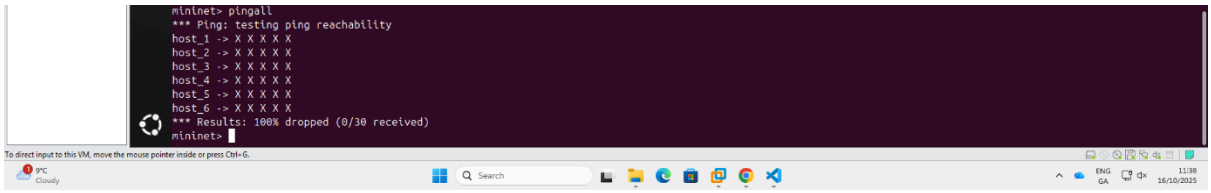


Once POX was activated and running, the python topology had to be given permission to run and be executed.

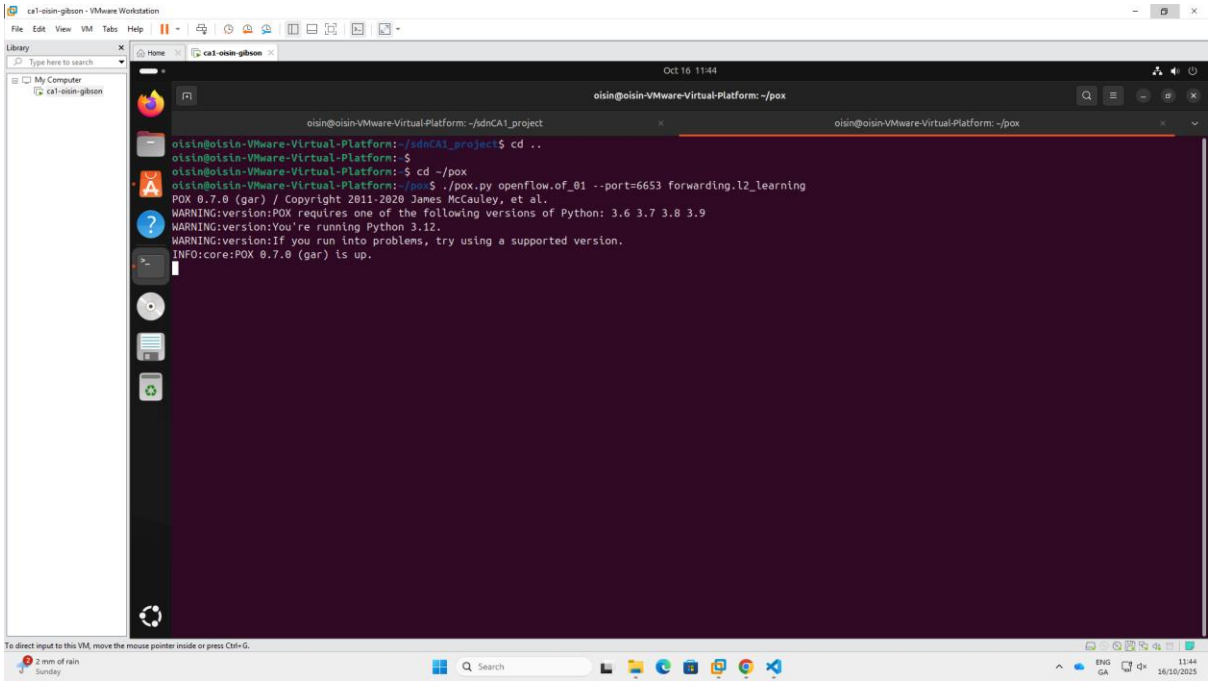


The topology was then activated. The same persistent issue of the pingall appeared again.

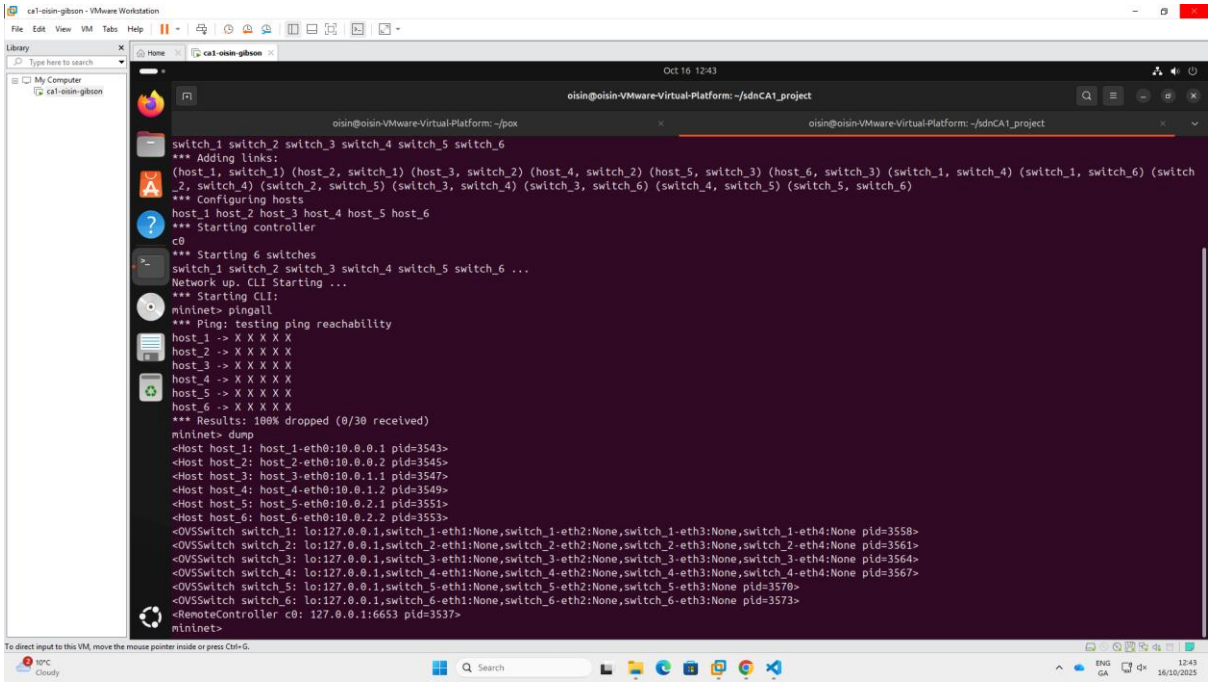




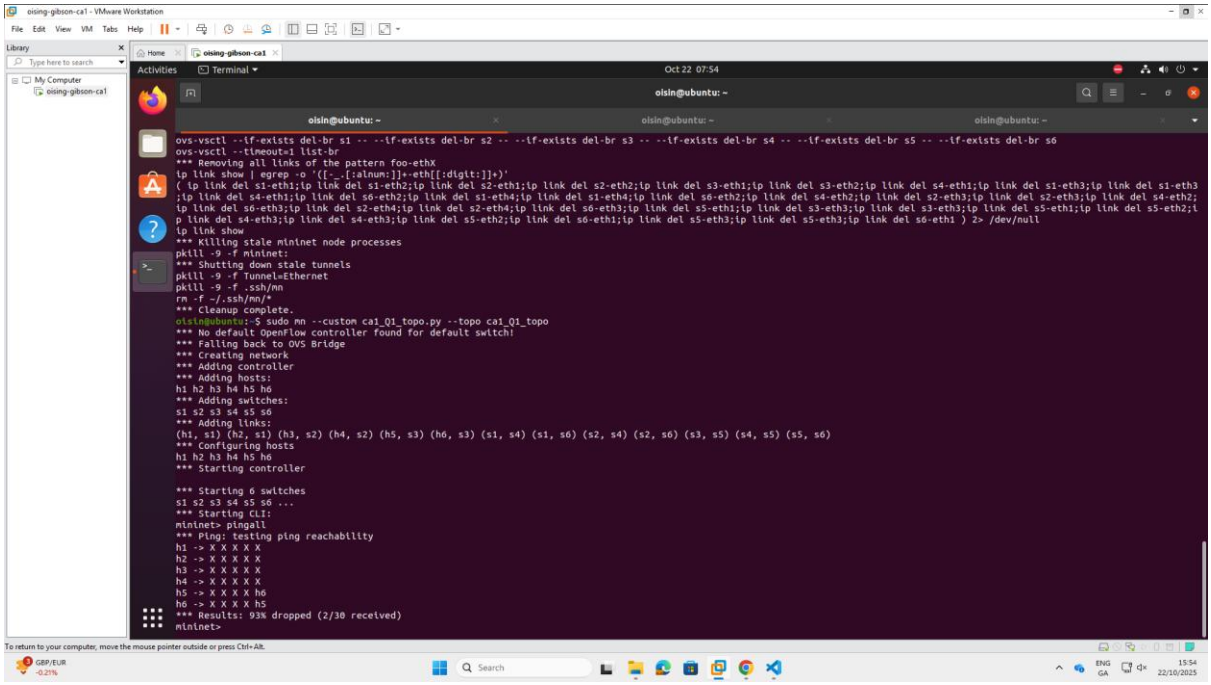
There may have been an issue with the POX itself being used, so a different version was installed and used.



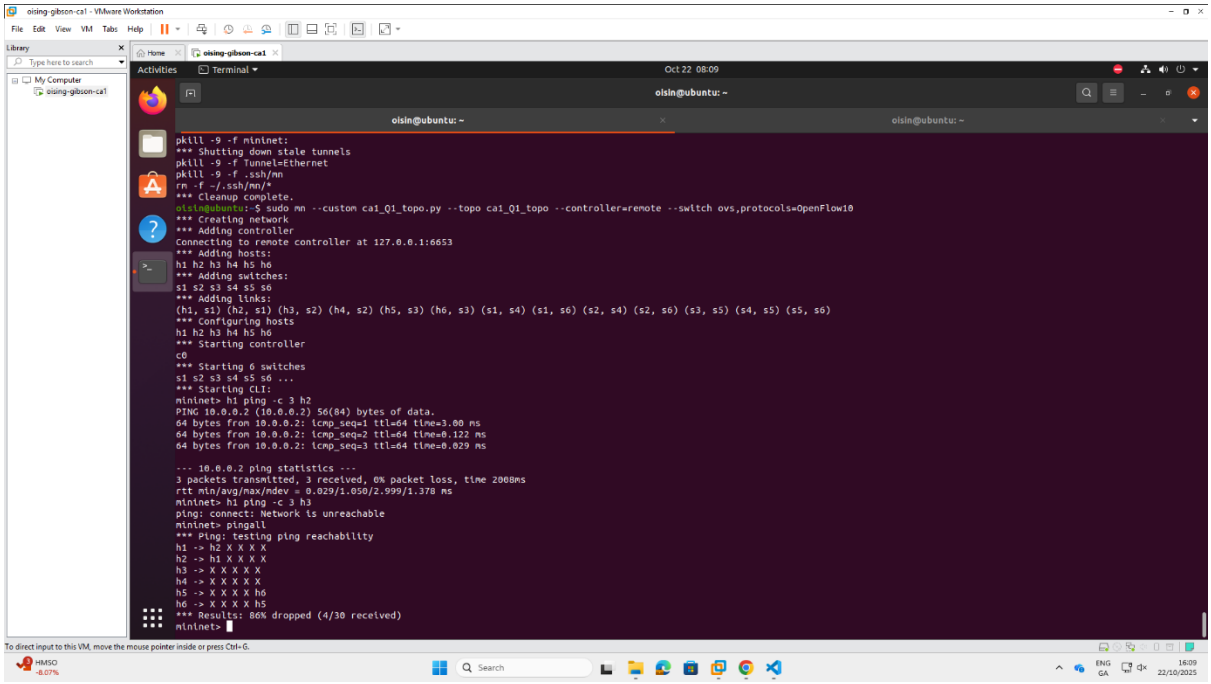
However, this proved to not be the issue and the pingall still would not work. There were no visible issues with the dump table.



This led to the idea of reworking the python topology. After some reworking, a 93% drop rate was achieved.



As shown in the figure below there was no issue with getting a 0% packet drop when using the h1 ping -c h2. This proved to be a perplexing issue as this would work but the pingall wouldn't work. This may point to a controller responsiveness or ARP issues. This led to another reworking of the python topology. Which proved to be a good course of action as the drop rate fell to 86%.



Possible Issues

Some research shows that it may be reactive controllers such as simple_switch_13 install flow rules only after receiving PacketIn events. This would then overwhelm the controller and delay

the flow instructions. It is also possible that the controller lacked explicit ARP flooding logic, meaning it couldn't forward in the OpenFlow networks.

Task 2

To start, the foundations needed to be setup. The controller installs a table-miss flow with priority 0, instructing the switch to forward unmatched packets to controller via packet-in messages. This makes sure that all the new traffic is visible to the controller before forwarding rules are established.

Firewall

The firewall will be built around a FIREWALL_BLOCKS list which will specify the forbidden flows. Each of the entries in the list defines matches on multiple OpenFlow fields, including eth_type, ipv4_src, ipv4_dst and ip_proto. The controller will then translate these definitions into proactive drop flows with a high priority of value 300. If any packets match these conditions are silently discarded. This enforces network segmentation and traffic control at the data plane level.

If a packet matches no proactive rule, it will then follow the standard forwarding path or it will be subjected to the ACL and DDoS logic. This proactive approach will minimise the load placed on the controller by preventing repeated packet-in events for known disallowed flows.

ACLs

ACLs were used in this assignment via a data structure known as ACL_ALLOW. This defines subnet pairs that are authorized to communicate. A default deny-all rule ensures that unless and IP pair is explicitly listed, communications will be blocked.

For this assignment, traffic between 10.0.1.0/24 and 10.0.2.0/24 was explicitly denied using rule with priority 250. This will prevent inter-subnet communication while also allowing other local flows to proceed. The ACL rules were installed proactively which ensures that immediate enforcement at the switch level without controller involvement once it is applied.

DDoS Detection and Mitigation

This mechanism is reactive and time sensitive. It will track packet-in events by recording the timestamps of packets per source IP in a deque data structure. To represent the observation period, a defined DDOS_WINDOW will be used. DDOS_PKT_THRESHOLD will specify the maximum allowed packet rate before triggering mitigation.

If a host exceeds the threshold, the controller will install a temporary drop flow targeting that ipv4_src with a very high priority of 400. This rule will block all the subsequent packets from the offender for a configurable period, DDOS_BLOCK_TIME. After this, the drop rule expires, allowing the normal traffic to flow again.

The above design effectively filters any abnormal traffic bursts at the switch level. This reduces both overhead for the controller and data-plane congestion during attacks.

Switch Learning and Forwarding Behaviour

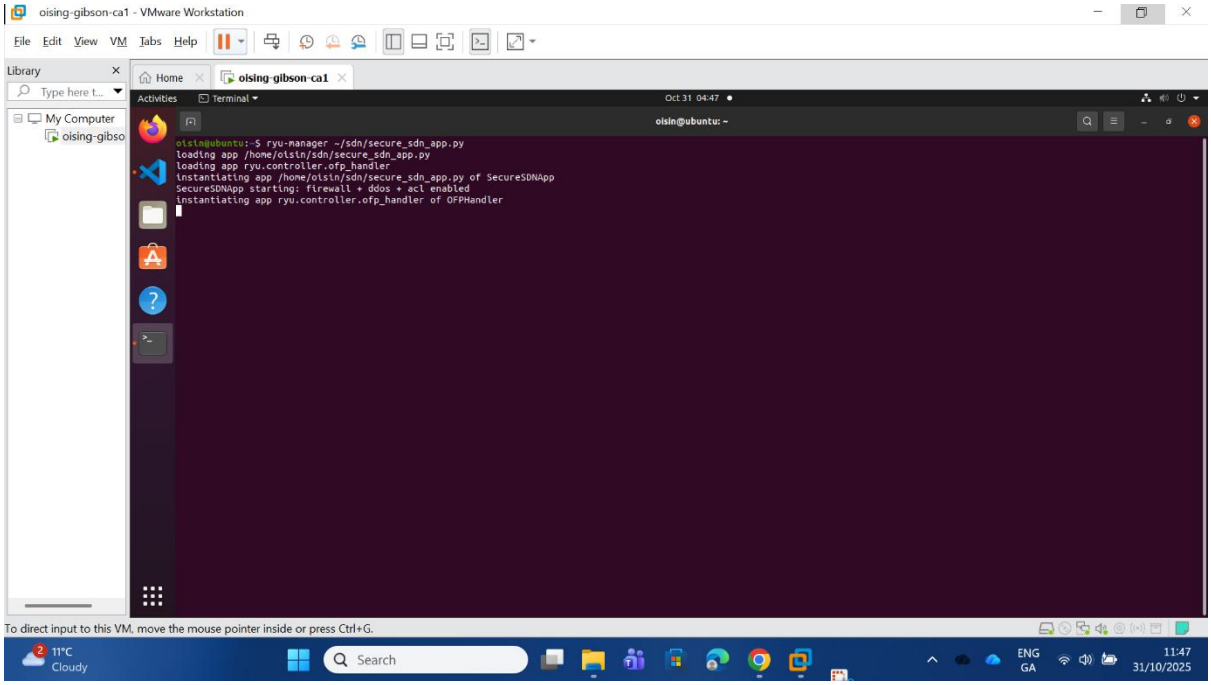
The controller will also implement standard MAC learning functionality. When the controller receives a packet-in message, it will record the source MAC and associated port. Once both source and destination mappings are known, a forwarding flow with a priority of 100 is then

installed to handle subsequent packets locally on the switch. This approach to the assignment aims to balance reactive learning and proactive security.

Implementation

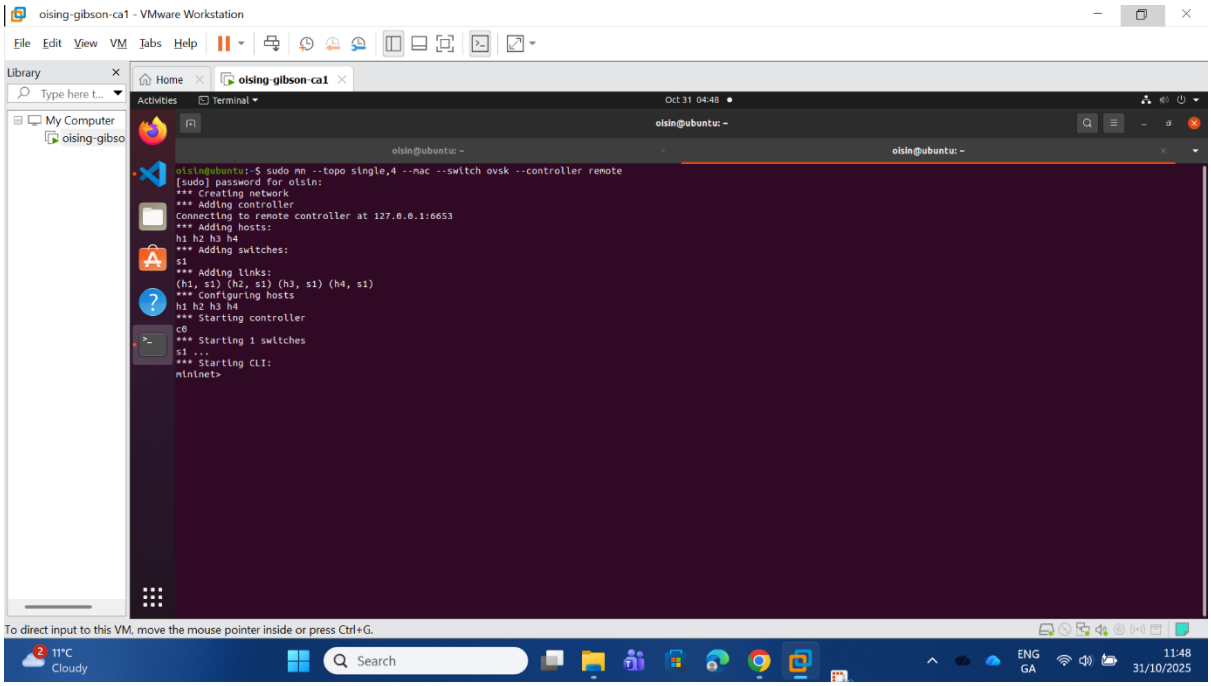
1. **Starting Ryu Controller**

   The Ryu Manager was launched. Successful startup was verified through log messages.



Initializing Mininet

A clean environment was created. After this Mininet was launched using a command which specified the single switch. The Ryu controller must be active before the Mininet is launched. This ensures the switch connects correctly via OpenFlow.



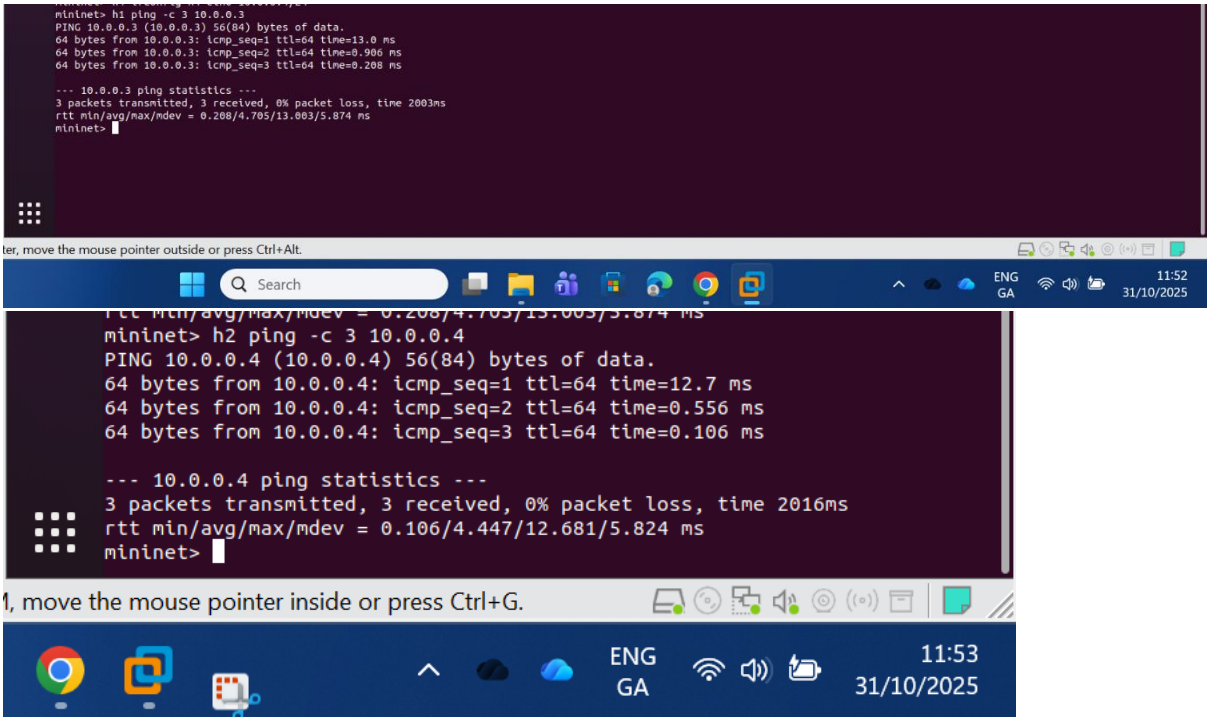
Configuring Host Ips

Each of the hosts were assigned an IP address within the 10.0.0.0/24 subnet. This simplified connectivity tests and policy enforcement.

```
mininet> h1 ifconfig h1-eth0 10.0.0.1/24
mininet> h2 ifconfig h2-eth0 10.0.0.2/24
mininet> h3 ifconfig h3-eth0 10.0.0.3/24
mininet> h4 ifconfig h4-eth0 10.0.0.4/24
```

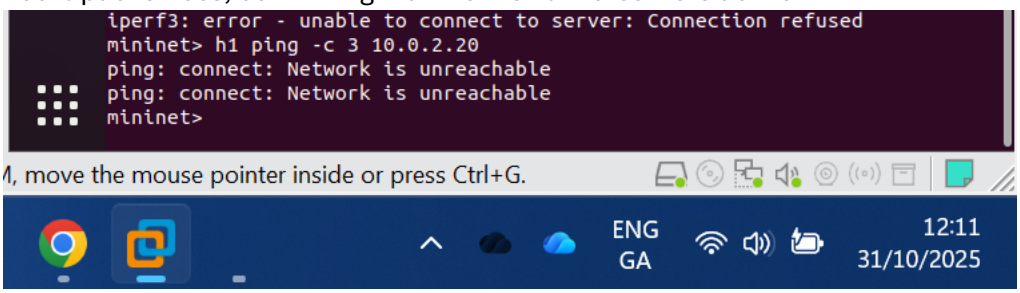
Verifying Connectivity

Connectivity between all the hosts was confirmed by executing ICMP ping. This confirmed proper layer 2 and 3 functionality before applying security policies.



Firewall Testing

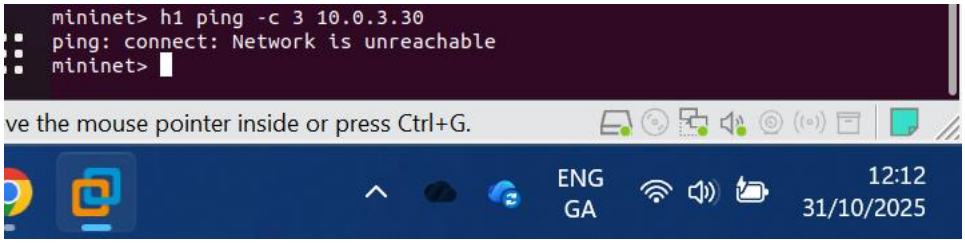
Specific pings between h1 and h2 were configured to be blocked. The test returned 100% packet loss, confirming that the firewall rules were active.



ACL Validation

Traffic between subnets 10.0.1.0 and 10.0.2.0 was denied by default. Intra-subnet communication remained successful. Flows dump confirmed that the deny rule was

present and functioning correctly.



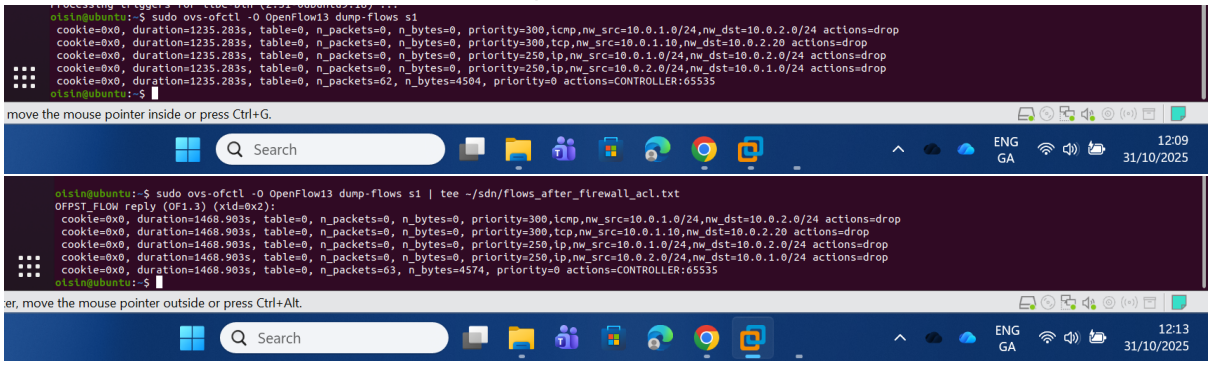
DDoS Detection

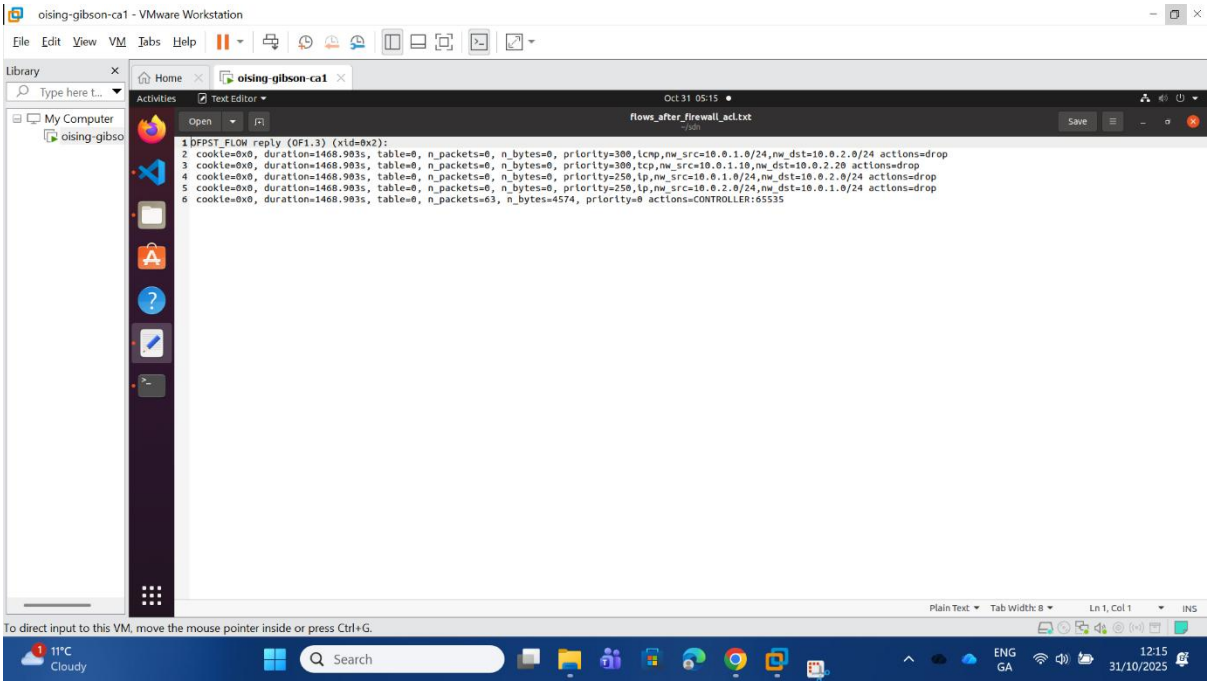
A UDP-based server was started. It was started on one host to simulate normal traffic service.

Once the packet-in rate exceeded the threshold set, the controller dynamically installed a drop rule. This would then target the attackers IP. This rule effectively isolated the offending host. The Ryu log showed evidence of detection and mitigation events showing the offending IP being blocked.

Flow Capture

To validate the results, flow tables were captured.

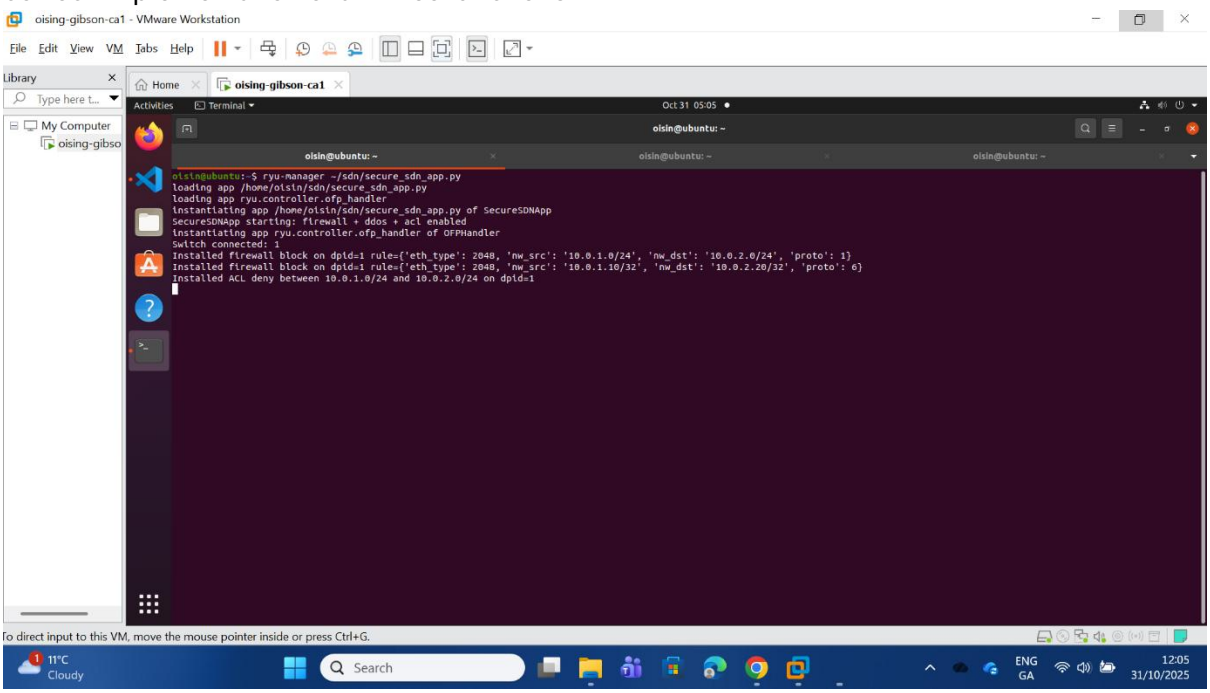




It confirmed that each of the feature's mitigation flows with their respective priorities and timeouts.

Final State

The logs for the Ryu Manger at the conclusion of all tests displayed the sequence of flow installations. They also show rule triggers and event handling, all of which confirms correct implementation of all three functions.



Discussion

Firewall rules effectively blocked traffic at the switch, reducing controller load.

ACLs enforced subnet isolation with proactive rules, improving performance.

The **DDoS system** showcased Ryu's event-driven capabilities. It used efficient timestamp tracking and temporary blocking to contain abnormal traffic. MAC learning enabled dynamic forwarding with minimal overhead.

Conclusion

Despite persistent drop rates (86–93%) in Phase 1, likely due to reactive controller behaviour and ARP handling issues, the topology was refined for better performance.

Phase 2 successfully implemented and validated firewall, ACL, and DDoS features using Ryu. Flow dumps and logs confirmed correct functionality and rule enforcement.

Appendix

Word Count (Report): 1498

Font: Aptos (Body)

Size: 11