


Statement and Confirmation of Own Work

***A signed copy of this form must be submitted with every assignment.
If the statement is missing your work may not be marked.***

Student Declaration

I confirm the following details:

Student Name:	Johnson Kenisha Corera
Student ID Number:	c23020001@cicra.edu.lk
Qualification:	Bachelor in Cyber Security
Unit:	SIT325 Advanced Network Security
Centre:	CICRA Campus
Word Count:	637
<p>I have read and understood both <i>Deakin Academic Misconduct Policy</i> and the <i>Referencing and Bibliographies</i> document. To the best of my knowledge my work has been accurately referenced and all sources cited correctly.</p> <p>I confirm that I have not exceeded the stipulated word limit by more than 10%.</p> <p>I confirm that this is my own work and that I have not colluded or plagiarized any part of it.</p>	
Candidate Signature:	J. 
Date:	03/12/2024

Task 3.1P

Table of contents

• Introduction.....	05
• Part A.....	06
• Part B.....	07
• Part C.....	12
• Part D	
✓ Question 01.....	13
✓ Question 02.....	14
✓ Question 03.....	15
• Conclusion.....	16
• References.....	17

Table of Figures

Figure 1.....	06
Figure 2.....	06
Figure 3.....	07
Figure 4.....	07
Figure 5.....	08
Figure 6.....	08
Figure 7.....	09
Figure 8.....	09
Figure 9.....	10
Figure 10.....	10
Figure 11.....	11
Figure 12.....	12
Figure 13.....	12

Table of Acronyms

Acronym	Full Form
MAC	Media Access Control
IP	Internet Protocol
TCAM	Ternary Content Addressable Memory
API	Application programming Interface
SDN	Software Defined Network
RPC	Remote Procedure Call
ONOS	Open Network Operating System
gRPC	googles Remote Procedure Call

Introduction

I began with a basic Python-based configuration of switches and hosts to experiment with constructing network topologies, with Mininet, for this assignment. To be able to identify the topology I assigned and drew the IP and MAC addresses on each of the devices. I also conducted some network tests to ensure every node in the networks was active or connected to the other one. The areas where the usage of TCAM is limited in SDN systems, the possible future for Open SDN in terms of scaling, and different types of network management API were the subject of my next theoretical discourse. This methodical approach enabled me to apply pragmatic knowledge to practical experience at the same time focusing on main concepts of SDN.

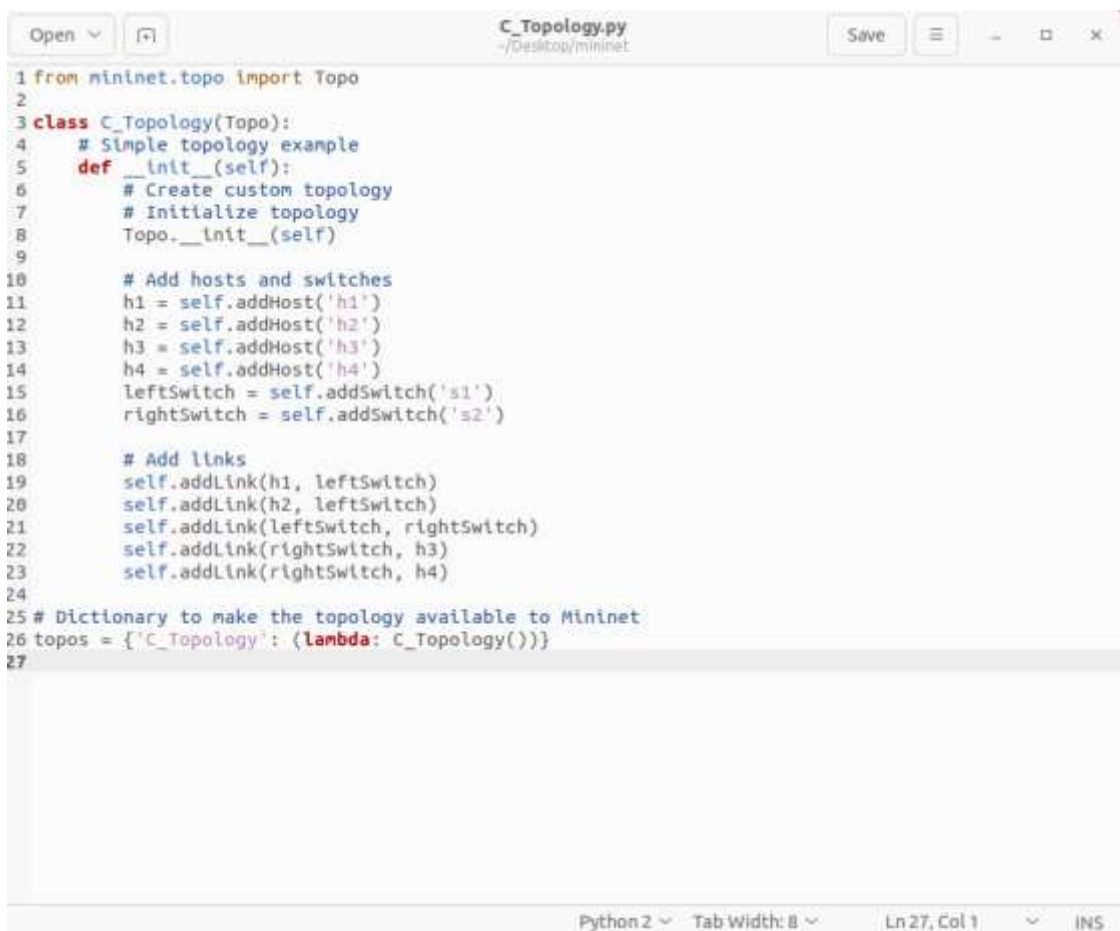
Part A

- The first step which was followed in the process was to develop a file in the Ubuntu virtual machine named C_Topology.py.

```
kenisha@kenisha-virtual-machine:~/Desktop/mininet$ touch C_Topology.py
kenisha@kenisha-virtual-machine:~/Desktop/mininet$ gedit C_Topology.py
^Z
[1]+  Stopped                  gedit C_Topology.py
```

Figure 1: Creating C_Topology.py

- The Python code provided in the job was then opened using the gedit editor and then copied then pasted into a notepad before the program was saved.



```
1 from mininet.topo import Topo
2
3 class C_Topology(Topo):
4     # Simple topology example
5     def __init__(self):
6         # Create custom topology
7         # Initialize topology
8         Topo.__init__(self)
9
10        # Add hosts and switches
11        h1 = self.addHost('h1')
12        h2 = self.addHost('h2')
13        h3 = self.addHost('h3')
14        h4 = self.addHost('h4')
15        leftSwitch = self.addSwitch('s1')
16        rightSwitch = self.addSwitch('s2')
17
18        # Add links
19        self.addLink(h1, leftSwitch)
20        self.addLink(h2, leftSwitch)
21        self.addLink(leftSwitch, rightSwitch)
22        self.addLink(rightSwitch, h3)
23        self.addLink(rightSwitch, h4)
24
25 # Dictionary to make the topology available to Mininet
26 topos = {'C_Topology': (lambda: C_Topology())}
27
```

Figure 2: Type the code into the C_Topology file

Part B

- I then run the code and created the needed network topology as requested. Top two were swapped and four happy were changed.

```

kenisha@kenisha-virtual-machine: ~/Desktop/mininet$ sudo mn --custom C_Topology.py --topo C_Topology
[sudo] password for kenisha:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s1) (s1, s2) (s2, h3) (s2, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:

```

Figure 3: Creating Topology

- After that, I checked the ip address, mac address and interworking interfaces of all the host and switch.

```

mininet> h1 ifconfig -a
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::802d:edff:fe37:ad9 prefixlen 64 scopeid 0x20<link>
    ether 82:2d:ed:37:0a:d9 txqueuelen 1000 (Ethernet)
    RX packets 55 bytes 6432 (6.4 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 9 bytes 726 (726.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Figure 4: Configuration of h1

```

mininet> h2 ifconfig -a
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.2 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::e010:cff:feea:67e2 prefixlen 64 scopeid 0x20<link>
    ether e2:10:0c:ea:67:e2 txqueuelen 1000 (Ethernet)
    RX packets 59 bytes 6766 (6.7 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 9 bytes 726 (726.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Figure 5: Configuration of h2

```

mininet> h3 ifconfig -a
h3-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.3 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::461:cfff:feb5:6133 prefixlen 64 scopeid 0x20<link>
    ether 06:61:cf:b5:61:33 txqueuelen 1000 (Ethernet)
    RX packets 73 bytes 7894 (7.8 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 11 bytes 866 (866.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Figure 6: Configuration of h3


```

mininet> h4 ifconfig -a
h4-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.4 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::9c5e:4eff:fe26:8fba prefixlen 64 scopeid 0x20<link>
    ether 9e:5e:4e:26:8f:ba txqueuelen 1000 (Ethernet)
    RX packets 72 bytes 7665 (7.6 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 12 bytes 956 (956.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Figure 7: Configuration of h4

```

mininet> s1 ifconfig -a
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.232.133 netmask 255.255.255.0 broadcast 192.168.232.255
    inet6 fe80::bfb:32a1:7e07:a361 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:a0:23:a3 txqueuelen 1000 (Ethernet)
    RX packets 43486 bytes 54660984 (54.6 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 12901 bytes 2444138 (2.4 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 1616 bytes 164974 (164.9 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1616 bytes 164974 (164.9 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ovs-system: flags=4098<BROADCAST,MULTICAST> mtu 1500
    ether b6:bc:8a:e4:26:1b txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

s1: flags=4098<BROADCAST,MULTICAST> mtu 1500
    ether 6a:c0:47:05:08:44 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 54 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

s2: flags=4098<BROADCAST,MULTICAST> mtu 1500
    ether 76:8b:cf:5d:db:4e txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 54 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Figure 8: Configuration of Switch 1

```

mininet> s2 ifconfig -a
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.232.133 netmask 255.255.255.0 broadcast 192.168.232.255
    inet6 fe80::bfb:32a1:7e07:a361 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:a8:23:a3 txqueuelen 1000 (Ethernet)
    RX packets 43486 bytes 54660984 (54.6 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 12901 bytes 2444138 (2.4 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 1631 bytes 165834 (165.8 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1631 bytes 165834 (165.8 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ovs-system: flags=4098<BROADCAST,MULTICAST> mtu 1500
    ether b6:bc:8a:e4:26:1b txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

s1: flags=4098<BROADCAST,MULTICAST> mtu 1500
    ether 6a:c0:47:05:08:44 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 54 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

s2: flags=4098<BROADCAST,MULTICAST> mtu 1500
    ether 76:8b:cf:5d:db:4e txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 54 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Figure 9: Configuration of Switch 2

- Subsequently, I used not only the show interface brief command but also the links command in order to identify which host is related to which switch.

```

mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
s1-eth3<->s2-eth1 (OK OK)
s2-eth2<->h3-eth0 (OK OK)
s2-eth3<->h4-eth0 (OK OK)

```

Figure 10: Connected to nodes

- I was able to generate the network diagram as I analyzed every configuration that was made on the network.

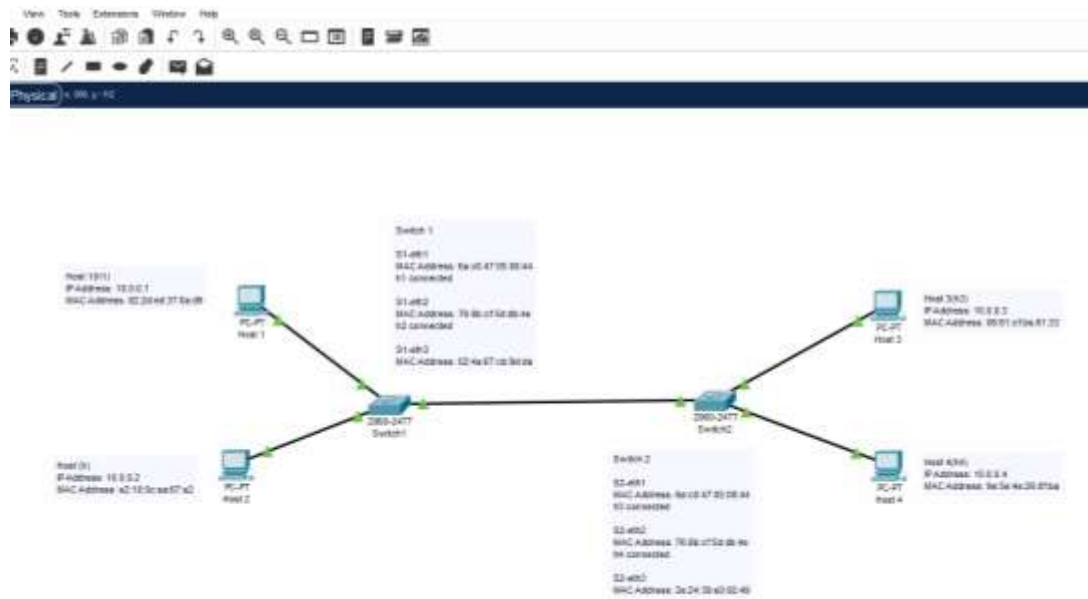


Figure 11: Cisco Network Diagram

Part C

- The next step is to find out how much each node is connected to the others. To discover the connectivity of nodes, I entered the command, “net.”

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s2-eth2
h4 h4-eth0:s2-eth3
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:s2-eth1
s2 lo: s2-eth1:s1-eth3 s2-eth2:h3-eth0 s2-eth3:h4-eth0
c0
```

Figure 12: Connectivity of network nodes

- I then tried to ping each of them to see how log they were interconnected.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Figure 13: pinging all the host

Part D

Q1. Why is the use of devices like TCAMs more challenging in an SDN environment than in a traditional switch?

- Hence, when retrieving routing and self-acting tables, the normal traditional network switches use a format known as Ternary Memory Storage Unit Technology (TCAM). The TCAM (ternary content addressable memory) is one of the typical representatives of conventional devices. It usually has a set of TCAM entries that cannot be changed as it has to perform very fast packet processing at a time.
- In an SDN structure the general mind directs the network strategy and the control plane and also the data plane are separated. Since flows that express entries should be changed often, issues such as power consumption and scalability errors may occur, thereby making the dynamics of an SDN switch architecture as being attributed to TCAM utilization. The light SDN controller can operate on the resources until they are utilized less environmentally, and TCAM has a limited capacity making it inefficient. Solving the problem of high performance but keeping the resource usage optimized is critical for TCAM in the SDN controlling process because it has to work with flow entries. (Jarraya, 2014)

(171 Words)

Q2. What might be some solutions for dealing with issues of scale in an Open SDN environment?

- It is also true there may be challenges while extending in an open SDN environment because the SDN controller holds the network's single point of control and it possibly become a limitation as network grows. The following are the ways to address these problems:
 - ✓ **Flow Aggregation:** When flows are grouped the controller only needs to handle or manipulate the flow entries of a group of flows. This technique reduces the burden of the controller since it groups related flows together into a single rule.
 - ✓ **Edge Computing:** It is possible to take some of the processing load to the edges of the network, in doing so the overall load that the central controller must perform will be less because more traffic is localized.
 - ✓ **Hierarchical and Distributed Controllers:** Instead of a single point controller, an implementation of architecture with hierarchical or distributed controller can be used. Thereby, in addition to increasing the amount of tolerated faults, it also allows for load balancing. They sync a global view together with each controller who handles a segment of the network.
 - ✓ **Enhanced Southbound Protocols:** All in all, enabling wildcard rules may be helpful to the effective protocol like Open Flow 1.3+ to deal with the flow tables entries. This leads to formulating traffic rules in a way that they are stated one at a time, thereby requiring few flow rules. (Zhang, 2017)

(229 Words)

Q3. What are the different types of APIs you noticed in the above task sheet (command and program). List (name and explain) two low-level, mid-level, and high-level APIs.

- APIs play the important roles in the frame of SDN and interactions provided by it. It helps in interaction with controllers and SDN applications and also with the control plane and data plane.
- **Low-level-API**
 - ✓ **Netconf:** Managed network devices have to be configured by the SDN controller and Netconf paradigm is used for this purpose for config/phy devices.
 - ✓ **OpenFlow:** One of the critical protocols of SDN is OpenFlow provides direct access to the data plane of underlay switches routers which may virtual junct or physical switches running over the hypervisor layer.
- **Mid-level-API**
 - ✓ **gRPC:** Thus, an RPC framework known as gRPC enhances micro-services communication by providing best excellent functionality in an SDN environment.
 - ✓ **REST API:** Application policies can be dynamically changed on the fly on the rules on the flow with RESTful API beside it maintain a standard way of interacting with southbound SDN controllers as OpenDaylight or ONOS.
- **High-level-API**
 - ✓ **Intent-Based APIs:** In this case, users indicate how the desired network would be achieved by adopting a higher level of technology than that which forms the substrate. To the last level of policy, the controller transforms this intention into specific network policies.
 - ✓ **Northbound APIs:** The abstractions that enable applications in a network to access the SDN controller to manage and control a network are referred to as northbound APIs. They ease network management as the generalization of lower levels of detail accompanies them. (Nick McKeown, 2008)

(237 Words)

Conclusion

As this project focused on the SDN and its practical implementation and testing, here, I acquired practical experience on designing and analysing the new network topology in detail using Mininet. During the procedure the hosts and switches have to be configured along with the MAC and IP addresses and; nodes within the network must also be checked for connectivity. This practical technique provided significant information for the subject of network architecture and workings.

I gained even more insights into the nuances of SDN architecture analysis through the corresponding theoretical topics that can be reviewed as follows

The limitations of TCAM use in SDN scenarios

Performance bottlenecks in Open SDN Network management APIs

classification

When SDN has been discussed, TCAM utilization and scalability problems demonstrate the need for SDN and resource management and imaginative solutions for SDN such as different controllers, edge computing, and flow grouping. How APIs assist the control plane, data plane, and applications in their interaction with each other also underlined their indispensability as tools for enhancing the management of a network.

In conclusion, I found that the focus of this assignment on as-built design, network diagram creation and related management processes has provided me with valuable practical experience that brought a lot of what I learned from other courses into perspective. It demonstrated how SDN's flexibility and (years) programmability enabled solutions to complex network issues and how architectural design and well-designed APIs are critical to the creation of sustainable network-scale solutions.

References

- A Survey and a Layered Taxonomy of Software-Defined Networking.
https://www.researchgate.net/publication/270633375_A_Survey_and_a_Layered_Taxonomy_of_Software-Defined_Networking
- Hamed, A. (2019). TCAM architecture and its challenges in SDN
https://www.researchgate.net/publication/318232218_TCAM_Space-efficient_Routing_on_Software_Defined_Network
- McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J. and Turner, J. (2008). OpenFlow: Enabling Innovation in Campus Networks.
<https://dl.acm.org/doi/10.1145/1355734.1355746>