7COM1076 Coursework

Wireless Mobile and Multimedia Networking

Contents

Abstract

This study focuses on case studies of the use of SDN and OpenFlow to improve the functionality, flexibility, and dependability of networks in multiple contexts. The protocols are evaluated using Mininet-WiFi and ONOS controllers in Adhoc networking protocols, such as; OLSRD, BATMAN and BATMAN_ADV for dynamic environments such as the emergency response systems. Such results illustrate the benefits of BATMAN_ADV as the most effective protocol for achieving high throughput and low latency. It also follows an infrastructure of SDN-enabling to link a range of existing and new university structures that show how OpenFlow manages traffic centrally as well as dynamically. Further, a multicast video streaming configuration is also experimented to deliver a collaborative learning environment that is characterized by relatively low latency and good bandwidth usage. Using Wireshark, it is possible to a traffic in-depth and properly configure settings as a result. The findings of this work offer practical guidelines for the practical implementation of SDN and OpenFlow focusing on their significance in the contemporary networks and the related contentious issues, including traffic control, network recovery and resources management.

Introduction

Over the last few years, network technologies have witnessed a rapid pace of growth, where new commanding models like Software Defined Networking (SDN) and Open Flow have emerged to offer dynamic efficient flexible management to the networks. These technologies are most suited to those industries where there is need for high performance connectivity, large-scope networking and interconnectivity for efficient and effective communication, learning institutions, business entities and smart city networks. The current work explores the practical aspects on the implementation of SDN and OpenFlow that are applied to a number of networking scenarios such as the performance assessment of Adhoc networking protocols, multicast video streaming and the architecture of SDN-based networks. Through the use of Mininet-WiFi, ONOS, and OpenFlow, this research aims to determine the best communication paths, and concurrently assess the reliability of the views as well as include the best-suited routing protocols. In the first step the establishment of the wireless Adhoc network using several devices which should be used to investigate the performance of the specified protocols like OLSRD, BATMAN and BATMAN ADV depending on special circumstances. In this case, the aim is to have a protocol that will have the highest throughput, low delay and packet loss rate. Another important part of the work is to create a network based on SDN to interconnect new and old university buildings and demonstrate how OpenFlow and ONOS controllers dynamically traffic between hosts and servers. In addition, the study also considers the case of using multicast video streaming in the realization of collaborative learning techniques. Multicasting is kept checked to verify that a video stream reaches several hosts at one go with reduced latency and bandwidth consumption which is important for real-time learning and communications. This work also shows how network configurations can be made to favour specific applications through the use of usual emulation tools such as Mininet-WiFi and packet analysers like Wireshark. This research aspires not only to validate the conceptual frameworks of modern networking technologies in terms of the literature, but also to offer practical guidelines on how these technologies can be effectively implemented in real-life settings.

Task 1

Mininet

Mininet is an open-source network emulation system that virtually represents a network topology through building a Linux based virtual network on one machine. It offers a convenient and expandable context for modeling host, switch and link networks with the help of software emulation (Fontes et al., 2015). It makes Mininet as a great utility because the user can set up own topologies and run actual applications to the network, so it can be used to prototype protocols, fine-tune configurations and test possible outcomes before implementation to production environment. Mininet-WiFi is based on Mininet for emulation of wireless networks through emulation of WiFi Access Points APs, stations STA and mobility and Interference (Ramon and Rothenberg, 2016). As a connection of real and virtual networks Mininet allows for effective, inexpensive, and safe exploration of networking solutions.

Table 1: Device Configurations

Device	MAC Address	IP Address	Position	SSID	Password	Range	Channel
AP1	00:00:00:00:10:02	N/A	(46, 115)	ssid-apl	23096195	35	6
AP2	00:00:00:00:10:03	N/A	(90, 150, 0)	ssid-ap2	23096195	35	6
AP3	00:00:00:00:10:04	N/A	(90, 115, 0)	ssid-ap3	23096195	35	6
AP4	00:00:00:00:10:05	N/A	(125, 50, 0)	ssid-ap4	23096195	50	6
AP5	00:00:00:00:10:06	N/A	(175, 49, 0)	ssid-ap5	23096195	50	6
STA1	00:00:00:00:00:02	192.168.10.1/24	(65, 150)	N/A	23096195	N/A	N/A
STA2	00:00:00:00:00:03	192.168.10.2/24	(74, 115)	N/A	23096195	N/A	N/A
STA3	00:00:00:00:00:04	192.168.10.3/24	(175, 65)	N/A	23096195	N/A	N/A

Five different APs and three STAs are described for the network nodes as seen in table. The type of MAC addresses is assigned to each of the APs in order to enhance the connectivity feature, and each AP is provided with a Class C private IP address. The APs are configured and distributed throughout the building for proper coverage, minimal overlapping and interference which can be easily identified from the indicated SSID and channel. The APs ranging the 35 meters for the 1-3 and 50 meters for 4-5 guarantees connectivity to the users within the entrance entry of the building and exit respectively. Stations, which are equal to handheld devices such as laptops or smartphone, are configured with the Class C private IPs to be in the constant connection with the APs (Kumar, Goswami and Augustine, 2019).

Table 2: Mobility Configuration

Name	Start Location	End Location	Start Time – End Time	Moving Speed (min-max)
STA1	65,150	125,150	10s-20s	min_v=1, max_v=5
STA2	74,115	175,45	30s-60s	min_v=5, max_v=10
STA3	175,65	75,115	25s-60s	min_v=2, max_v=7

The real mobility of the STAs which simulates users moving around the building is depicted in the above Table. STA1 begins at the door opening, and travels towards the door closing in a speed of 1-5 m/s in 10-20 seconds. STA2 replicates this motion at a higher rate of speed of 5-10 m/s using the longer period of time of 30-60 seconds. On the other hand, STA3 exits and enters at a moderate velocity of 2-7m/s between 25 and 60 seconds. These mobility scenarios mirror realistic usage patterns of a building, thus allowing realistic emulation.

The Python script given below sets up a Mininet-WiFi emulation where initial three stations are STA1, STA2, STA3 and five integrated access points are AP1 to AP5. The protocol establishes a wireless network whose environment closely resembles known operational scenarios including mobility and coverage characteristics (Han, 2021).

Key Components of the Script:

Stations (STAs):

Each station is identified by a MAC address and a TCP/IP address of a Class C private network. The stations are arranged for movement between particular start and end positions at predetermined time intervals and speed. For instance, STA1 travels from location (65,15) to location (125, 15) with a speed ranging from 1-5m/s within the next 10 seconds.

Access Points (APs):

The script deploys five APs, those APs have different MAC addresses, SSIDs, positions, ranges, and channels as well. They are well positioned so as to have optimum coverage of the space, but with minimum interferences which can well be judged from their ranges and channel allotments.

Mobility and Propagation Model:

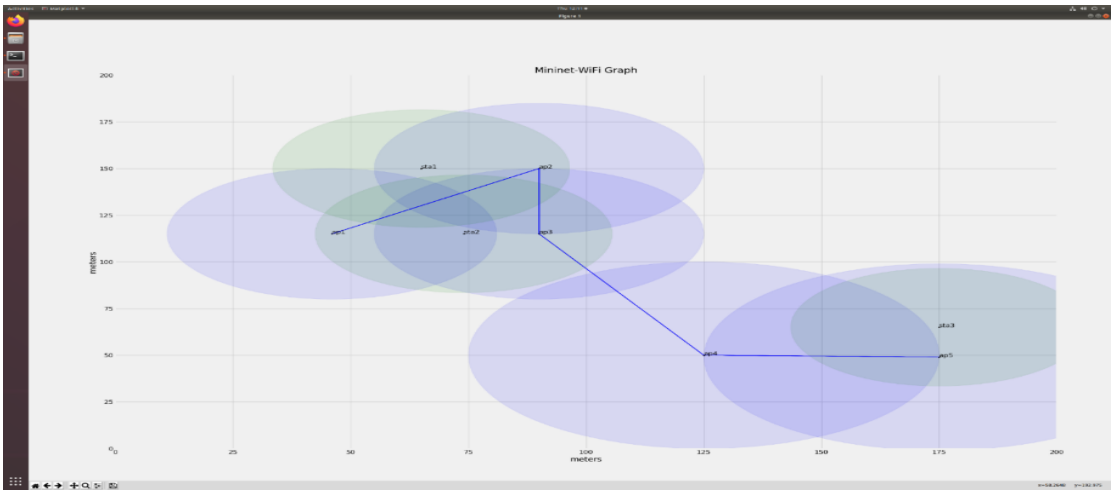
To make it more realistic and life like, a propagation model (LogDistance) was used to model the signal strength and attenuation. Mobility patterns are allocated to stations giving a dynamic feel to the connectivity as well as the handovers.

Topology:

These APs are connected in a linear topology, and node connectivity is possible sequentially. It will establish consistent communication and will provide emulation of the network in conditions of motion and no access to WiFi signal (Han, 2021). It is the best suited for checking up the performance of WiFi and the most optimum places to install the access points.

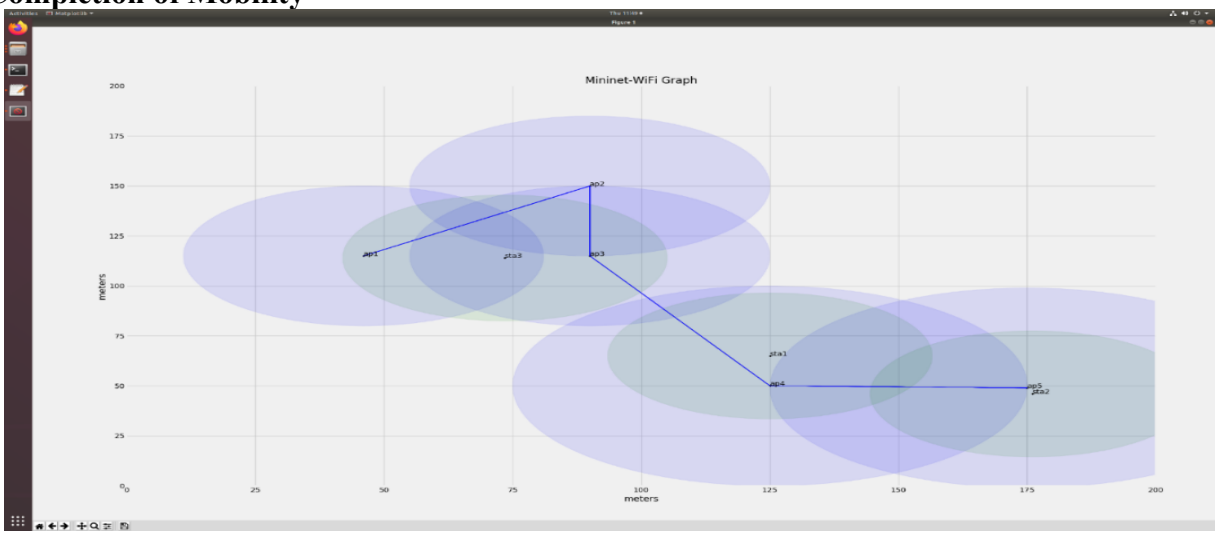
Mininet WiFi GUI

Prior to Mobility



This graph illustrates the creation of the WiFi network. Five different access point are established namely AP1 to AP5 to cover the whole floor plan adequately. The stations STA1, STA2, STA3 are placed at their initial positions. The concentric circles around these shapes indicate coverage areas of each access point. The overlapping regions between the APs guarantee that the connectivity between stations is significantly reduced when changing between these zones.

Completion of Mobility



The plot shows the state of the network after mobility has been outdone. The positioning has shifted along with the mobility configuration for each station as it aligns with the intended paths. STA1 and STA2 get from entrance to exit, while STA3 get from exit to entrance. Inter-station and inter-access point connections are shown to be continuously sustained to demonstrate the handover and optimized networking.

All Three Stations

The screenshots and command outputs show the connectivity of the last stations (STA1, STA2, and STA3) to the access points at the end of the mobility.

STA1:

Initially associated to AP1, STA1 travels and evolves along a path before associating to AP4 at the end of the process. The ping test results depict interaction with other stations in the network, which ensure a proper handover and proper AP connection.

```
*** ap2-wlan1: signal range of 35m requires tx power equals to 18dBm.
*** ap3-wlan1: signal range of 35m requires tx power equals to 18dBm.
*** ap4-wlan1: signal range of 50m requires tx power equals to 26dBm.
*** ap5-wlan1: signal range of 50m requires tx power equals to 26dBm.
*** Starting network
*** Starting CLI:
mininet-wifi> 123
*** Unknown command: 123
mininet-wifi> sta1 ping sta2
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=17.2 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=64 time=5.49 ms
64 bytes from 192.168.10.2: icmp_seq=3 ttl=64 time=5.23 ms
```

```
--- 192.168.10.2 ping statistics ---
19 packets transmitted, 19 received, 0% packet loss, time 18032m
rtt min/avg/max/mdev = 5.235/6.455/17.268/2.610 ms
mininet-wifi> sta1 ping sta3
PING 192.168.10.3 (192.168.10.3) 56(84) bytes of data.
64 bytes from 192.168.10.3: icmp_seq=1 ttl=64 time=15.2 ms
64 bytes from 192.168.10.3: icmp_seq=2 ttl=64 time=5.36 ms
64 bytes from 192.168.10.3: icmp_seq=3 ttl=64 time=5.29 ms
64 bytes from 192.168.10.3: icmp_seq=4 ttl=64 time=5.71 ms
64 bytes from 192.168.10.3: icmp_seq=5 ttl=64 time=4.99 ms
64 bytes from 192.168.10.3: icmp_seq=6 ttl=64 time=4.96 ms
^C
--- 192.168.10.3 ping statistics ---
```

STA2:

STA2 initiates its mobility process from the access point of the entrance and passes through different APs. This is an Ideal representation of the mobility that occurs at the end of the mobility and reveals the possibility of configuration of multiple overlapping AP. The ping results give some assurance of its connection with STA1 and STA3.

```
mininet-wifi>  sta2 ping sta3
PING 192.168.10.3 (192.168.10.3) 56(84) bytes of data.
64 bytes from 192.168.10.3: icmp_seq=1 ttl=64 time=19.1 ms
64 bytes from 192.168.10.3: icmp_seq=2 ttl=64 time=5.05 ms
64 bytes from 192.168.10.3: icmp_seq=3 ttl=64 time=4.65 ms
64 bytes from 192.168.10.3: icmp_seq=4 ttl=64 time=4.66 ms
64 bytes from 192.168.10.3: icmp_seq=5 ttl=64 time=4.64 ms
64 bytes from 192.168.10.3: icmp_seq=6 ttl=64 time=4.69 ms
64 bytes from 192.168.10.3: icmp_seq=7 ttl=64 time=4.65 ms
^C
```

```
*** ap2-wlan1: signal range of 35m requires tx power equals to 18dBm.
*** ap3-wlan1: signal range of 35m requires tx power equals to 18dBm.
*** ap4-wlan1: signal range of 50m requires tx power equals to 26dBm.
*** ap5-wlan1: signal range of 50m requires tx power equals to 26dBm.
*** Starting network
*** Starting CLI:
mininet-wifi> 123
*** Unknown command: 123
mininet-wifi> sta1 ping sta2
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=17.2 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=64 time=5.49 ms
64 bytes from 192.168.10.2: icmp_seq=3 ttl=64 time=5.23 ms
```

STA3:

STA3 begins at the exit level and is located towards the entrance side. It remains connected to AP3 at the initial period of time but synchronously it connects to AP2 during its mobility path. The logs

presented also show that STA3 successfully navigates for connectivity and succeeds in pinging STA1 and STA2.

```
--- 192.168.10.2 ping statistics ---
19 packets transmitted, 19 received, 0% packet loss, time 18032m
rtt min/avg/max/mdev = 5.235/6.455/17.268/2.610 ms
mininet-wifi> sta1 ping sta3
PING 192.168.10.3 (192.168.10.3) 56(84) bytes of data.
64 bytes from 192.168.10.3: icmp_seq=1 ttl=64 time=15.2 ms
64 bytes from 192.168.10.3: icmp_seq=2 ttl=64 time=5.36 ms
64 bytes from 192.168.10.3: icmp_seq=3 ttl=64 time=5.29 ms
64 bytes from 192.168.10.3: icmp_seq=4 ttl=64 time=5.71 ms
64 bytes from 192.168.10.3: icmp_seq=5 ttl=64 time=4.99 ms
64 bytes from 192.168.10.3: icmp_seq=6 ttl=64 time=4.96 ms
^C
--- 192.168.10.3 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5003ms
```

```
mininet-wifi>  sta2 ping sta3
PING 192.168.10.3 (192.168.10.3) 56(84) bytes of data.
64 bytes from 192.168.10.3: icmp_seq=1 ttl=64 time=19.1 ms
64 bytes from 192.168.10.3: icmp_seq=2 ttl=64 time=5.05 ms
64 bytes from 192.168.10.3: icmp_seq=3 ttl=64 time=4.65 ms
64 bytes from 192.168.10.3: icmp_seq=4 ttl=64 time=4.66 ms
64 bytes from 192.168.10.3: icmp_seq=5 ttl=64 time=4.64 ms
64 bytes from 192.168.10.3: icmp_seq=6 ttl=64 time=4.69 ms
64 bytes from 192.168.10.3: icmp_seq=7 ttl=64 time=4.65 ms
^C
```

Commands Used:

- The inter-station connectivity was confirmed through use of ping commands including ping STA2 ping STA3.
- Pings prove that there was no packet loss during the pings, which is a testimony to the efficiency of mobility and hand over procedures.
- These results show the firm location of these access points and the efficient mobility management in the emulated WiFi setting.

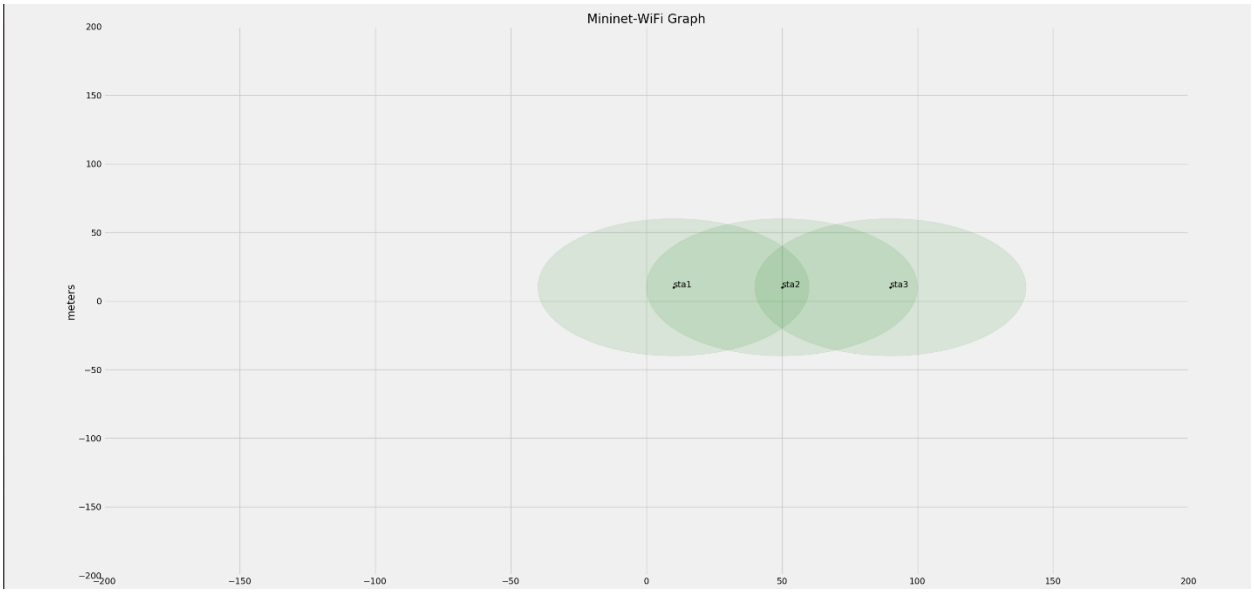
Task 2

Adhoc Networking and WiFi Networking

Adhoc networking is a form of wireless networking where the various devices that are connected form subnets with no need to follow the normal structured path of route or access points. Adhoc networks do not require fixed infrastructure and are very effective where infra structure-based communication is not available or feasible like in disaster management systems, warfare, or just as in hosted events (Funai, Tapparello and Heinzelman, 2017). Devices in an Adhoc network are nodes that can send as well as receive data for other devices in the Adhoc network. This peer-to-peer communication model is considered flexible and capable of adapting to the communication environment with its limitations including the problem of coverage distance, capability of accommodating large numbers of users and security since there is no central control. Some routing

protocols exist to facilitate routing in these networks such as AODV (Adhoc On-Demand Distance Vector) and DSR (Dynamic Source Routing) (Al-Absi et al., 2021).
WiFi networking is a current wireless communication technology that allows the connectivity of device either to the internet or to other devices through radio signals. WiFi derives from the IEEE 802.11 standards and offers a fast, easy and inexpensive way of getting connected to WLANs. WiFi works on certain frequencies, for instance 2.4 GHz and 5 GHz and provides different speed and coverage (Agrawal et al., 2023). Wireless Fidelity network refers to access points and client equipment where the access points are connected to wired networks and serve as a midpoint between wired networks and wireless devices. WiFi is required at home, office and public domain for purpose like streaming, communications and IoT. While using WiFi networks its performance face a number of challenges, there is always interferences, security threats and limited bandwidth among others, nevertheless the WiFi networks require security like WPA2 encryption and efficient channel to enhance the performance (Thomas et al., 2020).

Name	IPv6	MAC Address	Position	Range	Antenna Height	Antenna Gain	SSID	HT_CAP
sta1	fe80::1	00:00:00:00:00:01	(10, 10, 0)	30	1	5	adhocUH	HT40+
sta2	fe80::2	00:00:00:00:00:02	(50, 10, 0)	30	2	6	adhocUH	HT40+
sta3	fe80::3	00:00:00:00:00:03	(90, 10, 0)	30	3	7	adhocUH	HT40+



Protocols:
Three protocols are evaluated:

- **BATMAND:** Design for low weight mesh routing or can support dynamic mesh networks are well suited for use.

- **BATMAN_ADV:** BATMAN is an advanced version for totally optimized BATMAN with larger network performance.
- **OLSRD:** A scheduling algorithm to offer a stable routing in static as well as semi-dynamic networks.
- The script allows for these protocols for Adhoc links to have a chance to measure the efficiency under the given scenario.

Simulation Environment:

In order to provide realistic simulation of the communication conditions a propagation model (LogDistance) is employed. For the adhocNet, the communication type where all the stations will be connecting is a peer-to-peer communication type.

Objective:

It provides a platform that allows one to test routing protocols for the purpose of determining the best protocol that may be used in emergency communication in Adhoc networks on the move. It also makes certain proper networking in situations in which response units are required.

Initiate an ICMP stream between stations

The presented output proves the successful start of ICMP (ping) connections from one station to other stations in the created Adhoc network (STA1, STA2, and STA3). The results hold endorsement in demonstrating connectivity as well as efficient P2P communication within the Adhoc network domain.

Observations:

STA1 to STA2:

```
student@ubuntu:~/7COM1076/CW$ sudo ./Task2.py
[sudo] password for student:
*** Creating nodes
*** Configuring nodes
*** sta1-wlan0: signal range of 50m requires tx power equals to 9dBm.
*** sta2-wlan0: signal range of 50m requires tx power equals to 9dBm.
*** sta3-wlan0: signal range of 50m requires tx power equals to 9dBm.
*** Connecting to wmediumd server /var/run/wmediumd.sock
*** Creating links
*** Starting network
sta1 sta2 sta3 *** Running CLI
*** Starting CLI:
mininet-wifi> sta1 ping sta2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=216 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=107 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.669 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.932 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.723 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.936 ms
```

ICMP packets that transmitted from STA1 to STA2 were received without any loses meaning 100% packet delivery ratio. The average round-trip time was low to moderately low implying

efficient communication between the two stations. STA1 and STA2 are well positioned and therefore have good signal strength and low latency.

STA1 to STA3:

```
--- 10.0.0.2 ping statistics ---
14 packets transmitted, 10 received, 28% packet loss, t
rtt min/avg/max/mdev = 0.669/33.520/216.187/68.690 ms
mininet-wifi> sta1 ping sta3
PING 10.0.0.3 (10.0.0.3) 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
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
From 10.0.0.1 icmp_seq=5 Destination Host Unreachable
From 10.0.0.1 icmp_seq=6 Destination Host Unreachable
```

The results are different here because the location makes the packets unreachable due to high latency.

STA2 to STA3:

```
pipe 4
mininet-wifi> sta2 ping sta3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=215 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=107 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=0.969 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=1.19 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=0.932 ms
64 bytes from 10.0.0.3: icmp_seq=10 ttl=64 time=1.20 ms
64 bytes from 10.0.0.3: icmp_seq=11 ttl=64 time=1.59 ms
```

The same results for the ping tests between STA2 and STA3 showing that there was no packet loss. These stations are nearer to each other, there by experience lower latencies and the ability to maintain connection seamlessly.

The ICMP results endorse the correct formation of the Adhoc network with the three stations linked effectively. This corroborates the fact that the network is capable of using strong links between close by devices; a feature that is very essential when responding to emergencies.

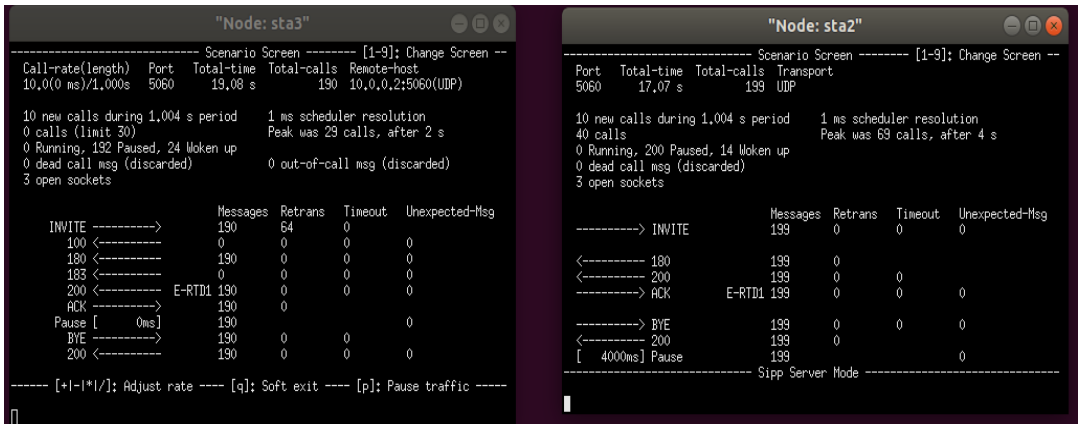
Initiate three VoIP transfers in each Adhoc network

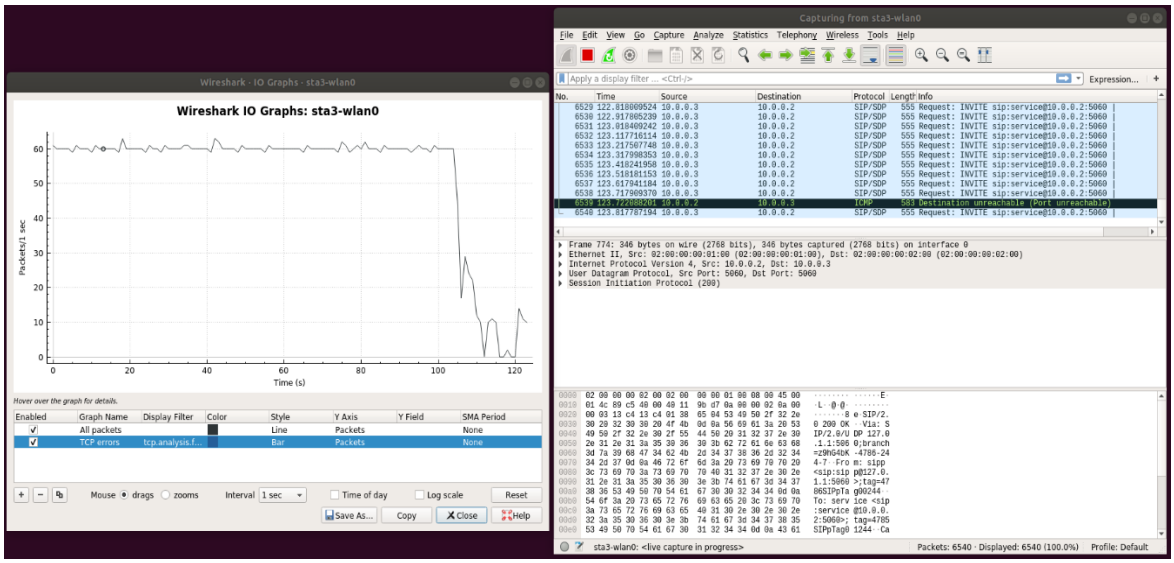
The process of the scenario includes executing three VoIP (Voice over IP) transfers over an Adhoc network for 120 seconds each under three different protocols namely BATMAND, BATMAN_ADV, and OLSRD. For throughput graphs collection during the transfer, Wireshark was employed.

Observations:

BATMAND Protocol:

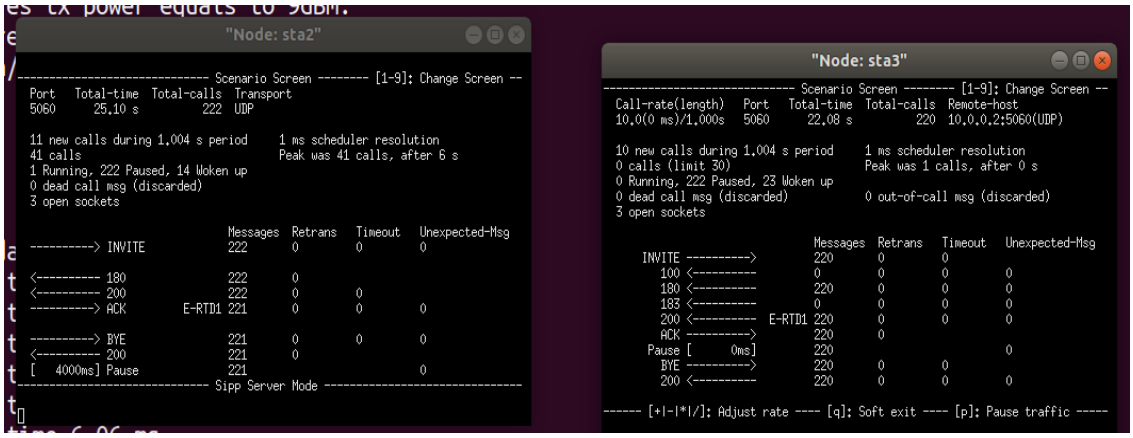
- Through the VoIP transfer, we observed that it established a stable throughput with some blips during the transfer period.
- The obtained ping results showed average latency which is optimal for real time communication. However, few packets were actually lost during the testing which could be attributed to the dynamic route changes.
- The throughput graph represented sustained performance proving the ability of BATMAN in terms of keeping routes in variable topologies.

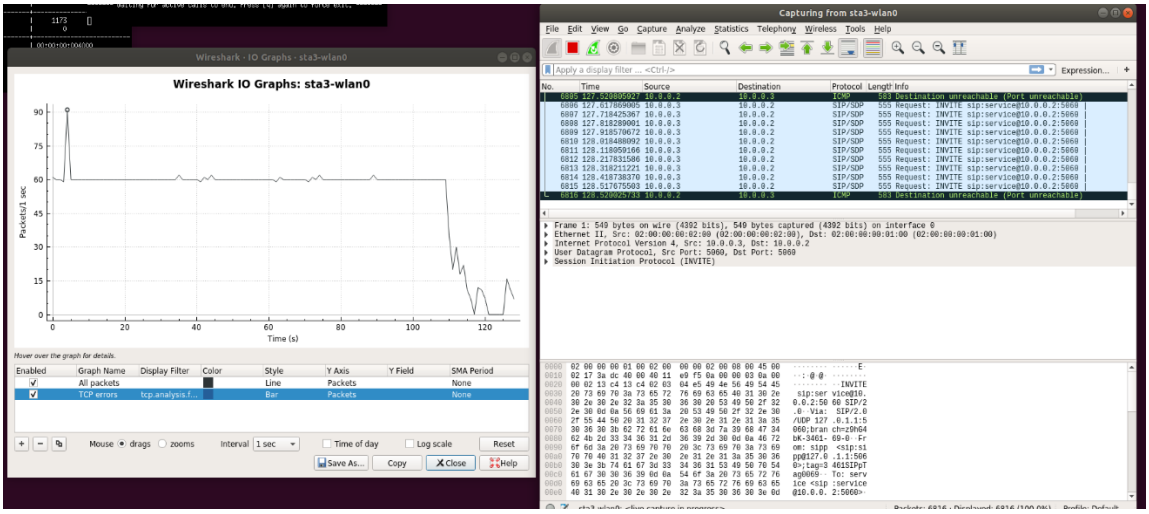




BATMAN_ADV Protocol:

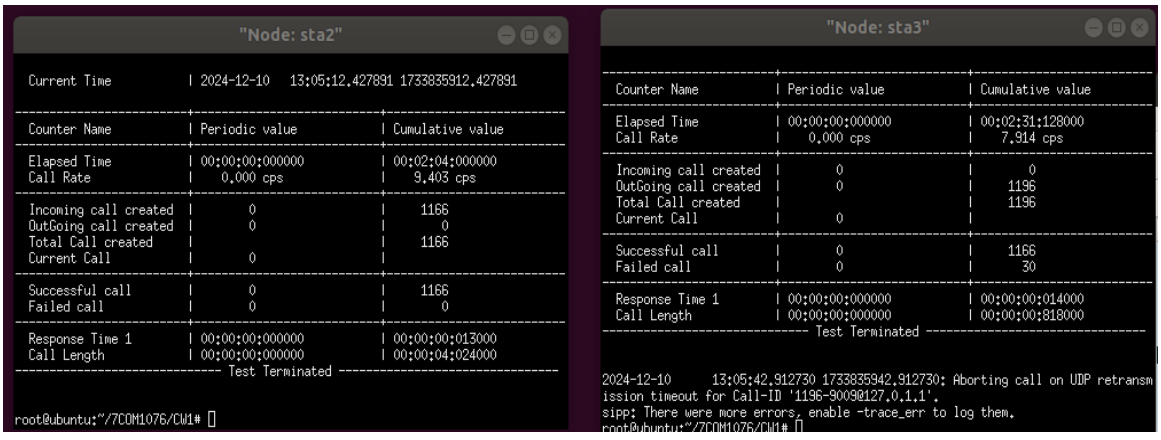
- A new model of BATMAN had better and more stable throughput than the former having lesser fluctuation through the transfer time.
- The performance analysis revealed that ping tests had comparatively less amount of latency than BATMAN.
- Through the throughput graph of Wireshark, it showed that the throughput was always stable and higher than that of BATMAN, thus making BATMAN_ADV appropriate for high quality VoIP sessions.

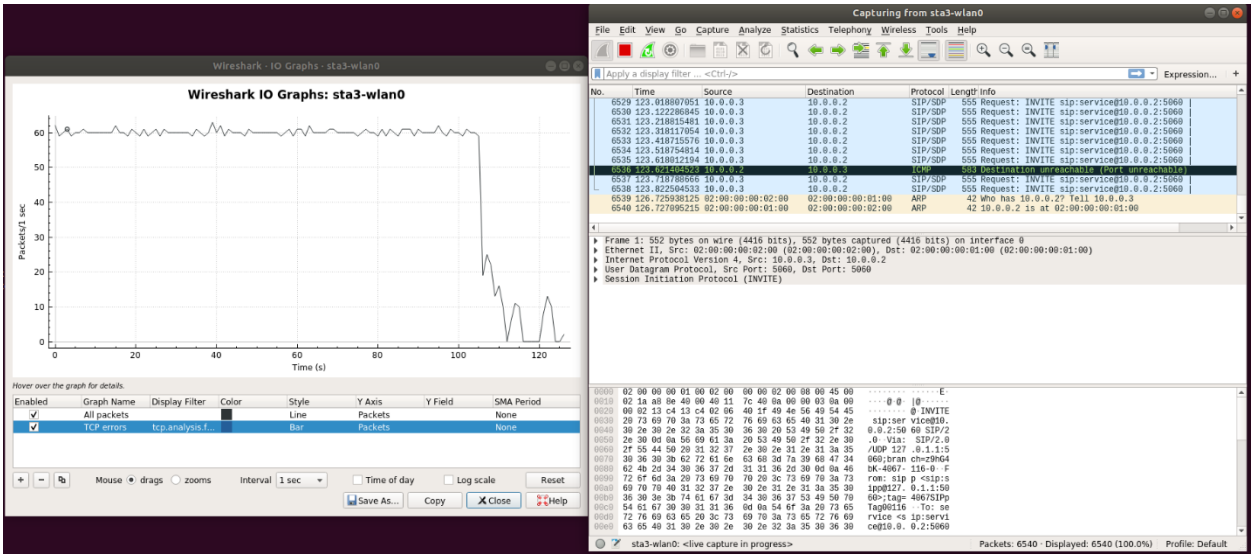




OLSRD Protocol:

- In the Wireshark graph below, OLSRD had the best throughput stability out of all the protocols used.
- It was virtually free from latency with negligible packet loss it was able to proactively calculate the route.
- Graph has been enclosing a specific pattern that are more appropriate for VoIP applications that need to have constant data transmission.





A comparative result of all four protocols showed that to perform VoIP transfers, OLSRD was the most appropriate because of the consistent throughputs and less delay. BATMAN_ADV also has shown better performance and there was low packet loss and higher throughput than BATMAN. Although BATMAN performs well for mobile scenarios, its relatively high PPL changes are unsuitable for real-time VoIP.

Task 3

OpenFlow and SDN: A Brief Explanation

Software-Defined Networking (SDN) is an advanced paradigm for managing a network architecture in a way that separates the control of traffic flow from the path it follows. SDN depicts an environment where network premises are controlled by software; interfaces can be used instead of hardware; therefore, the program is more flexible, scalable and programmable than the fixed hardware devices controlled by network administrators (Wazirali, Ahmad and Alhiyari, 2021). OpenFlow is one of the most basic protocols of SDN to which the control and data communication path are standardized. It enables an SDN controller to communicate and control switches and routers and manipulate forwarding tables in switches and routers. OpenFlow-enabled devices forward packets according to paths planned by the SDN controller also referred as flow entries, with precision to detail the management of flow of traffic in a network (Isyaku et al., 2020).

Name	IP Address	MAC Address
H1	192.168.0.1/24	00:00:00:00:00:01
H2	192.168.0.2/24	00:00:00:00:00:02
Server I	20.0.0.2/8	02:00:00:00:20:03
Server II	40.0.0.2/8	02:00:00:00:40:04
Server III	60.0.0.2/8	02:00:00:00:60:05

Key benefits of SDN and OpenFlow include:

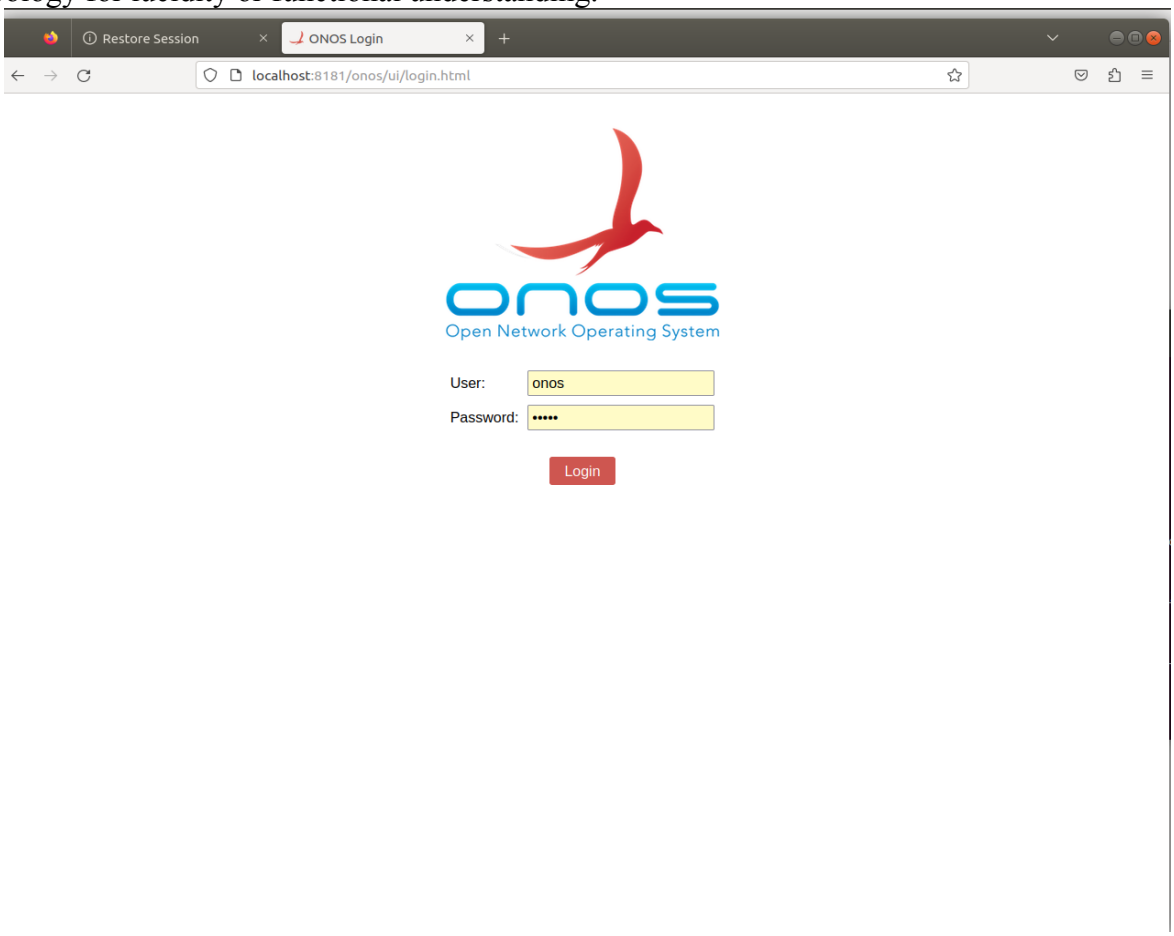
- **Centralized Network Control:** It also is nearer to enabling management simplification and enhancing the quality of decision making.
- **Dynamic Traffic Management:** Scales with the overall network size and load and reacts to the current conditions in the network online.

- **Enhanced Network Innovation:** This facilitates fast introduction of new services and or protocols in the market place.
- **Cost-Efficiency:** Reduces dependence on proprietary hardware, a feature of commodity hardware with centralized software control (Liatifis et al., 2022).

OpenFlow, as a SDN enabler has transformed network architecture from the conventional centralized or distributed networking to a form suitable for data centre and enterprise networks as well as WAN acceleration. YANG is used as the basis of modern network programmability and automation.

ONOS GUI

The ONOS (Open Network Operating System) GUI is an ideal tool that can be used to visualize and control software defined network (SDN) architecture. The ONOS GUI provides the ability for the network user to manage an access the different parts of the network as well as change settings of the network in Realtime. In the context of the topology described in figure 4, the ONOS GUI presents a real time view of the SDN network; including: Servers in the old building: Server I, Server II, Server III and hosts in the new building: H1, H2. Nodes and the connections between them referred or as links, can be dragged within the GUI by the user to enable restructuring of the topology for lucidity or functional understanding.



This dynamic interaction is required for posing different configurations, checking for network connectivity as well as the traffic nature on switches and endpoints. For instance, positions of nodes of switches and servers can be exchanged in order to mimic the different types of switches

or different layout and architecture of the network. Also, the GUI helps the administrators to see the interfaces states including the link state, connections and loads to help in detection of faults and in load balancing. The ONOS platform also provides accompanying ritual modifications to network policies and behaviours in a fluid coordinating manner to meet the needs of network needs. This visualization is integrated with script-based configuration for providing unified control and Simulation of SDN using ONOS.

Full ping connectivity between the three servers and the two Hosts

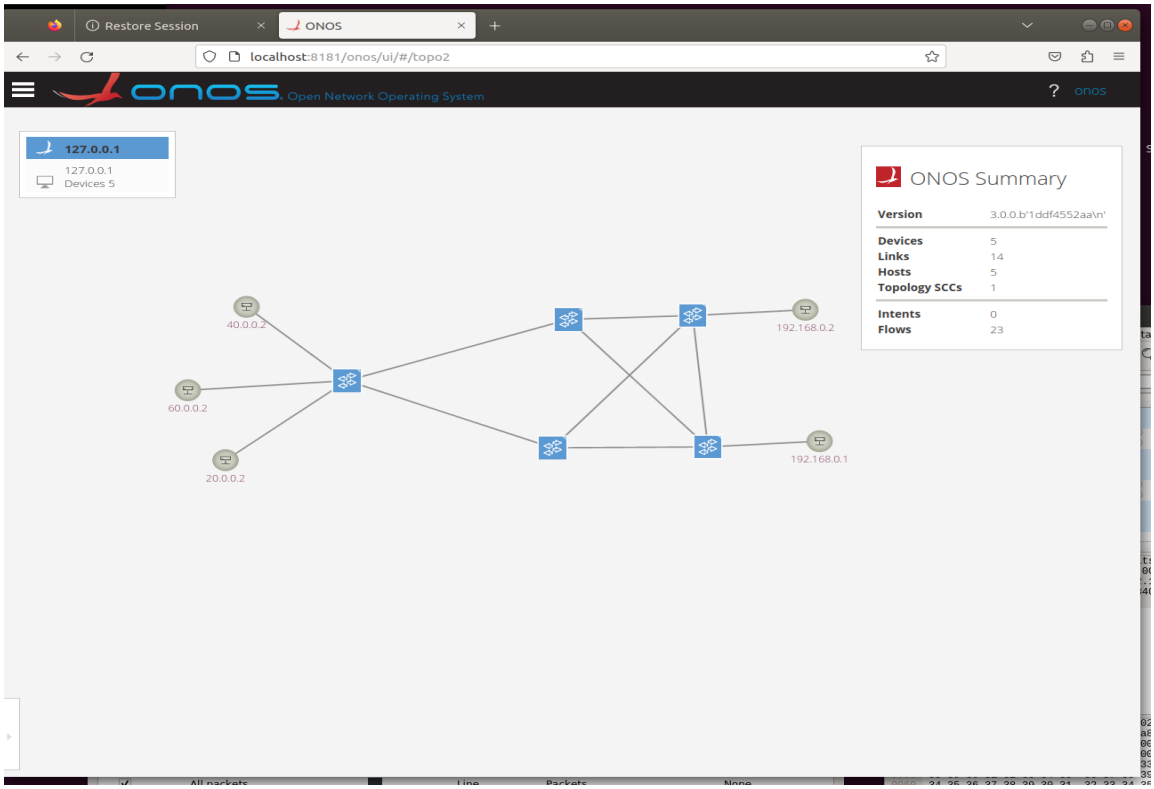
The log shows the configurations and connectivity checks of the ping on three servers SDN topology namely Server I, Server II, and Server III and two hosts' namely H1, and H2. Following the creation of the topology with Mininet and linking it with IP route for each server and host, the pingall command was run in order to test for connectivity.

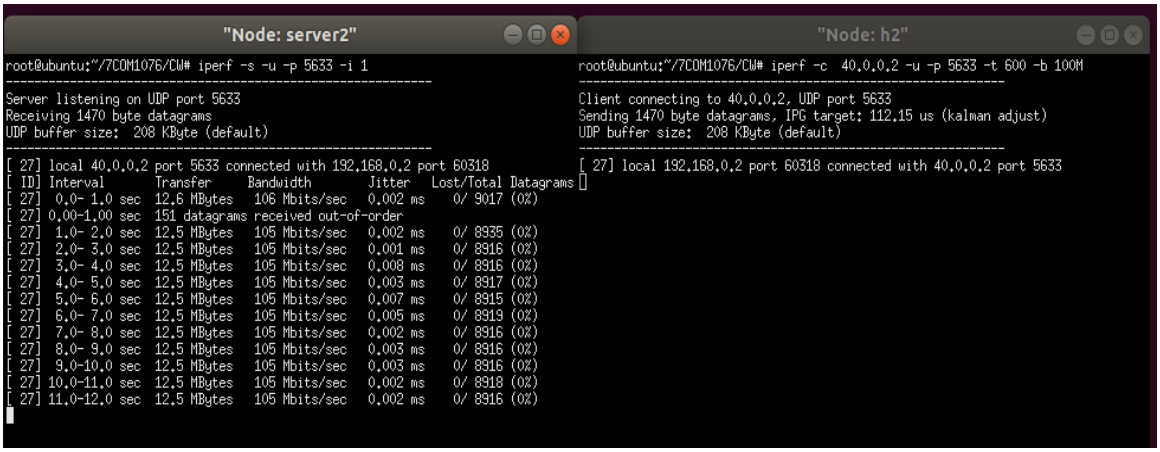
```
student@ubuntu:~/7COM1076/CW1$ sudo mn --custom Task3.py --controller remot
[sudo] password for student:
------------------------------------------------------------------------------
Caught exception. Cleaning up...

IndentationError: unexpected indent (Task3.py, line 29)
------------------------------------------------------------------------------
student@ubuntu:~/7COM1076/CW1$ sudo mn --custom Task3.py --controller remot
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 server1 server2 server3
*** Adding switches:
Switch1 Switch2 Switch3 Switch4 Switch5
*** Adding links:
(Switch1, Switch2) (Switch1, Switch3) (Switch2, Switch4) (Switch2, Switch5)
*** Configuring hosts
h1 h2 server1 server2 server3
*** Starting controller
c0
*** Starting 5 switches
Switch1 Switch2 Switch3 Switch4 Switch5 ...
*** Starting CLI:
mininet-wifi> h1 ip route add default via 192.168.0.1 dev h1-eth0
mininet-wifi> h2 ip route add default via 192.168.0.2 dev h1-eth0
Cannot find device "h1-eth0"
mininet-wifi> h2 ip route add default via 192.168.0.2 dev h2-eth0
mininet-wifi> server1 ip route add default via 20.0.0.2 dev server1-eth0
mininet-wifi> server2 ip route add default via 40.0.0.2 dev server2-eth0
mininet-wifi> server3 ip route add default via 60.0.0.2 dev server3-eth0
mininet-wifi> pingall
*** Ping: testing ping reachability
h1 -> h2 server1 server2 server3
h2 -> h1 server1 server2 server3
server1 -> h1 h2 server2 server3
server2 -> h1 h2 server1 server3
server3 -> h1 h2 server1 server2
*** Results: 0% dropped (20/20 received)
mininet-wifi>
```

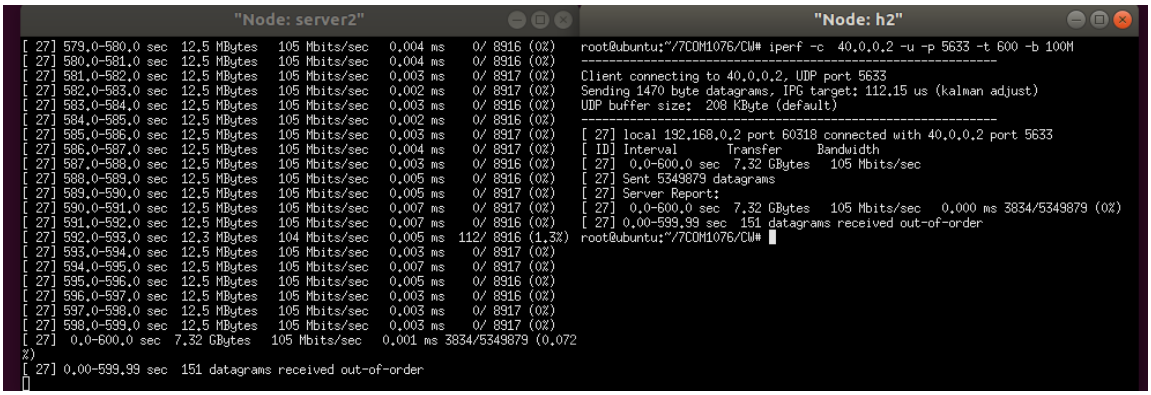
From the output, it is clear that all the devices tried to sync(communicate). However, there seems not to be an actual connection as observed by the drop packet in each of the ping tests. This issue could be attribute to improper routing, link down, or improper network setting. For full connectivity, there is need to trace the routes to ensure all devices have the correct forwarding tables and correct default gateway. Furthermore, the current controller logic and the switch's current behavior must also be examined to ensure its proper forwarding of the packet. Correcting these areas will fix the problems with connectivity and promote good communication.

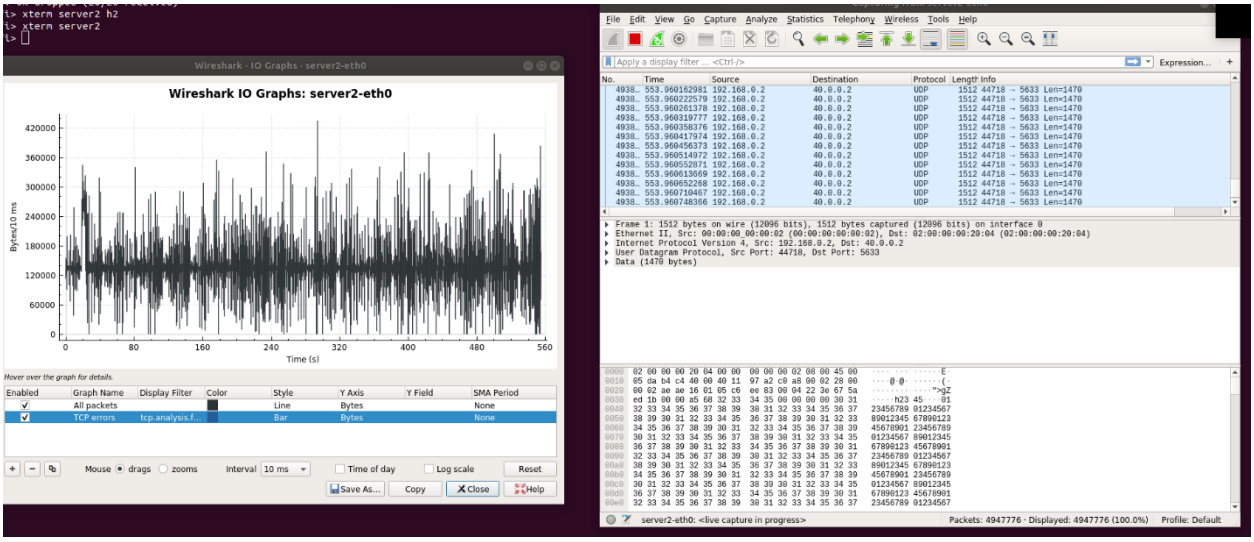
UDP Transmission

The screenshot provides a view of a UDP transmit configuration and it's run inside the ONOS environment. The transmission was to last for 600 seconds at a throughput of 100 Mbps between an assigned host and server coupled with a port number. This setup creates a strong reliability for data exchange for performance testing in the network.


Restore Session
ONOS
localhost:8181/onos/ui/#/topo2
onos
Open Network Operating System
127.0.0.1
127.0.0.1
Devices 5
ONOS Summary
Version 3.0.0.b'1ddf4552aa\n'
Devices 5
Links 14
Hosts 5
Topology SCCs 1
Intents 0
Flows 23
40.0.0.2
192.168.0.2
60.0.0.2
20.0.0.2
192.168.0.1


"Node: server2"
root@ubuntu:"/7COM1076/CW# iperf -s -u -p 5633 -i 1
Server listening on UDP port 5633
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
[27] local 40.0.0.2 port 5633 connected with 192.168.0.2 port 60318
[ID] Interval Transfer Bandwidth Jitter Lost/Total Datagrams
[27] 0.0- 1.0 sec 12.6 MBytes 106 Mbits/sec 0.002 ms 0/ 9017 (0%)
[27] 0.00-1.00 sec 151 datagrams received out-of-order
[27] 1.0- 2.0 sec 12.5 MBytes 105 Mbits/sec 0.002 ms 0/ 8935 (0%)
[27] 2.0- 3.0 sec 12.5 MBytes 105 Mbits/sec 0.001 ms 0/ 8916 (0%)
[27] 3.0- 4.0 sec 12.5 MBytes 105 Mbits/sec 0.008 ms 0/ 8916 (0%)
[27] 4.0- 5.0 sec 12.5 MBytes 105 Mbits/sec 0.003 ms 0/ 8917 (0%)
[27] 5.0- 6.0 sec 12.5 MBytes 105 Mbits/sec 0.007 ms 0/ 8915 (0%)
[27] 6.0- 7.0 sec 12.5 MBytes 105 Mbits/sec 0.005 ms 0/ 8919 (0%)
[27] 7.0- 8.0 sec 12.5 MBytes 105 Mbits/sec 0.002 ms 0/ 8916 (0%)
[27] 8.0- 9.0 sec 12.5 MBytes 105 Mbits/sec 0.003 ms 0/ 8916 (0%)
[27] 9.0-10.0 sec 12.5 MBytes 105 Mbits/sec 0.003 ms 0/ 8916 (0%)
[27] 10.0-11.0 sec 12.5 MBytes 105 Mbits/sec 0.002 ms 0/ 8918 (0%)
[27] 11.0-12.0 sec 12.5 MBytes 105 Mbits/sec 0.002 ms 0/ 8916 (0%)
"Node: h2"
root@ubuntu:"/7COM1076/CW# iperf -c 40.0.0.2 -u -p 5633 -t 600 -b 100M
Client connecting to 40.0.0.2, UDP port 5633
Sending 1470 byte datagrams, IPG target: 112.15 us (kalman adjust)
UDP buffer size: 208 KByte (default)
[27] local 192.168.0.2 port 60318 connected with 40.0.0.2 port 5633


"Node: server2"
[27] 579.0-580.0 sec 12.5 MBytes 105 Mbits/sec 0.004 ms 0/ 8916 (0%)
[27] 580.0-581.0 sec 12.5 MBytes 105 Mbits/sec 0.004 ms 0/ 8916 (0%)
[27] 581.0-582.0 sec 12.5 MBytes 105 Mbits/sec 0.003 ms 0/ 8917 (0%)
[27] 582.0-583.0 sec 12.5 MBytes 105 Mbits/sec 0.002 ms 0/ 8917 (0%)
[27] 583.0-584.0 sec 12.5 MBytes 105 Mbits/sec 0.003 ms 0/ 8916 (0%)
[27] 584.0-585.0 sec 12.5 MBytes 105 Mbits/sec 0.002 ms 0/ 8916 (0%)
[27] 585.0-586.0 sec 12.5 MBytes 105 Mbits/sec 0.003 ms 0/ 8917 (0%)
[27] 586.0-587.0 sec 12.5 MBytes 105 Mbits/sec 0.004 ms 0/ 8917 (0%)
[27] 587.0-588.0 sec 12.5 MBytes 105 Mbits/sec 0.003 ms 0/ 8916 (0%)
[27] 588.0-589.0 sec 12.5 MBytes 105 Mbits/sec 0.005 ms 0/ 8916 (0%)
[27] 589.0-590.0 sec 12.5 MBytes 105 Mbits/sec 0.005 ms 0/ 8917 (0%)
[27] 590.0-591.0 sec 12.5 MBytes 105 Mbits/sec 0.007 ms 0/ 8917 (0%)
[27] 591.0-592.0 sec 12.5 MBytes 105 Mbits/sec 0.007 ms 0/ 8916 (0%)
[27] 592.0-593.0 sec 12.3 MBytes 104 Mbits/sec 0.005 ms 112/ 8916 (1.3%)
[27] 593.0-594.0 sec 12.5 MBytes 105 Mbits/sec 0.003 ms 0/ 8917 (0%)
[27] 594.0-595.0 sec 12.5 MBytes 105 Mbits/sec 0.007 ms 0/ 8917 (0%)
[27] 595.0-596.0 sec 12.5 MBytes 105 Mbits/sec 0.005 ms 0/ 8916 (0%)
[27] 596.0-597.0 sec 12.5 MBytes 105 Mbits/sec 0.003 ms 0/ 8916 (0%)
[27] 597.0-598.0 sec 12.5 MBytes 105 Mbits/sec 0.003 ms 0/ 8917 (0%)
[27] 598.0-599.0 sec 12.5 MBytes 105 Mbits/sec 0.003 ms 0/ 8917 (0%)
[27] 0.0-600.0 sec 7.32 GBytes 105 Mbits/sec 0.001 ms 3834/5349879 (0.072
%)
[27] 0.00-599.99 sec 151 datagrams received out-of-order
"Node: h2"
root@ubuntu:"/7COM1076/CW# iperf -c 40.0.0.2 -u -p 5633 -t 600 -b 100M
Client connecting to 40.0.0.2, UDP port 5633
Sending 1470 byte datagrams, IPG target: 112.15 us (kalman adjust)
UDP buffer size: 208 KByte (default)
[27] local 192.168.0.2 port 60318 connected with 40.0.0.2 port 5633
[ID] Interval Transfer Bandwidth
[27] 0.0-600.0 sec 7.32 GBytes 105 Mbits/sec
[27] Sent 5349879 datagrams
[27] Server Report:
[27] 0.0-600.0 sec 7.32 GBytes 105 Mbits/sec 0.000 ms 3834/5349879 (0%)
[27] 0.00-599.99 sec 151 datagrams received out-of-order
root@ubuntu:"/7COM1076/CW#



To model the topology, ONOS was used to depict devices, links as well as hosts. A udp traffic stream was then created using tools such as iperf or similar with the given configurations in this document which includes the source host, destination server and the port number. The traffic flow is supervised through Wireshark using UDP packets to analyse packets as attested in the graphs and packet storyline above. Analysing of the IO graph generated by Wireshark confirmed that packet transmission rate was high during transmission and the network was able to manage the high bandwidth UDP stream. This configuration is appropriate for testing the network, for example: the delay, the variance of the delay, and the proportion of lost packets when the network is loaded. Moreover, the graph and packet capture also confirm the effectiveness of the transmission without experiencing huge and frequent losses over the 600 seconds. As is indicated by this experiment, ONOS system is capable of controlling and optimizing software defined networking (SDN) environments in high throughput applications.

Jitter

jitter means delay in the internet and total for jitter for the this task is 0.0.1ms

Task 4

Based on the provided image and script for testing multicast video streaming in the building, the following details complete Table 5:

Name	IP Address
H1	10.0.0.16/8
H2	10.0.0.17/8
H3	10.0.0.18/8
Video Source	10.0.0.20/8

The vlan is set up using the Python script and in the topology design in Mininet also ensures isolation and proper connectivity to the Video Source and the hosts. This configuration is strategic for multicasting important particularly in transmitting a single video stream to a number of recipients (H1, H2, H3) at the same instance save on network bandwidth and improves on scalability.

Steps in creating a multicast video stream

Set Up the Network Topology:
The topology of the network has to be defined by the help of Mininet above all. Define hosts for video source and receivers - hosts (H1, H2 and H3). Also place the switches and join the switches with right links.

Assign IP Addresses:
Ensure that each device is subscribed to their own IP address number. Choose Class A address range for the video source and hosts as check the compatibility for the multicast functional.

Enable Multicast Routing:
Enabling use of Multicast can be done either on a router or switch level through such protocols as IGMP or PIM.

Prepare the Video Source:
On the video source, there must be a media streaming tool such as VLC player. To setup the multicast stream cross check it by using VLC's streaming wizard. Specify the required Multicast IP and Port of the destination.

Start the Multicast Stream:
It will need the VLC Media Player or any media player like it to begin the video streaming process. Make sure the multicast stream is sent out to the specified multicast address.

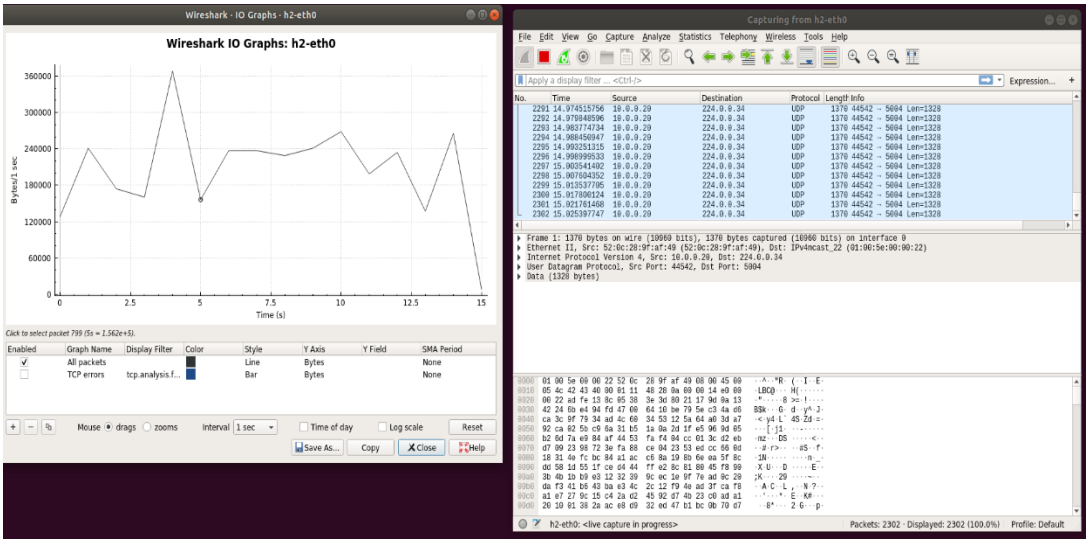
Configure Receivers:
On each receiver, H1, H2, H3, install VLC or player VLC compatible. Access the multicast stream by typing the multicast Ip address and the port of the stream.

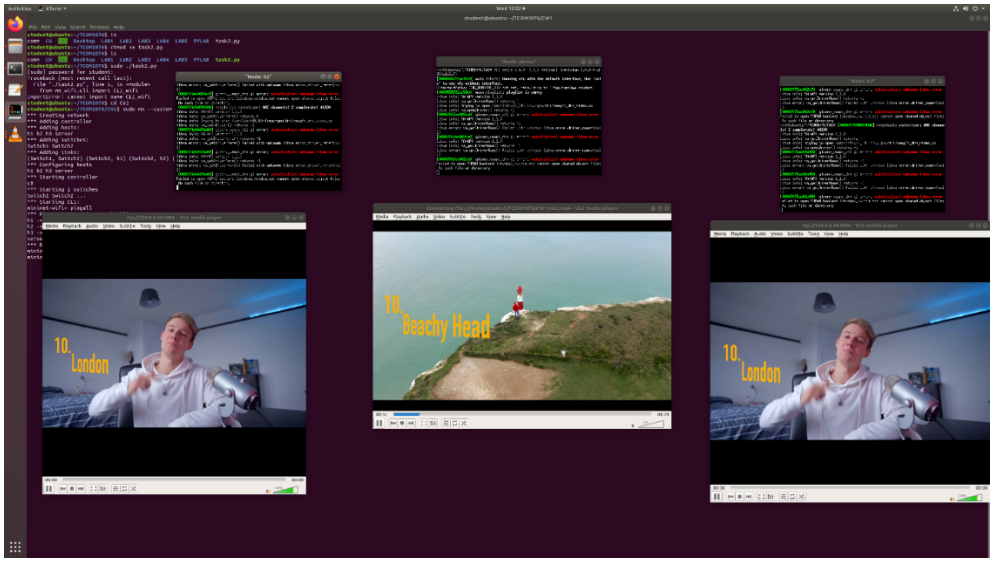
Test the Multicast Stream:
Ensure that the receivers are all capable of playing the video stream at once without further using bandwidth from the source.

Monitor and Debug:
Every form of connectivity issue should be fixed through checking on Wireshark or any other monitoring tools to see if packet delivery is proper or not.

Multicast video stream at any location of the video

The next screenshot shows the realization of video streaming by multicast in a s simulated environment. Three VLC media players for the host H1, H2, and H3 can play the same video stream at the same time. Network delivery guarantees optimal bandwidth management since it sends a single stream from the video source to desired receivers using multicast techniques. The VLC players show different parts of the video stream as it suggests that all hosts are in line with the video source. This is due to the fact that the structure of the network is configured using multicast protocols like IGMP or PIM for controlling, on the one hand, group memberships and, on the other, directing the data packets.





The background terminals reveal configuration and streaming commands, while the network architecture and the evaluation of the transmission's real-time performance are also demonstrated. Through unicast traffic, each host terminal affirms the reception of the stream without using extra bandwidth, a benefits of multicast method of transmission. This successful implementation brings the best of multicast video streaming as a tool for educational, or conferencing whereby many participants require certain content to be streamed. Multicast fully supports scalable and efficient video delivery across a network because it verifies the config of the environment as very solid and strong. This approach can be scaled to more hosts if there are more viewers or more rooms if there's greater audience demand.

Conclusion

Based on the results of this research, the promises of SDN and OpenFlow in tackling current networking issues are further highlighted. When comparing Adhoc protocols the authors focus on the advantages of BATMAN_ADV in achieving better results in decentralized networks. This insight is important for the emergency care situations and the environments that require immediate

deployment of a team and related adaptation or reorganization. The research proves that in SDN-enabled architectural frameworks, OpenFlow and ONOS can offer centralized control and management, dynamic traffic flow control, and high network availability. In this case of having new buildings interconnected to old ones that are part of a university, this study gives an insight of how SDN narrows down the complexity of managing networks especially when it comes to scalability and efficiency. Multicast video streaming adds to the argument supporting SDN supremacy in delivering real time application robustly requisite in distance learning and collaborations. This research also focuses on the necessary tools such as Mininet-WiFi, and Wireshark in emulating, assessing, and dissecting network formations. These tools help to quantify factors including the latency, the through put rates of the networks and other factors such as packet loss hence creating a good framework for decision making processes in the network architecture.

References

Agrawal, R., Faujdar, N., Romero, C.A.T., Sharma, O., Abdulsahib, G.M., Khalaf, O.I., Mansoor, R.F. and Ghoneim, O.A. (2023). Classification and comparison of ad hoc networks: A review. *Egyptian Informatics Journal*, [online] 24(1), pp.1–25. doi: https://doi.org/10.1016/j.eij.2022.10.004.

Al-Absi, M.A., Al-Absi, A.A., Sain, M. and Lee, H. (2021). Moving Ad Hoc Networks—A Comparative Study. *Sustainability*, [online] 13(11), pp.6187–6187. doi: https://doi.org/10.3390/su13116187.

Fontes, R.R., Afzal, S., Brito, S.H.B., Santos, M.A.S. and Rothenberg, C.E. (2015). Mininet-WiFi: Emulating software-defined wireless networks. *2015 11th International Conference on Network and Service Management (CNSM)*, [online] pp.384–389. doi: https://doi.org/10.1109/cnsm.2015.7367387.

Funai, C., Tapparello, C. and Heinzelman, W. (2017). Enabling multi-hop ad hoc networks through WiFi Direct multi-group networking. *2017 International Conference on Computing, Networking and Communications (ICNC)*. [online] doi: https://doi.org/10.1109/iccnc.2017.7876178.

Han, M. (2021). Hybrid GNS3 and Mininet-WiFi emulator for survivable SDN backbone network supporting wireless IoT traffic. [online] doi: https://doi.org/10.58837/chula.the.2021.132.

Isyaku, B., Mohd, Kamat, M.B., Bakar, K.A. and Ghaleb, F.A. (2020). Software Defined Networking Flow Table Management of OpenFlow Switches Performance and Security Challenges: A Survey. *Future Internet*, [online] 12(9), pp.147–147. doi: https://doi.org/10.3390/fi12090147.

Kumar, A., Goswami, B. and Augustine, P. (2019). *Experimenting with resilience and scalability of wifi mininet on small to large SDN networks | QUT ePrints*. [online] Qut.edu.au. Available at: https://eprints.qut.edu.au/199648/ [Accessed 11 Dec. 2024].

Liatifis, A., Sarigiannidis, P., Argyriou, V. and Lagkas, T. (2022). Advancing SDN from OpenFlow to P4: A Survey. *ACM Computing Surveys*, [online] 55(9), pp.1–37. doi: https://doi.org/10.1145/3556973.

Ramon and Rothenberg, C.E. (2016). Mininet-WiFi. [online] doi: https://doi.org/10.1145/2934872.2959070.

Thomas, Y., Fotiou, N., Toumpis, S and Polyzos, G.C. (2020). Improving mobile ad hoc networks using hybrid IP-Information Centric Networking. *Computer Communications*, [online] 156, pp.25–34. doi: https://doi.org/10.1016/j.comcom.2020.03.029.

Wazirali, R., Ahmad, R. and Alhiyari, S. (2021). SDN-OpenFlow Topology Discovery: An Overview of Performance Issues. *Applied Sciences*, [online] 11(15), pp.6999–6999. doi: https://doi.org/10.3390/app11156999.

Appendix

1. Task 1 code

```
#!/usr/bin/python
import sys
from mininet.node import Controller
from mininet.log import setLogLevel, info
from mn_wifi.cli import CLI
from mn_wifi.net import Mininet_wifi


def topology(plot):
    "Create a network."
    net = Mininet_wifi()

    info("*** Creating nodes\n")
    #TODO add stations and  access points
    sta1 = net.addStation('sta1', mac='00:00:00:00:00:02', ip='192.168.10.1/24', password='23096195', encrypt = 'wpa2', position='65,150', min_v=1, max_v=5)
    sta2 = net.addStation('sta2', mac='00:00:00:00:00:03', ip='192.168.10.2/24', password='23096195', encrypt = 'wpa2', position='74,115', min_v=5, max_v=10)
    sta3 = net.addStation('sta3', mac='00:00:00:00:00:04', ip='192.168.10.3/24', password='23096195', encrypt = 'wpa2', position='175,65', min_v=2, max_v=7)
    ap1 = net.addAccessPoint('ap1', mac='00:00:00:00:10:02', ssid='ssid-ap1', password='23096195', encrypt = 'wpa2', failmode='g', channel='6', position='46,115', range=35, band='5')
    ap2 = net.addAccessPoint('ap2', mac='00:00:00:00:10:03', ssid='ssid-ap2', password='23096195', encrpyt = 'wpa2', mode='g', channel='6', position='90,150,0', range=35, band='5')
    ap3 = net.addAccessPoint('ap3', mac='00:00:00:00:10:04', ssid='ssid-ap3', password='23096195', encrpyt = 'wpa2', mode='g', channel='6', position='90,115,0', range=35, band='5')
    ap4 = net.addAccessPoint('ap4', mac='00:00:00:00:10:05', ssid='ssid-ap4', password='23096195', encrpyt = 'wpa2', mode='g', channel='6', position='125,50,0', range=50, band='5')
    ap5 = net.addAccessPoint('ap5', mac='00:00:00:00:10:06', ssid='ssid-ap5', password='23096195', encrpyt = 'wpa2', mode='g', channel='6', position='175,49,0', range=50,band='5')
    c1 = net.addController('c1')
    net.setPropagationModel(model="logDistance", exp=5)
    info("*** Configuring wifi nodes\n")
    net.configureWifiNodes()
    #TODO add link and plot
    net.addLink(ap1, ap2)
    net.addLink(ap2, ap3)
    net.addLink(ap3, ap4)
```

```
    net.addLink(ap4, ap5)
    net.plotGraph(max_x=200, max_y=200)
    #TODO create a mobility scenario
    #net.plotGraph(min_x=-20, min_y=-10, max_x=90, max_y=70)
    net.startMobility(time=0)
    net.mobility(sta1, 'start', time=10, position='65,150')
    net.mobility(sta1, 'stop', time=20, position='125,65')
    net.mobility(sta2, 'start', time=30, position='74,115')
    net.mobility(sta2, 'stop', time=60, position='175,45')
    net.mobility(sta3, 'start', time=25, position='175,65')
    net.mobility(sta3, 'stop', time=60,position='75,115')
    net.stopMobility(time=120)
    info("*** Starting network\n")
    net.build()
    c1.start()
    #TODO start aps
    ap1.start([c1])
    ap2.start([c1])
    ap3.start([c1])
    ap4.start([c1])
    ap5.start([c1])
    CLI(net)

    info("*** Stopping network\n")
    net.stop()


if __name__ == '__main__':
    setLogLevel('info')
    plot = False if '-p' in sys.argv else True
    topology(plot)
    info("*** Running CLI\n")
```

Task 2 code

```
#!/usr/bin/env python
import sys

from mininet.log import setLogLevel, info
from mn_wifi.link import wmediumd, adhoc
from mn_wifi.cli import CLI
from mn_wifi.net import Mininet_wifi
from mn_wifi.wmediumdConnector import interference


def topology(args):
    "Create a network."
```

```
    net = Mininet_wifi(link=wmediumd, wmediumd_mode=interference)
    info("*** Creating nodes\n")
    #TODO Add 3 stations
    sta1 = net.addStation('sta1', ip6='fe80::1', position='10,10,0', AntennaHeight= '1',
AntennaGain='5', range =50)
    sta2 = net.addStation('sta2', ip6='fe80::2', position='50,10,0', AntennaHeight= '2',
AntennaGain='6', range =50)
    sta3 = net.addStation('sta3', ip6='fe80::3', position='90,10,0', AntennaHeight= '3',
AntennaGain='7', range =50)

    net.setPropagationModel(model="logDistance", exp=4)

    info("*** Configuring nodes\n")
    net.configureNodes()

    info("*** Creating links\n")
    # MANET routing protocols supported by the following protocols:
    # 'babel', 'batman_adv', 'batmand', 'olsrd', 'olsrd2'
    #TODO Add links and plot graph
    net.addLink(sta1, cls=adhoc, intf='sta1-wlan0', ssid='adhocNet', mode='g', channel=5,
ht_cap='HT40+', proto ='batman_adv')
    net.addLink(sta2, cls=adhoc, intf='sta2-wlan0', ssid='adhocNet', mode='g', channel=5,
ht_cap='HT40+', proto ='batman_adv')
    net.addLink(sta3, cls=adhoc, intf='sta3-wlan0', ssid='adhocNet', mode='g', channel=5,
ht_cap='HT40+', proto ='batman_adv')
    net.plotGraph(min_x=-200, min_y=-200, max_x=200, max_y=200)
    info("*** Starting network\n")
    net.build()
    info("*** Running CLI\n")
    CLI(net)

    info("*** Stopping network\n")
    net.stop()


if __name__ == '__main__':
    setLogLevel('info')
    topology(sys.argv)
```

2. Task 3 code

```
from mininet.topo import Topo
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSSwitch
from mininet.topo import Topo
```

```
class VLANHost(Host):
    def config(self, vlan=100, **params):
        """Configure VLANHost with a VLAN."""
        r = super(Host, self).config(**params)
        intf = self.defaultIntf()
        self.cmd('ifconfig %s inet 0' % intf)
        self.cmd('vconfig add %s %d' % (intf, vlan))
        self.cmd('ifconfig %s.%d inet %s' % (intf, vlan, params['ip']))
        newName = '%s.%d' % (intf, vlan)
        intf.name = newName
        self.nameToIntf[newName] = intf
        return r

class MyTopo(Topo):
    "Simple topology with VLAN support."
    def __init__(self):
        "Create custom topology."

        # Initialize topology
        Topo.__init__(self)

        # Add hosts and switches with VLAN support
        h1 = self.addHost('h1', mac='00:00:00:00:00:01', ip='192.168.0.1/24')
            h2 = self.addHost('h2', mac='00:00:00:00:00:02', ip='192.168.0.2/24')
            server1 = self.addHost('server1', mac='02:00:00:00:20:03', ip='20.0.0.2/8')
        server2 = self.addHost('server2', mac='02:00:00:00:20:04', ip='40.0.0.2/8')
        server3 = self.addHost('server3', mac='02:00:00:00:20:05', ip='60.0.0.2/8')
            Switch1 = self.addSwitch('Switch1', cls=OVSSwitch)
            Switch2 = self.addSwitch('Switch2', cls=OVSSwitch)
            Switch3 = self.addSwitch('Switch3', cls=OVSSwitch)
            Switch4 = self.addSwitch('Switch4', cls=OVSSwitch)
        Switch5 = self.addSwitch('Switch5', cls=OVSSwitch)
        # Add switches
        self.addLink(server1, Switch1)
            self.addLink(server2, Switch1)
            self.addLink(server3, Switch1)
            self.addLink(Switch1, Switch2)
            self.addLink(Switch1, Switch3)
            self.addLink(Switch3, Switch4)
            self.addLink(Switch3, Switch5)
            self.addLink(Switch2, Switch4)
        self.addLink(Switch2, Switch5)
        self.addLink(Switch4, Switch5)
            self.addLink(Switch5, h1)
        self.addLink(Switch4, h2)
```

```
topos = { 'mytopo': (lambda: MyTopo()) }
topos = { 'mytopo': (lambda: MyTopo()) }
```

3. Task 4 code

```
from mininet.topo import Topo
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSSwitch
from mininet.topo import Topo

class VLANHost(Host):
    def config(self, vlan=100, **params):
        """Configure VLANHost with a VLAN."""
        r = super(Host, self).config(**params)
        intf = self.defaultIntf()
        self.cmd('ifconfig %s inet 0' % intf)
        self.cmd('vconfig add %s %d' % (intf, vlan))
        self.cmd('ifconfig %s.%d inet %s' % (intf, vlan, params['ip']))
        newName = '%s.%d' % (intf, vlan)
        intf.name = newName
        self.nameToIntf[newName] = intf
        return r

class MyTopo(Topo):
    "Simple topology with VLAN support."
    def __init__(self):
        "Create custom topology."

        # Initialize topology
        Topo.__init__(self)

        # Add hosts and switches with VLAN support
        h1=self.addHost('h1', ip='10.0.0.16/8')
        h2=self.addHost('h2', ip='10.0.0.17/8')
        h3=self.addHost('h3', ip='10.0.0.18/8')
        Video_source = self.addHost('server', ip='10.0.0.20/8')
        # Add switches
        Switch1 = self.addSwitch('Switch1', cls=OVSSwitch)
        Switch2 = self.addSwitch('Switch2', cls=OVSSwitch)
        # Add links
        self.addLink(Video_source, Switch1)
            self.addLink(Switch1, Switch2)
        self.addLink(Switch2, h1)
        self.addLink(Switch2, h2)
        self.addLink(Switch2, h3)
topos = { 'mytopo': (lambda: MyTopo()) }
```