University of Hertfordshire

School of Physics, Engineering and Computer Science

20070088

December 5, 2024

Wireless Mobile and Multimedia Networking

Software Defined Networks(SDN) Emulation with Mininet and OpenFlow

Contents

List of Tables

List of Figures

Abstract

The world has been considered a digital society with the intervention of the Internet. These computer networks form a strong foundation in the manner we access information, communicate, and handle data, providing convenience in various aspects of life. This technology relies strongly on the operations of traditional IP networks and despite being widely adopted, they do get complex and hard to manage in terms of configuration according to predefined policies and reconfiguration in response to load, faults, and changes. On top of that, conventional networks are vertically integrated(the control plane and data plane are bundled together). Emerging technologies such as Software Defined Networking (SDN) provides a convenient approach in handling computer networks. SDN brings a proposed paradigm which changes state of affairs by separating the network's control logic from the data plane. This approach enables centralisation of network control and ability to program the network improving network availability, scalability, and management. This report presents a practical survey on SDN. It starts by utilising concepts of SDN in both a wireless and wired network scenario, an Ad-Hoc Network for emergency scenario, and multi-cast video streaming. We measure and test out performance metrics such as throughput and jitter for communications under Session Initiation Protocol (SIP), connection-less communication in server and client scenario. All activities are carried out on a single machine running Linux with underlying applications such as Mininet, Open Network Operating System(ONOS) and Wireshark.

4

1   Introduction

Since the conception and realisation of the Advanced Research Projects Agency Networks(ARPANET) [5], developments and innovations in computer networks has seen a significant growth in size and requirements and navigating traditional network switches has become a challenge. In traditional networks, the control plane operation has a distributed infrastructure that requires protocols such as OSPF, STP, EIGRP, to operate independently on network devices. The forwarding decision which is the process of determining how to send a packet from one device (such as a router or switch) to another device within a network, based on certain criteria is performed by these network devices. This decision is made based on the information available in the routing table (for routers) or the switching table (MAC table) (for switches). It is accepted that the network devices connect but there is no centralised machine to manage or summarise the whole network [2]. Also, the distributed control and transport protocols running inside routers and switches makes it complex to express desired high-level network policies since network operators must configure each individual network device separately [3]. This main issue is rectified by SDN introducing the needed network control, flexibility, and programmability.

1.1   Software-Defined Networking (SDN) and OpenFlow

Software-Defined Networking (SDN) is a computer technology that brings a change to the limitations of current computer networks infrastructure. SDN [4] decouples the control and data plane of a network leaving the switch with the simple function of forwarding packets based on a set of rules. By doing so, it breaks the vertical integration of conventional networks as the control logic is separated from the data plane i.e. the underlying switches and routers that forward the traffic [3]. To make possible the communication between the controller and the network devices, OpenFlow [4] is used in conjunction with SDN. OpenFlow standardises the communication between the switches and the software-based controller in an SDN architecture. Researchers found it difficult to test out new ideas in current hardware because the source code of the software running on the network device cannot be modified making the infrastructure 'rigid'. As a result, a standardised protocol(eg. OpenFlow) to control flow table of switches through software was provided by identifying common features in the flow tables of the ethernet switches [4].

The figures below show a comparison of the traditional network and SDN network and an overview of the SDN architecture.
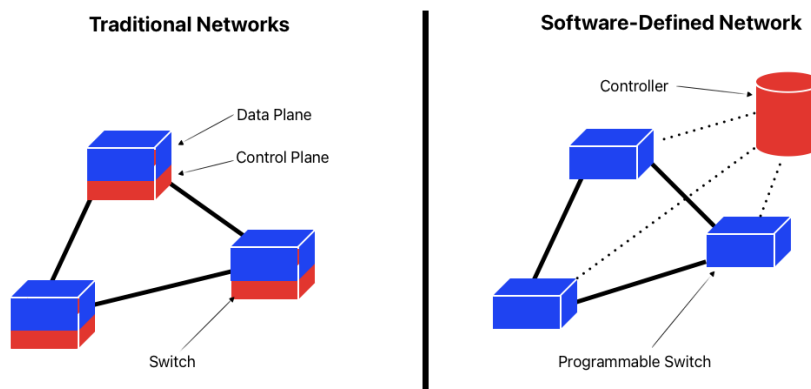

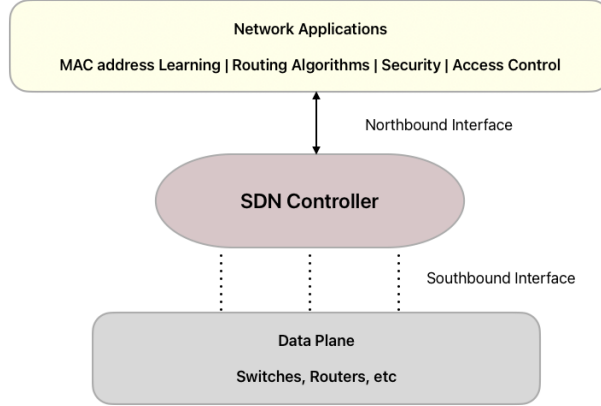Figure 1: Traditional Networks vs SDN Networks

Figure 2: SDN Architecture

The underlying network applications or packages which are the softwares for the logical operations within the network are installed and activated on the SDN controller. These applications are known as the Northbound APIs or interface as shown in fig.2. The Southbound interface consist the protocols that facilitates communication between the remote SDN controller and the network devices, for example the OpenFlow [6] protocol. Examples of such network applications can be STP, routing algorithms, and access control and examples of a SDN controller can be the ONOS controller. The prospect of such architecture is noticed in terms of ease of network management, programmability and openness to innovations and virtualisation.

1.2 Mininet

For the scope of this paper, we aim to determine and investigate the convenience of SDN networks in a variety of network scenarios and requirements. To test out the efficiency, reliabilty, cost, and flexibility of the SDN approach in our network designs, the experiment will be carried out on a network emulators known as Mininet. It is one of the many network simulation tools(another example is OMNETT++) that have been developed to virtualise and test network performance [2, 6].

Mininet [1] provides a system that allows rapid prototyping of large networks and creates scalable software-defined networks using lightweight virtualisation mechanism. This alleviates the downside of conventional networking by providing a centralised view of the network and paves the way for more controllability and managing how networks should operate regardless of their size or their complexity.

Since research on this topic is still in progress, there are not many devices such as routers and switches that implement SDN functionalities, moreover, the existing ones are very expensive. Thus, in order to enable researchers to perform experiments and test novel features of this new paradigm in practice at a low financial cost, one solution is to use virtual network emulators such as Mininet. It creates SDN elements, customise them, and share them among other networks and perform interactions [1].

1.3   Emulation environment specifications

For this experiment, we utilise a microcomputer MacBook with the following specifications: Processor 3 GHz 6-Core Intel Core i5, 16GB of ram, running the macOS 15.1, and VirtualBox Oracle VM version 7.1.2. In this microcomputer, under the management of VirtualBox, we installed the following guest operating systems: Mininet emulator version 2.0 on Linux operating system Ubuntu 64bits with 4GB of RAM; ONOS controller version ¡FIND OUT¿.

2   Task 1 - WiFi Network Simulation

The first experiment will simulate a wireless network topology for a given department floor. The scenario will consist of five access points spread out strategically within the floor to make space for a signal dead-zone(red-spotted) and three stations to mimic mobile nodes which will engage in mobility actions to emulate real-life instance of networks. During this work, we utilise a feature known as Mininet-GUI to view and position APs and nodes/stations and also observe mobile node association and handoff to access point, authentication of access point, full connectivity between all nodes in the network and briefly understand communication channels are used in a wireless network.

The figures below show the outline of the floor-plan in context and position of APs and stations in the design.
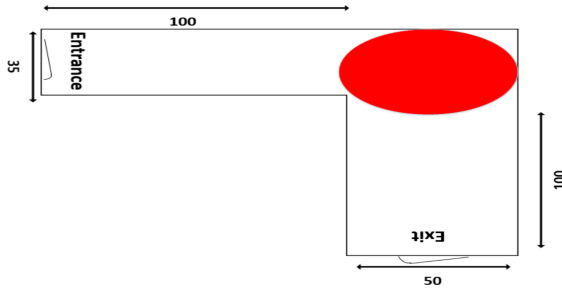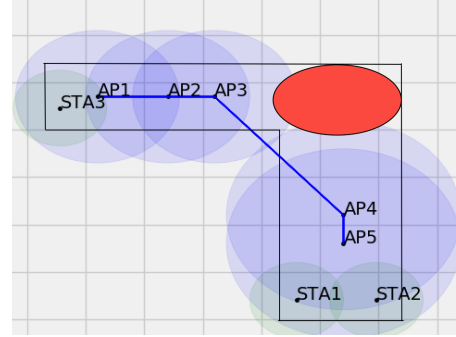

Figure 3: Floor-plan in context

Figure 4: Floor-plan with infrastructure

2.1   Design and Configuration

Both figures show the outline of the floor-plan in context with the dimensions included on fig. 3 while fig. 4 is a snapshot of the Mininet GUI with the positions of the stations and access points and the range of coverage of all devices. The 'red-spotted' zone as seen remains free of WiFi signals as required in the design. Table 1 contains the details of stations and access points which are used for the network configuration in a python script. All three stations share the same network address and have full authenticated access to each access points.

The access points are positioned strategically and linked serially to ensure the required zones within the floor have strong WiFi coverage and support full connectivity for connected nodes or stations. To emulate real networking scenario, mobility is included for the stations, details are shown in Table 2. We get to simulate a mobile node's association and handoff operation by access points. Different mobility attributes in terms of mobile node speed of movement are specified for each station.

The procedure was done in the python code for the network configuration which is stated in the appendix section of this document.

DEVICE	MAC	IPv4	(x,y)	SSID	PASSWORD	RANGE	CHANNEL
STA1	00:00:00:00:00:10	192.168.50.11/24	15,115	n/a	n/a	20	n/a
STA2	00:00:00:00:00:11	192.168.50.12/24	20,130	n/a	n/a	20	n/a
STA3	00:00:00:00:00:12	192.168.50.13/24	140,10	n/a	n/a	20	n/a
AP1	00:00:00:00:00:00	n/a	30,117.5	AP1	n/a	35	1
AP2	00:00:00:00:00:01	n/a	60,117.5	AP2	n/a	35	1
AP3	00:00:00:00:00:02	n/a	80,117.5	AP3	n/a	35	1
AP4	00:00:00:00:00:03	n/a	135,55	AP4	n/a	50	1
AP5	00:00:00:00:00:04	n/a	135,40	AP5	n/a	50	1

Table 1: Details of Stations and Access Points

DEVICE	START LOCATION	END LOCATION	START-STOP TIME	MOVING SPEED(min-max)
STA1	15,115	115,10	10s-20s	min_v=1, max_v=5
STA2	20,130	150,10	30s-60s	min_v=5, max_v=10
STA3	140,10	15,120	25s-60s	min_v=2, max_v=7

Table 2: Mobility of the Nodes

2.2 Results

To commence our test, we prepare the python script for the network topology with all necessary libraries then run it with Mininet on the Linux terminal. The command `sudo ./<pycode filename>` is used to start the network using the configuration from the python script on Mininet. Mininet creates the SDN elements such as the controller, stations, and access points and sets up the topology as described in the python script.

The controller used in this task is the default one used by Mininet since it is not specified. The access points are started and operate based on instructions from the controller.
Fig. 5 & 6 show the Mininet GUI view of the topology prior mobility and after mobility respectively. All stations are within strong WiFi signal coverage and successfully communicate with each other as shown in Fig. 8. The `ping -c 3` command was utilised to send out three Internet Control Message Protocol(ICMP) packets between specified stations to determine full connectivity within the network. The results show all three ICMP packets were transmitted and received successfully, no packet loss, and took an average time of 2000 milliseconds for all communications.

Prior mobility, we see that stations 1 & 2 would likely be associated with AP1 and station 3 with AP5 due to their close proximity to these APs. During mobility, the stations will lose association, go through a handoff operation, then get associated with the AP in close proximity after mobility. To verify the APs connected to each station, we use the `<station name> iwconfig` to pull the wireless interface information for each station or mobile node.
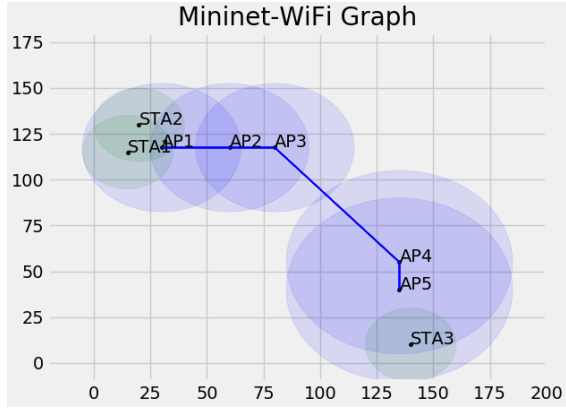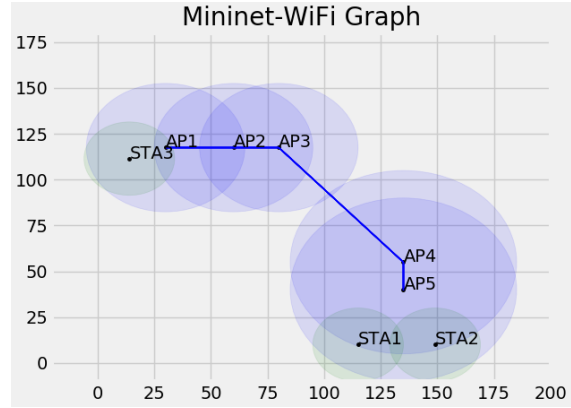
Figure 5: Prior Mobility

Figure 6: After Mobility

Figure 7: APs connected after mobility

Figure 8: Successful Connectivity

2.3   Analysis

3   Task 2 - Adhoc Network Simulation
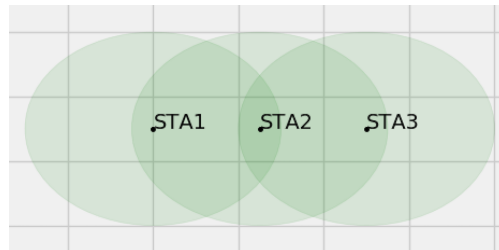
Figure 9: Adhoc Network with 3 Nodes

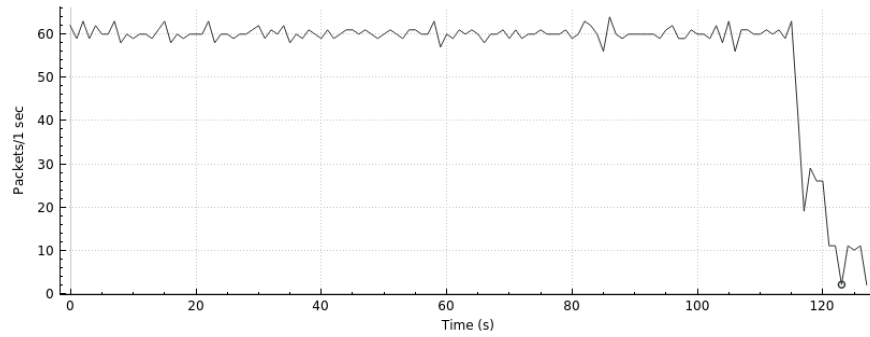| NAME	IPv6	MAC	POSITION	RANGE	A_HEIGHT	A_GAIN	SSID	HT_CAP
STA1	2024::11	00:00:00:00:01:11	20,10,0	30	1	5	adhocUH	HT40+
STA2	2024::12	00:00:00:00:01:12	45,10,0	30	2	6	adhocUH	HT40+
STA3	2024::13	00:00:00:00:01:13	70,10,0	30	3	7	adhocUH	HT40+

Table 3: Adhoc Nodes Information

3.1 Results

Figure 10: Throughput using BATMAN_ADV protocol

"Node: STA1"

| Last Reset Time | 2024-11-18 | 17:41:12,163259 1731951672,163259 |
| Current Time | 2024-11-18 | 17:41:12,163339 1731951672,163339 |

Counter Name	Periodic value	Cumulative value
Elapsed Time	00:00:00:000000	00:02:03:937000
Call Rate	0,000 cps	9,497 cps
Incoming call created	0	1177
OutGoing call created	0	0
Total Call created		1177
Current Call	0	0
Successful call	0	1177
Failed call	0	0
Response Time 1	00:00:00:000000	00:00:00:015000
Call Length	00:00:00:000000	00:00:04:029000

------------------ Test Terminated ------------------

"Node: STA2"

Call-rate(length)	Port	Total-time	Total-calls	Remote-host
10,0(0 ms)/1,000s	5060	201,96 s	1207	10,0,0,1:5060(UDP)

0 new calls during 1.001 s period   1 ms scheduler resolution
0 calls (limit 30)   Peak was 30 calls, after 120 s
0 Running, 2 Paused, 4 Woken up
0 dead call msg (discarded)   0 out-of-call msg (discarded)
3 open sockets

		Messages	Retrans	Timeout	Unexpected-Msg
INVITE ---------->		1207	150	30	
100 <----------		0	0	0	0
180 <----------		1176	0	0	0
183 <----------		0	0	0	0
200 <----------	E-RTD1	1177	0	0	0
ACK ---------->		1177	0		
Pause [0ms]		1177		0	
BYE ---------->		1177	0		
200 <----------		1177	0	0	0

------------------ Traffic Paused - Press [p] again to resume ------------------

Wireshark IO Graphs: STA2-wlan0

Figure 11: Throughput using BATMAND protocol
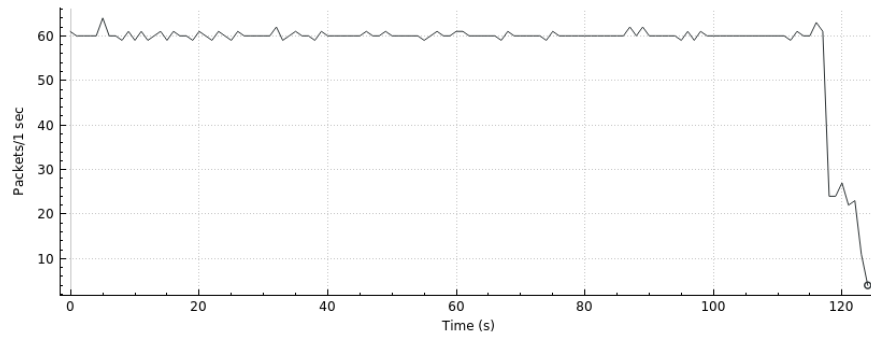
Wireshark IO Graphs: STA2-wlan0

Figure 12: Throughput using OLSRD protocol

3.2   Analysis

For this experiment, we aim to establish a successful connection between the closest stations in the setup i.e. between STA1 and STA2 or STA2 and STA3 and test out a VoIP connection between the stations under three different MANET protocols. From our test results, we observe that the different MANET protocols tested have identical similarity in terms of the average packets sent per second achieved during the VoIP connection for same amount of time.

4   Task 3 - SDN Simulation

This task activity utilises the concept of Software Defined Networking with an Openflow on the local machine to manage our network.

The network comprises five interconnected switches, three servers in an old building, and access to two hosts in a new building at the University of Hertfordshire. The table below contains the configuration information for hosts and servers in the network.

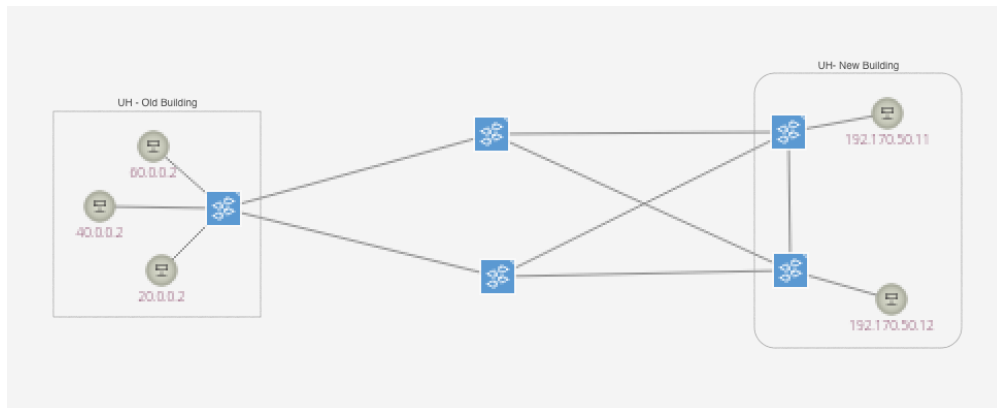| NAME	IPv4	MAC ADDRESS
H1	192.170.50.11	00:00:00:00:15:98
H2	192.170.50.12	00:00:00:00:15:99
SERVER1	20.0.0.2	00:00:00:00:16:00
SERVER2	40.0.0.2	00:00:00:00:16:01
SERVER3	60.0.0.2	00:00:00:00:16:02

Table 4: Network Configuration

Figure 13: Building Plan

4.1 Results

After setting up and running the network on mininet and configuring static routes to various networks on each host, we check the connectivity of hosts in the design. The following images are results from a 'pingall' command in mininet and server-client UDP connection for 600s at a bandwidth of 100Mbsec.

Figure 14: Full ping connectivity & Server-Client UDP connection
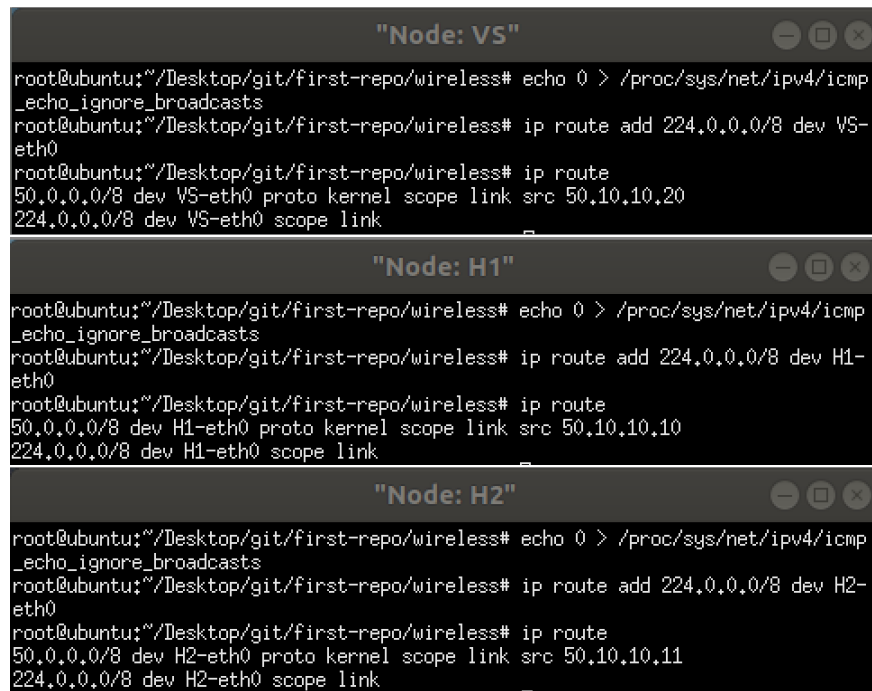
4.2 Analysis

5 Task 4 - Multicast Video Stream Service

NAME	IP ADDRESS
H1	50.10.10.10
H2	50.10.10.11
H3	50.10.10.12
VS	50.10.10.20

Table 5: Network Configuration

Figure 15: Activating multicast for required nodes
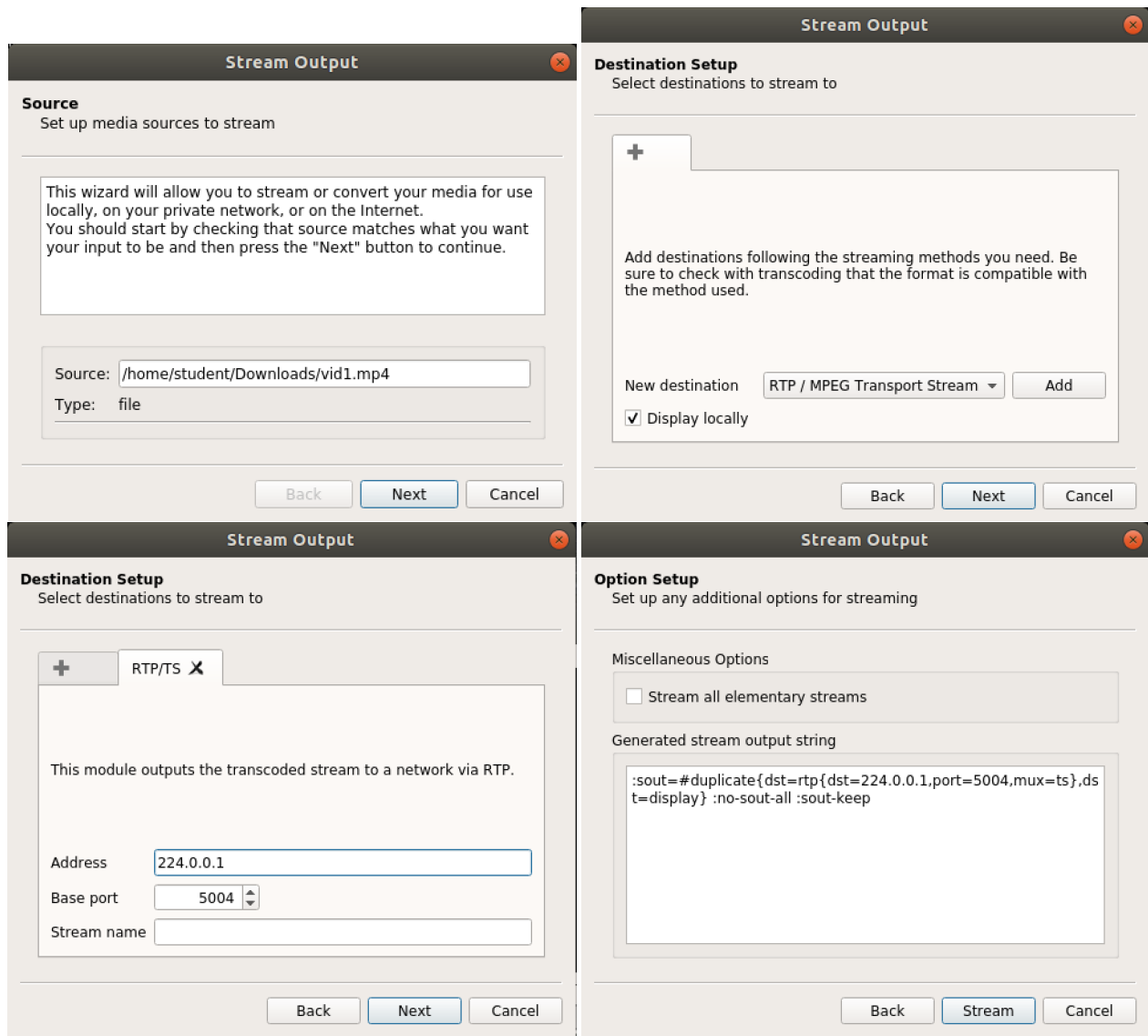
Figure 16: Steps for multicast video streaming

5.1 Results
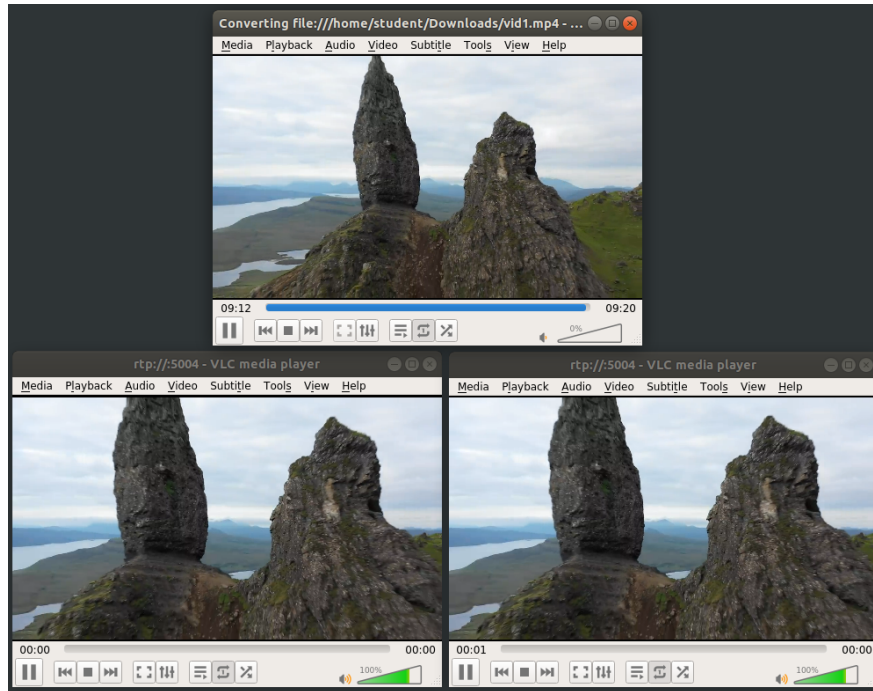
Figure 17: Multicast video stream from source to hosts

5.2   Analysis

6 References

References

[1] Rogério Leão Santos de Oliveira, Christiane Marie Schweitzer, Ailton Akira Shinoda, and Ligia Rodrigues Prete. Using mininet for emulation and prototyping software-defined networks. pages 1–6, 2014.

[2] Saad H. Haji, Subhi R. M. Zeebaree, Rezgar Hasan Saeed, Siddeeq Y. Ameen, Hanan M. Shukur, Naaman Omar, Mohammed A. M. Sadeeq, Zainab Salih Ageed, Ibrahim Mahmood Ibrahim, and Hajar Maseeh Yasin. Comparison of software defined networking with traditional networking. *Asian Journal of Research in Computer Science*, 9(2):1–18, May 2021.

[3] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[4] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy. Network innovation using openflow: A survey. *IEEE Communications Surveys gnore s Tutorials*, 16(1):493–512, 2014.

[5] Stephen Lukasik. Why the arpanet was built. *IEEE Annals of the History of Computing*, 33(3):4–21, 2011.

[6] Koketso Molemane Rodney Mokoena, Ramahlapane Lerato Moila, and Prof Mthulisi Velempini. Improving network management with software defined networking using openflow protocol. In *2023 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*, pages 1–5, 2023.

7 Appendix

a Task 1 code

```
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():
    info("**creating network**\n")
    net = Mininet_wifi()

    info("**creating nodes**\n")
    ap1 = net.addAccessPoint('ap1', mac='00:00:00:00:00:00', ssid='AP1', mode
        ='g', channel='1', position='30,117.5', band='5', range=35)
    ap2 = net.addAccessPoint('ap2', mac='00:00:00:00:00:01', ssid='AP2', mode
        ='g', channel='1', position='60,117.5', band='5', range=35)
    ap3 = net.addAccessPoint('ap3', mac='00:00:00:00:00:02', ssid='AP3', mode
        ='g', channel='1', position='80,117.5', band='5', range=35)
    ap4 = net.addAccessPoint('ap4', mac='00:00:00:00:00:03', ssid='AP4', mode
        ='g', channel='1', position='135,55', band='5', range=50)
    ap5 = net.addAccessPoint('ap5', mac='00:00:00:00:00:04', ssid='AP5', mode
        ='g', channel='1', position='135,40', band='5', range=50)
    sta1 = net.addStation('iphone5', mac='00:00:00:00:00:10', ip
        ='192.168.50.11/24', position='15,115', range=20, min_v=1, max_v=5)
    sta2 = net.addStation('ipad', mac='00:00:00:00:00:11', ip
        ='192.168.50.12/24', position='20,130', range=20, min_v=5, max_v=10)
    sta3 = net.addStation('macbook', mac='00:00:00:00:00:12', ip
        ='192.168.50.13/24', position='140,10', range=20, min_v=2, max_v=7)
    c0 = net.addController('c0')
    net.setPropagationModel(model="logDistance", exp=5)

    info("**configuring wifi nodes and mobility**\n")
    net.configureWifiNodes()
    net.addLink(ap1, ap2)
    net.addLink(ap2, ap3)
    net.addLink(ap3, ap4)
    net.addLink(ap4, ap5)
    net.plotGraph(min_x=-20, min_y=-10, max_x=200, max_y=180)

    net.startMobility(time=0)
    net.mobility(sta1, 'start', time=10, position='15,115')
    net.mobility(sta1, 'stop', time=20, position='115,10')
    net.mobility(sta2, 'start', time=30, position='20,130')
```

```python
    net.mobility(sta2, 'stop', time=60, position='150,10')
    net.mobility(sta3, 'start', time=25, position='140,10')
    net.mobility(sta3, 'stop', time=60, position='15,110')
    net.stopMobility(time=120)

    info("**starting network**\n")
    net.build()
    c0.start()
    ap1.start([c0])
    ap2.start([c0])
    ap3.start([c0])
    ap4.start([c0])
    ap5.start([c0])

    info("**running CLI**\n")
    CLI(net)

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

if __name__ == '__main__':
    setLogLevel('info')
    plot = False if '-p' in sys.argv else True
    topology()
```

b   Task 2 code

```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")
    sta1 = net.addStation('STA1', ip6='2024::11', mac='00:00:00:00:01:11',
        position='20,10,0', range=30, antennaGain=5, antennaHeight=1)
    sta2 = net.addStation('STA2', ip6='2024::12', mac='00:00:00:00:01:12',
        position='45,10,0', range=30, antennaGain=6, antennaHeight=2)
    sta3 = net.addStation('STA3', ip6='2024::13', mac='00:00:00:00:01:13',
        position='70,10,0', range=30, antennaGain=7, antennaHeight=3)
    net.setPropagationModel(model="logDistance", exp=4)

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

    info("*** Creating links\n")
    #testing 'batman_adv', 'batmand', 'olsrd' manet protocol
    net.plotGraph(min_x=-20, min_y=-40, max_x=120, max_y=90)
    net.addLink(sta1, cls=adhoc, intf='STA1-wlan0', ssid='adhocUH', mode='g',
        channel=5, ht_cap='HT40+', proto='batman_adv')
    net.addLink(sta2, cls=adhoc, intf='STA2-wlan0', ssid='adhocUH', mode='g',
        channel=5, ht_cap='HT40+', proto='batman_adv')
    net.addLink(sta3, cls=adhoc, intf='STA3-wlan0', ssid='adhocUH', mode='g',
        channel=5, ht_cap='HT40+', proto='batman_adv')

    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 )
```

c 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)

                h1 = self.addHost( 'H1', mac='00:00:00:00:15:98', ip
                    ='192.170.50.11/24' )
                h2 = self.addHost( 'H2', mac='00:00:00:00:15:99', ip
                    ='192.170.50.12/24' )
                SERVER1 = self.addHost( 'Sv1', mac='00:00:00:00:16:00', ip
                    ='20.0.0.2/8' )
                SERVER2 = self.addHost( 'Sv2', mac='00:00:00:00:16:01', ip
                    ='40.0.0.2/8' )
                SERVER3 = self.addHost( 'Sv3', mac='00:00:00:00:16:02', 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 )
```

```
            self.addLink( h1, Switch4 )
            self.addLink( h2, Switch5 )
            self.addLink( SERVER1, Switch1 )
            self.addLink( SERVER2, Switch1 )
            self.addLink( SERVER3, Switch1 )
            self.addLink( Switch1, Switch2 )
            self.addLink( Switch1, Switch3 )
            self.addLink( Switch2, Switch4 )
            self.addLink( Switch2, Switch5 )
            self.addLink( Switch3, Switch4 )
            self.addLink( Switch3, Switch5 )
            self.addLink( Switch4, Switch5 )

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

d   Task 4 code

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node
from mininet.log import setLogLevel, info
from mininet.cli import CLI


class MyTopo( Topo ):
    "Simple topology example."
    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # hosts and switches
        h1 = self.addHost('H1', ip='50.10.10.10/8')
        h2 = self.addHost('H2', ip='50.10.10.11/8')
        h3 = self.addHost('H3', ip='50.10.10.12/8')
        h4 = self.addHost('vidSrc', ip='50.10.10.20/8')
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')

        self.addLink(h1, s2)
        self.addLink(h2, s2)
        self.addLink(h3, s2)
        self.addLink(s1, s2)
        self.addLink(h4, s1)

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