DOI : https://doi.org/10.56452/110

ISSN: 0974-5823 Vol. 7 (Special Issue 5, December 2022)

International Journal of Mechanical Engineering

Secure Live Video Streaming, Network Slicing in an Emulated Network Environment

Shikha (M.Tech)[1], **Dr. Randeep Singh** (Professor)[2], **Ravinder Singh Madhan** (Asst. Professor)[3]

IEC University Department of Computer Science

Pinjore-Nalagarh Highway, District:- Solan, Himachal Pradesh

shikha.bhutani05@gmail.com[1], randeeppoonia@gmail.com[2], ravimadhan@gmail.com[3]

Abstract— **Network slicing allows many networking platforms to share the same underlying physical infrastructure. It's a diagram that shows how various components of the network are linked to various services and databases. The current research examines the potential benefits and drawbacks of slicing and virtualizing 5G networks and their functions. SDN, NFV, fog computing, and VMs are just some of the technologies that researchers are looking into in relation to 5G network slicing. Network slicing is the practise of establishing and managing many logical networks on a single physical infrastructure; it was first seen in 5G mobile communication networks. Technology advancements made possible by 5G, such as network slicing, will allow for resource sharing between the network's many segments. Through the network as infrastructure and subsequently into the network as a set of services, this technology enables the delivery of smart, critical, multiservice with various requirements. In this dissertation, we introduce a scalable framework for analyzing the efficacy of software-defined networking (SDN) multicast architectures and algorithms for real-time video streaming in comparison to traditional IP (unicast) methods. We use real HD video content to stream over a simulated network while we collect data on how the network and applications perform. SDN, or software-defined networking, is an up-and-coming method for network programmability that allows for the dynamic initialization, control, modification, and management of network behaviour through the use of standard interfaces. SDN's ascent allows for the possibility of delivering services like inter-domain network layer multicast for real-time video streaming, which are currently hampered by the Internet's legacy architecture's inflexibility and insularity. In this paper, we propose a framework for analyzing and contrasting the capabilities of SDN-based multicast topologies for real-time streaming with those of traditional IP unicast. In this work we used real HD video content to stream over a simulated network while we collect data on how the network and applications perform.**

Keywords*— *SDN, Live Video Streaming, Secure Live Video Streaming

1. Introduction

Providing real-time multimedia services over the Internet has been made possible because to recent advancements in computer and compression technology, as well as high-bandwidth storage devices and fast networks. As the name suggests, real-time multimedia is constrained by a sense of time. Audio and video files, for example, must be played back indefinitely. The play-out process will pause if the data is not received in a timely manner, which is inconvenient for human ears and sight. Using SDN, the control and forwarding planes are separated, making it possible to programme the network control directly and to abstract the underlying infrastructure for use by network applications. Network traffic can be monitored, forwarded and optimized at runtime by a centralized controller that has a global view of the network. Using this flexibility and control, services like inter-domain multicast can be provided. Because 40% of Internet video traffic is live, multicasting can save a lot of system and network resources. For some time now, it has been advocated to use SDN-based multicast frameworks to provide content and/or network providers with the control they need to implement a streaming video service. It has been increasingly difficult to maintain a high quality of service (QoS) in wireless networks during the past few decades. As a starting point, the concept of 5G Network Slicing was introduced. When it comes to network slice concepts, NGMN stated that SIL, NSI, and Resource layers were all part of the overall concept. "A Service Instance is a representation of a service," as stated in "A Service Instance is a service." It is also possible to use a Service Instance to represent a third-party service provided by an operator. NGMN also envisaged that network operators would make use of a NetworkSlice Blueprint in order to construct an NSI (NSB). NSIs are responsible for providing the network properties required by a Service Instance. It is possible for a network operator to share an NSI with many Service Instances. An NSI is a "collection of network functions, and resources to run these network functions, forming a complete instantiated logical network," according to the Service Instance(s). The Network Slice Blueprint defined all parts of the NSI's structure, configuration, and plans/workflows on how to instantiate and control the NSI. In this paper we present a platform to evaluate and compare SDN-based multicast architectures for live streaming and benchmark their performance against standard IP unicast.

2. Literature Review

Ibrahim Afolabi et. All (2018) With regard to the current state of 5G network slicing maturity, this essay takes a close look. For those who want to learn more about the history and evolution of network slicing, this article is for you. Use of NFV, SDN and cloud computing technologies allows for network slicing (cloud computing). According to this article, a comprehensive evaluation of 5G's impact on the radio access network (RAN), the core network (CN), and transport networks (TN) is offered. For example, by displaying key activities related to orchestration of end-to-end network slices and management, we may showcase the importance of network slicing as a major enabler for 5G Slicing the RAN and core networks, as well as presenting instances of how this might be done in practice, is explained in this survey. Finally, we examine and address open research issues associated with the deployment of 5G mobile networks' end-to-end network slicing. [1]

Alcardo Alex Barakabitzea et. All (2020) SDN and NFV are being adopted at an unprecedented rate by both academia and industry as solutions to overcome the management and orchestration of resources challenge in 5G networks and suit the requirements of different verticals. It is hoped that SDN and NFV would enable future 5G network control and administration to be programmable, cost-effective, and adaptive. Because of its central role in addressing increasingly stringent and business-critical vertical industry requirements, network slicing is at the heart of 5G and will play an important part in meeting ELAs/SLAs that are more stringent as well as business-critical. Using SDN and NFV, we present a complete overview of the current state-of-the-art technologies for network slicing in 5G networks. Before discussing 5G service quality and business requirements, we describe the basics, history, and many use cases of 5G network softwarization and slicing. [2]

P. Iyyanar et. All (2012) Live or pre-recorded video is the most significant component of real-time multimedia networking. When using streaming technology, the video file does not have to be downloaded in its entirety; instead, the video file is played out while the video file's contents are received and decoded. In order to stream video over the internet, video data compression and the construction of communication protocols are necessary. A low-bit-rate coder and UDP encryption are two possible solutions to this challenge. The creation of a new transport protocol could also be accomplished through the use of video coding. High levels of security are required for streaming applications. This study describes an efficient approach for encrypting video communications over un-trusted client–server networks. [3]

Huifen Huang et. All (2020) Quality of service (QoS) must be ensured for online video applications by avoiding network congestion, which increases packet loss and transmission latency. In SDN, the controller can readily receive information on the network's topology and connection bandwidth consumption. Based on the aforementioned benefits, this paper presents an SDN-based video multicast routing solution named as CAVM. CAVM collects network topology data, monitors bandwidth utilization, and calculates connection delays using Open-Flow, a popular SDN southbound interface. In this research, an SDN-based multicast routing solution for online video applications is provided. To keep track of the network and determine link delays, open-flow is employed. This paper introduced the DCMCCMR problem as well as a function for assessing congestion levels. [4]

Xenofon Foukas et. All (2017) The new 5G networks, which are designed to be multiservice environments capable of dynamically launching applications with a wide range of performance needs, can be used by mobile network operators, verticals, and OTT service providers. Flexible mobile network virtualization is essential to achieve this goal at a fair price. When it comes to mobile core virtualization, RAN virtualization is a relatively new concept. On the fly virtualization of base stations, flexible modification of slices to meet particular service requirements, and use in an end-to-end network slicing environment are all features of the Orion RAN slice system introduced here. Orion ensures functional and performance separation while allowing for maximum utilization of RAN resources among the slices. [5]

Rafael Montero et. All (2019) QoE/QoS-guaranteed network slicing is a key enabler of the next generation of 5G wireless networks. Many concerns must be addressed in order to ensure efficient service delivery at the end-to-end level, notwithstanding this. It is necessary for network slice installations that contemplate operation across numerous domains and network segments to incorporate features such as inter-domain settings and continuous monitoring in order to achieve and maintain their promised Key Performance Indicators (KPIs). It's imperative that optical networks be used in this case since they allow the connecting of various distant segments and Points of Presence (PoPs). This paper presents a design for 5G service chaining network slice provisioning in multi-segment/multi-domain optical network scenarios. The provided end-to-end (E2E) network slice can be monitored and controlled depending on policy. [6]

Christelle Al Hasrouty et .all (2018) Video conferences place a significant load on the access network due to the sheer volume of video data they generate. Upgrades to network operators' video conferencing systems save considerable bandwidth, allowing them to serve more customers while simultaneously improving their quality of service (QoE). New alternatives for better video management are becoming accessible with the introduction of software-defined networking (SDN). There are various ways to cut bandwidth consumption in the core network while still delivering the best possible visual quality to end users in this research, which takes advantage of SDN-enabled networks. Video streams are tailored to the user's access network characteristics while being sent via multicast trees via SVC layer changes in the network. According to testing, our strategy saves more bandwidth and increases network support for the number of simultaneous calls. [7]

R. Pereiraa et. All (2016) Technology-mediated communication is now widely used due to advancements in a variety of related fields such as processing, storage, networking, compression, and portable computing. There are now more ways than ever before for people to interact with each other and conduct business online, and this has had a profound impact on the way people live their daily lives. Media-rich networked and mobile applications are the driving force behind this paradigm shift in global lifestyles. To some degree or another, video streaming is used in everything from movie and TV streaming to videoconferencing to education to video telephony to gaming and social networking. [8]

Alexander Hefele et. All (2020) This study presents an ultra-reliable low-latency communication (URLLC) module for the 5G mobile backhaul. In this proposal, URLLC communications will be implemented via SDN-based network slicing. Network slicing is detailed in detail in this study, including the components and techniques involved. A hypothetical Mobile Backhaul network will be used to showcase the concept. To enable mobile backhaul network slicing, SDN has been examined as a potential solution. Measurement and monitoring tools can be used for OpenFlow-enabled networks, according to this article. URLLC group members are assigned trustworthy and low-latency pathways based on the acquired data. The controller application is tested on congested network links. As the URLLC group's packets get overloaded, SDN-managed Backhaul networks are able to provide alternative paths for their traffic. [9]

En-zhong et. All (2016) The current standard video conferencing link is now bridged by an MCU (multipoint control unit), which can cause substantial delays and communication bottlenecks. Allows for the creation of video conferencing systems that are more scalable, controllable, and flexible thanks to software-defined networking (SDN). A video coding method known as scalable video coding (SVC) can help. In this study, an SVC-enabled multicasting video conferencing architecture is suggested that eliminates the existing IGMP and MCU protocols. A variety of conference terminals, each with a different set of capabilities, rely on SVC multicast streaming to suit the system's requirements. A video feed's several levels must be actively managed and controlled by the SDN controller in the event of network congestion. Managing the conference attendees is made easier by the presence of a conference manager. [10]

Qiang Gao1 et. All (2015) In distributed shared memory systems, synchronization strategies such as broadcast or multicast are frequently employed. A lack of transparency about the time spent synchronizing is caused by interrupting unrelated activities in the same multicast group at the same time. For distributed shared memory, we describe a dynamic SDN multicast technique that utilizes an SDN multicast forwarding rule and a dynamic core-based tree algorithm. Experiments with the prototype system in a homogenous testing environment demonstrate that it is capable of delivering better response times and process planning than a traditional Ethernet-based solution. [11]

Shalitha Wijethilaka et. All (2021) The Internet of Things (IoT) is an emerging technology that can be used in a variety of ways to improve people's lives. Over the next several years, 5G wireless networks will be used to connect IoT devices. Since the Internet of Things is expected to grow rapidly, it has been designed with this in mind (IoT). The 5G architecture relies on slicing to separate the physical network (i.e., slices) into several logical networks (i.e. networks). Consequently, network slicing is an essential part of 5G IoT development. For heterogeneous IoT applications, network slicing can be utilized to create different slices for each application. In this research, the role of network slicing in IoT implementation is thoroughly investigated. IoT applications can benefit from network slicing, as well as some of the technical difficulties that it can address. Integration concerns and unsolved research questions related to network slicing are also addressed in this paper. [12]

Alcardo Alex Barakabitzea et. All (2019) As a result of the increasing demand for high-quality services, network administration has undergone a fundamental shift in the way we manage networks. With 5G, industry and academia alike are embracing next-generation vertical applications with a wide range of service requirements. Several distinct logical networks of various sizes and designs must be created within the physical network to fulfill the specific requirements of various service types in order for this 5G network vision to be realized (e.g., a slice for massive IoT devices, smart phones or autonomous cars, etc.). [13]

Christian Koch et. All (2017) Internet service providers' networks are clogged with video traffic. Demand for more and better films is rising quickly, but the network's capacity is being constrained. Many OTT video streams are now delivered via CDNs (CDNs). IP multicast is an efficient method of delivering ISP-internal videos such as IPTV channels. On the other hand, there are so many OTT videos and their popularity is so widely distributed that this strategy is unsuccessful. Using SDN-based multicast, ISPs can offer content based on network-level multicast. The technology has yet to be put to use in OTT video-on-demand (VoD) applications. This paper contributes in three different ways. [14]

Ahmed Khalid et. All (2017) Using software-defined networking (SDN), it is now possible to deliver network layer multicast for real-time video streaming. SDN-based architecture, mCast, is proposed in this paper as a means of reducing network and system resources for ISPs and content delivery networks (CDNs) (CDNs). In order to support mCast, we propose a communication mechanism between Internet service providers and content delivery networks. Because of mCast's transparency, CDNs are kept in check on user sessions. We created a testbed to allow us to conduct a full analysis and comparison. mCast, a new, scalable architecture for live streaming, is the subject of this article. mCast provides a framework for communication between ISPs and CDNs. As a result of simply sending a single copy of a video channel to the ISP, CDN egress has dropped dramatically. [15]

3. Methodology

3.1 Network Slicing In An Emulated Network Environment

The project has been implemented in Linux Virtual Environment Using Virtual Box & Ubuntu 18LTS. Network simulation is achieved by using Mininet. Apart from Mininet, WireShark has been installed for network traffic analysis & VLC has been installed for Video Streaming & Network Video Playback.

An example network has been set-up comprising of transport network having 08Nos Routers, 02Nos Edge Network Having 01Nos Switch & 01Nos Edge Node, 02Nos Access Networks each having 01Nos Switch & 02Nos Access Nodes. Following Nomenclature has been assigned to various network nodes.

S.No	Type	Name	Subset	Links
1.	Router	R1	Transport Network	R1-R2
2.	Router	R2	Transport Network	R2-R1, R2-R3, R2-R6
3.	Router	R3	Transport Network	R3-R2, R3-R4
4.	Router	R4	Transport Network	R4-R3, R4-R6, R4-R5

5.	Router	R5	Transport Network	R5-R4, R5-R7, R5-R8
6.	Router	R6	Transport Network	R6-R2, R6-R4, R6-R7
7.	Router	R7	Transport Network	R7-R6, R7-R5
8.	Router	R8	Transport Network	R5-R8
9.	Switch	S1	Access Network – 1	S1-R1, S1-H1, S1-H2
10.	Switch	S2	Access Network – 2	S2-R8, S2-H3, S2-H4
11.	Switch	S3	Edge Network – 1	S3-R1, S3-E1
12.	Switch	S4	Edge Network – 2	S4-R8, S4-E2
13.	Node	H1	Access Network – 1	H1-S1
14.	Node	H2	Access Network – 1	H2-S1
15.	Node	H3	Access Network – 2	H3-S2
16.	Node	H4	Access Network – 2	H4-S2
17.	Node	E1	Edge Network - 1	E1-S3
18.	Node	E2	Edge Network - 2	E2-S4

Table- 3.1 Network Components

Apart from these nodes, an Open Flow Switch Controller C0 is employed which is global to the network & is responsible for flow control for all of the 04Nos Switches.

Static Routing Paths have been defined from R1 to R8 and vice versa, however no direct flow control is defined for packets between S1 & S2. Open Flow Control Statements have been defined to allow traffic between each access network & its corresponding edge network. The access pattern is listed below.

S.No	**Node1**	**Node2**	**Access**
1.	H1	E1	Yes
2.	H2	E1	Yes
3.	H3	E1	No
4.	H4	E1	No
5.	H1	E2	No
6.	H2	E2	No
7.	H3	E2	Yes
8.	H4	E2	Yes
9.	E1	E2	No
10.	E2	E1	No

Table- 3.2 OpenFlow Defined Access

For Communication Between Selected Nodes of Access Network - 1 & Access Network -2, GENEVE tunnels are employed, employing the concept of Network Slicing. 02Nos of GENEVE tunnels have been defined to provide virtual encapsulation between selected nodes of both access networks. The access pattern is listed below.

S.No	Node1	Node2	Access	Medium
1.	H1	H2	Yes	Local Network
2.	H1	H3	Yes	GENEVE-1
3.	H1	H4	No	No Medium
4.	H2	H1	Yes	Local Network
5.	H2	H3	No	No Medium
6.	H2	H4	Yes	GENEVE-2
7.	H3	H4	Yes	Local Network

Table- 3.3 GENEVE Tunnel Based Access

VLC Media player is employed for both streaming video from File & as a network video player. Operation of video streaming is verified by employing Node H1 as video server for Video-1, Node H3 has a separate instance of VLC running which receives and plays Video-1. Node H4 cannot access Video-1. Simultaneous separate sliced video streaming operation is verified by running another instance of VLC on Node H4 for Streaming Video-2 & receiving & playing the same on Node H2. Node H1 cannot access Video-2.

3.2 Network Architecture

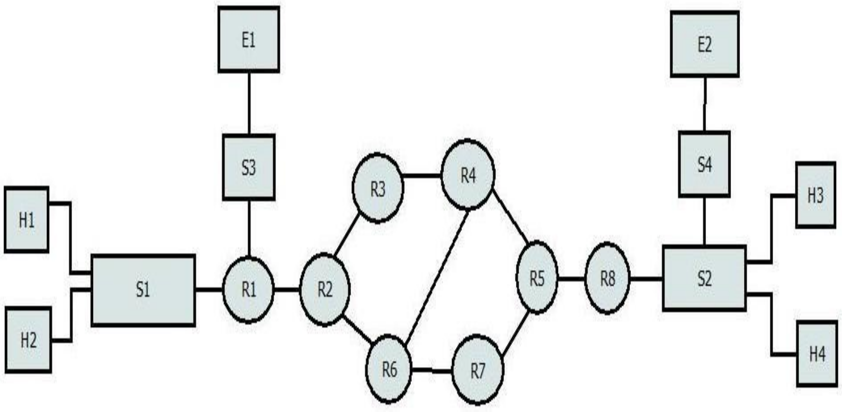
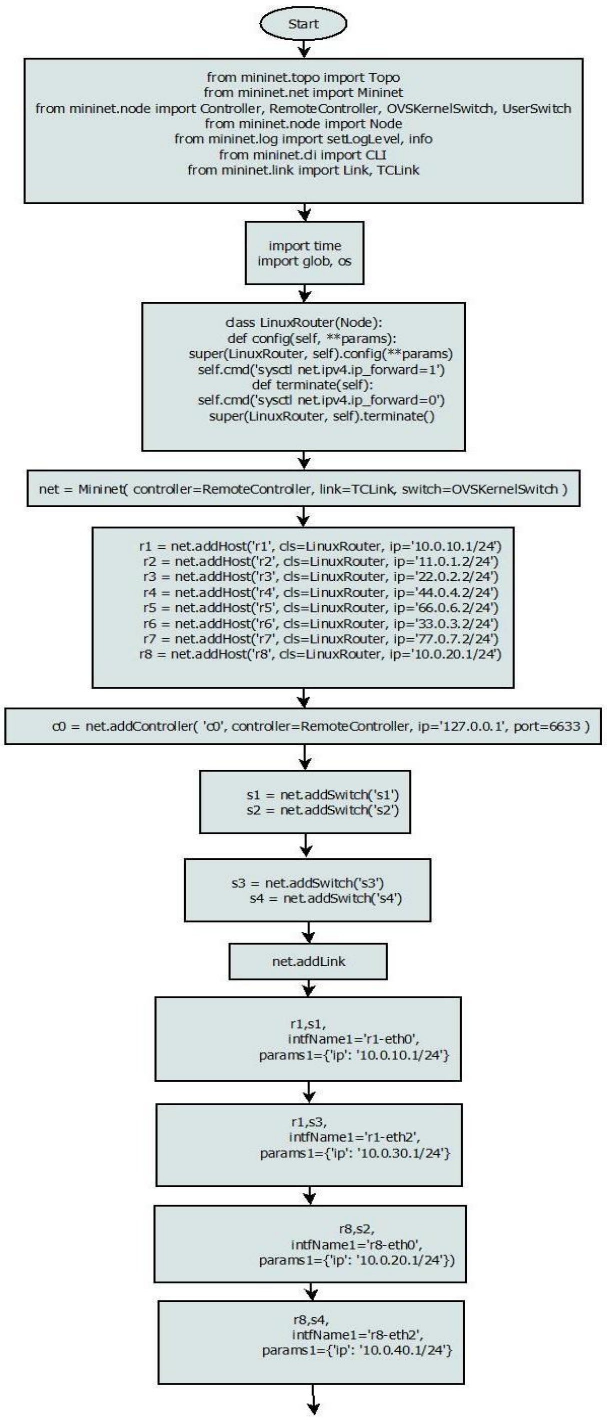


Figure: 3.1 Network Architecture

We define a linux router and network topology and define how all nodes connecting together. here we can see number of router is 8 which are on the transport network, and we have two access switch which connected to the host, we have 4 host and 2 host are edge network host there we can stream content. all connection diagram we can see in figure number 3.1, we are used edge network because we need short delay data, so here is 2 edge network, 4 host, 4 open flow controller and 8 routers. We make a protocol for encapsulate data communication between H1-H3 using geneve tunnel, data not transmitted H1 to H4 because there is no define tunnel. H1to H2 data can be transmitted because both are connected with the same switch, and H2-H3 data not transmitted because there is no connection, and data can be transmitted H2-H4 using second geneve tunnel. So here is two tunnels both tunnels streams two video.

3.3 Main Flow Chart



```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSKernelSwitch, UserSwitch
from mininet.node import Node
from mininet.log import setLogLevel, info
from mininet.cli import CLI
from mininet.link import Link, TCLink
```

```
import time
import glob, os
```

```
class LinuxRouter(Node):
def config(self, **params):
super(LinuxRouter, self).config(**params)
self.cmd('sysctl net.ipv4.ip_forward=1')
def terminate(self):
self.cmd('sysctl net.ipv4.ip_forward=0')
super(LinuxRouter, self).terminate()
```

```
net = Mininet( controller=RemoteController, link=TCLink, switch=OVSKernelSwitch )
```

```
r1 = net.addHost('r1', cls=LinuxRouter, ip='10.0.10.1/24')
r2 = net.addHost('r2', cls=LinuxRouter, ip='11.0.1.2/24')
r3 = net.addHost('r3', cls=LinuxRouter, ip='22.0.2.2/24')
r4 = net.addHost('r4', cls=LinuxRouter, ip='44.0.4.2/24')
r5 = net.addHost('r5', cls=LinuxRouter, ip='66.0.6.2/24')
r6 = net.addHost('r6', cls=LinuxRouter, ip='33.0.3.2/24')
r7 = net.addHost('r7', cls=LinuxRouter, ip='77.0.7.2/24')
r8 = net.addHost('r8', cls=LinuxRouter, ip='10.0.20.1/24')
```

```
c0 = net.addController( 'c0', controller=RemoteController, ip='127.0.0.1', port=6633 )
```

```
s1 = net.addSwitch('s1')
s2 = net.addSwitch('s2')
```

```
s3 = net.addSwitch('s3')
s4 = net.addSwitch('s4')
```

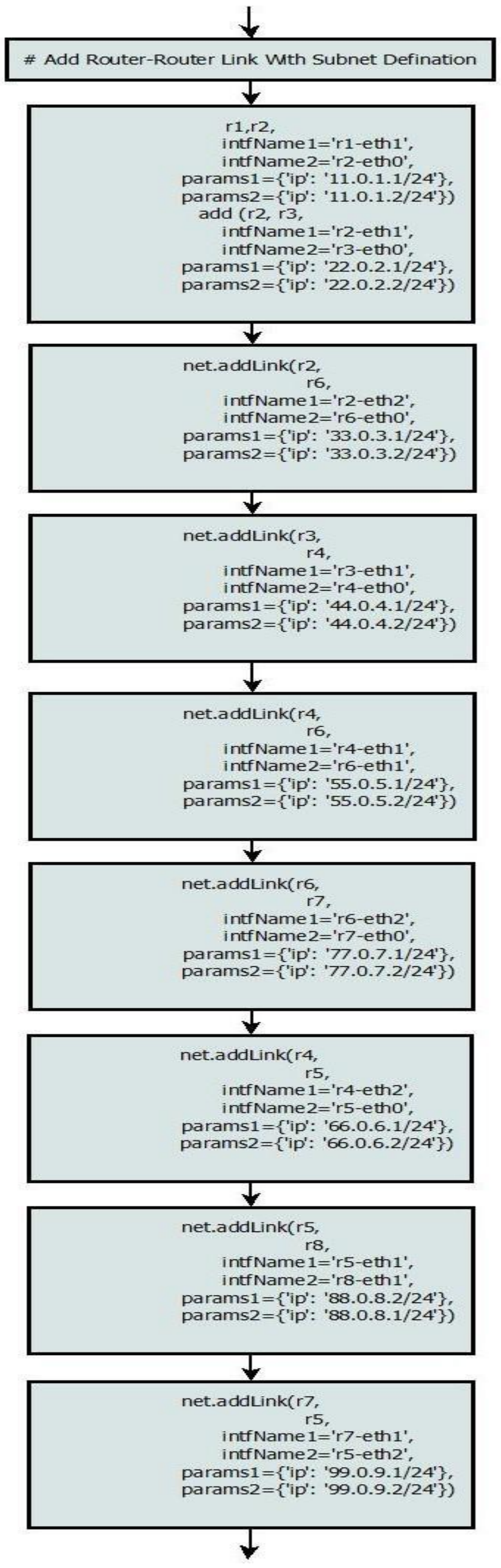
```
net.addLink
```

```
r1,s1,
intfName1='r1-eth0',
params1={'ip': '10.0.10.1/24'}
```

```
r1,s3,
intfName1='r1-eth2',
params1={'ip': '10.0.30.1/24'}
```

```
r8,s2,
intfName1='r8-eth0',
params1={'ip': '10.0.20.1/24'})
```

```
r8,s4,
intfName1='r8-eth2',
params1={'ip': '10.0.40.1/24'}
```



```
# Add Router-Router Link With Subnet Defination
```

```
r1,r2,
    intfName1='r1-eth1',
    intfName2='r2-eth0',
params1={'ip': '11.0.1.1/24'},
params2={'ip': '11.0.1.2/24'})
  add (r2, r3,
    intfName1='r2-eth1',
    intfName2='r3-eth0',
params1={'ip': '22.0.2.1/24'},
params2={'ip': '22.0.2.2/24'})
```

```
net.addLink(r2,
              r6,
    intfName1='r2-eth2',
    intfName2='r6-eth0',
params1={'ip': '33.0.3.1/24'},
params2={'ip': '33.0.3.2/24'})
```

```
net.addLink(r3,
              r4,
    intfName1='r3-eth1',
    intfName2='r4-eth0',
params1={'ip': '44.0.4.1/24'},
params2={'ip': '44.0.4.2/24'})
```

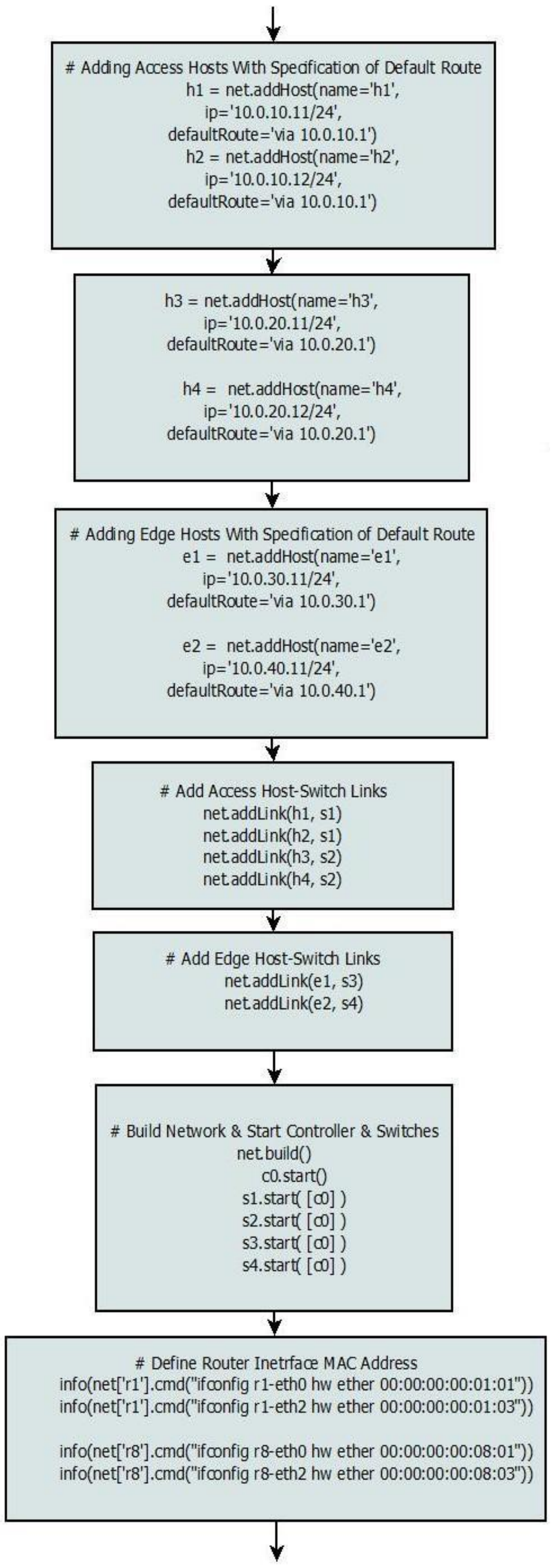
```
net.addLink(r4,
              r6,
    intfName1='r4-eth1',
    intfName2='r6-eth1',
params1={'ip': '55.0.5.1/24'},
params2={'ip': '55.0.5.2/24'})
```

```
net.addLink(r6,
              r7,
    intfName1='r6-eth2',
    intfName2='r7-eth0',
params1={'ip': '77.0.7.1/24'},
params2={'ip': '77.0.7.2/24'})
```

```
net.addLink(r4,
              r5,
    intfName1='r4-eth2',
    intfName2='r5-eth0',
params1={'ip': '66.0.6.1/24'},
params2={'ip': '66.0.6.2/24'})
```

```
net.addLink(r5,
              r8,
    intfName1='r5-eth1',
    intfName2='r8-eth1',
params1={'ip': '88.0.8.2/24'},
params2={'ip': '88.0.8.1/24'})
```

```
net.addLink(r7,
              r5,
    intfName1='r7-eth1',
    intfName2='r5-eth2',
params1={'ip': '99.0.9.1/24'},
params2={'ip': '99.0.9.2/24'})
```



```
# Adding Access Hosts With Specification of Default Route
h1 = net.addHost(name='h1',
ip='10.0.10.11/24',
defaultRoute='via 10.0.10.1')
h2 = net.addHost(name='h2',
ip='10.0.10.12/24',
defaultRoute='via 10.0.10.1')
```

```
h3 = net.addHost(name='h3',
ip='10.0.20.11/24',
defaultRoute='via 10.0.20.1')

h4 =  net.addHost(name='h4',
ip='10.0.20.12/24',
defaultRoute='via 10.0.20.1')
```

```
# Adding Edge Hosts With Specification of Default Route
e1 =  net.addHost(name='e1',
ip='10.0.30.11/24',
defaultRoute='via 10.0.30.1')

e2 =  net.addHost(name='e2',
ip='10.0.40.11/24',
defaultRoute='via 10.0.40.1')
```

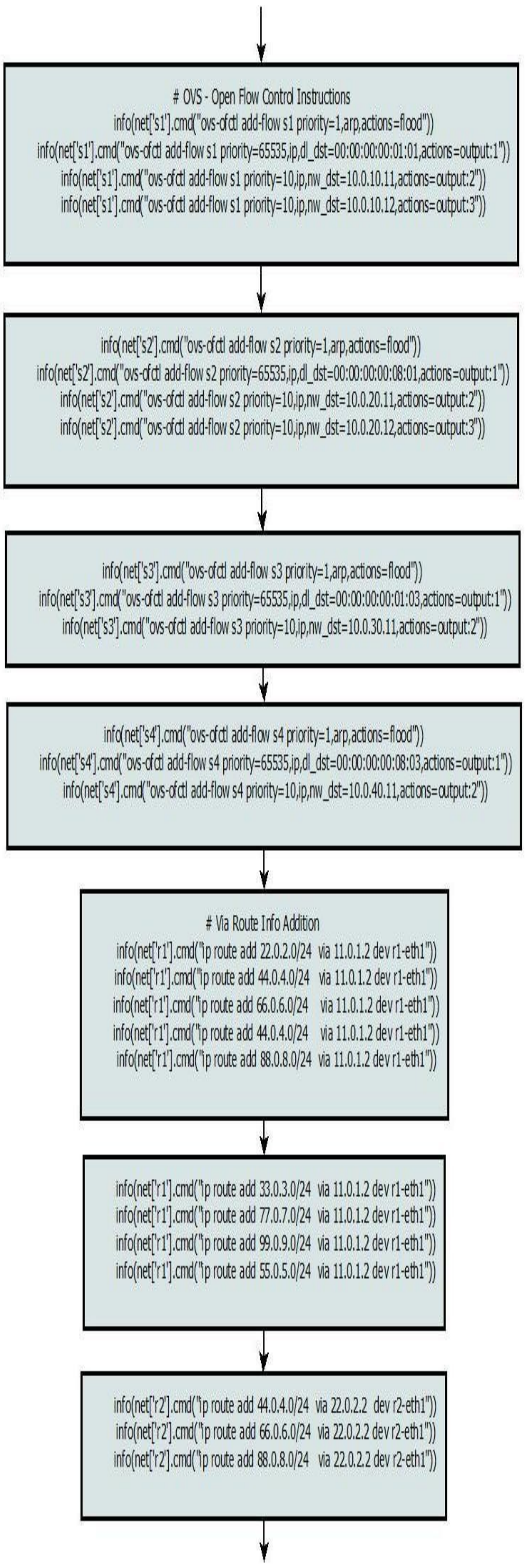
```
# Add Access Host-Switch Links
net.addLink(h1, s1)
net.addLink(h2, s1)
net.addLink(h3, s2)
net.addLink(h4, s2)
```

```
# Add Edge Host-Switch Links
net.addLink(e1, s3)
net.addLink(e2, s4)
```

```
# Build Network & Start Controller & Switches
net.build()
c0.start()
s1.start( [c0] )
s2.start( [c0] )
s3.start( [c0] )
s4.start( [c0] )
```

```
# Define Router Inetrface MAC Address
info(net['r1'].cmd("ifconfig r1-eth0 hw ether 00:00:00:00:01:01"))
info(net['r1'].cmd("ifconfig r1-eth2 hw ether 00:00:00:00:01:03"))

info(net['r8'].cmd("ifconfig r8-eth0 hw ether 00:00:00:00:08:01"))
info(net['r8'].cmd("ifconfig r8-eth2 hw ether 00:00:00:00:08:03"))
```

DOI : https://doi.org/10.56452/110



```
# OVS - Open Flow Control Instructions
info(net['s1'].cmd("ovs-ofctl add-flow s1 priority=1,arp,actions=flood"))
info(net['s1'].cmd("ovs-ofctl add-flow s1 priority=65535,ip,dl_dst=00:00:00:00:01:01,actions=output:1"))
info(net['s1'].cmd("ovs-ofctl add-flow s1 priority=10,ip,nw_dst=10.0.10.11,actions=output:2"))
info(net['s1'].cmd("ovs-ofctl add-flow s1 priority=10,ip,nw_dst=10.0.10.12,actions=output:3"))
```

```
info(net['s2'].cmd("ovs-ofctl add-flow s2 priority=1,arp,actions=flood"))
info(net['s2'].cmd("ovs-ofctl add-flow s2 priority=65535,ip,dl_dst=00:00:00:00:08:01,actions=output:1"))
info(net['s2'].cmd("ovs-ofctl add-flow s2 priority=10,ip,nw_dst=10.0.20.11,actions=output:2"))
info(net['s2'].cmd("ovs-ofctl add-flow s2 priority=10,ip,nw_dst=10.0.20.12,actions=output:3"))
```

```
info(net['s3'].cmd("ovs-ofctl add-flow s3 priority=1,arp,actions=flood"))
info(net['s3'].cmd("ovs-ofctl add-flow s3 priority=65535,ip,dl_dst=00:00:00:00:01:03,actions=output:1"))
info(net['s3'].cmd("ovs-ofctl add-flow s3 priority=10,ip,nw_dst=10.0.30.11,actions=output:2"))
```

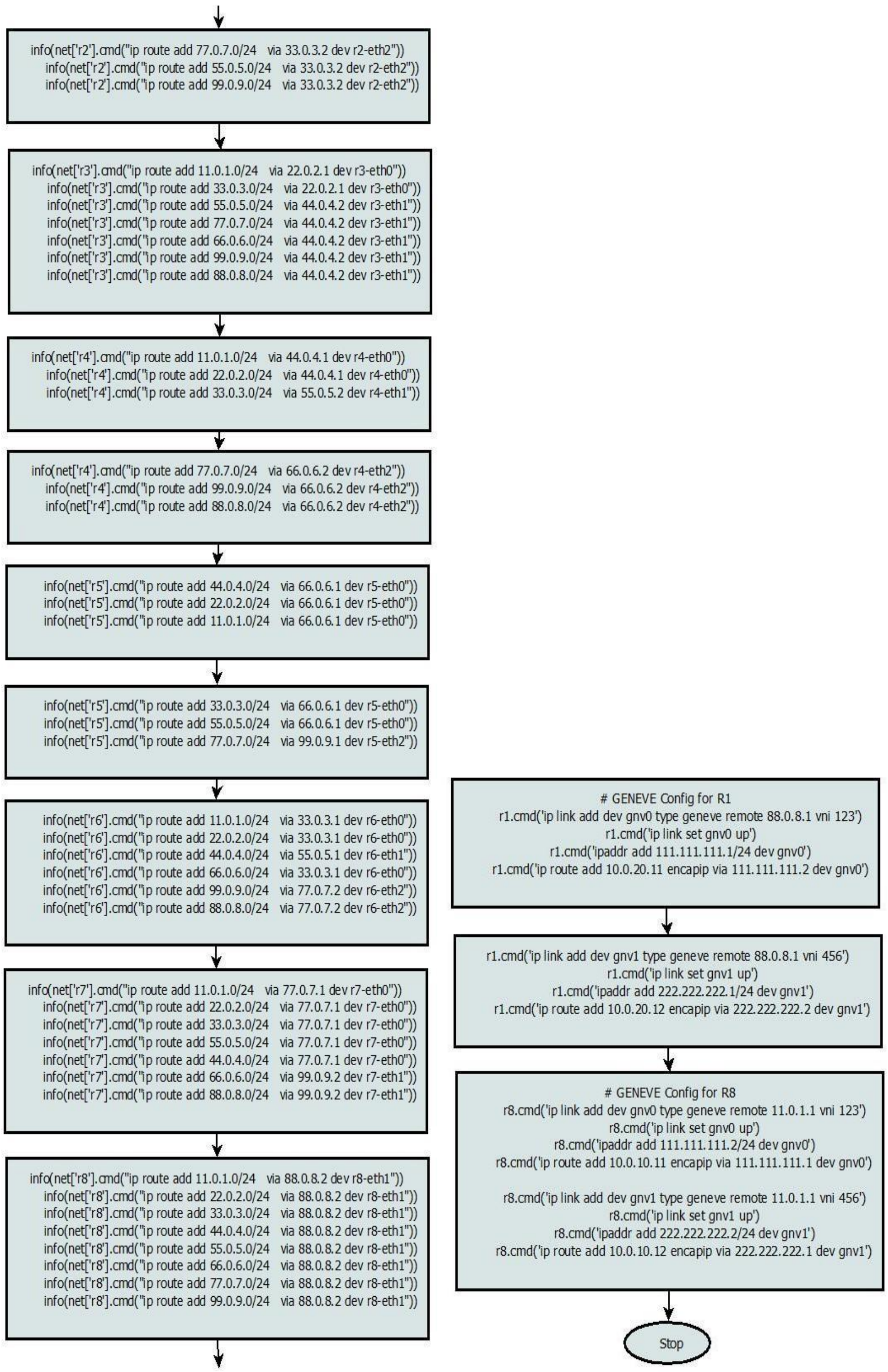
```
info(net['s4'].cmd("ovs-ofctl add-flow s4 priority=1,arp,actions=flood"))
info(net['s4'].cmd("ovs-ofctl add-flow s4 priority=65535,ip,dl_dst=00:00:00:00:08:03,actions=output:1"))
info(net['s4'].cmd("ovs-ofctl add-flow s4 priority=10,ip,nw_dst=10.0.40.11,actions=output:2"))
```

```
# Via Route Info Addition
info(net['r1'].cmd("ip route add 22.0.2.0/24  via 11.0.1.2 dev r1-eth1"))
info(net['r1'].cmd("ip route add 44.0.4.0/24  via 11.0.1.2 dev r1-eth1"))
info(net['r1'].cmd("ip route add 66.0.6.0/24  via 11.0.1.2 dev r1-eth1"))
info(net['r1'].cmd("ip route add 44.0.4.0/24  via 11.0.1.2 dev r1-eth1"))
info(net['r1'].cmd("ip route add 88.0.8.0/24  via 11.0.1.2 dev r1-eth1"))
```

```
info(net['r1'].cmd("ip route add 33.0.3.0/24  via 11.0.1.2 dev r1-eth1"))
info(net['r1'].cmd("ip route add 77.0.7.0/24  via 11.0.1.2 dev r1-eth1"))
info(net['r1'].cmd("ip route add 99.0.9.0/24  via 11.0.1.2 dev r1-eth1"))
info(net['r1'].cmd("ip route add 55.0.5.0/24  via 11.0.1.2 dev r1-eth1"))
```

```
info(net['r2'].cmd("ip route add 44.0.4.0/24  via 22.0.2.2  dev r2-eth1"))
info(net['r2'].cmd("ip route add 66.0.6.0/24  via 22.0.2.2 dev r2-eth1"))
info(net['r2'].cmd("ip route add 88.0.8.0/24  via 22.0.2.2 dev r2-eth1"))
```

DOI : https://doi.org/10.56452/110



```
info(net['r2'].cmd("ip route add 77.0.7.0/24   via 33.0.3.2 dev r2-eth2"))
info(net['r2'].cmd("ip route add 55.0.5.0/24   via 33.0.3.2 dev r2-eth2"))
info(net['r2'].cmd("ip route add 99.0.9.0/24   via 33.0.3.2 dev r2-eth2"))
```

```
info(net['r3'].cmd("ip route add 11.0.1.0/24   via 22.0.2.1 dev r3-eth0"))
info(net['r3'].cmd("ip route add 33.0.3.0/24   via 22.0.2.1 dev r3-eth0"))
info(net['r3'].cmd("ip route add 55.0.5.0/24   via 44.0.4.2 dev r3-eth1"))
info(net['r3'].cmd("ip route add 77.0.7.0/24   via 44.0.4.2 dev r3-eth1"))
info(net['r3'].cmd("ip route add 66.0.6.0/24   via 44.0.4.2 dev r3-eth1"))
info(net['r3'].cmd("ip route add 99.0.9.0/24   via 44.0.4.2 dev r3-eth1"))
info(net['r3'].cmd("ip route add 88.0.8.0/24   via 44.0.4.2 dev r3-eth1"))
```

```
info(net['r4'].cmd("ip route add 11.0.1.0/24   via 44.0.4.1 dev r4-eth0"))
info(net['r4'].cmd("ip route add 22.0.2.0/24   via 44.0.4.1 dev r4-eth0"))
info(net['r4'].cmd("ip route add 33.0.3.0/24   via 55.0.5.2 dev r4-eth1"))
```

```
info(net['r4'].cmd("ip route add 77.0.7.0/24   via 66.0.6.2 dev r4-eth2"))
info(net['r4'].cmd("ip route add 99.0.9.0/24   via 66.0.6.2 dev r4-eth2"))
info(net['r4'].cmd("ip route add 88.0.8.0/24   via 66.0.6.2 dev r4-eth2"))
```

```
info(net['r5'].cmd("ip route add 44.0.4.0/24   via 66.0.6.1 dev r5-eth0"))
info(net['r5'].cmd("ip route add 22.0.2.0/24   via 66.0.6.1 dev r5-eth0"))
info(net['r5'].cmd("ip route add 11.0.1.0/24   via 66.0.6.1 dev r5-eth0"))
```

```
info(net['r5'].cmd("ip route add 33.0.3.0/24   via 66.0.6.1 dev r5-eth0"))
info(net['r5'].cmd("ip route add 55.0.5.0/24   via 66.0.6.1 dev r5-eth0"))
info(net['r5'].cmd("ip route add 77.0.7.0/24   via 99.0.9.1 dev r5-eth2"))
```

```
info(net['r6'].cmd("ip route add 11.0.1.0/24   via 33.0.3.1 dev r6-eth0"))
info(net['r6'].cmd("ip route add 22.0.2.0/24   via 33.0.3.1 dev r6-eth0"))
info(net['r6'].cmd("ip route add 44.0.4.0/24   via 55.0.5.1 dev r6-eth1"))
info(net['r6'].cmd("ip route add 66.0.6.0/24   via 33.0.3.1 dev r6-eth0"))
info(net['r6'].cmd("ip route add 99.0.9.0/24   via 77.0.7.2 dev r6-eth2"))
info(net['r6'].cmd("ip route add 88.0.8.0/24   via 77.0.7.2 dev r6-eth2"))
```

```
info(net['r7'].cmd("ip route add 11.0.1.0/24   via 77.0.7.1 dev r7-eth0"))
info(net['r7'].cmd("ip route add 22.0.2.0/24   via 77.0.7.1 dev r7-eth0"))
info(net['r7'].cmd("ip route add 33.0.3.0/24   via 77.0.7.1 dev r7-eth0"))
info(net['r7'].cmd("ip route add 55.0.5.0/24   via 77.0.7.1 dev r7-eth0"))
info(net['r7'].cmd("ip route add 44.0.4.0/24   via 77.0.7.1 dev r7-eth0"))
info(net['r7'].cmd("ip route add 66.0.6.0/24   via 99.0.9.2 dev r7-eth1"))
info(net['r7'].cmd("ip route add 88.0.8.0/24   via 99.0.9.2 dev r7-eth1"))
```

```
info(net['r8'].cmd("ip route add 11.0.1.0/24   via 88.0.8.2 dev r8-eth1"))
info(net['r8'].cmd("ip route add 22.0.2.0/24   via 88.0.8.2 dev r8-eth1"))
info(net['r8'].cmd("ip route add 33.0.3.0/24   via 88.0.8.2 dev r8-eth1"))
info(net['r8'].cmd("ip route add 44.0.4.0/24   via 88.0.8.2 dev r8-eth1"))
info(net['r8'].cmd("ip route add 55.0.5.0/24   via 88.0.8.2 dev r8-eth1"))
info(net['r8'].cmd("ip route add 66.0.6.0/24   via 88.0.8.2 dev r8-eth1"))
info(net['r8'].cmd("ip route add 77.0.7.0/24   via 88.0.8.2 dev r8-eth1"))
info(net['r8'].cmd("ip route add 99.0.9.0/24   via 88.0.8.2 dev r8-eth1"))
```

```
# GENEVE Config for R1
r1.cmd('ip link add dev gnv0 type geneve remote 88.0.8.1 vni 123')
r1.cmd('ip link set gnv0 up')
r1.cmd('ipaddr add 111.111.111.1/24 dev gnv0')
r1.cmd('ip route add 10.0.20.11 encapip via 111.111.111.2 dev gnv0')
```

```
r1.cmd('ip link add dev gnv1 type geneve remote 88.0.8.1 vni 456')
r1.cmd('ip link set gnv1 up')
r1.cmd('ipaddr add 222.222.222.1/24 dev gnv1')
r1.cmd('ip route add 10.0.20.12 encapip via 222.222.222.2 dev gnv1')
```

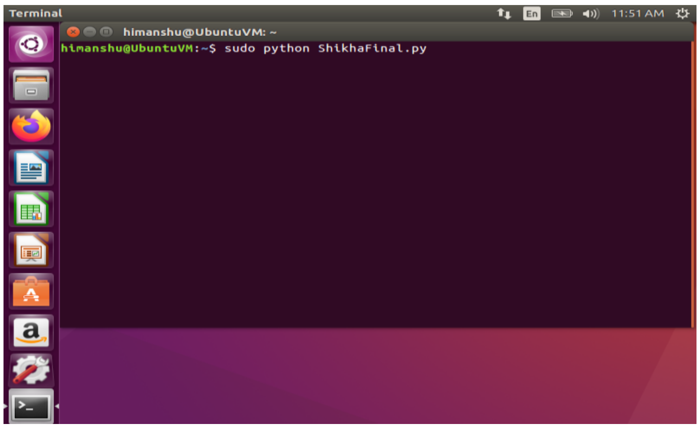
```
# GENEVE Config for R8
r8.cmd('ip link add dev gnv0 type geneve remote 11.0.1.1 vni 123')
r8.cmd('ip link set gnv0 up')
r8.cmd('ipaddr add 111.111.111.2/24 dev gnv0')
r8.cmd('ip route add 10.0.10.11 encapip via 111.111.111.1 dev gnv0')

r8.cmd('ip link add dev gnv1 type geneve remote 11.0.1.1 vni 456')
r8.cmd('ip link set gnv1 up')
r8.cmd('ipaddr add 222.222.222.2/24 dev gnv1')
r8.cmd('ip route add 10.0.10.12 encapip via 222.222.222.1 dev gnv1')
```

Figure: 3.2 Main Code Flow Chart

Here we can see 1st we install mininet then all the libraries like openv kernal switch, remote controller, constructer etc. then install general libraries of linux like time, globel OS etc. then initialize ip for all routers then add remote controller at port- 6633 then initialize all the switches, then establish all the connections, then add router to router link with subnet definition then adding access hosts with specification of default route, then adding edge hosts with specification of default route then add access host to switch link then add edge host to switch links, then add edge host switch links then build network and start controller and switches. Then define router interface MAC address then initialize open flow control instructions and route info addition then make geneve config for R1, and make geneve config for R8.

4. Results

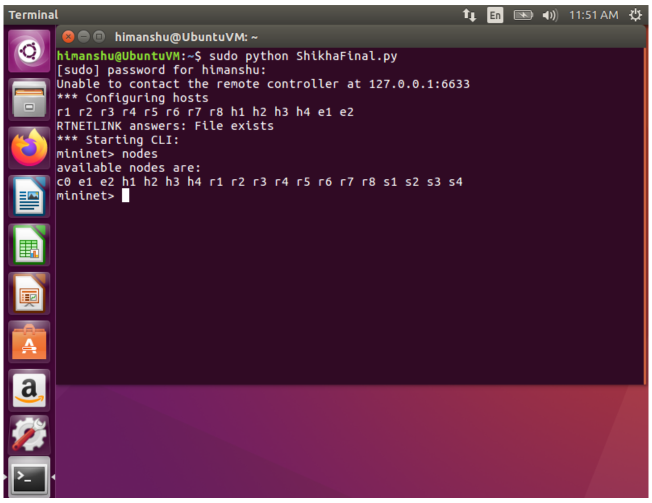
4.1 Final Outputs



```
himanshu@UbuntuVM:~$ sudo python ShikhaFinal.py
```

Figure: 4.1 Run Python Code

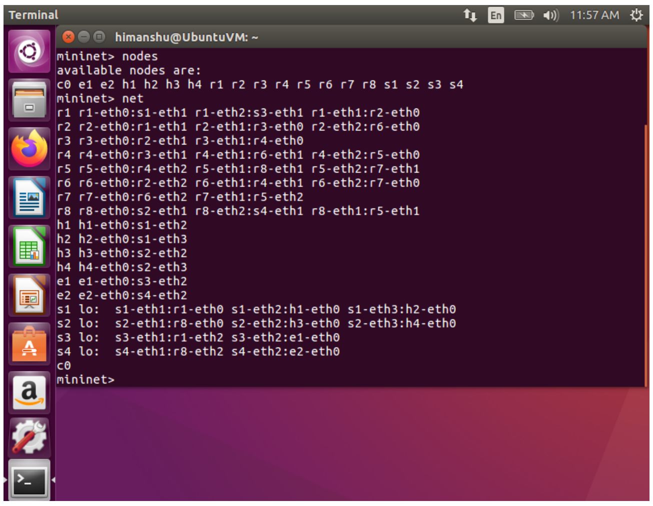
In this figure we can see here we run the our main python code that named as Shikhafinal.py.



```
himanshu@UbuntuVM:~$ sudo python ShikhaFinal.py
[sudo] password for himanshu:
Unable to contact the remote controller at 127.0.0.1:6633
*** Configuring hosts
r1 r2 r3 r4 r5 r6 r7 r8 h1 h2 h3 h4 e1 e2
RTNETLINK answers: File exists
*** Starting CLI:
mininet> nodes
available nodes are:
c0 e1 e2 h1 h2 h3 h4 r1 r2 r3 r4 r5 r6 r7 r8 s1 s2 s3 s4
mininet> 
```

Figure: 4.2 Available nodes

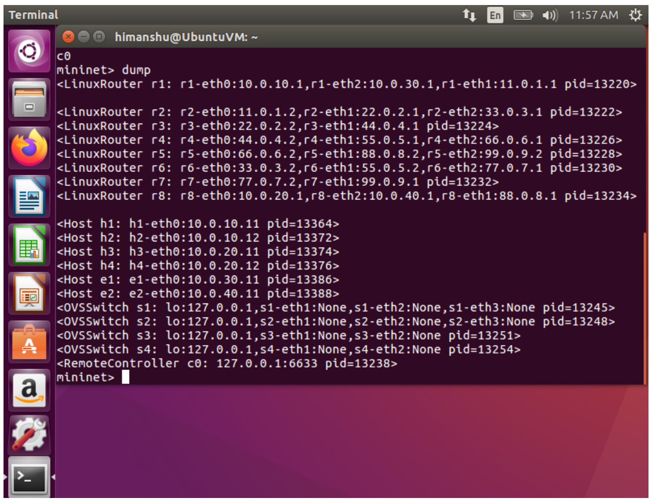
In this we figure we can see now we give “node” command for see the elements.



```
mininet> nodes
available nodes are:
c0 e1 e2 h1 h2 h3 h4 r1 r2 r3 r4 r5 r6 r7 r8 s1 s2 s3 s4
mininet> net
r1 r1-eth0:s1-eth1 r1-eth2:s3-eth1 r1-eth1:r2-eth0
r2 r2-eth0:r1-eth1 r2-eth1:r3-eth0 r2-eth2:r6-eth0
r3 r3-eth0:r2-eth1 r3-eth1:r4-eth0
r4 r4-eth0:r3-eth1 r4-eth1:r6-eth1 r4-eth2:r5-eth0
r5 r5-eth0:r4-eth2 r5-eth1:r8-eth1 r5-eth2:r7-eth1
r6 r6-eth0:r2-eth2 r6-eth1:r4-eth1 r6-eth2:r7-eth0
r7 r7-eth0:r6-eth2 r7-eth1:r5-eth2
r8 r8-eth0:s2-eth1 r8-eth2:s4-eth1 r8-eth1:r5-eth1
h1 h1-eth0:s1-eth2
h2 h2-eth0:s1-eth3
h3 h3-eth0:s2-eth2
h4 h4-eth0:s2-eth3
e1 e1-eth0:s3-eth2
e2 e2-eth0:s4-eth2
s1 lo:  s1-eth1:r1-eth0 s1-eth2:h1-eth0 s1-eth3:h2-eth0
s2 lo:  s2-eth1:r8-eth0 s2-eth2:h3-eth0 s2-eth3:h4-eth0
s3 lo:  s3-eth1:r1-eth2 s3-eth2:e1-eth0
s4 lo:  s4-eth1:r8-eth2 s4-eth2:e2-eth0
c0
mininet>
```

Figure: 4.3 Nodes and Ports links

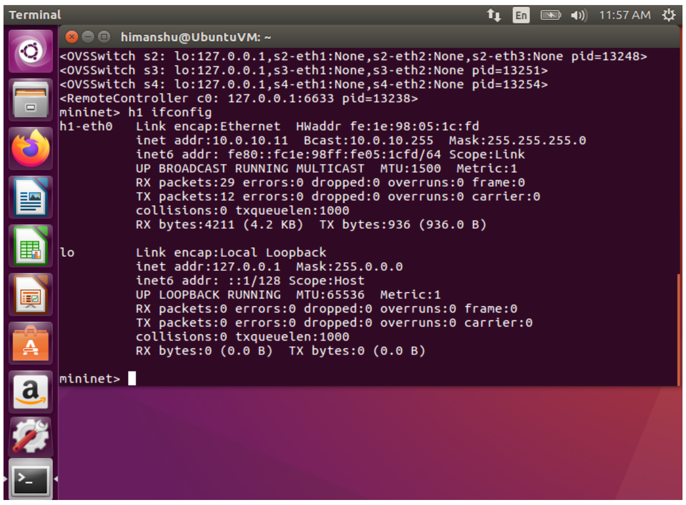
In this figure we can see now we give command "net" for see links of all the connections. In this figure we can see now we give command "net" for see links of all the connections. c0 is the controller, e1,e2 edge node, h1,h2,h3,h4 are the host, r1,r2,r3,r4,r5,r6,r7,r8 are the routers and s1,s2,s3,s4 are the switches.



```
c0
mininet> dump
<LinuxRouter r1: r1-eth0:10.0.10.1,r1-eth2:10.0.30.1,r1-eth1:11.0.1.1 pid=13220>

<LinuxRouter r2: r2-eth0:11.0.1.2,r2-eth1:22.0.2.1,r2-eth2:33.0.3.1 pid=13222>
<LinuxRouter r3: r3-eth0:22.0.2.2,r3-eth1:44.0.4.1 pid=13224>
<LinuxRouter r4: r4-eth0:44.0.4.2,r4-eth1:55.0.5.1,r4-eth2:66.0.6.1 pid=13226>
<LinuxRouter r5: r5-eth0:66.0.6.2,r5-eth1:88.0.8.2,r5-eth2:99.0.9.2 pid=13228>
<LinuxRouter r6: r6-eth0:33.0.3.2,r6-eth1:55.0.5.2,r6-eth2:77.0.7.1 pid=13230>
<LinuxRouter r7: r7-eth0:77.0.7.2,r7-eth1:99.0.9.1 pid=13232>
<LinuxRouter r8: r8-eth0:10.0.20.1,r8-eth2:10.0.40.1,r8-eth1:88.0.8.1 pid=13234>

<Host h1: h1-eth0:10.0.10.11 pid=13364>
<Host h2: h2-eth0:10.0.10.12 pid=13372>
<Host h3: h3-eth0:10.0.20.11 pid=13374>
<Host h4: h4-eth0:10.0.20.12 pid=13376>
<Host e1: e1-eth0:10.0.30.11 pid=13386>
<Host e2: e2-eth0:10.0.40.11 pid=13388>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=13245>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=13248>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=13251>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None pid=13254>
<RemoteController c0: 127.0.0.1:6633 pid=13238>
mininet>
```

Figure: 4.4 IP address

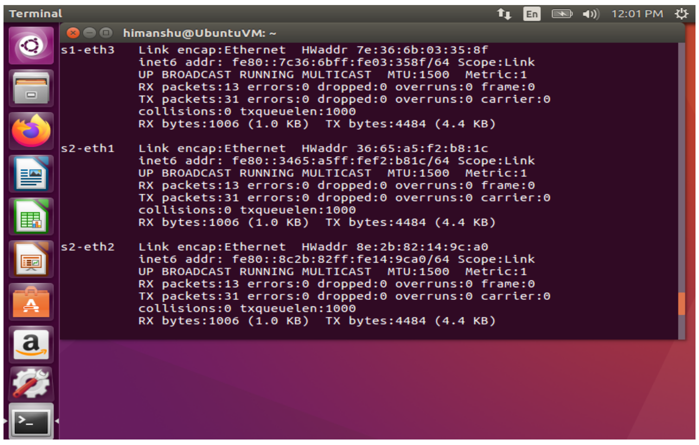
In this figure we can see now we give command "dump" for see IP address of all elements and process id's.



```
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=13248>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=13251>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None pid=13254>
<RemoteController c0: 127.0.0.1:6633 pid=13238>
mininet> h1 ifconfig
h1-eth0   Link encap:Ethernet  HWaddr fe:1e:98:05:1c:fd
          inet addr:10.0.10.11  Bcast:10.0.10.255  Mask:255.255.255.0
          inet6 addr: fe80::fc1e:98ff:fe05:1cfd/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:29 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4211 (4.2 KB)  TX bytes:936 (936.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet>
```

Figure: 4.5 h1 IP and other details

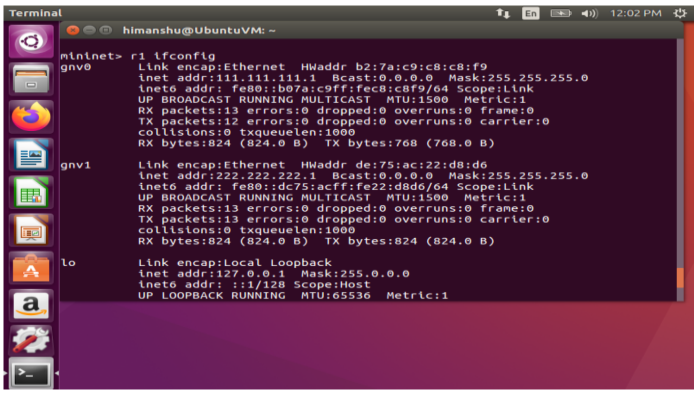
In this figure we can see IP address other interface details of h1 using h1 ifconfig command. We can see all the host IP and other details like this for h,h2,h3,h4 hosts.



```
s1-eth3   Link encap:Ethernet  HWaddr 7e:36:6b:03:35:8f
          inet6 addr: fe80::7c36:6bff:fe03:358f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:13 errors:0 dropped:0 overruns:0 frame:0
          TX packets:31 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1006 (1.0 KB)  TX bytes:4484 (4.4 KB)

s2-eth1   Link encap:Ethernet  HWaddr 36:65:a5:f2:b8:1c
          inet6 addr: fe80::3465:a5ff:fef2:b81c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:13 errors:0 dropped:0 overruns:0 frame:0
          TX packets:31 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1006 (1.0 KB)  TX bytes:4484 (4.4 KB)

s2-eth2   Link encap:Ethernet  HWaddr 8e:2b:82:14:9c:a0
          inet6 addr: fe80::8c2b:82ff:fe14:9ca0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:13 errors:0 dropped:0 overruns:0 frame:0
          TX packets:31 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1006 (1.0 KB)  TX bytes:4484 (4.4 KB)
```

Figure: 4.6 Switch1 Ethernet Interface

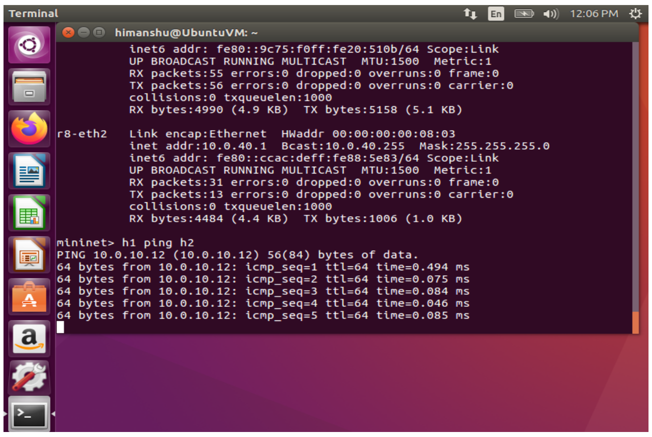
In this figure we can see switch 1 Ethernet interface with IP address, Links and other details. Same procedure follow for switch 2, switch 3 and switch 4.



```
mininet> r1 ifconfig
gnv0      Link encap:Ethernet  HWaddr b2:7a:c9:c8:c8:f9
          inet addr:111.111.111.1  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::b07a:c9ff:fec8:c8f9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:13 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:824 (824.0 B)  TX bytes:768 (768.0 B)

gnv1      Link encap:Ethernet  HWaddr de:75:ac:22:d8:d6
          inet addr:222.222.222.1  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::dc75:acff:fe22:d8d6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:13 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:824 (824.0 B)  TX bytes:824 (824.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
```

Figure: 4.7 Router 1 Interface

In this figure we can see router 1 Ethernet interface with IP address, Links and other details. And we can also see geneve tunnel 0 and geneve tunnel 1 details. We can see all the router interface (r1 to r2) using ifconfig command.

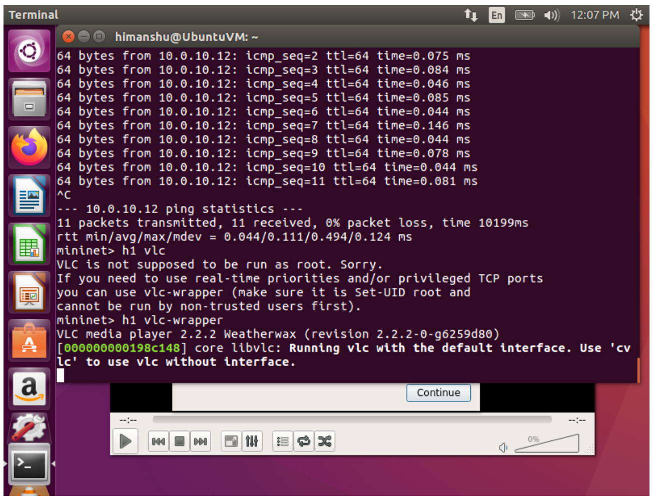


```
          inet6 addr: fe80::9c75:f0ff:fe20:510b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:55 errors:0 dropped:0 overruns:0 frame:0
          TX packets:56 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4990 (4.9 KB)  TX bytes:5158 (5.1 KB)

r8-eth2   Link encap:Ethernet  HWaddr 00:00:00:00:08:03
          inet addr:10.0.40.1  Bcast:10.0.40.255  Mask:255.255.255.0
          inet6 addr: fe80::ccac:deff:fe88:5e83/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:31 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4484 (4.4 KB)  TX bytes:1006 (1.0 KB)

mininet> h1 ping h2
PING 10.0.10.12 (10.0.10.12) 56(84) bytes of data.
64 bytes from 10.0.10.12: icmp_seq=1 ttl=64 time=0.494 ms
64 bytes from 10.0.10.12: icmp_seq=2 ttl=64 time=0.075 ms
64 bytes from 10.0.10.12: icmp_seq=3 ttl=64 time=0.084 ms
64 bytes from 10.0.10.12: icmp_seq=4 ttl=64 time=0.046 ms
64 bytes from 10.0.10.12: icmp_seq=5 ttl=64 time=0.085 ms
```

Figure: 4.8 host 1 ping to host 2

In this figure we can see now we ping host 1 (h1) to host 2 (h2), and also see statistics details.

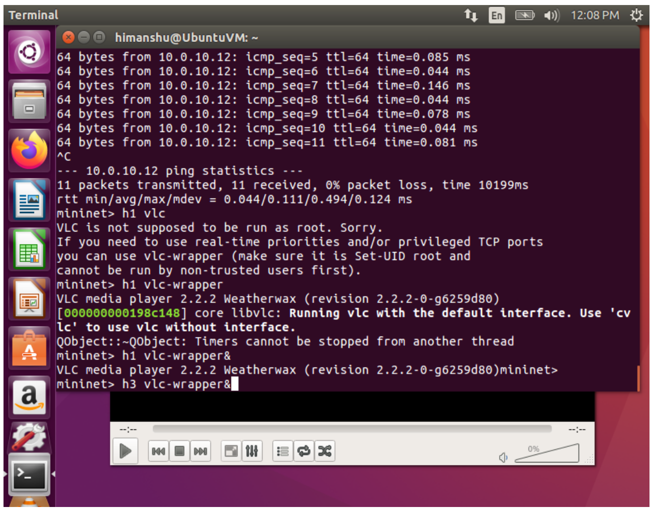
DOI : https://doi.org/10.56452/110



```
64 bytes from 10.0.10.12: icmp_seq=2 ttl=64 time=0.075 ms
64 bytes from 10.0.10.12: icmp_seq=3 ttl=64 time=0.084 ms
64 bytes from 10.0.10.12: icmp_seq=4 ttl=64 time=0.046 ms
64 bytes from 10.0.10.12: icmp_seq=5 ttl=64 time=0.085 ms
64 bytes from 10.0.10.12: icmp_seq=6 ttl=64 time=0.044 ms
64 bytes from 10.0.10.12: icmp_seq=7 ttl=64 time=0.146 ms
64 bytes from 10.0.10.12: icmp_seq=8 ttl=64 time=0.044 ms
64 bytes from 10.0.10.12: icmp_seq=9 ttl=64 time=0.078 ms
64 bytes from 10.0.10.12: icmp_seq=10 ttl=64 time=0.044 ms
64 bytes from 10.0.10.12: icmp_seq=11 ttl=64 time=0.081 ms
^C
--- 10.0.10.12 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10199ms
rtt min/avg/max/mdev = 0.044/0.111/0.494/0.124 ms
mininet> h1 vlc
VLC is not supposed to be run as root. Sorry.
If you need to use real-time priorities and/or privileged TCP ports
you can use vlc-wrapper (make sure it is Set-UID root and
cannot be run by non-trusted users first).
mininet> h1 vlc-wrapper
VLC media player 2.2.2 Weatherwax (revision 2.2.2-0-g6259d80)
[000000000198c148] core libvlc: Running vlc with the default interface. Use 'cv
lc' to use vlc without interface.
```

Figure: 4.9 Play VLC on h1

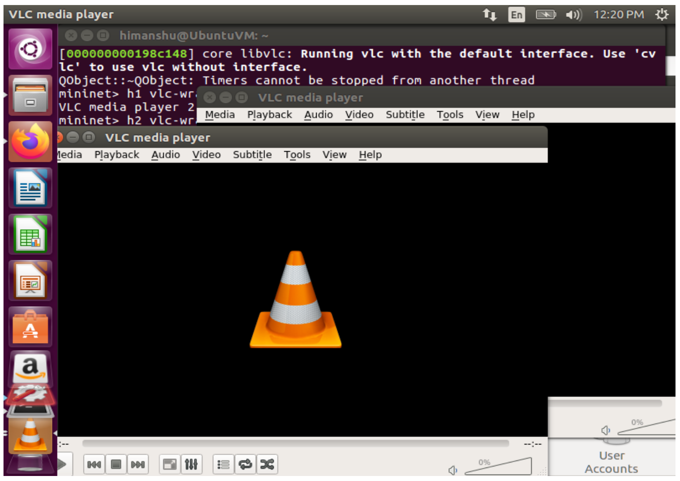
In this figure we can h1 is successfully ping with h2 and now we are playing VLC media player on host 1 (h1).



```
64 bytes from 10.0.10.12: icmp_seq=5 ttl=64 time=0.085 ms
64 bytes from 10.0.10.12: icmp_seq=6 ttl=64 time=0.044 ms
64 bytes from 10.0.10.12: icmp_seq=7 ttl=64 time=0.146 ms
64 bytes from 10.0.10.12: icmp_seq=8 ttl=64 time=0.044 ms
64 bytes from 10.0.10.12: icmp_seq=9 ttl=64 time=0.078 ms
64 bytes from 10.0.10.12: icmp_seq=10 ttl=64 time=0.044 ms
64 bytes from 10.0.10.12: icmp_seq=11 ttl=64 time=0.081 ms
^C
--- 10.0.10.12 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10199ms
rtt min/avg/max/mdev = 0.044/0.111/0.494/0.124 ms
mininet> h1 vlc
VLC is not supposed to be run as root. Sorry.
If you need to use real-time priorities and/or privileged TCP ports
you can use vlc-wrapper (make sure it is Set-UID root and
cannot be run by non-trusted users first).
mininet> h1 vlc-wrapper
VLC media player 2.2.2 Weatherwax (revision 2.2.2-0-g6259d80)
[000000000198c148] core libvlc: Running vlc with the default interface. Use 'cv
lc' to use vlc without interface.
QObject::~QObject: Timers cannot be stopped from another thread
mininet> h1 vlc-wrapper&
VLC media player 2.2.2 Weatherwax (revision 2.2.2-0-g6259d80)mininet>
mininet> h3 vlc-wrapper&
```

Figure: 4.10 Play VLC on h3

In this window we can see now we are playing VLC media player on host 3 (h3).



```
[000000000198c148] core libvlc: Running vlc with the default interface. Use 'cv
lc' to use vlc without interface.
QObject::~QObject: Timers cannot be stopped from another thread
mininet> h1 vlc-wr
VLC media player 2
mininet> h2 vlc-wr
```

VLC media player
12:20 PM
himanshu@UbuntuVM: ~
VLC media player
Media Playback Audio Video Subtitle Tools View Help
VLC media player
Media Playback Audio Video Subtitle Tools View Help
User Accounts

Figure: 4.11 video stream h1 to h3

In this window we can see live video stream is start via host 1 to host 3, because data only send with h1 to h3 using geneve tunnel.

5. Conclusion

To make network control directly programmable and the underlying infrastructure abstracted for applications and network services, SDN is a network architecture that divides the network into a control plane and a forwarding plane. Traffic flows can be monitored, forwarding decisions can be made, and effective policies can be implemented in real time thanks to a logically centralised controller with a global perspective of the network. Services like cross-domain network layer multicast can take advantage of this pliability and control. SDN, or software-defined networking, is an up-and-coming method for network programmability that allows for the dynamic initialization, control, modification, and management of network behaviour through the use of standard interfaces. SDN's ascent allows for the possibility of delivering services like inter-domain network layer multicast for real-time video streaming, which are currently hampered by the Internet's legacy architecture's inflexibility and insularity. Here we show off a system for gauging the efficacy of SDN-based multicast topologies for real-time streaming in comparison to the status quo of traditional IP unicast. And we saw the debut of a brand-new multicast architecture for real-time video transmission over the Internet that relied on software-defined networking (SDN). To show how SDN-based multicast architectures and algorithms for real-time video streaming measure up against the industry standard of IP unicast, we've built a modular framework for comparing and contrasting their respective performance. In this work we used real HD video content to stream over a simulated network while we collect data on how the network and applications perform.

6. Future scope

Increases in wireless connectivity have enabled its use in previously untapped markets including the healthcare and automobile sectors. Not all customers in the wireless industry have the same demands; some need very high bandwidth, while others need very low latency. It will be vital to have dedicated networks that can perform dynamic resource utilization if future wireless cellular Internet of Things (IoT) applications like 5G, Narrow-Band IoT (NB-IoT), and machine-to-machine (M2M) communications are to offer various services with a mix of requirements. With network slicing, it will be possible to create isolated, virtualized sections of the network for certain purposes. We will conduct further tests in future with CCN and analyse the performance of alternative network designs like HIMALIS and INP in subsequent developments. In addition to helping with general workloads and content/video distribution in the IoT, these architectures may also be useful for other use cases.

References

[1] Ibrahim Afolabi, Tarik Taleb, Konstantinos Samdanis “Network Slicing & Softwarization: A Survey on Principles, Enabling Technologies & Solutions” IEEE 2018.

[2] Alcardo Alex Barakabitzea, Arslan Ahmadb , Rashid Mijumbi “5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges” Computer Networks 2020.

[3] P. Iyyanar, M. Chitra, and P. Sabarinath “Effective and Secure Scheme for Video Streaming Using SRTP” International Journal of Machine Learning and Computing 2012.

[4] Huifen Huang, Zhihong Wu, Jin Ge, and Lu Wang “Toward Building Video Multicast Tree with Congestion Avoidance Capability in Software-Defined Networks” The International Arab Journal of Information Technology 2020.

[5] Xenofon Foukas, Mahesh K. Marina, Kimon Kontovasilis “Orion: RAN Slicing for a Flexible and Cost-Eective Multi-Service Mobile Network Architecture” MobiCom 2017.

[6] Rafael Montero, Albert Pagès, Fernando Agraz “Supporting QoE/QoS-aware end-to-end network slicing in future 5G-enabled optical networks” Research Gate 2019.

[7] Christelle Al Hasrouty, Mohamed Lamine Lamali, Vincent Autefage “Adaptive Multicast Streaming for Videoconferences on Software-Defined Networks” Hal Open Science 2018.

[8] R. Pereiraa, E.G. Pereira “Video streaming: Overview and challenges in the internet of things” Pervasive Computing 2016.

[9] Alexander Hefele, Jose Costa-Requena “SDN managed Network Slicing in Mobile Backhaul” IEEE 2020.

DOI : https://doi.org/10.56452/110

[10] En-zhong YANG, Lin-kai ZHANG, Zhen YAO, Jian YANG "A video conferencing system based on SDN-enabled SVC multicast"Springer 2016.

[11] Qiang Gao1, Weiqin Tong, Samina Kausar and Shengan Zheng "A Design and Implementation of SDN Multicast for Distributed Shared Memory" IEEE 2015.

[12] Shalitha Wijethilaka, Madhusanka Liyanage "Survey on Network Slicing for Internet of Things Realization in 5G Networks" Research Gate 2021.

[13] Alcardo Alex Barakabitzea , Arslan Ahmadb , Rashid Mijumbi "5G Network Slicing using SDN and NFV: A Survey of Taxonomy, Architectures and Future Challenges" Elsevier 2019.

[14] Christian Koch, Stefan Hacker and David Hausheer "VoDCast: Efficient SDN-based Multicast for Video on Demand" IEEE 2017.

[15] Ahmed Khalid, Ahmed H. Zahran, Cormac J. Sreenan "mCast: An SDN-based Resource-Efficient Live Video Streaming Architecture with ISP-CDN Collaboration" IEEE 2017.

[16] Xiantao Jiang, F. Richard Yu "A Survey on Multi-Access Edge Computing Applied to Video Streaming: Some Research Issues and Challenges" IEEE 2021.

[17] Xenofon Foukas, Mahesh K. Marina, Kimon Kontovasilis "Orion: RAN Slicing for a Flexible and Cost-Eective Multi-Service Mobile Network Architecture" ACM 2017.

[18] Panagiotis Georgopoulosa, Matthew Broadbentb, Arsham Farshad "Using Software Defined Networking to Enhance the Delivery of Video-on-Demand" Preprint submitted to Computer Communications 2015.

[19] Algimantas Venčkauskas, Nerijus Morkevicius, Kazimieras Bagdonas "A Lightweight Protocol for Secure Video Streaming" Sensors 2018.

[20] Johanna Andrea Hurtado Sánchez, Katherine Casilimas and Oscar Mauricio Caicedo Rendon "Deep Reinforcement Learning for Resource Management on Network Slicing: A Survey" Sensors 2022.

[21] B. Susila, C. Veeralakshmi "Software Defined Networking – an Overview" IJERT 2015.