|  |  |  |
| --- | --- | --- |
| **Description: Logo, company nameDescription automatically generated**  **جامعة المنصورة**  **كلية الهندسة**  **قسم إتصالات وإلكترونيات** |  | **Description: LogoDescription automatically generated with medium confidence** |

**Software Defined Network (SDN)**

***Supervisor******s***

**Asocc. Prof. Mohamed M. Ashour**

**Dr. Haitham Mahmoud Mohammed**

**Academic Year: 2022-2023**

**Team**

|  |  |
| --- | --- |
| 1. **Samah Atef Abd Elgelil** | **سماح عاطف عبد الجليل** |
| 1. **Mohammed Ahmed Ali** | **محمد أحمد على** |
| 1. **Mahmoud Shaker Shabana** | **محمود شاكر شبانة** |
| 1. **Mahmoud Elsayed Mosaad** | **محمود السيد مسعد** |
| 1. **Mohammed Ahmed Abd Ellatif** | **محمد أحمد عبد اللطيف** |
| 1. **Marwan Hisham Ali** | **مروان هشام على** |

**Abstract:…………………………………………………………………………………………4**

**Chapter 1:**

**Software Defined Network (SDN)**

**1.1 SDN Introduction:………………………………………………………..…………….5**

**1.2 SDN Architecture:…………………………………………………………..………….6**

**1.3 SDN Test bed Installation:…………………….…………………………..………….7**

**1.4 Option1: Prebuilt VM Image:………………….……………………..……………….8**

**1.5 Installation:………………………………………………………………………………8**

**1.6 Testing:…………………………………………………………………………………..9**

**1.7 Networking Concepts: ……………………………………………………………….13**

**1.8 Traditional L2 Switch: ………………………………………………………………..17**

**1.9 SDN Switch(OpenFlow Switch) : ………………………………………...…………18**

**Chapter 2**

**Mininet**

**2.1 Basic Operations: ……………………………………………………………………18**

**2.2 Running Traffic Tests: ………………………………………………………………23**

**2.3 Writing Custom Topology in Mininet: ……………………………….……………25**

**Chapter 3**

**OpenFlow Theory**

**3.1 Introduction: …………………………………………………………………………26**

**3.2 Switch Components: ………………………………….……………………………27**

**3.3 OpenFlow channel: …………………………………………………………………28**

**3.4 OpenFlow Flow table: ………………………………………………………………28**

**3.5 OpenFlow Ports: …………………………………………………….……..…………36**

**3.6 Important Take aways: ………………………………………………………………37**

**Chapter 4**

**Controllers**

**4.1 RYU Controller Introduction: ………………………………………………………37**

**4.2 Reactive/Proactive Flows: ………………………………………………………….41**

**4.3 OpenFlow Applications: ……………………………………………………………43**

**4.4 Simple Switch Application (in built) : ……………………………………………43**

**4.5 RYU Programming - Basics: ………………………………………………………47**

**4.6 Closer look on Simple\_Switch\_13 application: ………………………………..48**

**4.7 Developing Switching Applications: …………………………………………….50**

**4.8 HUB Application: ……………………………………………………………………51**

**4.9 Switch Application with L3 Match: ……………………………….………………52**

**4.10 Statistics Collection: ………………………………………………………………53**

**4.11 RYU Thread API: ……………………………………………………………………53**

**4.12 Flow Statistics: ………………………………………………..…………………....54**

**4.13 Topology Discovery: ………………………………………………………………58**

**4.14 How it works: ………………………………………………………………………..58**

**4.15 Coding: ………………………………………………………………………………59**

**4.16 Demo: ………………………………………………………………………………..60**

**4.17 Multicontrollers: …………………………………………………………………...60**

**4.18 Cluster of SDN Controllers: ……………………………………………………..62**

**4.19 Controller Roles: …………………………………………………………………..62**

**4.20 Demo - Equal role: ………………………………………………………….………63**

**4.21 Demo- Master/Slave Role: …………………………………………………………64**

**Chapter 5**

**Simulation Steps And Results:**

**5.1 Traffic Tests - Part1: …………………………………………………………………65**

**5.2 TCP Tests: ……………………………………………………………………………..66**

**5.3 UDP Tests with IPERF: ………………………………………………………………69**

**5.4 Video Stream Traffic Testing: ………………………………………………………69**

**5.5 VOIP Tests: …………………………………………………………………………….70**

**5.6 Installations: …………………………………………………………………………...72**

**5.7 python code for custom Topology: ……………...………………………………..73**

**5.8 Mini-edit: ………………………………………………………………………………..74**

**5.9 Implementation: ………………………………………………………………………..77**

**5.10 Wireshark: ……………………………………………………………………….……82**

**Conclusion:…………………………………………………………………………………...88**

**References:…………………………………………………………………………………...89**

**Abstract:**

This book presents a comprehensive exploration of video streaming over Software-Defined Networking (SDN) infrastructure. The rapid growth of online video consumption has necessitated the development of efficient and scalable solutions to deliver high-quality streaming experiences to end-users. SDN, with its centralized control and programmability, offers a promising approach to optimize video streaming performance.

The research conducted in this book focuses on leveraging SDN principles and technologies to enhance video streaming in terms of quality, reliability, and adaptability. The investigation includes the design and implementation of a video streaming system using the Mininet and MiniEdit network simulators, with support from Wireshark, VLC Media Player, and OpenFlow protocol. The system's performance is evaluated through various experiments and simulations, which analyze factors such as network traffic, bandwidth allocation, and latency.

Throughout the book, the chapters delve into the foundational concepts of SDN, video streaming protocols, and the OpenFlow protocol's role in enabling dynamic network control. The experiments conducted provide insights into the advantages of SDN-based video streaming, such as improved Quality of Service (QoS) and reduced network congestion. Furthermore, the simulations showcase the flexibility of SDN in adapting to changing network conditions and optimizing resource allocation for video streaming applications.

The results obtained from the experiments highlight the significant enhancements achieved through the integration of SDN and video streaming. The findings demonstrate the potential of SDN to enhance video streaming experiences, thereby paving the way for future research and development in this domain. By leveraging the programmability and centralization of SDN, network operators and service providers can offer users a superior video streaming experience, characterized by reduced buffering, enhanced video quality, and efficient bandwidth utilization.

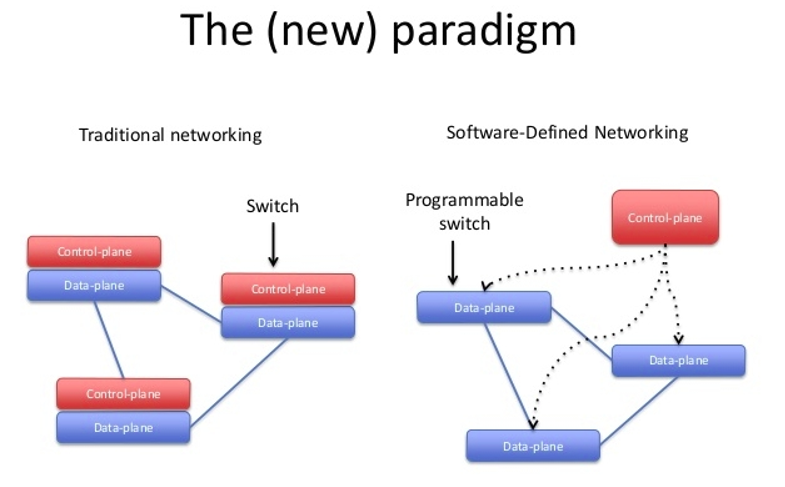
This book serves as a valuable resource for researchers, network engineers, and students interested in exploring the convergence of SDN and video streaming. It not only provides a comprehensive overview of the subject matter but also offers practical insights and guidelines for implementing SDN-based video streaming solutions. Through this research, we contribute to the growing body of knowledge in SDN and highlight the potential of this technology to revolutionize the field of video streaming.

**Chapter 1:Software Defined Network (SDN)**

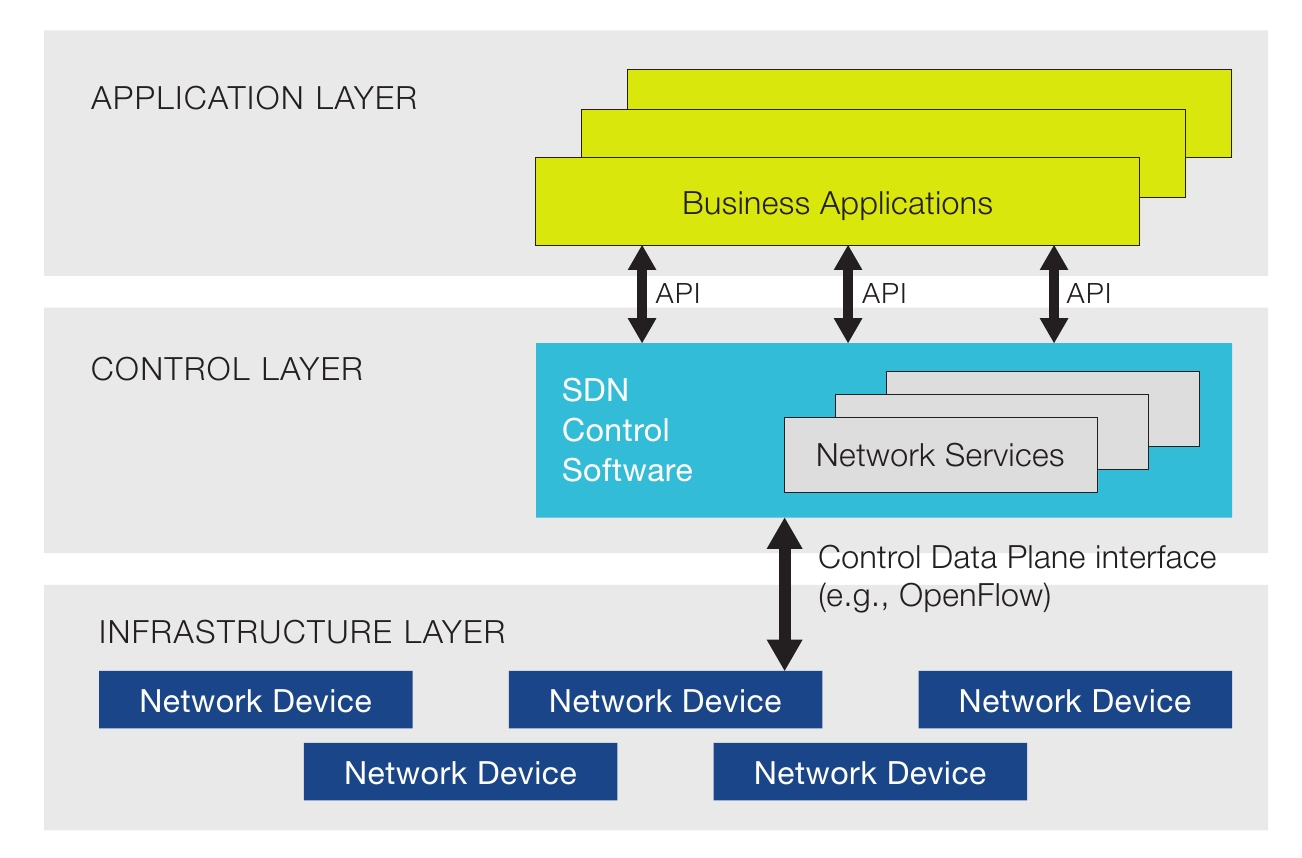
**1.1 SDN Introduction**

**SDN** suggests to centralize network intelligence in one network component by disassociating the forwarding process of network packets (data plane) from the routing process (control plane). The control plane consists of one or more controllers which are considered as the brain of SDN network where the whole intelligence is incorporated.

Enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The OpenFlow protocol is a foundational element for building SDN solutions.



**1.2 SDN Architecture**



SDN architectures generally have three components or groups of functionality:

The SDN networking devices control the forwarding and data processing capabilities for the network. This includes forwarding and processing of the data path. Example: Openvswitch (OpenFlow protocol)

**Southbound Interface:**

Southbound interface is the connection between the controller and the physically networking hardware

**SDN Controller**

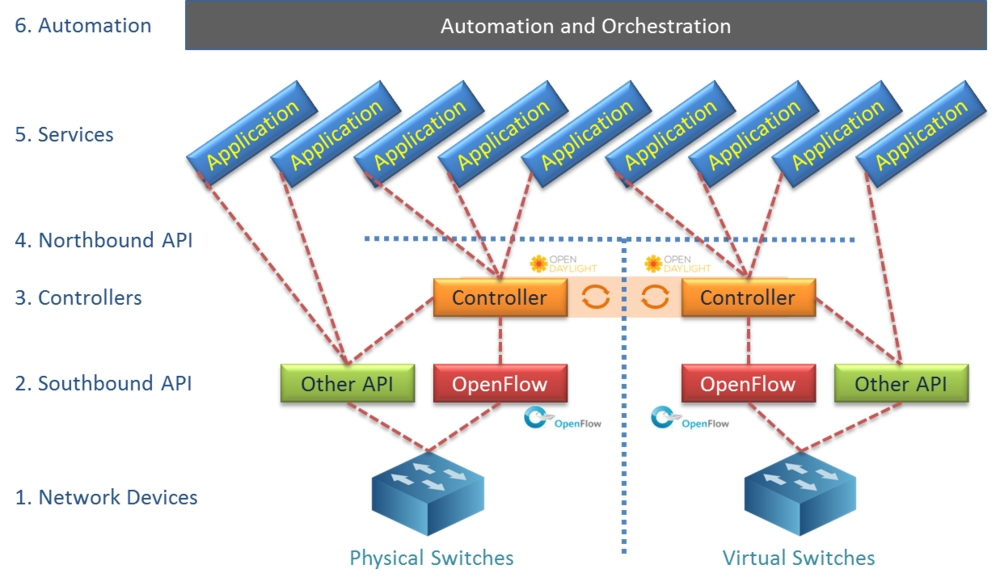
The SDN Controller is a logical entity that receives instructions or requirements from the SDN Application layer and relays them to the networking components. The controller also extracts information about the network from the hardware devices and communicates back to the SDN Applications with an abstract view of the network, including statistics and events about what is happening.

Example:  
**RYU, Open Daylight, ONOS, Floodlight etc.**

**SDN Applications:**

SDN Applications are programs that communicate behaviors and needed resources with the SDN Controller via application programming interface (APIs). These applications could include networking management, analytics, or business applications used to run large data centers. This is the place for research, innovations, new ideas etc.

A Northbound interface is defined as the connection between the controller and applications.



**1.3 SDN Test bed Installation**

Setup the SDN Test environment to practice OpenFlow use cases with RYU SDN Controller.

There are two option. 1. Download the prebuilt SDN Test bed VM image(ubuntu 20.04 desktop) and use it 2. Setup the SDN Test bed by your self(Fresh Installation on ubuntu 20.04)

**1.4 Option1: Prebuilt VM Image**

This is the easiest option(for new comer).

Prebuilt VM Image (OVA Format), Size 3.3GB can be download from this below link,

**username : test** **password : test**

This can be imported in **Oracle VirtualBox**. Just double click the OVA file it will be imported automatically.

This should be imported in **VMWARE** also.

**1.5 Installation**

If you want to setup the testbed by yourself follow this

**Requirements:**

OS: Ubuntu 20.04

CPU: 2 Cores +

RAM: 4GB +

HDD: 15GB+

Please run the below commands in UBUNTU Terminal

sudo apt update

sudo apt install python3 python3-pip xterm iperf hping3 net-tools wireshark apache2-utils curl

sudo apt install mininet

sudo pip3 install ryu

sudo pip3 install mininet

sudo cp /usr/bin/python3 /usr/bin/python

ryu-manager --version

sudo mn --version

**To check the python version:**

python3 --version or python --version

Python 3.8.5

To verify :

ovs-vsctl --version

ovs-vsctl (Open vSwitch) 2.13.1

DB Schema 8.2.0

**1.6 Testing**

Open 4 Terminals:

1. In Terminal1,

sudo wireshark

And start the capture for "loopback" or "any" interface.

1. In Terminal2,

ryu-manager ryu.app.simple\_switch\_13

1. In Terminal3,

sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4

you will get the Mininet prompt. In Mininet prompt, type pingall command

pingall

Logs:

suresh@suresh-vm:~$ sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4

\*\*\* Creating network

\*\*\* Adding controller

Connecting to remote controller at 127.0.0.1:6653

\*\*\* Adding hosts:

h1 h2 h3 h4

\*\*\* Adding switches:

s1

\*\*\* Adding links:

(h1, s1) (h2, s1) (h3, s1) (h4, s1)

\*\*\* Configuring hosts

h1 h2 h3 h4

\*\*\* Starting controller

c0

\*\*\* Starting 1 switches

s1 ...

\*\*\* Starting CLI:

mininet> pingall

\*\*\* Ping: testing ping reachability

h1 -> h2 h3 h4

h2 -> h1 h3 h4

h3 -> h1 h2 h4

h4 -> h1 h2 h3

\*\*\* Results: 0% dropped (12/12 received)

mininet>

1. In Terminal 4,

sudo ovs-vsctl show

sudo ovs-ofctl -O OpenFlow13 dump-flows s1

Logs:

suresh@suresh-vm:~$ sudo ovs-vsctl show

[sudo] password for suresh:

a315e8b4-dd3f-42f6-b84a-e967e02660a4

Bridge "s1"

Controller "tcp:127.0.0.1:6653"

is\_connected: true

Controller "ptcp:6654"

fail\_mode: secure

Port "s1-eth4"

Interface "s1-eth4"

Port "s1"

Interface "s1"

type: internal

Port "s1-eth1"

Interface "s1-eth1"

Port "s1-eth3"

Interface "s1-eth3"

Port "s1-eth2"

Interface "s1-eth2"

ovs\_version: "2.13.1"

suresh@suresh-vm:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

cookie=0x0, duration=115.430s, table=0, n\_packets=3, n\_bytes=238, priority=1,in\_port="s1-eth2",dl\_src=00:00:00:00:00:02,dl\_dst=00:00:00:00:00:01 actions=output:"s1-eth1"

cookie=0x0, duration=115.421s, table=0, n\_packets=2, n\_bytes=140, priority=1,in\_port="s1-eth1",dl\_src=00:00:00:00:00:01,dl\_dst=00:00:00:00:00:02 actions=output:"s1-eth2"

cookie=0x0, duration=115.410s, table=0, n\_packets=3, n\_bytes=238, priority=1,in\_port="s1-eth3",dl\_src=00:00:00:00:00:03,dl\_dst=00:00:00:00:00:01 actions=output:"s1-eth1"

cookie=0x0, duration=115.404s, table=0, n\_packets=2, n\_bytes=140, priority=1,in\_port="s1-eth1",dl\_src=00:00:00:00:00:01,dl\_dst=00:00:00:00:00:03 actions=output:"s1-eth3"

cookie=0x0, duration=115.391s, table=0, n\_packets=3, n\_bytes=238, priority=1,in\_port="s1-eth4",dl\_src=00:00:00:00:00:04,dl\_dst=00:00:00:00:00:01 actions=output:"s1-eth1"

cookie=0x0, duration=115.380s, table=0, n\_packets=2, n\_bytes=140, priority=1,in\_port="s1-eth1",dl\_src=00:00:00:00:00:01,dl\_dst=00:00:00:00:00:04 actions=output:"s1-eth4"

cookie=0x0, duration=115.370s, table=0, n\_packets=3, n\_bytes=238, priority=1,in\_port="s1-eth3",dl\_src=00:00:00:00:00:03,dl\_dst=00:00:00:00:00:02 actions=output:"s1-eth2"

cookie=0x0, duration=115.368s, table=0, n\_packets=2, n\_bytes=140, priority=1,in\_port="s1-eth2",dl\_src=00:00:00:00:00:02,dl\_dst=00:00:00:00:00:03 actions=output:"s1-eth3"

cookie=0x0, duration=115.361s, table=0, n\_packets=3, n\_bytes=238, priority=1,in\_port="s1-eth4",dl\_src=00:00:00:00:00:04,dl\_dst=00:00:00:00:00:02 actions=output:"s1-eth2"

cookie=0x0, duration=115.359s, table=0, n\_packets=2, n\_bytes=140, priority=1,in\_port="s1-eth2",dl\_src=00:00:00:00:00:02,dl\_dst=00:00:00:00:00:04 actions=output:"s1-eth4"

cookie=0x0, duration=115.346s, table=0, n\_packets=3, n\_bytes=238, priority=1,in\_port="s1-eth4",dl\_src=00:00:00:00:00:04,dl\_dst=00:00:00:00:00:03 actions=output:"s1-eth3"

cookie=0x0, duration=115.344s, table=0, n\_packets=2, n\_bytes=140, priority=1,in\_port="s1-eth3",dl\_src=00:00:00:00:00:03,dl\_dst=00:00:00:00:00:04 actions=output:"s1-eth4"

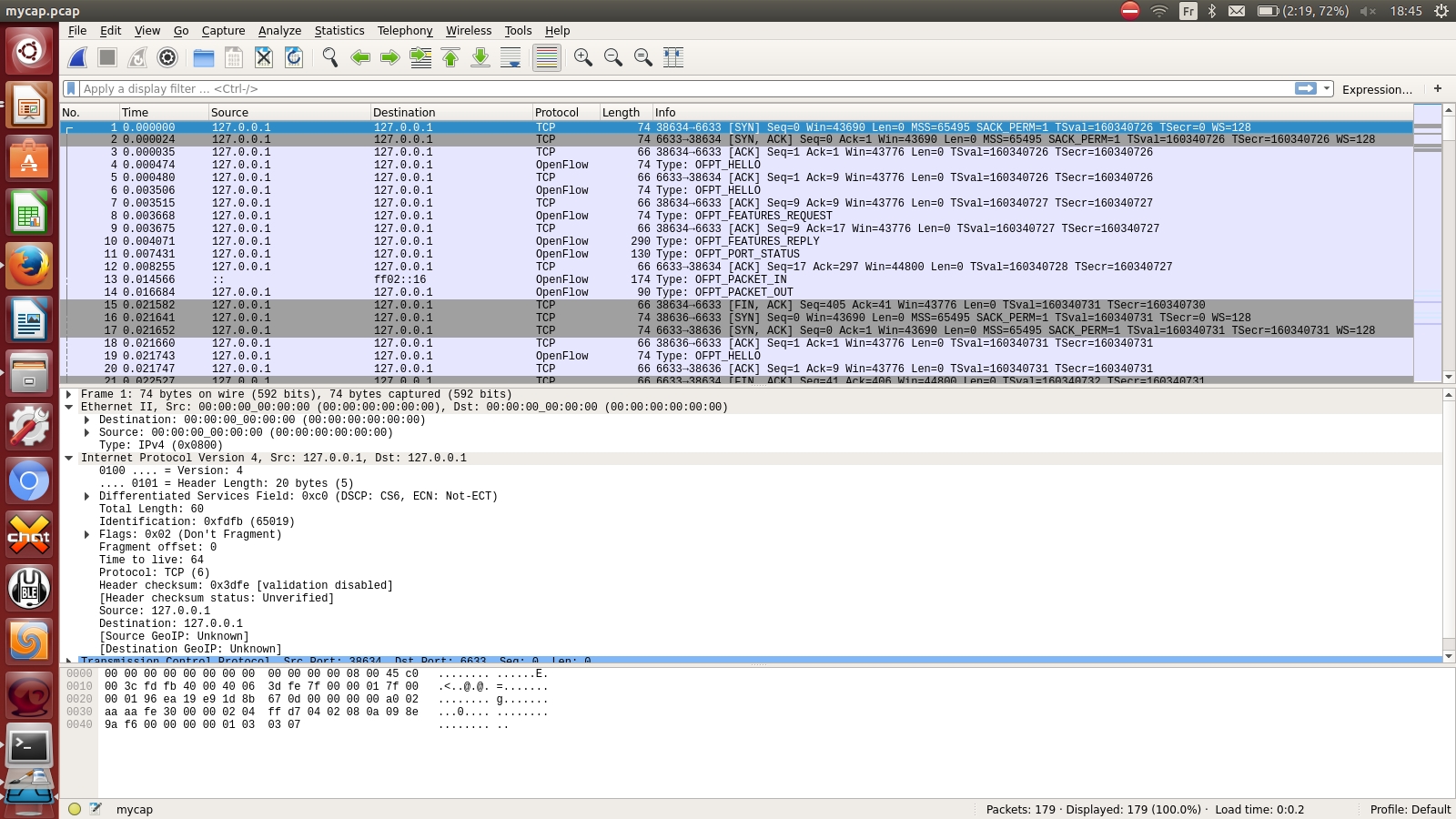
cookie=0x0, duration=164.572s, table=0, n\_packets=58, n\_bytes=4276, priority=0 actions=CONTROLLER:65535

suresh@suresh-vm:~$

1. check the OpenFlow messages in Wireshark

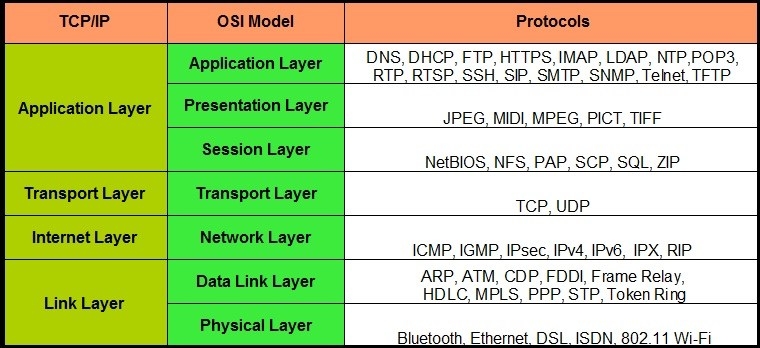
Stop the Wireshark capture,

In the filter type "openflow\_v4" to see the OPENFLOW Messages.



**1.7 Networking Concepts**

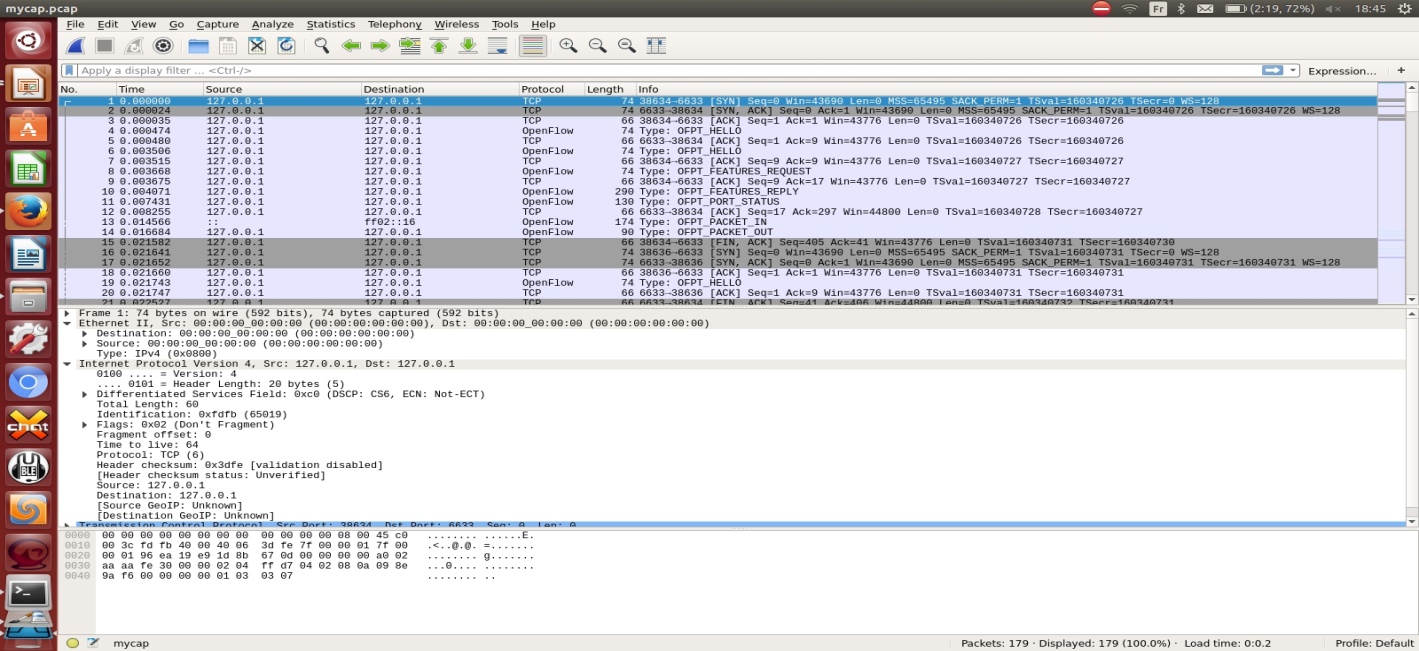
**TCP/IP Layers**



**Network Packet Capture & Protocol Analyzer:**

**Wireshark**

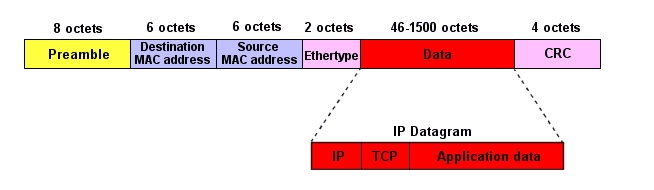
* Traffic Capture & Network protocol analyzer
* GUI
* Capture the Traffic from the interface
* Open the capture file and analyze it



**Ethernet (Layer2) Concepts**

* Ethernet is Layer2 Protocol,
* Majorly used as LAN connectivity
* Interface (eth0, eth1, ….)
* MAC Address is L2 Address, Associated with the interface, its 8 bytes (HWaddr 02:42:ac:11:00:02)

**Ethernet Frame**



Destination Mac address : Target machine MAC address

Source Mac address: Source machine MAC address

Ethertype: Next layer protocol

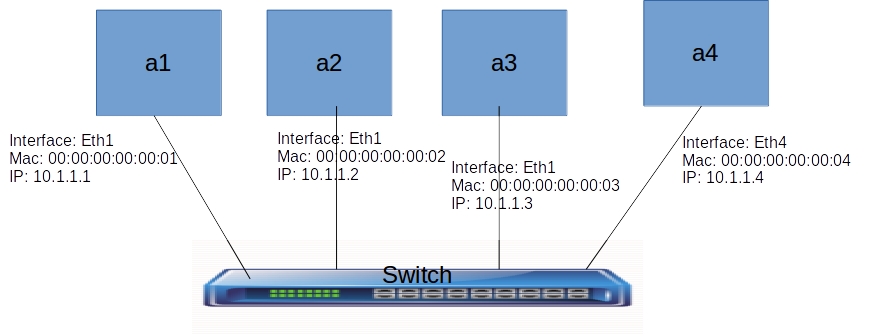
0X0800 - IP

0X0806 - ARP

0X86DD - IPv6

**Neighbour Discovery**

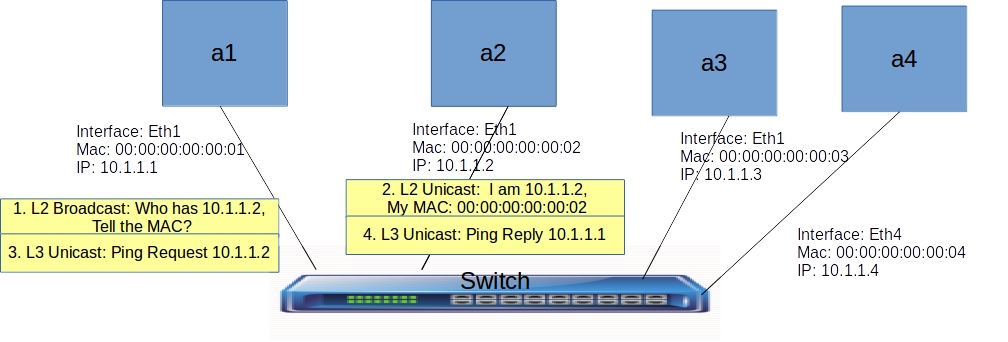
The MAC address is how machines on a subnet communicate. When machine A sends packets to another machine on its subnet, it sends it using the MAC address. When sending a packet to a machine on the public Internet, the packet is sent to the MAC address of the router interface that is the default gateway. IP addresses are used to figure out the MAC address to send to using ARP.



**ARP Basics**

ARP stands for Address Resolution Protocol. When you try to ping an IP address on your local network, say 192.168.1.1, your system has to turn the IP address 192.168.1.1 into a MAC address. This involves using ARP to resolve the address, hence its name.

Systems keep an ARP look-up table where they store information about what IP addresses are associated with what MAC addresses.



1. When trying to send a packet to an IP address, the system will first consult this table to see if it already knows the MAC address. If there is a value cached, ARP is not used.
2. If the IP address is not found in the ARP table, the system will then send a broadcast packet to the network using the ARP protocol to ask "who has 192.168.1.1".
3. Because it is a broadcast packet, it is sent to a special MAC address that causes all machines on the network to receive it.
4. Any machine with the requested IP address will reply with an ARP packet that says "I am 192.168.1.1", and this includes the MAC address which can receive packets for that IP.

**ARP Table**

**List ARP Table**

arp -a

**Delete ARP entry**

arp -d 10.1.1.2

**ARP Demo**

1. Start the Mininet topology (linear topology , not connected to SDN Controller)

sudo python traditional\_switch.py

1. in Mininet console, run 'links' command to identify the link names

mininet> links

h1-eth0<->s1-eth1 (OK OK)

h2-eth0<->s1-eth2 (OK OK)

h3-eth0<->s1-eth3 (OK OK)

h4-eth0<->s1-eth4 (OK OK)

mininet>

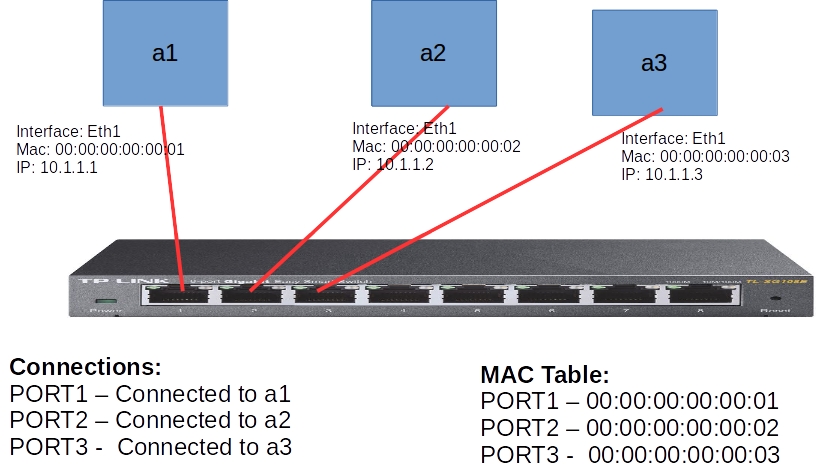
So if we want to capture Wireshark traces for h2, we have to capture the s1-eth2 interface.

1. capture Wireshark traces for h2
2. in Mininet console run 'h1 ping h2'
3. Analyze the ARP Packets from Wireshark traces
4. check the ARP entries in h1 and h2

mininet>h1 arp -a

**1.8 Traditional L2 Switch**

A network switch (MAC bridge) is a computer networking device that connects devices together on a computer network by using packet switching to receive, process, and forward data to the destination device.



**Control Plane**

Learns the MAC Address from the incoming packet and populate the MAC Table

**Data Plane**

* Receives the Packet
* Read the Destination MAC from the Packet
* Look up the MAC Table for the destination Port
* Forward the Packet to the destination Port

Openvswitch is soft switch, which works as normal switch(traditional) as well as SDN(OpenFlow)switch

**Demo**

**Same steps as** **Neighbour discovery Demo**

Lets run OVS commands to check the switch

sudo ovs-vsctl --version

sudo ovs-vsctl show

sudo ovs-appctl fdb/show s1

sudo ovs-dpctl dump-flows

**How it works**

1. Traditional Switch have built in Control Plane + Data Plane.
2. Mac Table( a.k.a Forwarding table) is Empty when the switch starts.
3. Control Plane updates the Mac Table with the MAC and the PORT Number. This information is extracted from incoming Packet.
4. Control Plane keeps on building/updating the Mac Table.
5. When the packet arrives, Switch Data plane looks the Mac table, if the destination MAC matches in the Mac table, it forwards the packet to the respective Port.

**1.9 SDN Switch(OpenFlow Switch)**

1. Run the Linear Mininet Topology,

sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --switch=ovsk,protocols=OpenFlow13 --topo=linear,4

1. Start Wireshark Capture
2. RYU L3 Application

ryu-manager ryu.app.simple\_switch\_13

1. In the Mininet cli, ping h1 to h2.

mininet>h1 ping h2

1. Check the OpenFlow flows

sudo ovs-ofctl -O OpenFlow13 dump-flows s1

**How it works**

1. Switch is configured with SDN Controller IP and OpenFlow protocol version.
2. Switch establishes the communication with SDN Controller.
3. SDN Controller installs the default OpenFlow rule (TABLE MISS ENTRY) in the switch Flow table.
4. TABLE MISS Entry OpenFlow rule matches with all the packets and send it to the CONTROLLER. The priority is lowest in the table(0)
5. When the Host Data Packet arrives in the Switch, It will be matched with TABLE MISS ENTRY , and the packet will be sent it to Controller ( PACKET IN Message)
6. Controller receives the packet and build the Switch Logic with the packets.
7. Controller adds the OPENFLOW flows to the switch.
8. Now, Switch data path is built with flows. So next time, when the Packet arrives it will be matched with the Flow table and forward the packet to respective port.

**Chapter 2: Mininet**

**2.1 Basic Operations**

**1. To check the Mininet version**

mn --version

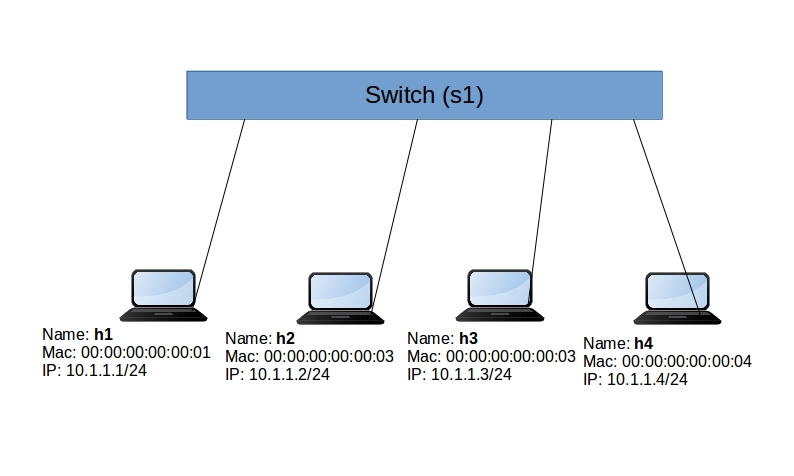
**2. To clean up the existing ovs bridges and namespaces**

Note: sometime we mistakenly closed the Mininet shell, or Mininet crashed. But the topology components will continue to exists. To clean such stuff, cleanup command is used.

mn -c

**3. Our First Topology (Single)**

Topology with Single Switch and 4 Nodes.



RYU SDN Controller

ryu-manager ryu.app.simple\_switch\_13

Mininet Topology

sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --switch=ovsk,protocols=OpenFlow13 --topo=single,4

| **options** | **Description** |
| --- | --- |
| --controller | type of controller local/remote and remote controller Ip. |
|  |  |
| --mac | mac address starts with 00:00:00:00:00:01 |
|  |  |
| -i | IP Subnets for the Topology |
|  |  |
| --switch | Switch type (ovsk - openvswitch kernel module), and OpenFlow version. |
|  |  |
| --topo | topology type(linear,minimal,reversed,single,torus,tree) and params. |

Example:

mininet> exit

\*\*\* Stopping 1 controllers

c0

\*\*\* Stopping 4 links

....

\*\*\* Stopping 1 switches

s1

\*\*\* Stopping 4 hosts

h1 h2 h3 h4

\*\*\* Done

completed in 2.604 seconds

suresh@suresh-vm:~$

**4. Mininet Basic Shell Commands**

**Informative commands**

help

dump

net

links

**Action commands**

pingall

**Execute the commands in HOST/Node:**

Option1:

we can login to each host using 'xterm' command

xterm h1

mininet>xterm h1

It will open a xterm terminal for the host. Now we can execute the command inside that terminal

Option2:

we can directly execute from the Mininet shell.

mininet><hostname> command

Example:

mininet>h1 ifconfig

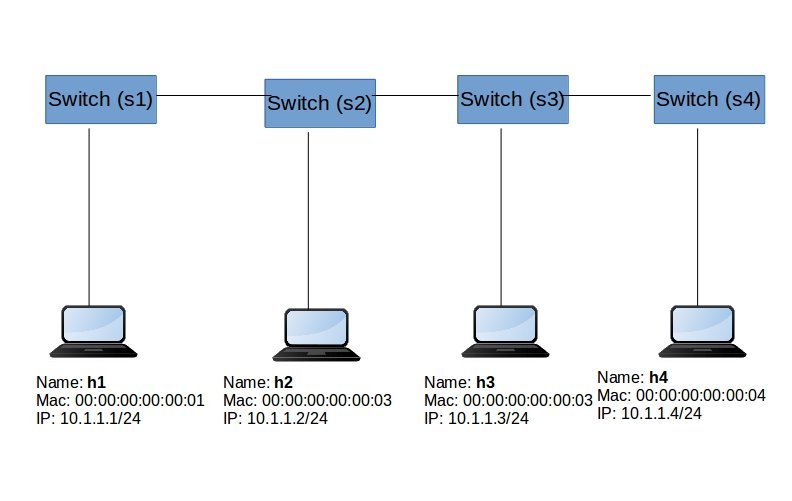
mininet>h1 ping h2

mininet>h1 ip route

we can use either method to run the Traffic tests or executing the commands.

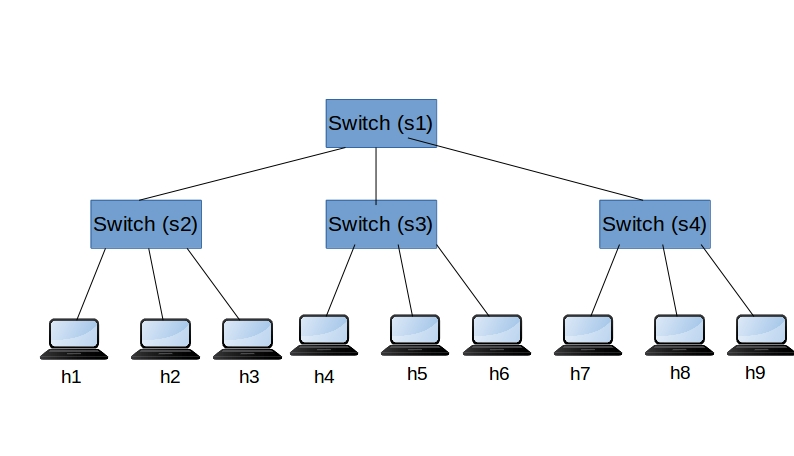
**5. Linear Topology**

linear topology (where each switch has one host, and all switches connect in a line)



sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --switch=ovsk,protocols=OpenFlow13 --topo=linear,4

**6. Tree Topology**



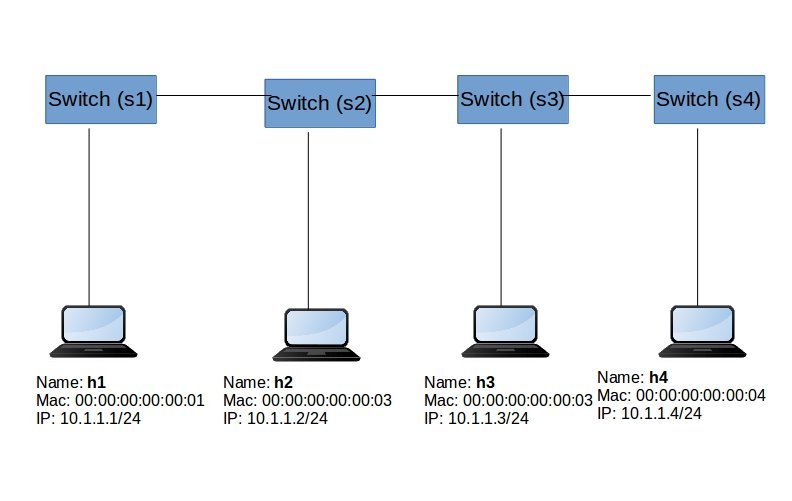
sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --topo=tree,depth=2,fanout=3

fanout : each switch is connected to these many childs depth : depth of the tree

**2.2 Running Traffic Tests**

1. TCP/UDP Traffic Tests

a. Setup the Topology with xterms options (open terminal for each Node)



sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --switch=ovsk,protocols=OpenFlow13 --topo=linear,4 -x

b. TCP Traffic Test between h1 to h4

Run IPERF TCP Server in h4

iperf -s

-s means server mode

RUN IPERF TCP Client in h1

iperf -c 10.1.1.4 -i 10 -t 30

iperf -c 10.1.1.4 -i 10 -b 10m -t 30

iperf -c 10.1.1.4 -i 10 -P 10 -t 30

-c : means client mode.

-I : means reporting interval

-t : means test duration in seconds

-b : means bandwidth 10m means 10Mbps

-P : means parallel connections

c. UDP Traffic Test between h1 to h4

Run IPERF UDP Server in h4

iperf -u -s

-u : means udp

RUN IPERF UDP Client in h1

iperf -u -c 10.1.1.4 -b 10m -i 10 -t 30

iperf -u -c 10.1.1.4 -b 10m -i 10 -P 10 -t 30

-b : means bandwidth 10m means 10Mbps

**HTTP Traffic Tests**

Run Python Simple Web Server in h4

python -m SimpleHTTPServer 80

From H1, Access the Web server

curl http://10.1.1.4/

curl utility used as web client to access the web server.

If we want to simulate the 1000s users accessing the web server on the same time (load), we can use ab(Apache bench) tool

ab -n 500 -c 50 http://10.1.1.4/

-c 50 means parallel request per second (50 Requests per second)

-n 500 means total request for this test (500 requests)

**2.3 Writing Custom Topology in Mininet**

Mininet exposes the python API. We can create a custom topology using the python API with few lines of code.

**How to write Custom Topology in Mininet**

Steps are below.

1. Import the python required libraries

from mininet.topo import Topo

from mininet.net import Mininet

1. Write the Topology definition class

class SingleSwitchTopo(Topo):

def build(self):

s1 = self.addSwitch('s1')

h1 = self.addHost('h1')

h2 = self.addHost('h2')

h3 = self.addHost('h3')

h4 = self.addHost('h4')

self.addLink(h1, s1)

self.addLink(h2, s1)

self.addLink(h3, s1)

self.addLink(h4, s1)

Important Topology definition APIs:

addSwitch

addHost

addLink

1. Run the Topology as below,

* Create the Topology object
* Create the Mininet with Topology object
* Start the Mininet

if \_\_name\_\_ == '\_\_main\_\_':

topo = SingleSwitchTopo()

c1 = RemoteController('c1', ip='127.0.0.1')

net = Mininet(topo=topo, controller=c1)

net.start()

**How to run**

1. start the RYU SDN Controller

ryu-manager ryu.app.simple\_switch\_13

1. Run the Mininet topology file

sudo python <topology file name>

1. Perform your tests/operation etc.

**Chapter 3: OpenFlow Theory**

**3.1 Introduction**

The OPENFLOW specification covers the components and the basic functions of the switch, and the OpenFlow switch protocol to manage an OpenFlow switch from a remote OpenFlow controller.

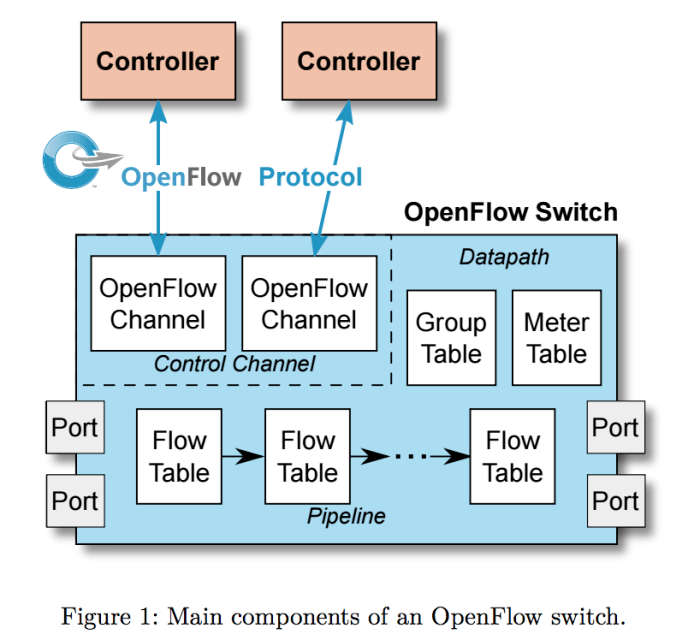
OpenFlow Version Details:

OpenFlow 1.1 OpenFlow 1.2 OpenFlow 1.3 OpenFlow 1.4 OpenFlow 1.5

Most widely used: 1.3

**3.2 Switch Components**

An OpenFlow Logical Switch consists of one or more flow tables and a group table, which perform packet lookups and forwarding, and one or more OpenFlow channels to an external controller (Figure 1).



* The switch communicates with the controller and the controller manages the switch via the OpenFlow switch protocol.
* Using the OpenFlow switch protocol, the controller can add, update, and delete flow entries in flow tables, both reactively (in response to packets) and proactively.
* Each flow table in the switch contains a set of flow entries; each flow entry consists of match fields, counters, and a set of instructions to apply to matching packets.
* Matching starts at the first flow table and may continue to additional flow tables of the pipeline
* Flow entries match packets in priority order, with the first matching entry in each table being used.
* If a matching entry is found, the instructions associated with the specific flow entry are executed.
* If no match is found in a flow table, the outcome depends on configuration of the table-miss flow entry: for example, the packet may be forwarded to the controllers over the OpenFlow channel, dropped, or may continue to the next flow table
* Actions included in instructions describe packet forwarding, packet modification and group table processing.
* Flow entries may forward to a port. This is usually a physical port, but it may also be a logical port defined by the switch(such as link aggregation groups, tunnels or loopback interfaces) or a reserved port defined by this specification.

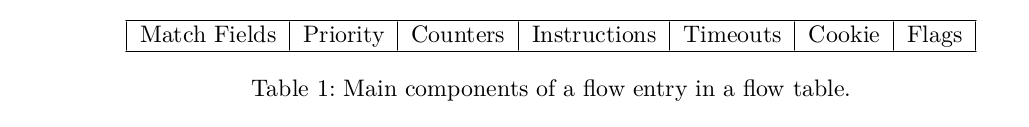
**3.3 OpenFlow channel**

* The OpenFlow channel is the interface that connects each OpenFlow Logical Switch to an OpenFlow controller. Through this interface, the controller configures and manages the switch, receives events from the switch, and sends packets out the switch.
* The Control Channel of the switch may support a single OpenFlow channel with a single controller, or multiple OpenFlow channels enabling multiple controllers.
* The OpenFlow channel is usually encrypted using TLS, but may be run directly over TCP
* Default Port number : 6653

**3.4 OpenFlow Flow table**

A flow table entry is identified by its match fields and priority: the match fields and priority taken together identify a unique flow entry in a specific flow table.

A flow table consists of flow entries.



**Example Flows with MAC Match**

cookie=0x0, duration=4.742s, table=0, n\_packets=2, n\_bytes=196, priority=1,in\_port="s1-eth2",dl\_src=00:00:00:00:11:12,dl\_dst=00:00:00:00:11:11 actions=output:"s1-eth1"

cookie=0x0, duration=4.738s, table=0, n\_packets=1, n\_bytes=98, priority=1,in\_port="s1-eth1",dl\_src=00:00:00:00:11:11,dl\_dst=00:00:00:00:11:12 actions=output:"s1-eth2"

cookie=0x0, duration=5.781s, table=0, n\_packets=29, n\_bytes=3102, priority=0 actions=CONTROLLER:65535

**Example Flows with IP Match**

cookie=0x0, duration=12.927s, table=0, n\_packets=2, n\_bytes=196, priority=1,ip,nw\_src=192.168.1.1,nw\_dst=192.168.1.2 actions=output:"s1-eth2"

cookie=0x0, duration=12.918s, table=0, n\_packets=2, n\_bytes=196, priority=1,ip,nw\_src=192.168.1.2,nw\_dst=192.168.1.1 actions=output:"s1-eth1"

cookie=0x0, duration=12.959s, table=0, n\_packets=37, n\_bytes=3844, priority=0 actions=CONTROLLER:65535

**Example Flows with TCP/UDP Ports Match**

suresh@suresh-vm:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

cookie=0x0, duration=3.933s, table=0, n\_packets=238752, n\_bytes=11069190464, priority=1,tcp,nw\_src=192.168.1.2,nw\_dst=192.168.1.1,tp\_src=37304,tp\_dst=5001 actions=output:"s1-eth1"

cookie=0x0, duration=3.906s, table=0, n\_packets=192421, n\_bytes=12699810, priority=1,tcp,nw\_src=192.168.1.1,nw\_dst=192.168.1.2,tp\_src=5001,tp\_dst=37304 actions=output:"s1-eth2"

cookie=0x0, duration=31.495s, table=0, n\_packets=43, n\_bytes=4309, priority=0 actions=CONTROLLER:65535

suresh@suresh-vm:~$

**match fields:** to match against packets. These consist of the ingress port and packet headers, and optionally other pipeline fields such as metadata specified by a previous table

**priority:** matching precedence of the flow entry.

**counters:** updated when packets are matched.

**instructions:** to modify the action set or pipeline processing.

**timeouts:** maximum amount of time or idle time before flow is expired by the switch.

**cookie:** opaque data value chosen by the controller. May be used by the controller to filter flow entries affected by flow statistics, flow modification and flow deletion requests. Not used when processing packets.

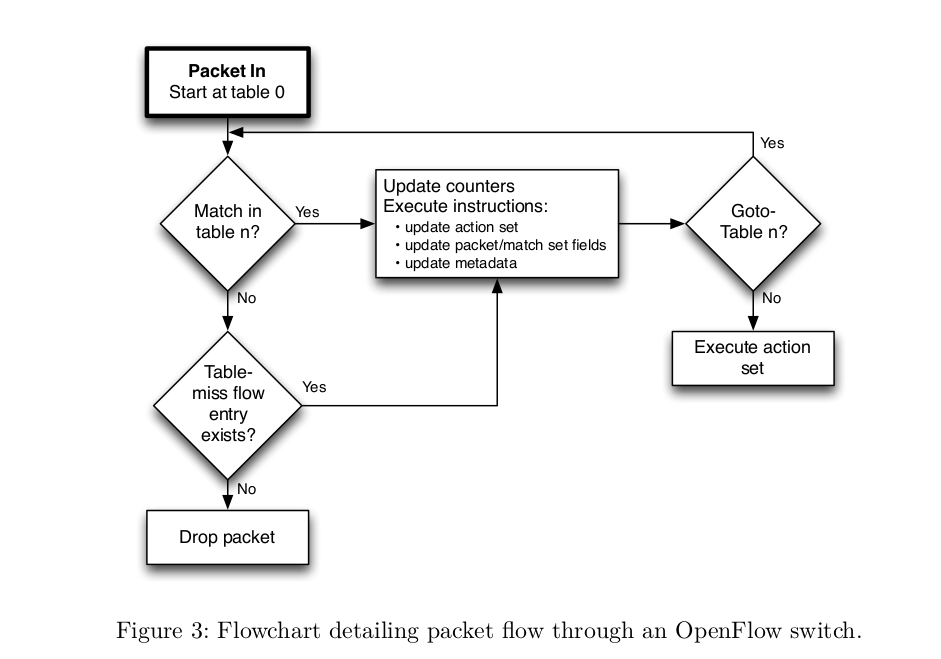
**flags:** flags alter the way flow entries are managed, for example the flag OFPFF\_SEND\_FLOW\_REM triggers flow removed messages for that flow entry.

The flow entry that wildcards all match fields (all fields omitted) and has priority equal to 0 is called the table-miss flow entry.

The table-miss flow entry must support at least sending packets to the controller using the CONTROLLER reserved port.

**OpenFlow Matching**

On receipt of a packet, an OpenFlow Switch performs the functions shown as below.



The switch starts by performing a table lookup in the first flow table, and based on pipeline processing, may perform table lookups in other flow tables

Switch input port. \*/

Switch physical input port. \*

Ethernet destination address. \*/

Ethernet source address. \*/

Ethernet frame type. \*/

VLAN id. \*/

VLAN priority. \*/

IP DSCP (6 bits in ToS field). \*/

IP ECN (2 bits in ToS field). \*/

IP protocol. \*/

IPv4 source address. \*/

IPv4 destination address. \*/

TCP source port. \*/

TCP destination port. \*/

UDP source port. \*/

UDP destination port. \*/

SCTP source port. \*/

SCTP destination port. \*/

ICMP type. \*/

ICMP code. \*/

ARP opcode. \*/

ARP source IPv4 address. \*/

ARP target IPv4 address. \*/

ARP source hardware address. \*/

ARP target hardware address. \*/

IPv6 source address. \*/

IPv6 destination address. \*/

IPv6 Flow Label \*/

ICMPv6 type. \*/

ICMPv6 code. \*/

Target address for ND.

Source link-layer for ND. \*/

Target link-layer for ND. \*/

MPLS label. \*/

MPLS TC. \*/

MPLS BoS bit. \*/

PBB I-SID. \*/

Logical Port Metadata. \*/

IPv6 Extension Header pseudo-field \*/

* Match fields come in two types, header match fields and pipeline match fields.
* Header match fields are match fields matching values extracted from the packet headers. Most header match fields map directly to a specific field in the packet header defined by a Datapath protocol.
* All header match fields have different size, prerequisites and masking capability
* Pipeline match fields are match fields matching values attached to the packet for pipeline processing and not associated with packet headers, such as META\_DATA, TUNNEL\_ID. Refer : Page 66.

**Prerequisites Example:**

If you want to include the SRC IP Address in the match, Prerequisites is ETH\_TYPE should be 0X0800. It means, you need to include the ETH\_TYPE match also.

**Masking example:**

You can match the Source IP against with Subnet. Refer page 64 for detailed table.

**Instructions**

Each flow entry contains a set of instructions that are executed when a packet matches the entry. These instructions result in changes to the packet, action set and/or pipeline processing.

**Apply-Actions action(s):**

* Applies the specific action(s) immediately, without any change to the Action Set.
* This instruction may be used to modify the packet between two tables or to execute multiple actions of the same type.
* The actions are specified as a list of actions

**Clear-Actions:**

* Clears all the actions in the action set immediately.

**Go to-Table next-table-id:**

* Indicates the next table in the processing pipeline. The table-id must be greater than the current table-id.

**Write-Actions action(s):**

* Merges the specified set of action(s) into the current action set

**Meter id:**

* Direct packet to the specified meter.

**Action Set and Actions:**

Action set contains set of actions.

Example actions are:

* Output port no
* Group id
* Drop
* Set-Queue queue id
* Push-Tag/Pop-Tag ether type (VLAN, MPLS, PBP)
* Set-Field field type value
* Change-TTL (IP TTL, MPLS TTL)

**Flow Removal:**

Flow entries are removed from flow tables in two ways, either at the request of the controller or via the switch flow expiry mechanism.

**Flow expiry:**

Each flow entry has an idle\_timeout and a hard\_timeout associated with it

Example:

cookie=0x0, duration=7.619s, table=0, n\_packets=3, n\_bytes=238, idle\_timeout=10, hard\_timeout=30, priority=1,in\_port="s1-eth2",dl\_src=00:00:00:00:11:12,dl\_dst=00:00:00:00:11:11 actions=output:"s1-eth1"

cookie=0x0, duration=7.605s, table=0, n\_packets=2, n\_bytes=140, idle\_timeout=10, hard\_timeout=30, priority=1,in\_port="s1-eth1",dl\_src=00:00:00:00:11:11,dl\_dst=00:00:00:00:11:12 actions=output:"s1-eth2"

cookie=0x0, duration=8.652s, table=0, n\_packets=33, n\_bytes=3527, priority=0 actions=CONTROLLER:65535

**Hard\_timeout :** If the hard\_timeout field is non-zero, the switch must note the flow entry’s arrival time, as it may need to evict the entry later. A non-zero hard\_timeout field causes the flow entry to be removed after the given number of seconds, regardless of how many packets it has matched.

**idle\_timeout :** If the idle\_timeout field is non-zero, the switch must note the arrival time of the last packet associated with the flow, as it may need to evict the entry later. A non-zero idle\_timeout field causes the flow entry to be removed when it has matched no packets in the given number of seconds. The switch must implement flow expiry and remove flow entries from the flow table when one of their timeouts is exceeded.

**Counters:**

statistics are maintained by the OpenFlow switch as below,

* Per flow entry
* Per flow table
* Per Switch Port
* Per Queue
* Per Group
* Per Group Bucket
* Per Meter
* Per Meter Band

**Three types of messages.**

**1. Controller to Switch**

Controller-to-switch messages are initiated by the controller and used to directly manage or inspect the state of the switch

* Feature Request
* Packet Out
* Modify Flow Table
* Modify Group Table
* Modify Meter Table
* OpenFlow Switch Description Request
* OpenFlow Port Description Request
* OpenFlow Statistics Request (Flow, Port, Flow table, Aggregate, Group, Meter, Queue )
* Role Request
* Barrier Request

**2. Asynchronous**

Asynchronous messages are initiated by the switch and used to update the controller about network events and changes to the switch state

* Packet In
* Flow Removed

**3. Symmetric**

Symmetric messages are initiated by either the switch or the controller and sent without solicitation.

* Hello Message
* Echo Message

**Message transaction - during the Topology Setup**

1. Hello
2. Feature request/Response
3. Switch/Port Description Request/Response
4. Modify Flow Entry (To install table Miss entry)
5. Packet IN (Switch to Controller)
6. Packet Out (Controller to Switch)
7. Modify Flow Entry (Install a flow)
8. Echo

**Hello Message:**

Hello messages are exchanged between the switch and controller upon connection startup.

* Switch sends OpenFlow Hello Message(includes version number) to the Controller
* Controller responds with the Hello Message if version is supported.
* Failure Case(Version Mismatch): If different OpenFlow Version is user between the Controller and Switch, Hello Message will fail.
* You will see similar error msg in the controller.

Error:

unsupported version 0x1. If possible, set the switch to use one of the versions [3]

**Echo Message:**

Echo request/reply messages can be sent from either the switch or the controller, and must return an echo reply. They are mainly used to verify the liveness of a controller-switch connection and may as well be used to measure its latency or bandwidth. (Default: 5sec interval)

A. Switch sends Echo Request to the Controller.

B. Controller responds back with Echo Reply.

**Features Request/Reply:**

* The controller may request the identity and the basic capabilities of a switch by sending a features request;
* The switch must respond with a features reply that specifies the identity and basic capabilities (Datapath ID, buffers, number of tables, statistics) of the switch.
* This is commonly performed upon establishment of the OpenFlow channel.

**Packet-in:**

Transfer the control of a packet to the controller. For all packets forwarded to the CONTROLLER reserved port using a flow entry or the table-miss flow entry, a packet-in event is always sent to controllers

**Packet-out Message:**

These are used by the controller to send packets out of a specified port on the switch, and to forward packets received via Packet-in messages. Packet-out messages must contain a full packet or a buffer ID referencing a packet stored in the switch.

The message must also contain a list of actions to be applied in the order they are specified; an empty list of actions drops the packet.

**Modify Flow Entry Message:**

Modifications to a flow table from the controller are done with the OFPT\_FLOW\_MOD message:

* To add, remove, modify the flow in the switch, controller using this message.
* Controller Sends the Flow Modification message to the switch with this important params (Command, Match, Instruction, action.)
* Command: ADD, MODIFY, MODIFY\_STRICT, DELETE, DELETE\_STRICT

**3.5 OpenFlow Ports:**

**Physical ports:**

The OpenFlow physical ports are switch defined ports that correspond to a hardware interface of the switch. In the Virtualized environment it represents the virutal interface.

Example: "s1-eth1"

**Logical ports:**

Logical ports are higher level abstractions that may be defined in the switch using non-OpenFlow methods (e.g. link aggregation groups, tunnels, loopback interfaces).

Example: "vxlan0"

**Reserved ports:**

The OpenFlow reserved ports are defined by this specification. They specify generic forwarding actions such as sending to the controller, flooding, or forwarding using non-OpenFlow methods, such as “normal” switch processing. FLOOD, ALL, CONTROLLER, IN PORT, LOCAL, NORMAL,

Example: FLOOD

**3.6 Important Take aways**

1) OpenFlow version should match between the switch and Controller

2) Our Controller Application (our RYU project/exercise) should process Packet IN (Message), to build the Switching/Routing logic.

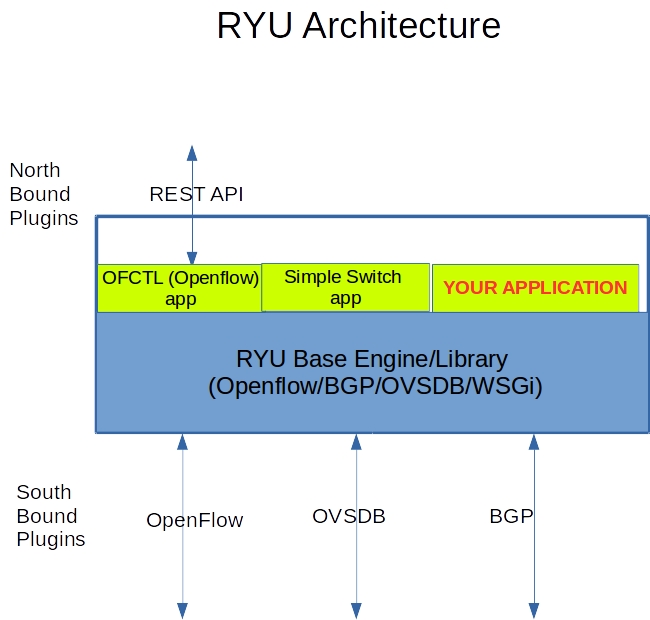
3) Our Controller Application (our RYU project/exercise) should use Flow Modification message to add/modify/delete the flows in the switch.

4) Our Controller Application (our RYU project/exercise) should use Flow Stats, Port Stats request message to get the statistics (Packets Sent/Received, etc.) of the flows, Ports .

**Chapter 4: Controllers**

**4.1 Ryu Controller**

**Ryu** is a component-based software defined networking framework. Ryu provides software components with well-defined API that make it easy for developers to create new network management and control applications. Ryu supports various protocols for managing network devices, such as OpenFlow, OVSDB, BGP. About OpenFlow, Ryu supports fully 1.0, 1.2, 1.3, 1.4, 1.5 and Nicira Extensions. All of the code is freely available under the Apache 2.0 license.



**How to run RYU applications**

**run the Ryu application:**

ryu-manager ryu.app.application-name

Example:

ryu-manager ryu.app.simple\_switch\_13

In built applications are available in

<https://github.com/osrg/ryu/tree/master/ryu/app>

**Some of the applications are**

1. simple\_switch
2. simple\_monitor
3. ofctl\_rest
4. rest\_qos
5. rest\_firewall
6. rest\_router

ryu-manager ryu.app.simple\_switch\_13

Example:

suresh@suresh-vm:~$ ryu-manager ryu.app.simple\_switch\_13

loading app ryu.app.simple\_switch\_13

loading app ryu.controller.ofp\_handler

instantiating app ryu.app.simple\_switch\_13 of SimpleSwitch13

instantiating app ryu.controller.ofp\_handler of OFPHandler

**Check the OpenFlow port status**

RYU Manager listens on OpenFlow ports(6653) are in listening state.

netstat -ap | grep 6653

suresh@suresh-vm:~$ netstat -ap | grep 6653

(Not all processes could be identified, non-owned process info

will not be shown, you would have to be root to see it all.)

tcp 0 0 0.0.0.0:6653 0.0.0.0:\* LISTEN 19190/python

suresh@suresh-vm:~$

**How to stop the Ryu controller**

CTRL + C (Kill the Process)

**How to run your (custom developed) applications.**

RYU application is a python script.

ryu-manager <python-file-name>

Example:

ryu-manager l3\_switch.py

**How to run your multiple applications.**

RYU can run multiple applications in a single initiation.

ryu-manager <application1> <application2>

Example:

ryu-manager ryu.app.simple\_switch\_13 ryu.app.ofctl\_rest

**RYU Controller command line options**

**To know all the available options:**

ryu-manager --help

**To enable the debug logs:**

ryu-manager --verbose application\_name

**To use custom OpenFlow port number:**

ryu-manager --ofp-tcp-listen-port 6634 application\_name

Example:

ryu:

ryu-manager --ofp-tcp-listen-port 6634 ryu.app.simple\_switch\_13

Mininet:

sudo mn --controller=remote,ip=127.0.0.1:6634 --switch=ovsk,protocols=OpenFlow13 --topo=linear,4

**To use topology discovery**

ryu-manager --observe-links application\_name

Example:

ryu-manager --observe-links ryu.app.simple\_switch\_13

**4.2 Reactive/Proactive Flows**

**Reactive Flows:**

* When the new packet enters in the switch, if it doesn't match on the existing flows, Switch sends it to the controller.
* controller inspect the packet, and build the logic
* install the flow for that session(match) in the switch Packet IN /Packet Out

**Proactive Flows:**

* OpenFlow controller will install flow tables ahead of time for all traffic matches.

**Simple Proactive Hub Application**

Install the OpenFlow flow in the switch which performs FLOOD action, when switch connects to the controller.

**Testing**

1. Run Mininet topology

sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4

suresh@suresh-vm:~/sdn$ sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4

[sudo] password for suresh:

\*\*\* Creating network

\*\*\* Adding controller

Unable to contact the remote controller at 127.0.0.1:6653

Unable to contact the remote controller at 127.0.0.1:6633

Setting remote controller to 127.0.0.1:6653

\*\*\* Adding hosts:

h1 h2 h3 h4

\*\*\* Adding switches:

s1

\*\*\* Adding links:

(h1, s1) (h2, s1) (h3, s1) (h4, s1)

\*\*\* Configuring hosts

h1 h2 h3 h4

\*\*\* Starting controller

c0

\*\*\* Starting 1 switches

s1 ...

\*\*\* Starting CLI:

mininet>

1. Run RYU hub application

ryu-manager hub.py

suresh@suresh-vm:~/sdn$ ryu-manager hub.py

loading app hub.py

loading app ryu.controller.ofp\_handler

instantiating app ryu.controller.ofp\_handler of OFPHandler

instantiating app hub.py of SimpleSwitch13

1. Check the flows

sudo ovs-ofctl -O OpenFlow13 dump-flows s1

suresh@suresh-vm:~/sdn$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

[sudo] password for suresh:

cookie=0x0, duration=16.055s, table=0, n\_packets=5, n\_bytes=350, priority=0 actions=FLOOD

suresh@suresh-vm:~/sdn$

1. Perform Ping between the hosts in Mininet

mininet> pingall

\*\*\* Ping: testing ping reachability

h1 -> h2 h3 h4

h2 -> h1 h3 h4

h3 -> h1 h2 h4

h4 -> h1 h2 h3

\*\*\* Results: 0% dropped (12/12 received)

mininet>

1. Watch the flows again

suresh@suresh-vm:~/sdn$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

cookie=0x0, duration=654.092s, table=0, n\_packets=72, n\_bytes=5040, priority=0 actions=FLOOD

suresh@suresh-vm:~/sdn$

**4.3 OpenFlow Applications**

**Applications are Part of SDN Controller**

* Most of the RYU Applications are this category. example, simple\_switch application, monitor application.
* To develop this application, we should know the RYU Python API, and RYU Programming guidelines.
* Most of the academic projects will be developed in this type.

**Application sits externally, and communicate with SDN Controller thru North bound plugin**

* User using Northbound interfaces (REST API) to add the flows,
* Packet In /Packet Out will not be considered as the flow control is handled externally by the user or external application.
* No Packet generation capability (Packet out)

**Basic Openvswitch Commands**

ovs-vsctl show

ovs-ofctl -O OpenFlow13 dump-flows <bridgename>

ovs-ofctl -O OpenFlow13 show <bridgename>

**4.4 Simple Switch Application (in built)**

Simple Switch Application is RYU inbuilt basic switching application works in reactive model.

* Install the Table Miss entry to the switch
* When the packet comes to Switch, it matches with Table Miss Entry, then Switch send it to the Controller(PACKET IN message)
* Controller look the src mac of the packet and updates in its db(port to mac mapping)
* Controller look the destination mac of the packet, and decides on the output port.
* Controller send the packet to switch (PACKET OUT message)

**Testing**

1. Run Mininet topology

sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4

suresh@suresh-vm:~/sdn$ sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4

[sudo] password for suresh:

\*\*\* Creating network

\*\*\* Adding controller

Unable to contact the remote controller at 127.0.0.1:6653

Unable to contact the remote controller at 127.0.0.1:6633

Setting remote controller to 127.0.0.1:6653

\*\*\* Adding hosts:

h1 h2 h3 h4

\*\*\* Adding switches:

s1

\*\*\* Adding links:

(h1, s1) (h2, s1) (h3, s1) (h4, s1)

\*\*\* Configuring hosts

h1 h2 h3 h4

\*\*\* Starting controller

c0

\*\*\* Starting 1 switches

s1 ...

\*\*\* Starting CLI:

mininet>

1. Run RYU simple switch application

ryu-manager ryu.app.simple\_switch\_13

suresh@suresh-vm:~/sdn$ ryu-manager ryu.app.simple\_switch\_13

loading app ryu.app.simple\_switch\_13

loading app ryu.controller.ofp\_handler

instantiating app ryu.app.simple\_switch\_13 of SimpleSwitch13

instantiating app ryu.controller.ofp\_handler of OFPHandler

packet in 1 00:00:00:00:00:03 33:33:00:00:00:02 3

packet in 1 00:00:00:00:00:04 33:33:00:00:00:02 4

packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1

packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2

1. Check the switch & flows

Table Miss entry to be present

suresh@suresh-vm:~/sdn$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

cookie=0x0, duration=39.737s, table=0, n\_packets=8, n\_bytes=560, priority=0 actions=CONTROLLER:65535

suresh@suresh-vm:~/sdn$

1. Do h1 to h2 ping from Mininet prompt

mininet> h1 ping h2

PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

64 bytes from 10.0.0.2: icmp\_seq=1 ttl=64 time=18.5 ms

64 bytes from 10.0.0.2: icmp\_seq=2 ttl=64 time=0.672 ms

64 bytes from 10.0.0.2: icmp\_seq=3 ttl=64 time=0.119 ms

64 bytes from 10.0.0.2: icmp\_seq=4 ttl=64 time=0.148 ms

^C

--- 10.0.0.2 ping statistics ---

4 packets transmitted, 4 received, 0% packet loss, time 3027ms

rtt min/avg/max/mdev = 0.119/4.869/18.540/7.896 ms

mininet>

1. Check the flows again

suresh@suresh-vm:~/sdn$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

cookie=0x0, duration=35.152s, table=0, n\_packets=5, n\_bytes=434, priority=1,in\_port="s1-eth2",dl\_src=00:00:00:00:00:02,dl\_dst=00:00:00:00:00:01 actions=output:"s1-eth1"

cookie=0x0, duration=35.140s, table=0, n\_packets=4, n\_bytes=336, priority=1,in\_port="s1-eth1",dl\_src=00:00:00:00:00:01,dl\_dst=00:00:00:00:00:02 actions=output:"s1-eth2"

cookie=0x0, duration=139.207s, table=0, n\_packets=19, n\_bytes=1302, priority=0 actions=CONTROLLER:65535

suresh@suresh-vm:~/sdn$

1. Look the Priority, Match and Action field

**Priority:**

priority=1

Match:

in\_port="s1-eth2",dl\_src=00:00:00:00:00:02,dl\_dst=00:00:00:00:00:01

Action:

output:"s1-eth1"

**Simple L3 Switch**

This exercise is same as Simple Switch Application, except Match is based on IP address.

1. Run Mininet topology

sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4

1. Run RYU l3switch application

ryu-manager l3\_switch.py

1. Check the switch & flows
2. Do h1 to h2 ping from Mininet prompt
3. Check the flows again

suresh@suresh-vm:~/sdn$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1

cookie=0x0, duration=10.369s, table=0, n\_packets=1, n\_bytes=98, priority=1,ip,nw\_src=10.0.0.1,nw\_dst=10.0.0.2 actions=output:"s1-eth2"

cookie=0x0, duration=10.361s, table=0, n\_packets=1, n\_bytes=98, priority=1,ip,nw\_src=10.0.0.2,nw\_dst=10.0.0.1 actions=output:"s1-eth1"

cookie=0x0, duration=17.720s, table=0, n\_packets=10, n\_bytes=644, priority=0 actions=CONTROLLER:65535

suresh@suresh-vm:~/sdn$

1. Look the Priority, Match and Action field

Priority:

priority=1

Match:

ip,nw\_src=10.0.0.1,nw\_dst=10.0.0.2

Action:

output:"s1-eth2"

**4.5 RYU Programming - Basics**

**Ryu** applications are just Python scripts so you can save the file as (.py) on any location.

**Components**

ryu-manager :

* main executable

ryu.base.app\_manager:

* The central management of Ryu applications
* Load Ryu applications
* Provide contexts to Ryu applications

ryu.ofproto:

* OpenFlow wire protocol encoder and decoder:

ryu.controller:

* OpenFlow controller implementation
* generates OpenFlow events

ryu.packet:

* all packet parsing libraries

Events:

RYU Application works based on Events. RYU Controller emits the events for the OpenFlow Messages received. This can be handled by the RYU Applications.

Example Events:

ofp\_event.EventOFPSwitchFeatures ofp\_event.EventOFPPacketIn ofp\_event.EventOFPFlowStatsReply

**4.6 Closer look on Simple\_Switch\_13 application**

Let's have quick Recap on the OpenFlow Message transactions between Controller and Switch.

1. Hello Message
2. Feature Request/Response Message
3. Flow Modification message to install Table Miss Entry
4. Packet In Message
5. Packet Out Message
6. Flow Modification Message to Install the Flows

we can classify the program in few important parts.

1.Import the base classes / library

from ryu.base import app\_manager

from ryu.controller import ofp\_event

from ryu.controller.handler import CONFIG\_DISPATCHER, MAIN\_DISPATCHER

from ryu.controller.handler import set\_ev\_cls

from ryu.ofproto import ofproto\_v1\_3

from ryu.lib.packet import packet

from ryu.lib.packet import ethernet

from ryu.lib.packet import ether\_types

1. Application Class (derived from app\_manager)

class SimpleSwitch13(app\_manager.RyuApp):

OFP\_VERSIONS = [ofproto\_v1\_3.OFP\_VERSION]

def \_\_init\_\_(self, \*args, \*\*kwargs):

super(SimpleSwitch13, self).\_\_init\_\_(\*args, \*\*kwargs)

self.mac\_to\_port = {}

1. Important openflow events handled

Features\_Response\_Received

PacketIn\_Message\_Received

we can handle this event, in our application as below,

@set\_ev\_cls(ofp\_event.EventOFPSwitchFeatures, CONFIG\_DISPATCHER)

def switch\_features\_handler(self, ev):

datapath = ev.msg.datapath

.....skipped......

@set\_ev\_cls(ofp\_event.EventOFPPacketIn, MAIN\_DISPATCHER)

def \_packet\_in\_handler(self, ev):

# If you hit this you might want to increase

# the "miss\_send\_length" of your switch

if ev.msg.msg\_len < ev.msg.total\_len:

self.logger.debug("packet truncated: only %s of %s bytes",

ev.msg.msg\_len, ev.msg.total\_len)

.....skipped......

In the switch\_features\_handler event, we add the TABLE MISS Entry . So we get the packet in messages.

In the packet\_in\_handler event, we process the PACKETs and parse the src\_mac and dst\_mac address and build the switching logic and install the flow.

**Inserting(Adding) a flow :**

Flow table consists of Match, Actions and Counters.

**A. Create a Match**

This match is with no match fields. it means matching all the packets.

match = parser.OFPMatch()

This matches with in\_port, eth\_dst and eth\_src field.

match = parser.OFPMatch(in\_port=in\_port, eth\_dst=dst, eth\_src=src)

**B. Create a Actions**

Below action is send it to CONTROLLER (Reserved port).

actions = [parser.OFPActionOutput(ofproto.OFPP\_CONTROLLER,ofproto.OFPCML\_NO\_BUFFER)

Below action, send it to port number 1.

out\_port = 1

actions = [parser.OFPActionOutput(out\_port)]

**C. Create an Instruction list with Actions**

inst = [parser.OFPInstructionActions(ofproto.OFPIT\_APPLY\_ACTIONS,actions)]

**D. Send OFP Flow Modification message for creating a new flow**

Most of the parameter are default. table\_id, timeouts, cookies etc.

mod = parser.OFPFlowMod(datapath=datapath, buffer\_id=buffer\_id,

priority=priority, match=match,

instructions=inst)

In Below example, explicitly specify table id, timeouts.

mod = parser.OFPFlowMod(datapath=datapath, table\_id=10,

buffer\_id=buffer\_id,idle\_timeout=10,

hard\_timeout=30, priority=priority,

match=match, instructions=inst)

**4.7 Developing Switching Applications**

we use simple\_switch\_13 application as base application, and will build hub, L3, L4 Match applications.

**4.8 HUB Application**

**Logic**

Create a Flow, all Matches with FLOOD Action

No need of TABLE MISS Entry.

So, we can simply modify the TABLE MISS Entry action as FLOOD to achieve the HUB Operations.

**Code changes**

In the Switch Features handler, modify the action as FLOOD.

actions = [parser.OFPActionOutput(port=ofproto.OFPP\_FLOOD)]

We will never get Packet\_in event, So we can remove those routines.

Save this file as hub.py

**Demo**

1. Run Mininet topology

sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4

1. Run RYU hub application

ryu-manager hub.py

1. Check the openvswitch flows

sudo ovs-ofctl -O OpenFlow13 dump-flows s1

1. do pingall from mininet.

**4.9 Switch Application With L3 Match**

**Logic**

* simple\_switch\_13 application as a base application and switch learning process is same
* But Flow will be based on Layer3 Match (src ip and destination ip) instead of src\_mac and dst\_mac

**Code changes**

1. include the IP library

from ryu.lib.packet import ipv4

1. Populate the Match based on IP.

* Check the packet is IP Packet, then decode the srcip and dstip from the packet header
* Populate the Match based on srcip and dstip.

# check IP Protocol and create a match for IP

if eth.ethertype == ether\_types.ETH\_TYPE\_IP:

ip = pkt.get\_protocol(ipv4.ipv4)

srcip = ip.src

dstip = ip.dst

match = parser.OFPMatch(eth\_type=ether\_types.ETH\_TYPE\_IP,

ipv4\_src=srcip,

ipv4\_dst=dstip

)

**Demo**

1. Run Mininet topology

sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4

1. Run RYU hub application

ryu-manager l3\_switch.py

1. Do pingall from Mininet.
2. Check the openvswitch flows.

sudo ovs-ofctl -O OpenFlow13 dump-flows s1

**4.10 Statistics Collection**

OpenFlow protocol provides extensive statistics, such as,

* flow statistics
* flow aggregate statistics
* port statistics
* group table statistics
* meter statistics

User can collect these statistics from the OpenFlow switch and use it(its used for many applications)

User needs to send the statistics request message (Example: OpenFlow Flow Statistics Request Message). Switch reply with statistics response message.

In this tutorial, we are going to write a RYU Code to display the statistics.

**4.11 RYU Thread API**

RYU comes with inbuilt thread implementation (hub library.)

Import the library

from ryu.lib import hub

We will write a piece of code, which will be executed in separate thread forever.

def myfunction(self):

self.logger.info(" started new thread")

i = 0

while True:

hub.sleep(30)

self.logger.info("printing %d", i)

i = i + 1

Note: its a continuous run, so we use hub.sleep() to sleep for a time.

To initiate a thread

hub.spawn(self.myfunction)

**Demo**

ryu-manager thread.py

Example:

$ ryu-manager thread1.py

loading app thread1.py

loading app ryu.controller.ofp\_handler

instantiating app ryu.controller.ofp\_handler of OFPHandler

instantiating app thread1.py of SimpleSwitch13

started new thread

printing 0

printing 1

printing 2

**4.12 Flow Statistics**

Measure the flows statistics or utilization in regular interval(10second) and print it.

**Logic**

* Use Ryu Thread(HUB) mechanism,
* In the Thread, Send OpenFlow Flow Statistics Request message at regular interval
* Flow Statistics Response will be received as event, and display the bytes & packets counter values.

**Coding**

1. Use Simple Switch application as base application
2. we require data paths object of all the switches for generating the stats request message. hence, we store the data paths in the dictionary.

In the init routine, we declare it.

self.datapaths = {}

In the switch features handler function, we store the Datapath object in the data paths dictionary.

self.datapaths[datapath.id] = datapath

1. Include RYU Thread library

from ryu.lib import hub

1. Start the thread in the init routine

self.monitor\_thread = hub.spawn(self.monitor)

Here thread function name is monitor.

1. Thread function

The thread function to be placed in side the RYU manager application class.

In the thread function, we process the data paths(switches) dictionary, and generate the flow stats request message(without any match filter). It means we query the stats of all the flows in the switch.

def monitor(self):

self.logger.info("start flow monitoring thread")

while True:

hub.sleep(10)

for datapath in self.datapaths.values():

ofp = datapath.ofproto

ofp\_parser = datapath.ofproto\_parser

req = ofp\_parser.OFPFlowStatsRequest(datapath)

datapath.send\_msg(req)

1. Flow Stats Response handling routine.

@set\_ev\_cls([ofp\_event.EventOFPFlowStatsReply,

], MAIN\_DISPATCHER)

def stats\_reply\_handler(self, ev):

for stat in ev.msg.body:

self.logger.info("Flow details: %s ",stat)

self.logger.info("byte\_count: %d ", stat.byte\_count)

self.logger.info("packet\_count: %d ", stat.packet\_count)

**Demo**

1. start the Mininet single topology

sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --switch=ovsk,protocols=OpenFlow13 --topo=single,4

1. run our application.

ryu-manager flow\_stats.py

1. ping h1 to h2 in Mininet

mininet> h1 ping h2

PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.

64 bytes from 10.1.1.2: icmp\_seq=1 ttl=64 time=6.33 ms

64 bytes from 10.1.1.2: icmp\_seq=2 ttl=64 time=0.501 ms

64 bytes from 10.1.1.2: icmp\_seq=3 ttl=64 time=0.112 ms

1. In the Ryu console, we can see the output

Flow details: OFPFlowStats(byte\_count=18480,cookie=0,duration\_nsec=20000000,duration\_sec=206,flags=0,hard\_timeout=0,idle\_timeout=0,instructions=[OFPInstructionActions(actions=[OFPActionOutput(len=16,max\_len=65509,port=1,type=0)],len=24,type=4)],length=104,match=OFPMatch(oxm\_fields={'eth\_src': '00:00:00:00:00:02', 'eth\_dst': '00:00:00:00:00:01', 'in\_port': 2}),packet\_count=192,priority=1,table\_id=0)

byte\_count: 18480

packet\_count: 192

**Aggregate Flow Stats**

Measure Total number flows in regular interval(10second) and print it.

**Logic**

* Use Ryu Thread(HUB) mechanism,
* In the Thread, Send OpenFlow Aggregate Flow Statistics Request message at regular interval
* Aggregate Flow Statistics Response will be received as event, and display the bytes & packets counter values.

Coding same as flow statistics example for thread stuff. But we send Aggregate flow stats request.

def monitor(self):

self.logger.info("start flow monitoring thread")

while True:

hub.sleep(10)

for datapath in self.datapaths.values():

ofp = datapath.ofproto

ofp\_parser = datapath.ofproto\_parser

cookie = cookie\_mask = 0

match = ofp\_parser.OFPMatch()

req = ofp\_parser.OFPAggregateStatsRequest(datapath, 0,

ofp.OFPTT\_ALL,

ofp.OFPP\_ANY,

ofp.OFPG\_ANY,

cookie, cookie\_mask,

match)

datapath.send\_msg(req)

Aggregate Flow Stats Response handling routine.

@set\_ev\_cls([ofp\_event.EventOFPAggregateStatsReply,

], MAIN\_DISPATCHER)

def stats\_reply\_handler(self, ev):

result = ev.msg.body

self.logger.info('AggregateFlowStats %s', result)

self.logger.info('FlowCount %d', result.flow\_count)

**demo**

1. start the Mininet single topology

sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --switch=ovsk,protocols=OpenFlow13 --topo=single,4

1. run our application.

ryu-manager agg\_flow\_stats.py

1. ping h1 to h2 in Mininet

mininet> pingall

\*\*\* Ping: testing ping reachability

h1 -> h2 h3 h4

h2 -> h1 h3 h4

h3 -> h1 h2 h4

h4 -> h1 h2 h3

\*\*\* Results: 0% dropped (12/12 received)

mininet>

1. In the Ryu console, we can see the output

Result:

AggregateFlowStats OFPAggregateStats(packet\_count=296,byte\_count=24752,flow\_count=13)

FlowCount 13

**4.13 Topology Discovery**

**RYU** has Topology discovery mechanism. RYU uses LLDP (Link Layer Discovery Protocol) to discover the switches and links. Topology discovery feature will be enabled with **"--observe-links"** option.

Topology discovery feature is used in many applications such as identify the shortest path, avoid loops, multipath with load balancing etc.

**4.14 How it works**

**Switch & Link Discovery:**

* RYU Controller generates LLDP Packet(Datapath id, port number) from each switch and each port number.
* LLDP Packet reaches the other end of the link (its another switch port).
* This packet will send it to the controller
* Controller decodes the packet and learn the Switch, and Link information from the LLDP Packet (sender switch details (Datapath ID, Port number) and received switch details ((Datapath ID, Port number)) )

**Host discovery:**

Host discovery is based on the incoming packet. Usually, the host machines are keep sending some broadcast/multicast packets (some services will trigger this). RYU Controller uses these packets to discover the hosts.

**4.15 Coding**

**Logic:**

Topology discovery process may take few seconds to complete it . So we use RYU thread and print the topology information after 10seconds. We assume within 10 seconds, the topology discovery process will be completed.

1. Include the header

from ryu.topology.api import get\_switch, get\_link, get\_host

1. Initiate a thread

hub.spawn(self.myfunction)

1. Printing the topology discovery information

def myfunction(self):

self.logger.info("started new thread")

hub.sleep(10)

switch\_list = get\_switch(self.topology\_api\_app, None)

self.switches = [switch.dp.id for switch in switch\_list]

links\_list = get\_link(self.topology\_api\_app, None)

self.links = [(link.src.dpid, link.dst.dpid, {'port': link.src.port\_no}) for link in links\_list]

host\_list = get\_host(self.topology\_api\_app, None)

self.hosts = [(host.mac, host.port.dpid, {'port': host.port.port\_no}) for host in host\_list]

self.logger.info("\*\*\*\*\*\*\*\*\*Topology Information\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.logger.info("Switches %s", self.switches)

self.logger.info("Links %s", self.links)

self.logger.info("Hosts %s", self.hosts)

we use three APIs **get\_switch, get\_link, get\_host** . These api provides the discovered switch,link and host details in the list of objects.

This below line, we are processing the switch list, and get only switch dp id.

self.switches = [switch.dp.id for switch in switch\_list]

Similar to other link and host.

**4.16 Demo**

1. Run Mininet Linear Topology

sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --switch=ovsk,protocols=OpenFlow13 --topo=linear,4

1. Start the Wireshark Capture with loopback interface
2. Run Ryu controller with "--observe-links" options

ryu-manager --observe-links topology\_discovery.py

1. Topology information will be printed in the RYU Console.

\*\*\*\*\*\*\*\*\*Topology Information\*\*\*\*\*\*\*\*\*\*\*\*\*

Switches [1, 2, 3, 4]

Links [(2, 3, {'port': 3}), (3, 2, {'port': 2}), (3, 4, {'port': 3}), (2, 1, {'port': 2}), (4, 3, {'port': 2}), (1, 2, {'port': 2})]

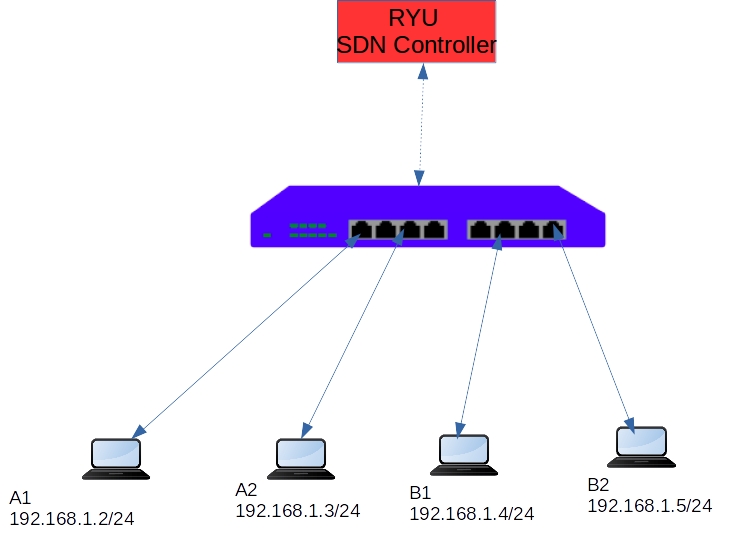
Hosts [('00:00:00:00:00:03', 3, {'port': 1}), ('00:00:00:00:00:02', 2, {'port': 1}), ('00:00:00:00:00:01', 1, {'port': 1}), ('00:00:00:00:00:04', 4, {'port': 1})]

1. Analyze the Wireshark traces(LLDP Packets)

**4.17 Multicontrollers**

**Problem Statement**

1. Controller failure in SDN Network is major failure affects the entire network topology
2. Various reasons for controller failure - security threats, hardware /software issues, manual errors etc.
3. Using Single Controller in SDN Network is high risk, high possibility for failure, not scalable and fault tolerant.



**Demo - Single Controller**

1. Running Mininet topology with Linear topology.

sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --switch=ovsk,protocols=OpenFlow13 --topo=single,4

1. Running a Ryu controller with simple switch application(idle timeout/hard timeout to 30seconds)

ryu-manager flow\_timeout.py

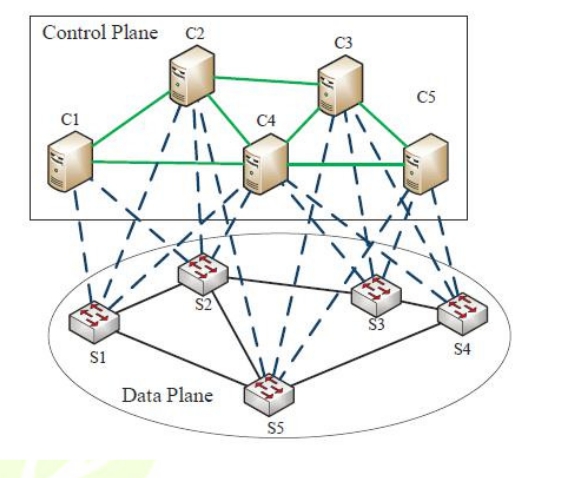
1. Ping all.
2. Stop the controller.
3. Check the flows

Observation: Once the controller is stopped, topology works till the flows are timeout(hard) after that, the entire topology Datapath is not working. hosts cannot reach other hosts.

It’s a single point failure.

**4.18 Cluster of SDN Controllers:**

OpenFlow specification supports the Multiple controllers environment. User can configure the multiple controllers in the switch.



**4.19 Controller Roles**

**Equal Role**

It means, all the controllers configured in the switch have full control to update/modify the flows.

Switch must send the PACKET IN Message to all the Controllers. Also switch process the PACKET OUT, FLOW UPDATES etc. from all the controllers.

**Master Role**

It means, MASTER controller will be responsible for managing the switch OpenFlow dataplane. Switch will send the control messages to only MASTER controller.

**Slave Role**

Slave plays backup role for MASTER. it also receives the HELLO and KEEPALIVE Message. But it cannot send and receive the Control message.

**ROLE\_REQUEST and RESPONSE:**

When the controller comes up, it will send the ROLE REQUEST with ROLE MASTER, other controller should send a role as SLAVE. Switch will communicate with MASTER.

**4.20 Demo - Equal role**

1. controller1

ryu-manager --ofp-tcp-listen-port 6653 flow\_timeout.py

1. controller2

ryu-manager --ofp-tcp-listen-port 6654 flow\_timeout.py

1. Mininet topology

sudo python topology.py

1. Start the Wireshark capture for traces
2. pingall

**Note** : we can see the packet in comes from both controller

**4.21 Demo- Master/Slave Role**

Here one controller going to act as Master Role, another controller going to act as SLAVE or BACKUP role.

1. Start the Wireshark capture for traces in loopback interface
2. Mininet topology

sudo python topology.py

1. start the master controller

ryu-manager --ofp-tcp-listen-port 6653 l3switch\_master.py

1. start the backup controller

ryu-manager --ofp-tcp-listen-port 6654 l3switch\_slave.py

1. Check the traces in controller terminals. you can see only MASTER terminal you can see packet in traces.
2. pingall

only MASTER Controller manages the switches.

1. Analyze the ROLE Messages in the Wireshark traces.

**Data Synchronization**

In Multicontroller environment(specifically Master/slave), data(intelligence built by the controller. For example mac\_to\_port structure) needs to be synchronized across the controller.

There are many mechanisms available such as DB, in Memory DB, Message Brokers etc.

**Further experiments**

* Use in memory DB or Message Broker for Data Synchronization
* Implement Master Election Algorithm/ Failure Detection /Auto failover.
* Distributed Master/Slaves

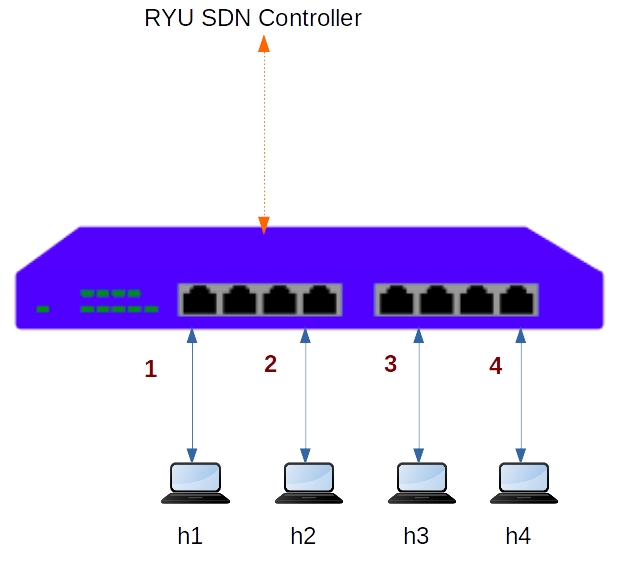
**Chapter 5: Simulation Steps and Results**

**5.1 Traffic Tests - Part1**

Usually we need to perform various traffic tests between the hosts to test our network topology & SDN application performance.

Some of the test includes:

1. TCP Tests between two hosts
2. UDP Tests
3. UDP Tests with different Packet size (64 bytes, 1024 bytes etc.), different packet rate (50 pkts/s, 100 pkts/s etc.)
4. Variable Traffic rate (10Pkts/s for first 60s, then 100Pkts/s for rest of the test)
5. Multiple/Parallel Streams/Sessions
6. VoIP Traffic Test
7. Video Streaming mp4 video file will be used as video source. and will generate the video stream using VLC and will be received by client using VLC Video Player.



**5.2 TCP Tests**

IPERF is widely accepted traffic generation tool to perform TCP Tests.

* TCP tests generally used for measure the bandwidth.
* Latency, Jitter, Packet loss cannot be measured using TCP Tests.

For all our traffic tests, we will use Simple Topology and l4 switch application.

1. Create the topology

sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4

1. Run the RYU Controller l4 switch application.

ryu-manager l4\_switch.py

1. In Mininet shell, do ping h1 to h2.

mininet>h1 ping -c 5 h2

4.check the flows

sudo ovs-ofctl -O OpenFlow13 dump-flows s1

1. understand the Host & Switch Port numbers.

mininet>links

mininet>ports

1. To check the switch statistics

sudo ovs-ofctl -O OpenFlow13 dump-ports s1

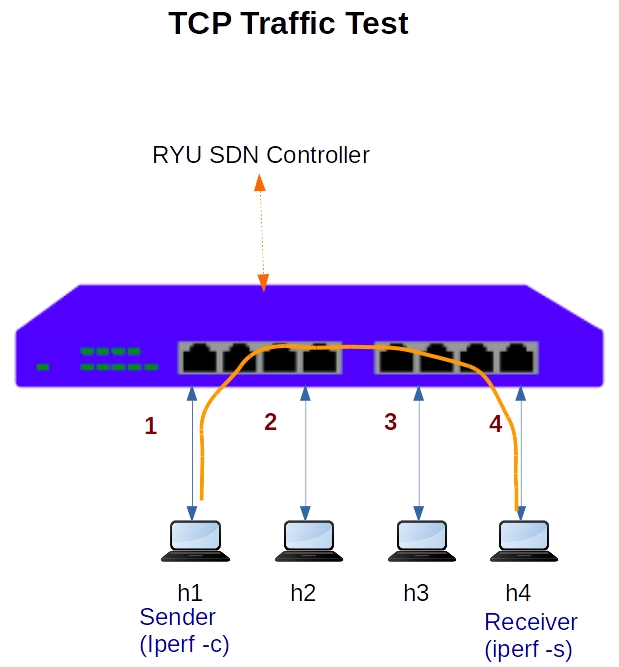
Now all set to run our traffic tests.

**A. Traffic test from h1 to h4 :**

Objective: Generate TCP Traffic from h1 to h4. (Measure bandwidth from h1 to h4)

h1 is sender.

h4 is receiver.



1. Start IPERF Server in h4

h4 iperf -s &

1. Start the IPERF Client in h1 and connecting to h4

h1 iperf -c h4

1. Analyze the results by flows

sudo ovs-ofctl -O OpenFlow13 dump-flows s1

we observe traffic in both directions.

h1 to h4 traffic is very high (in Gbps). This is data traffic. h4 to h1 traffic is very less. This is TCP Acknowledge traffic.

1. Analyze the results by ports.

sudo ovs-ofctl -O OpenFlow13 dump-ports s1

h1 --------port1, port4--------h4

**Forward traffic:**

* h1 transmits. port1 receives.
* port4 transmits, h4 receives.

**Acknowledge:**

* h4 transmits. port4 receives.
* port1 transmits. h1 receives.

**B. Bidirectional Traffic test h1 to h4(sequentially).**

1. Start IPERF Server in h4

h4 iperf -s &

1. Start the IPERF Client in h1 and connecting to h4

h1 iperf -c h4 -r

Its sequential test , once h1 to h4 traffic test is completed it will start h4 to h1 traffic test.

**C. Bidirectional Traffic test h1 to h4(parallel).**

1. Start IPERF Server in h4

h4 iperf -s &

1. Start the IPERF Client in h1 and connecting to h4

h1 iperf -c h4 -d

Its parallel test , both direction h1 to h4 as well h4 to h1 traffic tests started.

**D. Traffic test from h1 to h4 with Multiple Sessions.**

1. Start IPERF Server in h4

h4 iperf -s &

1. Start the IPERF Client in h1 and connecting to h4

h1 iperf -c h4 -P 5

**5.3 UDP Tests with IPERF**

UDP Tests are quite flexible in the nature of playing around packet sizes (64 bytes) and number of packets(10 Packets/s) you want to send. It means, user can control the bandwidth of UDP Traffic.

* Bandwidth, Latency, Jitter, Packet loss can be measured using UDP Tests.

IPERF doesn't provide much flexibility in UDP Tests.

1. Start IPERF UDP Server in h4

h4 iperf -u -s &

1. Start the IPERF Client in h1 and connecting to h4 and generate bandwidth of 10Mbsp

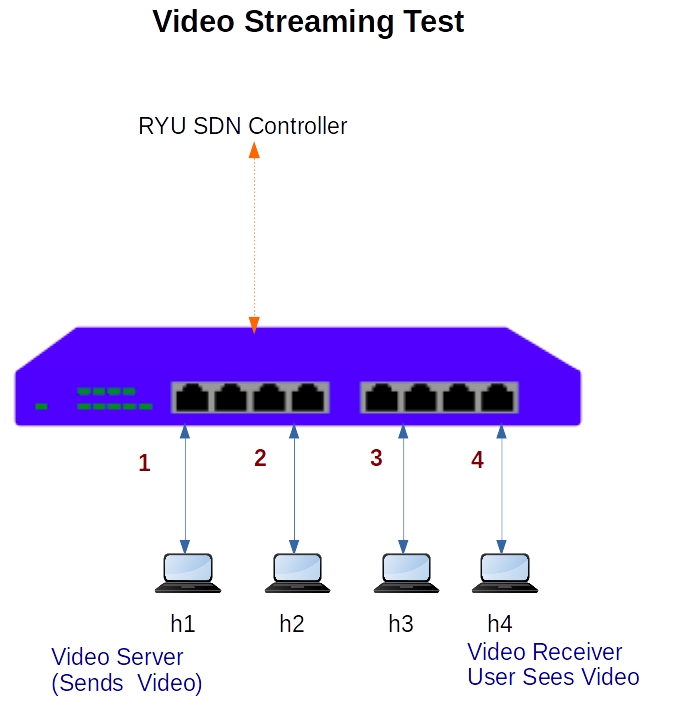
h1 iperf -u -c h4 -b 10m

**5.4 Video Stream Traffic Testing**

we use VLC media player for Video Streaming testing.

installation procedure

sudo apt-get install vlc vlc-nox



**Testing**

1. Create the topology

sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4

1. Run the RYU Controller l4 switch application.

ryu-manager l4\_switch.py

1. Download the sample video from internet (or use your own video file)
2. h4 is a Video receiver. Start the video receiver in h4 xterm shell

mininet> xterm h4

su - user-name

vlc rtp://@10.0.0.4:9001

1. h1 is video sender. Start the video streaming sender in h1 xterm shell

mininet> xterm h1

su - user-name

cvlc /home/sureh/cat3.mp4 --sout '#rtp{proto=udp,mux=ts,dst=10.0.0.4,port=9001}'

**5.5 VOIP Tests**

VOIP Traffic is UDP Stream. This can be simulated thru IPERF UDP Tests.

**Objective 1.** Test the single VoIP call 64Kbps 2. Test parallel VoIP calls

**Testing**

1. Create the topology

sudo mn --controller=remote,ip=127.0.0.1 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4

1. Run the RYU Controller l4 switch application.

ryu-manager l4\_switch.py

**A. Single 64Kbps VOICE CALL Test**

start the 64Kbps VOIP Traffic Test between h1 to h4 for 60 seconds, and vice versa.

1) Run the IPERF UDP server in h4

In Mininet CLI

mininet>h4 iperf --server --udp --len 300 --tos 184 -fk --interval 5 &

Here we are starting udp server, and setting the TOS field to 184

2) Run the IPERF UDP Client in h1

In Mininet CLI

h1 iperf -c 10.1.1.4 --udp --len 300 --bandwidth 67000 --dualtest --tradeoff --tos 184 -fk --interval 5 --time 60 --listenport 5002

**B. Multiple Parallel calls VOIP calls test**

1) Run the IPERF UDP server in h4

mininet>h4 iperf --server --udp --len 300 --tos 184 -fk --interval 5 --parallel 4

2) Run the IPERF UDP Client in h1

mininet>iperf -c 10.1.1.4 --udp --len 300 --bandwidth 67000 --dualtest --tradeoff --tos 184 -fk --interval 5 --time 60 --listenport 5002 --parallel 4

**5.6 Installations**

**To use Ryu with Mininet, you can follow these steps:**

* Install Ryu and Mininet on your system. You can follow the instructions on their respective websites to do this.
* Create a topology using Mininet. This can be done by writing a Python script that defines the topology, such as the number and types of switches and hosts.
* Start Mininet with the topology you created. This will create a virtual network that you can use to test your SDN implementation.
* Write a Ryu application that implements the desired SDN functionality, such as routing or load balancing. This can be done using the Ryu API, which provides a set of functions for controlling the network.
* Start the Ryu controller and connect it to the virtual network created by Mininet. This will allow the Ryu application to control the network traffic.
* Test your SDN implementation by sending traffic between the hosts in the Mininet network and observing the behavior of the Ryu controller.

Overall, using Ryu with Mininet can be a powerful tool for testing and developing SDN applications in a virtual environment.

**SDN Test bed Installation**

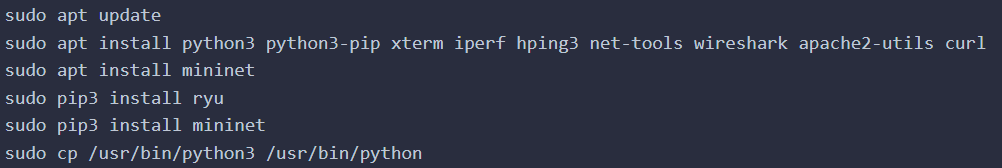
Setup the SDN Test environment to practice OpenFlow use cases with RYU SDN Controller.

Prebuilt VM Image (OVA Format), Size 3.3GB

**username : test** **password : test**

This can be imported in **Oracle VirtualBox**. Just double click the OVA file it will be imported automatically.

**To install Ryu and Mininet at ubuntu use this command:-**

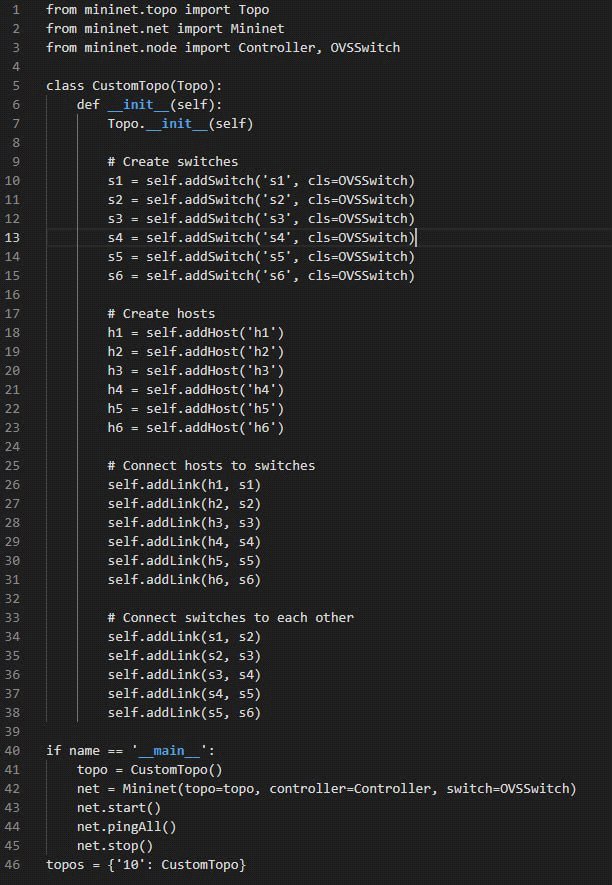


**5.7 python code for Custom Topology :**

Mininet exposes the python API. We can create a custom topologies using the python API with few lines of code.

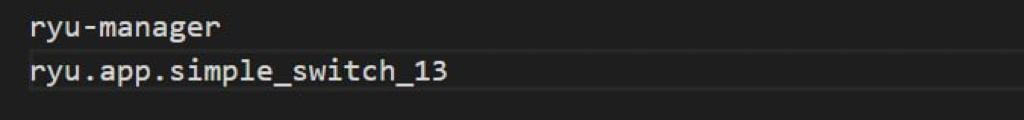
**How to write Custom Topology in Mininet**

Steps are below (for 6 host & 6 switch & 1 controller)



**How to run**

start the RYU SDN Controller



Run the Mininet topology file

Copy



Perform your tests/operation etc.

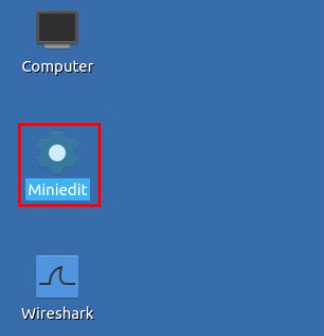
**5.8 Mini-edit**

Mininet is a system that supports the creation of lightweight logical nodes that can be connected into networks. These nodes are sometimes called containers, or, more accurately, network namespaces. Virtual-machine technology is not used

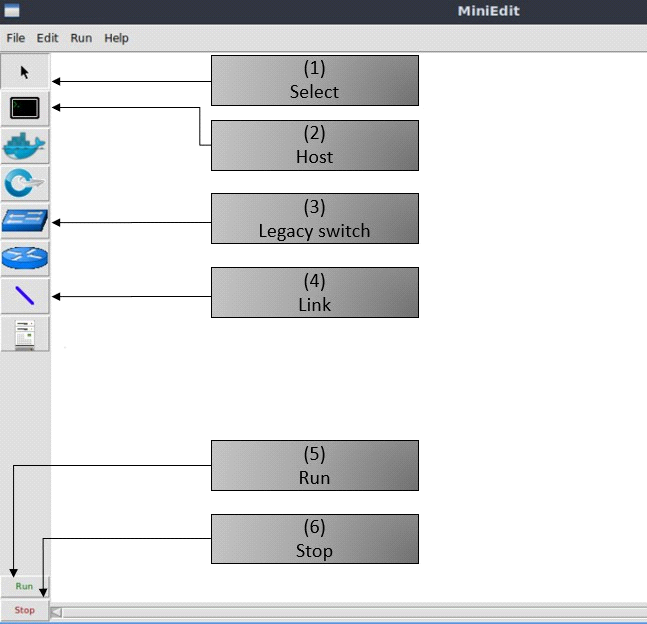
* **Build the network topology**

**Step 1.** A shortcut to MiniEdit is located on the machine’s Desktop. Start MiniEdit by

clicking on MiniEdit’s shortcut. When prompted for a password, type password.



MiniEdit will start, as illustrated below.

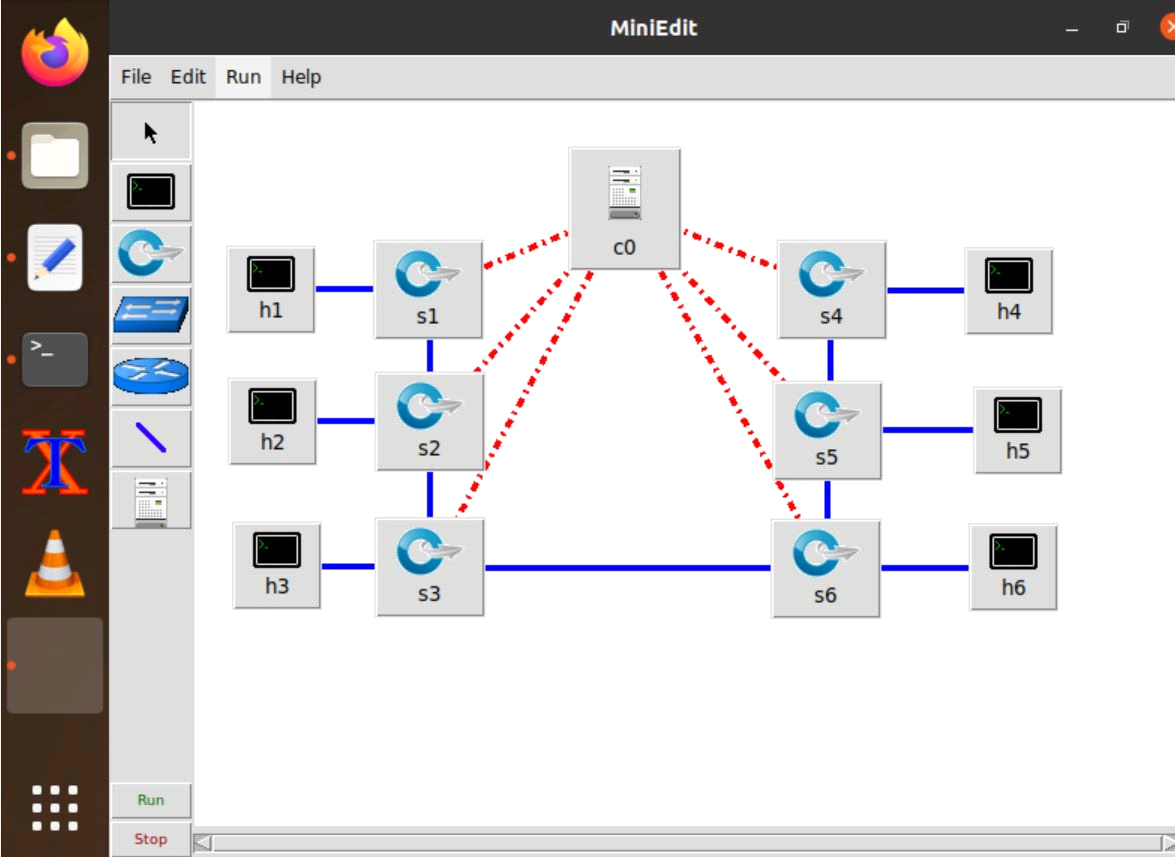


MiniEdit Graphical User Interface (GUI).

**The main buttons in this lab are:**

* **Select:** allows selection/movement of the devices. Pressing *Del* on the keyboard after selecting the device removes it from the topology.
* **Host:** allows addition of a new host to the topology. After clicking this button, click anywhere in the blank canvas to insert a new host.
* **Legacy switch:** allows addition of a new legacy switch to the topology. After clicking this button, click anywhere in the blank canvas to insert the switch.
* **Link:** connects devices in the topology (mainly switches and hosts). After clicking this button, click on a device and drag to the second device to which the link is to be established.
* **Run:** starts the emulation. After designing and configuring the topology, click the run button.
* **Stop:** stops the emulation.

**Step 2.** To build the topology illustrated in Figure six hosts and six switch must be deployed. Deploy these devices in MiniEdit, as shown below.



**5.9 Implementation**

To implement video streaming over SDN, we used the following tools and technologies:

Mininet: A network emulator that allowed us to create a virtual network topology and test our SDN application.

VLC Media Player: A media player that we used to stream the video from the source host to the destination host.

Ryu SDN controller: The controller that we used to manage the network and enforce OpenFlow rules on the switches.

To run our SDN application, follow these steps:

1) Start the virtual network: Start the virtual machines and switches using a virtualization software like VirtualBox or VMware.

2) Configure the network topology: Define the network topology that the SDN application will control using the **Custom Topo** class in your code.

3) Start the SDN controller: Start the Ryu controller software by running the following command in a terminal window:

4) Deploy the SDN application: Deploy the SDN application on the controller by running the following command in another terminal window:

where **sdn\_application.py** is the filename of your SDN application code.

5) Test the SDN application: Test the SDN application by sending traffic through the network and verifying that it is being correctly controlled by the application. This can be done using network testing tools such as ping or through an SDN-specific testing tool like Mininet. For example, you can run the following command to test connectivity between all hosts:

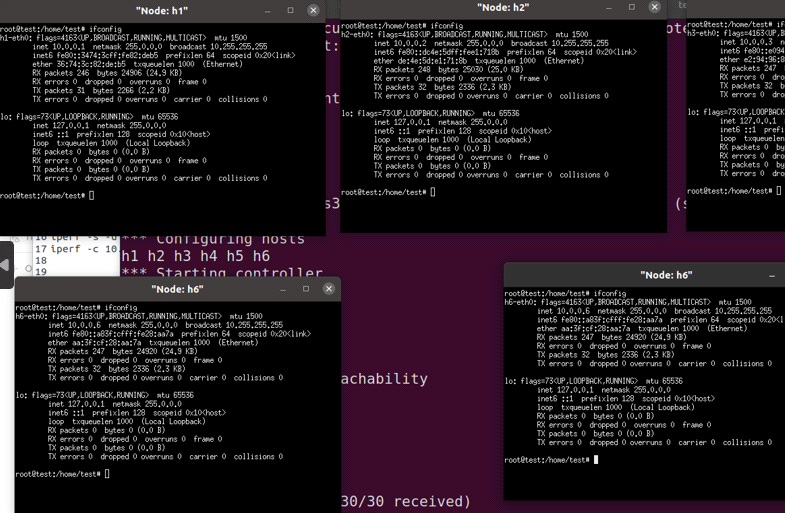
6) Then in the Mininet CLI, run the following command to test connectivity between **h1** and **h6**:

If the SDN application is correctly controlling the network, the ping should be successful.

Make sure to Install and configure VLC media player on all hosts that will be streaming video, audio, or data. Configure the VLC player to stream content over the network by setting the appropriate streaming options.

**Streaming:**

In this project we are going to use **h1** and **h6** to stream the video,**h2** and **h6** for the data, and lastly **h3** and **h6** for the audio.

1) We open the 6 nodes to check the connection using **xterm** command:

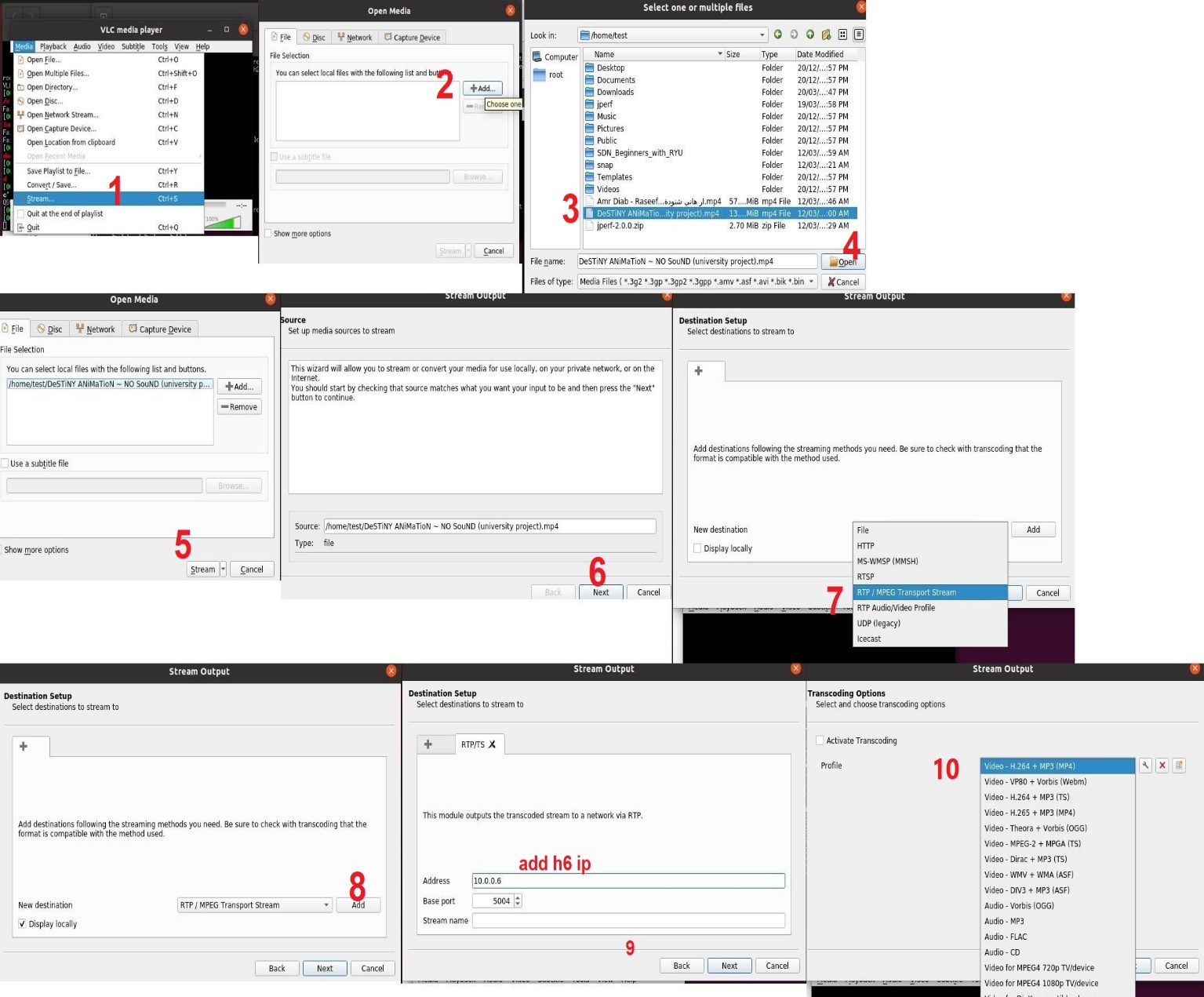
2) We use the following command to use **VLC** on host 1 and 6:



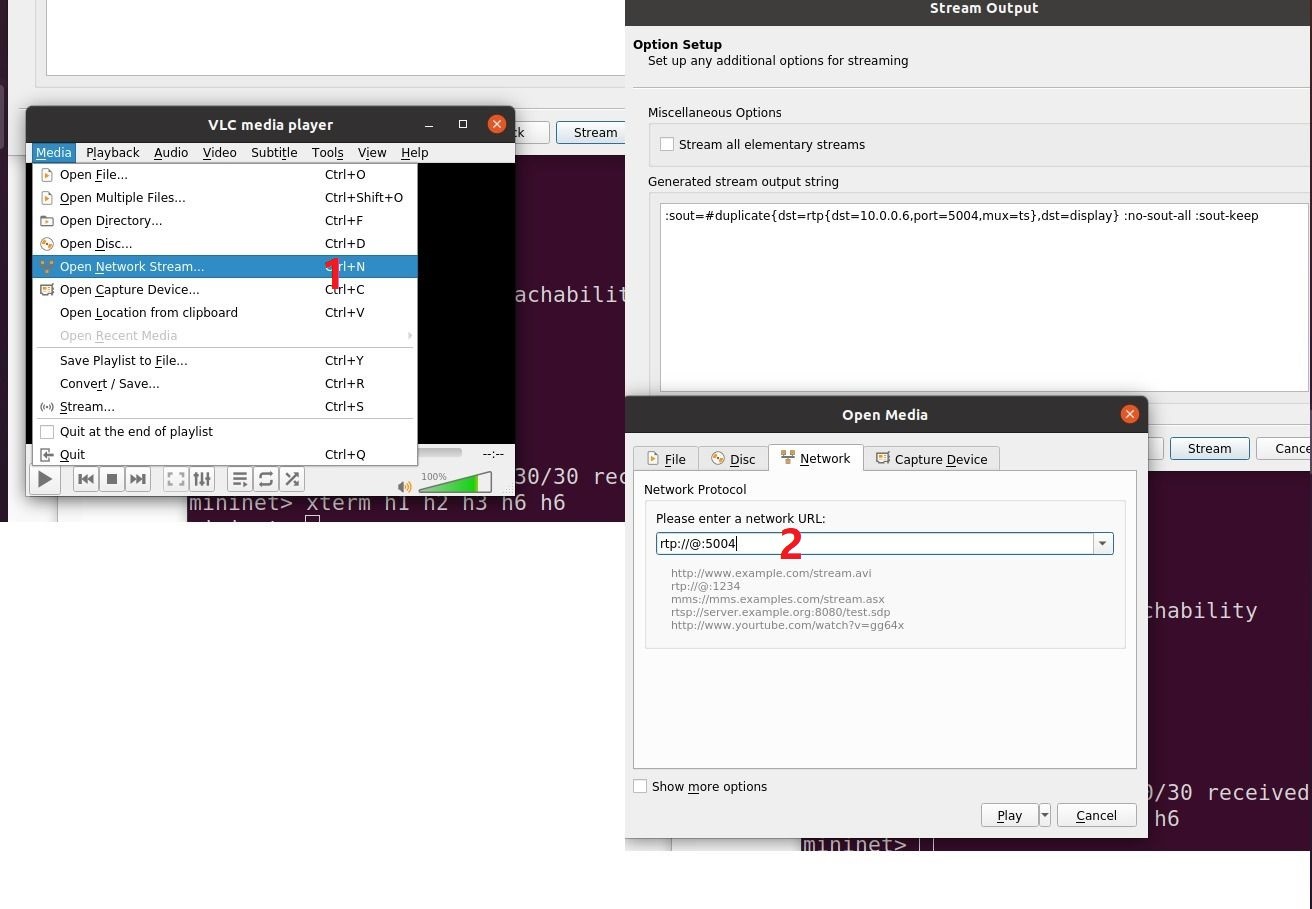
3) On host 1,from the VLC media player we click on the menu----->stream----->file---->add the video we want to stream-----click stream.

In the destination setup, check the display locally box and add RTP/MPEG Transport Stream.

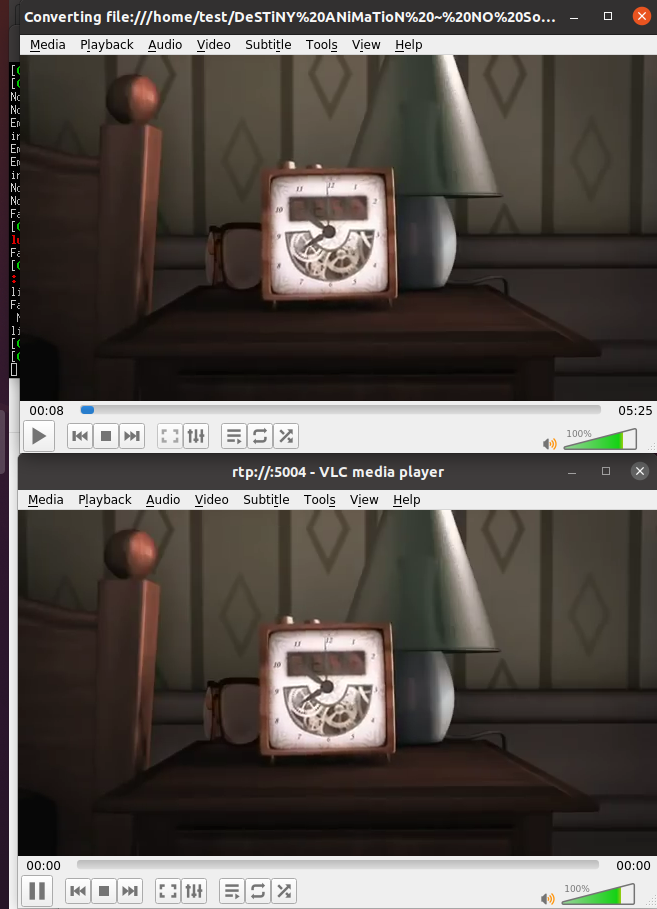
After clicking next , in the Address bar type the address you want to use for the stream and the port in which the host is connected on the switch.



4) On host 6,from the VLC media player we click on the menu---->Open Network Stream----->in the URL type “rtp://@:(port number)” make sure the port number is the same one used in host 1.



After making these steps the video will start streaming and you can view it h1&h6



5) On host 2 we streaming data to host 6 by using this command:



6) On host 3 we streaming voice to host 6 by using this command:



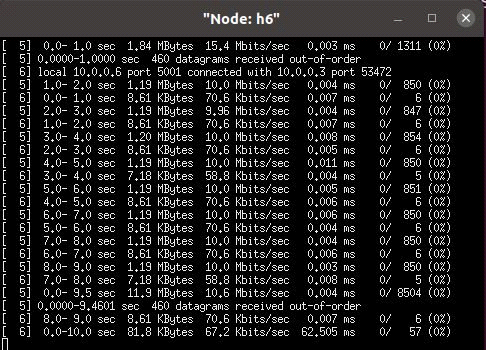
7) Then make host 6 be server to receive this traffic by this command :



**Monitoring Traffic:**

Now we want to analyze our stream and make sure there is no delay and it’s working great.First we need to gather data using the **iperf**

It will look like this:



**5.10 Wireshark**

**Network Analysis and Sniffing**

Process of capturing, decoding, and analyzing network traffic.

* Why is the network slow.
* What is the network traffic pattern.
* How is the traffic being shared between nodes.

Also known as traffic analysis, protocol analysis, sniffing, packet analysis, eavesdropping, etc.

A combination of hardware and software tools which can detect, decode, and manipulate traffic on the network.

* Passive monitoring (detection) – Difficult to detect.
* Active (attack).

**What is Wireshark?**

Wireshark is a network packet/ protocol analyzer.

A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible.

Wireshark is perhaps one of the best open-source packet analyzers available today for UNIX and Windows.

**Some intended purposes**

* Network administrators use it to troubleshoot network problems.
* Network security engineers use it to examine security problems.
* Developers use it to debug protocol implementations.
* People use it to learn network protocol internals.
* Wireshark isn’t an intrusion detection system.
* Wireshark will not manipulate things on the network, it will only “measure” things from it.

**Features:**

* Available for UNIX and Windows.
* Capture live packet data from a network interface.
* Display packets with very detailed protocol information.
* Open and Save packet data from and to a lot of other capture programs.
* Filter packets on many criteria.
* Colorize packet display based on filters.
* Create various statistics.
* Wireshark can capture traffic from many different network media types and despite its name including wireless LAN as well. Which media types are supported, depends on many things like the operating system you are using.
* Wireshark is an open source software project, and is released under the GNU. You can freely use Wireshark on any number of computers you like, without worrying about license keys or fees or such. In addition, all source code is freely available under the GPL. Because of that, it is very easy for people to add new protocols to Wireshark, either as plugins, or built into the source, and they often do.

**Disadvantages:**

* Wireshark isn’t an intrusion detection system. It will not warn you when someone does strange things on your network that he/ she isn’t allowed to do.
* Wireshark will not manipulate things on the network, it will only “measure” things from it. Wireshark doesn’t send packets on. The network or do other active things.

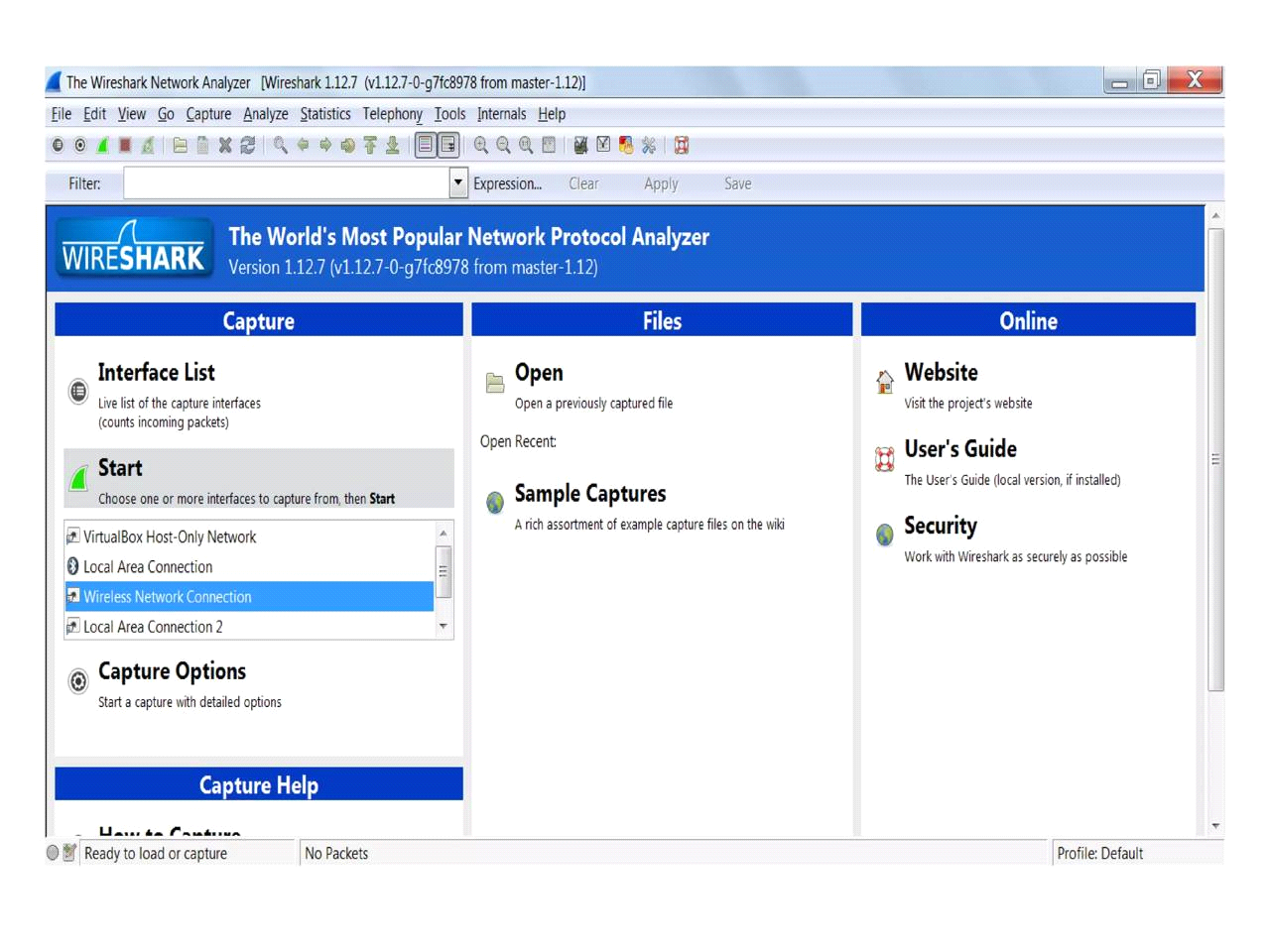
**Wireshark Fundamentals**

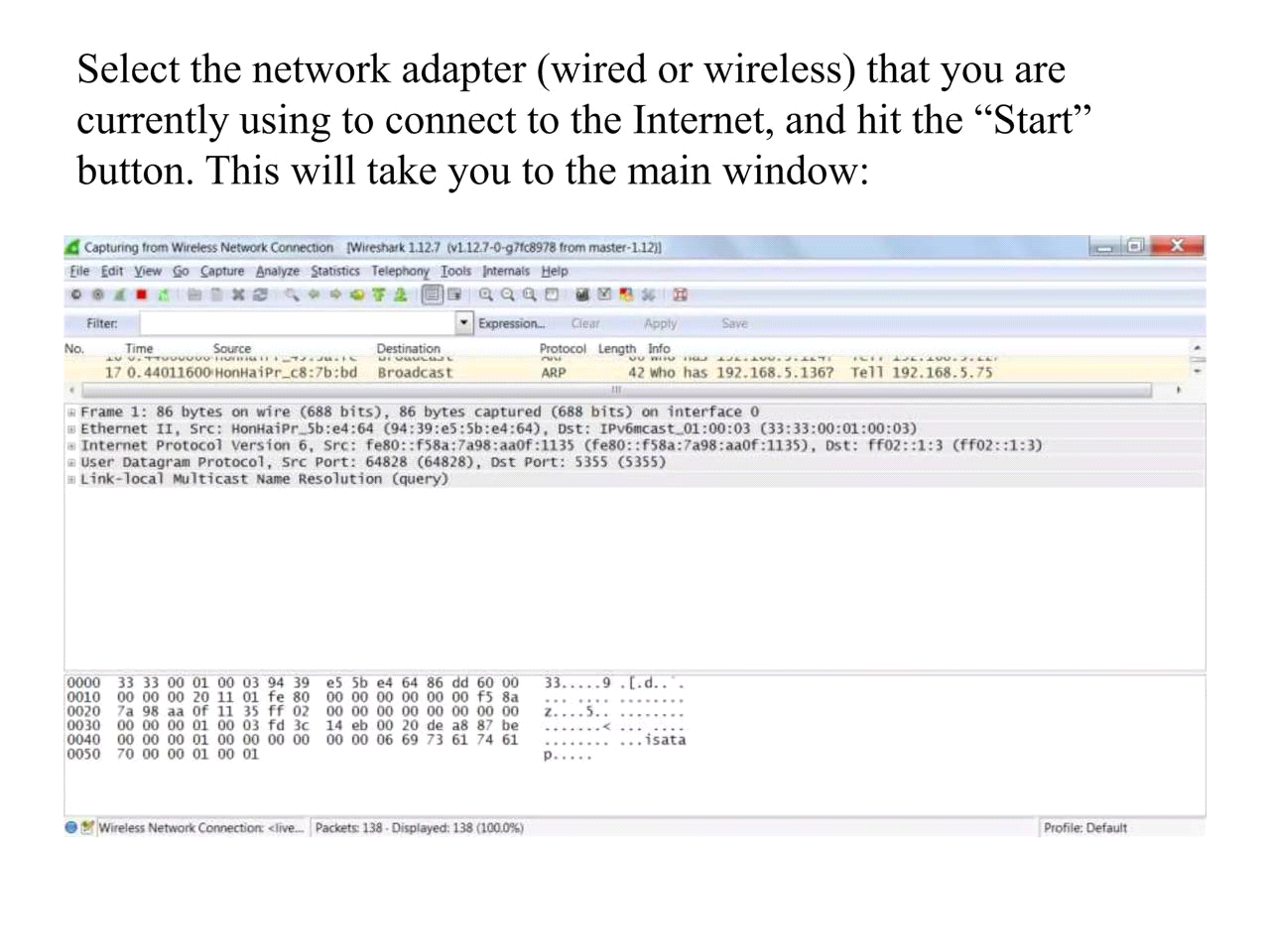
Your first packet capture

* First, there is always something wrong on the network.
* Secondly, there doesn’t need to be something wrong in order for you to perform packet analysis.
* More broadly, in order to find anomalies in daily network activity, you must know what normal daily network activity looks like.
* When your network is running smoothly, you can set your baseline so that you will know what its traffic looks like in a normal state.
* So let’s capture some packets.
* Open Wireshark
* From the main drop-down menu, select Capture and then Interfaces
* Choose the interface you wish to use and click Start, or simply click the interface under the Interface List sections of the welcome page.
* Wait about a minute or so, and when you are ready to stop the capture and view your data, click the Stop button from the capture drop-down menu.

**Getting Started Wireshark**

Start Wireshark. Under the “Capture” header, select the “Interface List” option; or click on the “Interfaces” button on the toolbar: This will bring up a list of network interfaces that Wireshark is able to capture packets from:

 Select the network adapter (wired or wireless) that you are currently using to connect to the Internet, and hit the “Start” button. This will take you to the main window:

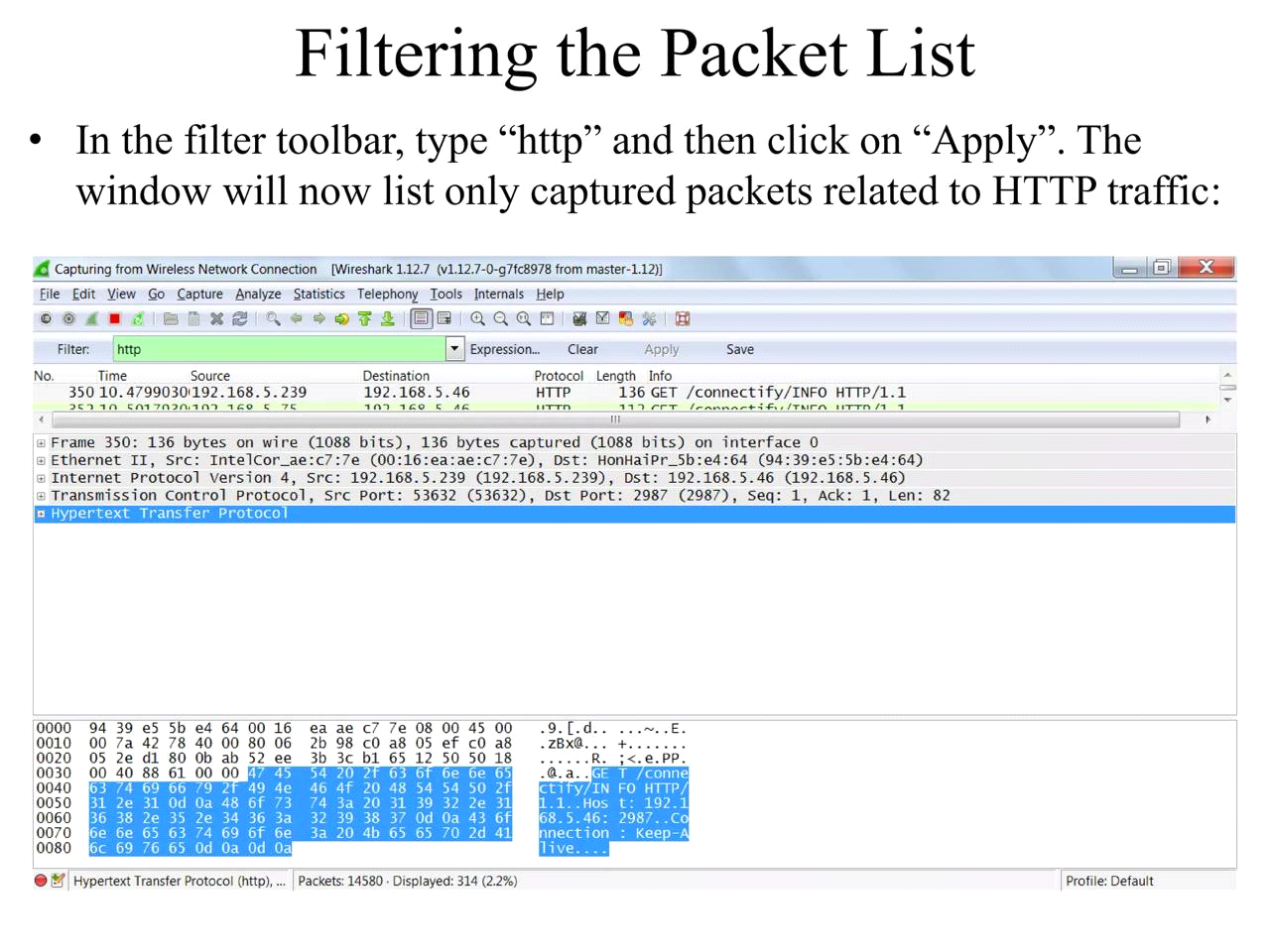


Wireshark is now capturing live network activity on your network interfaces. Notice that the list of packets is color-coded to highlight different types of network traffic.

Open your web browser and navigate to a few random web pages – observe that the network packets corresponding to your web browsing activity are captured and show up in Wireshark as well.

**Filtering the Packet List**

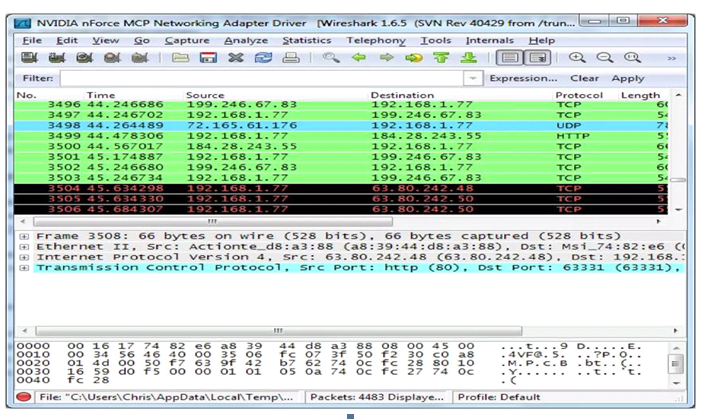
In the filter toolbar, type “http” and then click on “Apply”. The window will now list only captured packets related to HTTP traffic:



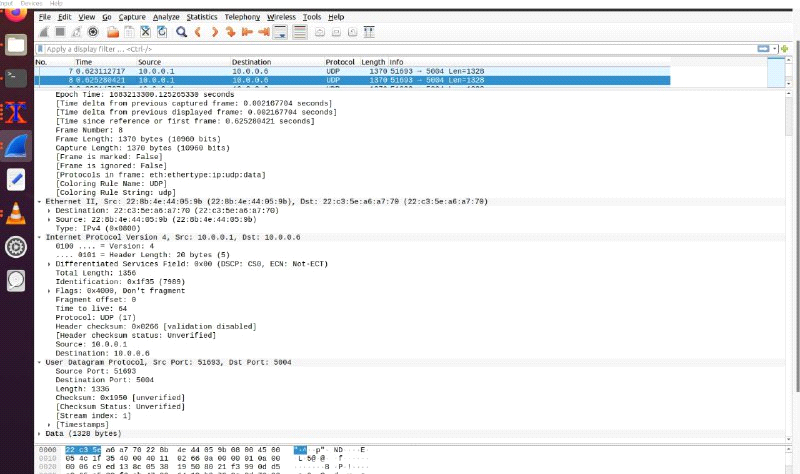
**Colour Coding**

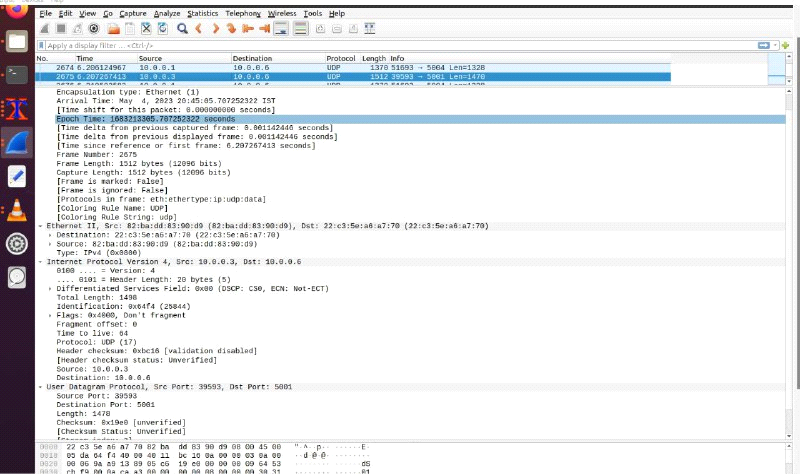
You will probably see packets highlighted in green, blue, and black. Wireshark uses colours to help you identify the types of traffic at a glance. By default, green is TCP traffic, and black identifies TCP packets with problems.

For Example, they could have been delivered out-of-order.



***Examples***





**Conclusion**:

In conclusion, this graduation project book has provided a comprehensive investigation into the integration of video streaming and Software-Defined Networking (SDN). The research conducted aimed to explore the potential of SDN to enhance video streaming performance, addressing the increasing demand for high-quality streaming experiences.

Throughout the chapters, we have examined the foundational concepts of SDN, video streaming protocols, and the OpenFlow protocol's role in enabling dynamic network control. The implementation of a video streaming system using Mininet, MiniEdit, Wireshark, and VLC Media Player allowed us to simulate and evaluate the system's performance under various scenarios.

The results obtained from the experiments and simulations showcased the advantages of SDN-based video streaming. The integration of SDN principles led to improvements in Quality of Service (QoS), reduced network congestion, and optimized resource allocation. These enhancements contribute to a superior streaming experience for end-users, characterized by reduced buffering, enhanced video quality, and efficient bandwidth utilization.

The findings of this research highlight the potential of SDN as a transformative technology in the realm of video streaming. By leveraging the programmability and centralization of SDN, network operators and service providers can adapt to changing network conditions, optimize resource allocation, and deliver a seamless streaming experience to users.

As the demand for video streaming continues to rise, further research and development in SDN-based video streaming are warranted. Future studies could explore additional optimizations, such as content-aware routing and adaptive bitrate streaming, to further enhance the streaming experience and accommodate diverse network conditions.

In conclusion, this graduation project book contributes to the body of knowledge surrounding video streaming over SDN infrastructure. It provides valuable insights for researchers, network engineers, and students interested in this domain, paving

the way for future advancements and innovations in the field. With the continued integration of SDN and video streaming, we can anticipate a future where high-quality, reliable streaming experiences are seamlessly delivered to users worldwide.

**References**:

https://www.opennetworking.org/sdn-definition/

https://en.wikipedia.org/wiki/Software-defined\_networking

https://www.sdxcentral.com/sdn-nfv-use-cases/

https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

https://github.com/mininet/mininet/tree/master/examples

http://mininet.org/api/annotated.html

https://kiranvemuri.info/dynamic-topology-changes-in-mininet-advanced-users-5c452f7f302a

https://stackoverflow.com/questions/43070043/mininet-wifi-custom-topologyhttps://stackoverflow.com/questions/43070043/mininet-wifi-custom-topology

http://mininet.org/walkthrough/

https://github.com/PrincetonUniversity/Coursera-SDN/tree/master/assignments/mininet-topology

https://www.pythonforbeginners.com/modules-in-python/how-to-use-simplehttpserver/

https://ryu.readthedocs.io/en/latest/writing\_ryu\_app.html