

Load balancing on any fat tree topology using SDN Controller

ABSTRACT

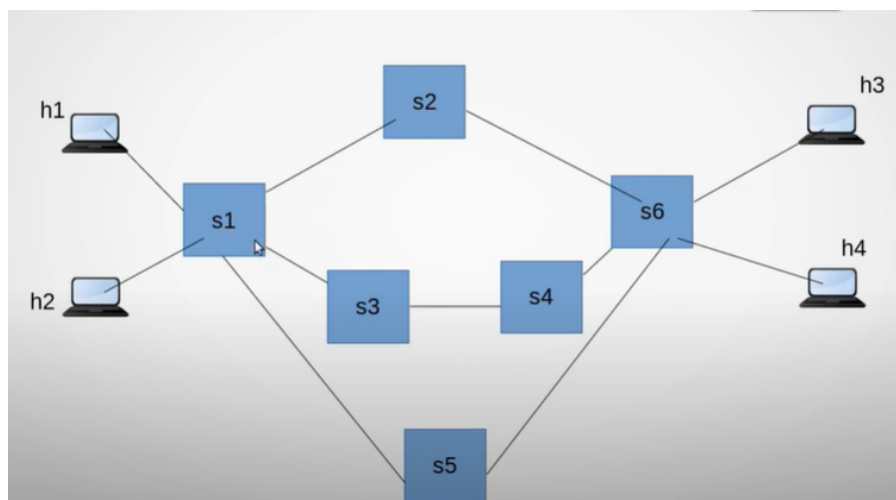
Software-Defined Networking (SDN) is an emerging technology in the field of networking. Nowadays clients are using large amount of data which needs to be handled by the network which creates a lot of traffic. For a single server it becomes difficult to handle all the load. The solution to this is to use multiple servers. The requests are sent to the load balancer. The client requests are then forwarded to the servers depending on the load balancing strategy used. Earlier hardware was used which turned out to be expensive and inefficient. Traditional load balancer are vendor locked, non-programmable because network administrators cannot create their own algorithms. On the other hand SDN load balancers are programmable and allow you to design and implement your own load balancing strategy. Additional use of SDN load balancer is it does not need dedicated hardware. In this project we are implementing and comparing our algorithm using RYU controller ,Mininet.

1. Concepts Used

Multipath Topology

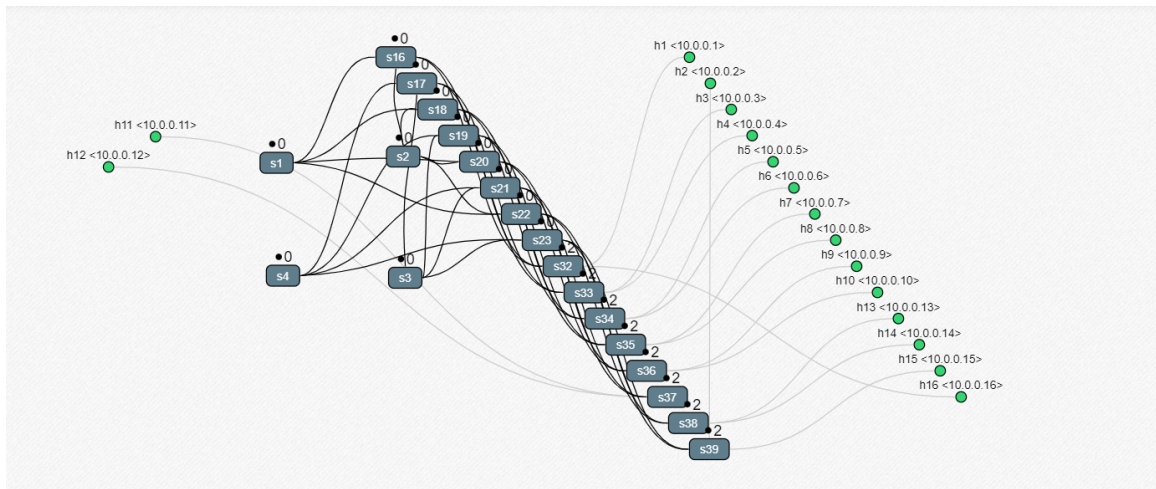
In computer networks, a multipath topology refers to a network configuration in which there are multiple paths between two nodes, allowing for redundant connections and increased fault tolerance. This can be achieved through the use of multiple physical links, such as cables or wireless connections, or through the use of virtual paths created through software or routing protocols. One common use of multipath topology is in load balancing, where traffic is distributed across multiple paths to prevent overloading of any one link. This can improve network performance and reduce the risk of congestion.

Another use of multipath topology is in providing redundancy and fault tolerance. In the event of a failure of one path, data can be transmitted over the remaining paths, maintaining connectivity and minimizing disruptions.



Fat Tree Topology

A Fat Tree Topology is a type of hierarchical network topology that is commonly used in data center networks. It is a type of Clos network topology, which is a type of non-blocking network topology that uses multiple layers of switches to connect a large number of devices. Fat Tree Topology consists of three layers: the core, the aggregate and the edge layers. The core layer is the central point of the network where all the data is routed and the aggregate layer is the layer where the core connects with the edge layer. The edge layer is the point of access to the network. The devices themselves are connected to the edge layer switches.



When we compare both the topologies, each has its own advantages and disadvantages,

In multipath topology, there are multiple paths between any two devices in the network. This means that if one path fails, there are other paths that can be used to maintain connectivity. Multipath topology provides redundancy and improves the reliability of the network. However, it can also increase the complexity of the network and make it more difficult to troubleshoot.

Fat tree topology is scalable and efficient, it also allows for good bandwidth and low latency. Additionally, it's easy to troubleshoot and maintain.

The choice of topology depends on the specific requirements of the network and the organization. For example, a network that needs a high degree of reliability and redundancy may choose to use multipath topology, while a network that needs to scale and manage high traffic may choose fat-tree topology. For fat-tree topology if

Number of pods = k

Number of servers in each pod = $(k/2)^2$

Number of Aggregation Switches = $(k/2)$

Number of Edge Switches = $(k/2)$

Number of Core Switches = $(k/2)^2$

Software Defined Networking

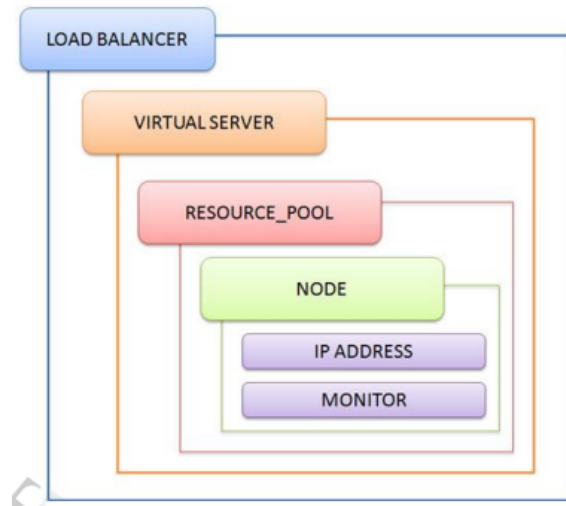
- ❖ Software-Defined Networking (SDN) is a network architecture paradigm that aims to make the network more flexible and programmable by decoupling the control plane (the part of the network that makes decisions about where traffic should be sent) from the data plane (the part of the network that forwards traffic).
- ❖ In traditional networks, the control plane and data plane are closely tied together, making it difficult to change the network's behavior or add new features. With SDN, the control plane is separated and implemented in software, running on a centralized controller. This controller can communicate with the data plane devices (such as switches and routers) to configure and manage the network.
- ❖ This separation allows for more flexibility and programmability in the network, as the controller can make decisions based on higher-level policies and abstractions, rather than low-level configurations. It also allows for centralized management and monitoring of the network, and enables the integration of new technologies and services more easily.
- ❖ Additionally, SDN allows for network virtualization, where multiple virtual networks can be created on top of a single physical infrastructure, providing isolation and security between different groups of users or applications.
- ❖ Overall, SDN aims to make the network more flexible, programmable, and easier to manage and automate, which can lead to more efficient and effective use of network resources.

Load Balancing

Load Balancing is a technique in which the workload on the resources of a node is shifted to respective resources on the other node in a network without disturbing the running task. A standard way to scale web applications is by using a hardware-based load balancer.

The load balancer assumes the IP address of the web application, so all communication with the web application hits the load balancer first. The load balancer is connected to one or more identical web servers in the back-end. Depending on the user session and the load on each web server, the load balancer forwards packets to different web servers for processing. The hardware-based load balancer is designed to handle high-level of load, so it can easily scale.

However, a hardware-based load balancer uses application specific hardware-based components, thus it is typically expensive. Because of cloud's commodity business model, a hardware-based load balancer is rarely occurred by cloud providers as a service. Instead, one has to use a software based load balancer running on a generic server



Goals of Load Balancing

- To improve the performance substantially.
- To have a backup plan in case the system fails even partially.
- To maintain the system stability.
- To accommodate future modification in the system.

Types of Load Balancing Algorithms

Static Algorithms:

Static algorithms divide the traffic equivalently between servers. By this approach the traffic on the servers will be disdained easily and consequently it will make the situation more imperfectly. This algorithm, which divides the traffic equally, is announced as round robin algorithm. However, there were lots of problems appeared in this algorithm. Therefore, weighted round robin was defined to improve the critical challenges associated with round robin. In this algorithm each servers have been assigned a weight and according to the highest weight they received more connections. In the situation that all the weights are equal, servers will receive balanced traffic.

Dynamic Algorithms:

Dynamic algorithms designated proper weights on servers and by searching in whole network a

lightest server preferred to balance the traffic. However, selecting an appropriate server needed real time communication with the networks, which will lead to extra traffic added on system. In comparison between these two algorithms, although round robin algorithms based on simple rule, more loads conceived on servers and thus imbalanced traffic discovered as a result.

There are several methods of load balancing, including:

- **Round-robin:** The load balancer distributes requests to servers in a round-robin fashion, sending the first request to the first server, the second request to the second server, and so on.
- **Least connections:** The load balancer sends requests to the server with the fewest current connections.
- **IP hash:** The load balancer uses a hash function to determine which server a request should be sent to, based on the IP address of the client.
- **Least response time:** It uses the information from the health-check probes to determine which server is currently the most responsive and routes traffic to that server.
- **Application-based:** It uses application-level information to make the routing decision.
- Load balancing can be performed at various layers of the OSI model, such as layer 4 (transport) or layer 7 (application). Load balancers can be implemented in hardware, software, or a combination of both.
- Load balancing is an essential component of a high-availability infrastructure, as it allows for the distribution of traffic across multiple servers, ensuring that no single point of failure can bring down the entire system. It also enables the addition of more servers to a system to handle an increase in traffic.

Load Balancer

A load balancer is an entity in the network that distributes incoming traffic among multiple outgoing links. Based on the load balancing algorithm implemented, the load balancer selects one or more outgoing links for the incoming traffic and forwards the traffic so that each outgoing link carries an equal amount of traffic during the operation time. The following figure shows an example of a load balancer in the network.

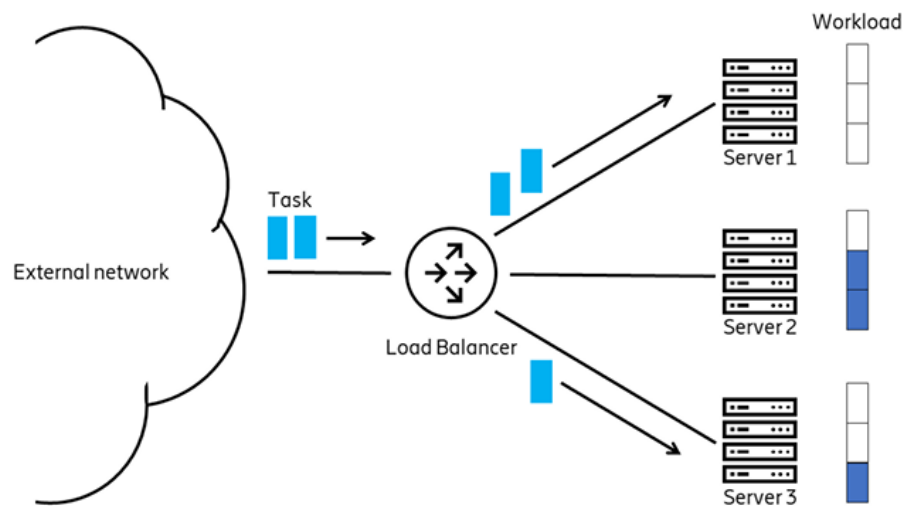


Figure: A simple example of load balancer. The load balancer distributes tasks according to the workload of each server behind the links.

Load balancers are commonly used in web applications in order to distribute the number of connections evenly across all web servers. This has the advantage of centralizing control over the network, providing a more comprehensive view of network status and increasing the chances of achieving optimal results. However, load balancers can also form single points of failure, where a crash could cause the entire network to go down. Additionally, load balancers can add complexity when scaling a network, as more processes are needed to upgrade hardware and software, or more load balancers are needed in the network.

Stateless load balancing

A stateless architecture or application is a type of Internet protocol where the state of the previous transactions is neither stored nor referenced in subsequent transactions.

Each request sent between the sender and receiver can be interpreted and does not need earlier requests for its execution.

This is a protocol where a client and server request and response are made in a current state.

Statefull load balancing

Stateful architecture or application describes a structure that allows users to store, record, and return to already established information and processes over the internet.

It entails transactions that are performed using past transactions as a reference point. In stateful applications, the current transaction can be affected by the previous ones.

Because of this, a stateful application uses the same server to process its requests. Stateful transactions can be likened to an ongoing discussion with statements made based on already established facts

RYU CONTROLLER

RYU is a Python-based OpenFlow controller that supports various protocols for managing network devices. It is an open-source software that can be used to build and manage Software-Defined Networks (SDN). RYU allows network administrators to programmatically control and configure the network using software, rather than manually configuring each device.

RYU is built on top of the Ryu Application Framework (RAF), which is a library of Python modules that provide the necessary functionality for building network controllers. The RAF provides an API for interacting with OpenFlow switches, as well as other features such as event handling, packet parsing, and network topology discovery.

RYU supports various protocols and features, such as:

- OpenFlow 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, and 1.10
- Open vSwitch Database Management Protocol (OVSDB)
- OpenFlow Secure Channel (OF-SCTP)
- OpenFlow Quality of Service (OFQoS)
- OpenFlow Group Table
- OpenFlow Meter Table

RYU also provides a simple and flexible architecture that allows developers to write custom applications to run on top of the controller. These applications can be used to implement a wide range of network management and control functions, such as load balancing, traffic monitoring, and security.

RYU is designed to be highly modular and extensible, with a clear separation between the core controller and the applications that run on top of it. This allows developers to easily add new features and functionality to the controller, and to create custom applications that can be integrated with existing network management systems.

RYU is widely used in the industry and academia for network research and testing, it's also used as a learning platform for students and researchers. It's compatible with various types of hardware and platforms, including Raspberry Pi and Mininet.

Overall, RYU is a powerful and flexible SDN controller that provides a wide range of features and can be extended to support custom applications. It's a well-documented and widely used software that can be used to create a variety of SDN solutions, and it's a good choice for researchers and developers who want to build and test new network management and control solutions.

Ryu Controller for Load Balancing

The purpose of the Ryu controller in load balancing is to distribute network traffic across multiple servers to improve the overall performance and availability of a service. The Ryu load balancer uses a variety of algorithms, such as round-robin and least-connection, to determine which server should handle a given incoming request.

By distributing the traffic across multiple servers, the Ryu load balancer helps to:

- Improve the scalability of the service by allowing it to handle a larger number of requests.
- Increase the availability of the service by ensuring that if one server goes down, traffic can still be directed to the remaining servers.
- Improve the performance of the service by ensuring that no single server is overwhelmed with too many requests.

Additionally, the Ryu load balancer supports virtual IPs, which allow multiple servers to share a single IP address and appear as a single entity to clients. This allows for easy and efficient management of the service, and also allows for easy failover in case a server goes down.

Shortest Path Routing

There are many different shortest path routing algorithms but we will be using dijkstra's algorithm because of its efficiency over the other algorithms. The time complexity of $O(E \log V)$, where E is the number of edges and V is the number of nodes. This is considered to be more efficient than other algorithms such as Bellman-Ford, which has a time complexity of $O(VE)$ or A* algorithm which is $O(E + V \log V)$ not only time complexity but also the space complexity of

$O(V)$, which is the space required to store the distance and previous vertex for each node in the graph is considered to be more efficient than other algorithms.

Dijkstras's Algorithm as Shortest Path Routing Algorithm

Dijkstra's algorithm can be used in a fat tree topology to find the shortest path between any two nodes in the network. One of the main advantages of using Dijkstra's algorithm in a fat tree topology is its ability to efficiently handle the hierarchical structure of the network. In a fat tree topology, the core, aggregate, and edge layers are connected in a tree-like structure, which allows for a more efficient search for the shortest path. By finding multiple paths of the same length, Dijkstra's algorithm can focus on a smaller region of the topology, reducing the search space and minimizing the number of nodes that need to be visited.

Additionally, Dijkstra's algorithm can also be used to perform traffic engineering in a fat tree topology. By finding multiple paths of the same length, the algorithm can balance the traffic load across the network, avoiding congestion and minimizing response time. Finally, Dijkstra's algorithm can also be used to provide network fault-tolerance, by finding multiple paths that can be used to reroute traffic in the event of a link or node failure, minimizing the impact on network availability.

To align with the tree topology model, the algorithm takes a conceptual supernode as a start and runs the Dijkstra's algorithm with the supernode as the root. By setting the distance between the supernode and each donor to 0, we guarantee that every donor is a direct child of the supernode, and the sub-tree under each donor forms an independent tree topology. This routing scheme then takes the reciprocal of link rate, i.e. $1/rate$ to serve as the distance for each link, and performs the Dijkstra's algorithm to derive the complete network topology.

PSEUDO CODE FOR DIJKSTRA'S ALGORITHM

Considering the topology as a weighted graph below is pseudo code for dijkstra's algorithm

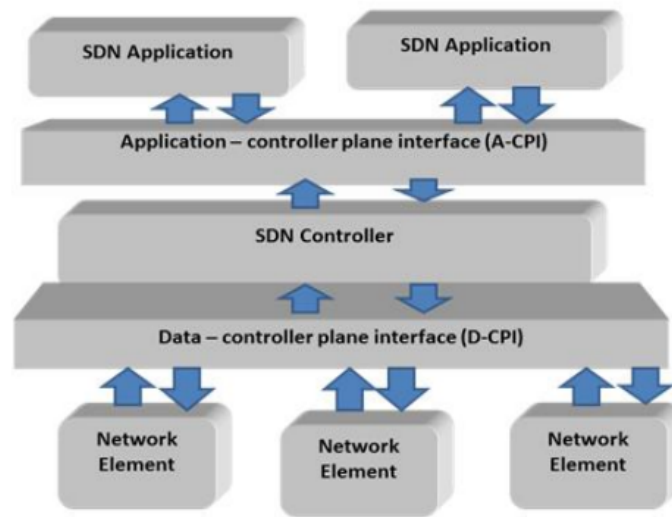
1. Create a set of unvisited nodes and initialize the distance to each node as infinity, except for the starting node which should have a distance of 0.
2. Select the node with the smallest distance from the unvisited set and mark it as visited.

3. For each neighbor of the current node, calculate the distance to that neighbor by adding the edge weight to the current node's distance. If the calculated distance is less than the current distance of the neighbor, update the distance.
4. Repeat steps 2 and 3 until all nodes have been visited or the destination node has been reached.
5. The shortest path to the destination node can be reconstructed by tracing the path of updated distances from the destination node to the starting node.

USE OF SDN IN LOAD BALANCING

- ☐ In Software-Defined Networking (SDN), load balancing refers to the process of distributing network traffic across multiple network paths or devices to ensure that no single device becomes a bottleneck. Load balancing in SDN can be implemented in a variety of ways, including using a dedicated load balancer device, or by incorporating load balancing functionality into the SDN controller.
- ☐ One of the key benefits of using SDN for load balancing is the ability to programmatically control the distribution of traffic. The SDN controller can use various algorithms, such as round-robin, least connections, and IP hash, to determine the next hop for a given packet. This allows for more flexibility and granular control over network traffic, as well as the ability to quickly adapt to changes in network conditions.
- ☐ Another benefit of using SDN for load balancing is the ability to monitor the network in real-time and make decisions based on current network conditions. For example, the SDN controller can monitor the utilization of each network device and redirect traffic to less utilized devices to prevent overloading.
- ☐ Additionally, SDN allows for network virtualization, where multiple virtual networks can be created on top of a single physical infrastructure, providing isolation and security between different groups of users or applications. This allows for different load balancing policies to be applied to different virtual networks, providing more granular control over traffic distribution.
- ☐ Overall, SDN-based load balancing allows for more flexibility, granularity, and programmability in the distribution of network traffic, making it a powerful tool for ensuring the efficient and effective use of network resources.

SYSTEM MODEL



The above figure shows the SDN components. There are three layers infrastructure, control and application layer. The infrastructure layer consists of network elements and it interacts with controller plane using controller plane interface. SDN application interacts with controller via Application controller plane interface. SDN controller translates the network requirements to network elements through controller.

PROPOSED METHODOLOGY

We have used the software tool mininet and RYU controller and our basic aim is to achieve the efficient load balancing using SDN. So by using the above described software tools we have implemented the load balancing. We have used the shortest path first and to implement is we have used python language.

Implementation

In Raspberry Pi:

Finding Raspberry Pi IP Address:

```
pi@raspberrypi:~ $ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 169.254.202.202  netmask 255.255.0.0  broadcast 169.254.255.255
    inet6 fe80::ce56:c3d:83a:7c1c  prefixlen 64  scopeid 0x20<link>
    ether b8:27:eb:38:27:3f  txqueuelen 1000  (Ethernet)
    RX packets 23555  bytes 2026286 (1.9 MiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 29899  bytes 22926975 (21.8 MiB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
    loop txqueuelen 1000  (Local Loopback)
    RX packets 81626  bytes 11181170 (10.6 MiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 81626  bytes 11181170 (10.6 MiB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.80.19  netmask 255.255.255.0  broadcast 192.168.80.255
    inet6 2401:4900:6321:7098:d3db:4f99:121a:c414  prefixlen 64  scopeid 0x0
```

Raspberry Pi IP Address: 192.168.80.19

Implementing Custom Controller in Raspberry pi:

Run-controller.sh

```
#!/bin/bash

controller="./controllers/controller.py"
observe=1
rest=1

usage(){
    echo "usage: run-controller.h [-f file] [-o] [-r]"
```

```

}

while [ "$1" != "" ]; do
    case $1 in
        -f | --file ) shift
            controller=$1
            ;;
        -o | --observe-links )
            observe=0
            ;;
        -r | --rest )
            rest=0
            ;;
        * )      usage
            exit 1
    esac
    shift
done

if [ "$observe" == "1" ] && [ "$rest" == "1" ]; then
    ryu-manager $controller ryu.app.ofctl_rest --observe-links
elif [ "$observe" == "1" ]; then
    ryu-manager $controller --observe-links
elif [ "$rest" == "1" ]; then
    ryu-manager $controller ryu.ap.ofctl_rest
else
    ryu-manager $controller
fi

```

Controller.py

```

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

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 = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                           ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions, buffer_id=None):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]

        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                     priority=priority, match=match,
                                     instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                     match=match, instructions=inst)
        datapath.send_msg(mod)

    @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)

msg = ev.msg
datapath = msg.datapath
ofproto = datapath.ofproto
parser = datapath.ofproto_parser
in_port = msg.match['in_port']

pkt = packet.Packet(msg.data)
eth = pkt.get_protocols(ethernet.ethernet)[0]

if eth.ethertype == ether_types.ETH_TYPE_LLDP or eth.ethertype ==
ether_types.ETH_TYPE_IPV6:
    # ignore lldp packet
    return

dst = eth.dst
src = eth.src

dpid = datapath.id
self.mac_to_port.setdefault(dpid, {})

self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

#STEP 1-----
if not src in self.mac_to_port[dpid]:
    self.mac_to_port[dpid][src] = in_port
    out_port = ofproto.OFPP_FLOOD
    actions = [parser.OFPAActionOutput(out_port)]
else:
    if self.mac_to_port[dpid][src] != in_port:
        out_port = -1;
        actions = []
    else:
        out_port = ofproto.OFPP_FLOOD
        actions = [parser.OFPAActionOutput(out_port)]

#-----

```

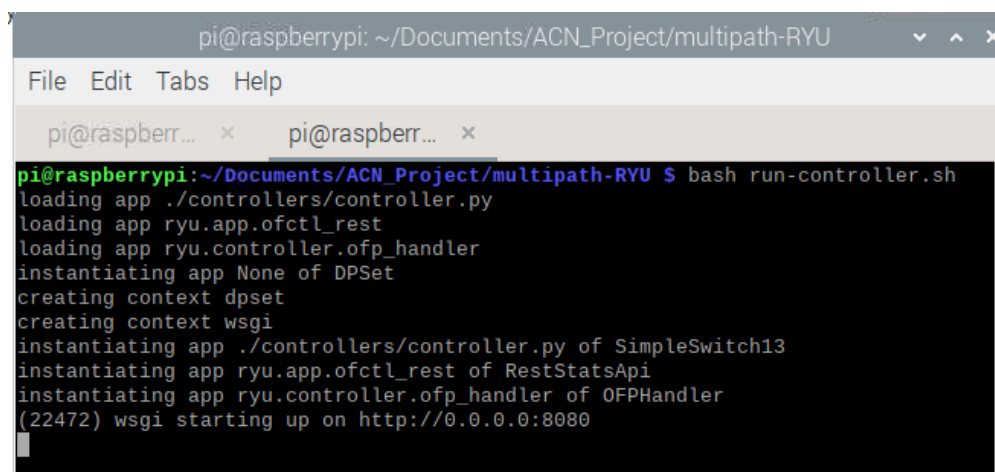


```

#-----
# install a flow to avoid packet_in next time
if out_port != ofproto.OFPP_FLOOD:
    match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
    # verify if we have a valid buffer_id, if yes avoid to send both
    # flow_mod & packet_out
    if msg.buffer_id != ofproto.OFP_NO_BUFFER:
        self.add_flow(datapath, 1, match, actions, msg.buffer_id)
        return
    else:
        self.add_flow(datapath, 1, match, actions)
data = None
if msg.buffer_id == ofproto.OFP_NO_BUFFER:
    data = msg.data

out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                          in_port=in_port, actions=actions, data=data)
datapath.send_msg(out)

```



The screenshot shows a terminal window titled 'pi@raspberrypi: ~/Documents/ACN_Project/multipath-RYU'. The terminal output shows the execution of 'bash run-controller.sh', which loads several Python modules and instantiates various components. The output ends with 'wsgi starting up on http://0.0.0.0:8080'.

```

pi@raspberrypi: ~/Documents/ACN_Project/multipath-RYU
File Edit Tabs Help

pi@raspberr... x pi@raspberr... x

pi@raspberrypi:~/Documents/ACN_Project/multipath-RYU $ bash run-controller.sh
loading app ./controllers/controller.py
loading app ryu.app.ofctl_rest
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ./controllers/controller.py of SimpleSwitch13
instantiating app ryu.app.ofctl_rest of RestStatsApi
instantiating app ryu.controller.ofp_handler of OFPHandler
(22472) wsgi starting up on http://0.0.0.0:8080

```

Creating Virtual Network in Virtual Machine using the Remote controller in Raspberry pi:

build-topo.sh

```
#!/bin/bash

topology="./topologies/mn_ft.py"
topo="ft"

usage(){
    echo "usage: run.sh [[-f file]] [[-t topo]]"
}

while [ "$1" != "" ]; do
    case $1 in
        -f | --file ) shift
            topology=$1
            ;;
        -t | --topo ) shift
            topo=$1
            ;;
        * )          usage
                    exit 1
    esac
    shift
done

sudo mn --custom $topology --topo $topo --mac --switch ovs --controller remote,ip=192.168.80.19
sudo mn -c; clear
exit 0
```

In bash file we are assigning the IP address of Raspberry Pi in the remote controller IP Address

Custom Fat Tree Topology in python:

fat_tree.py

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSController
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
```

```

from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def myNetwork():

    net = Mininet( topo=None,
                  build=False,
                  ipBase='10.0.0.0/8',
                  autoSetMacs = True)

    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                        controller=RemoteController,
                        ip='192.168.80.19',
                        protocol='tcp',
                        port=6633)
# Assigning the IP Address of Raspberry pi for remote controller IP Address

    info( '*** Add switches\n')
    s10 = net.addSwitch('s10', cls=OVSKernelSwitch)
    s20 = net.addSwitch('s20', cls=OVSKernelSwitch)
    s6 = net.addSwitch('s6', cls=OVSKernelSwitch)
    s11 = net.addSwitch('s11', cls=OVSKernelSwitch)
    s5 = net.addSwitch('s5', cls=OVSKernelSwitch)
    s14 = net.addSwitch('s14', cls=OVSKernelSwitch)
    s12 = net.addSwitch('s12', cls=OVSKernelSwitch)
    s15 = net.addSwitch('s15', cls=OVSKernelSwitch)
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
    s7 = net.addSwitch('s7', cls=OVSKernelSwitch)
    s16 = net.addSwitch('s16', cls=OVSKernelSwitch)
    s4 = net.addSwitch('s4', cls=OVSKernelSwitch)
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
    s17 = net.addSwitch('s17', cls=OVSKernelSwitch)
    s8 = net.addSwitch('s8', cls=OVSKernelSwitch)
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
    s18 = net.addSwitch('s18', cls=OVSKernelSwitch)

```

```
s13 = net.addSwitch('s13', cls=OVSKernelSwitch)
s9 = net.addSwitch('s9', cls=OVSKernelSwitch)
s19 = net.addSwitch('s19', cls=OVSKernelSwitch)

info( '*** Add hosts\n')
h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
h8 = net.addHost('h8', cls=Host, ip='10.0.0.8', defaultRoute=None)
h5 = net.addHost('h5', cls=Host, ip='10.0.0.5', defaultRoute=None)
h4 = net.addHost('h4', cls=Host, ip='10.0.0.4', defaultRoute=None)
h10 = net.addHost('h10', cls=Host, ip='10.0.0.10', defaultRoute=None)
h15 = net.addHost('h15', cls=Host, ip='10.0.0.15', defaultRoute=None)
h7 = net.addHost('h7', cls=Host, ip='10.0.0.7', defaultRoute=None)
h12 = net.addHost('h12', cls=Host, ip='10.0.0.12', defaultRoute=None)
h16 = net.addHost('h16', cls=Host, ip='10.0.0.16', defaultRoute=None)
h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
h14 = net.addHost('h14', cls=Host, ip='10.0.0.14', defaultRoute=None)
h11 = net.addHost('h11', cls=Host, ip='10.0.0.11', defaultRoute=None)
h13 = net.addHost('h13', cls=Host, ip='10.0.0.13', defaultRoute=None)
h9 = net.addHost('h9', cls=Host, ip='10.0.0.9', defaultRoute=None)
h6 = net.addHost('h6', cls=Host, ip='10.0.0.6', defaultRoute=None)

info( '*** Add links\n')
net.addLink(h1, s13)
net.addLink(s13, h2)
net.addLink(s14, h3)
net.addLink(s14, h4)
net.addLink(s15, h5)
net.addLink(s15, h6)
net.addLink(s16, h7)
net.addLink(s16, h8)
net.addLink(s17, h9)
net.addLink(s17, h10)
net.addLink(s18, h11)
net.addLink(s18, h12)
net.addLink(s19, h13)
net.addLink(s19, h14)
net.addLink(s20, h15)
net.addLink(s20, h16)
```

```
net.addLink(s13, s5)
net.addLink(s13, s6)
net.addLink(s14, s5)
net.addLink(s14, s6)
net.addLink(s15, s7)
net.addLink(s15, s8)
net.addLink(s16, s7)
net.addLink(s16, s8)
net.addLink(s17, s9)
net.addLink(s17, s10)
net.addLink(s18, s9)
net.addLink(s18, s10)
net.addLink(s19, s11)
net.addLink(s19, s12)
net.addLink(s20, s11)
net.addLink(s20, s12)
net.addLink(s5, s1)
net.addLink(s5, s2)
net.addLink(s6, s3)
net.addLink(s6, s4)
net.addLink(s7, s1)
net.addLink(s7, s2)
net.addLink(s8, s3)
net.addLink(s8, s4)
net.addLink(s9, s1)
net.addLink(s9, s2)
net.addLink(s10, s3)
net.addLink(s10, s4)
net.addLink(s11, s1)
net.addLink(s11, s2)
net.addLink(s12, s3)
net.addLink(s12, s4)

info( '*** Starting network\n')
net.build()
info( '*** Starting controllers\n')
for controller in net.controllers:
    controller.start()
```

```

info( '*** Starting switches\n')
net.get('s10').start([c0])
net.get('s20').start([c0])
net.get('s6').start([c0])
net.get('s11').start([c0])
net.get('s5').start([c0])
net.get('s14').start([c0])
net.get('s12').start([c0])
net.get('s15').start([c0])
net.get('s1').start([c0])
net.get('s7').start([c0])
net.get('s16').start([c0])
net.get('s4').start([c0])
net.get('s3').start([c0])
net.get('s17').start([c0])
net.get('s8').start([c0])
net.get('s2').start([c0])
net.get('s18').start([c0])
net.get('s13').start([c0])
net.get('s9').start([c0])
net.get('s19').start([c0])

info( '*** Post configure switches and hosts\n')

CLI(net)
net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    myNetwork()

```

Algorithms Used:

dijkstra.py

```

import networkx as nx
import copy

```

```

def dijskrta(G, source_node):
    #initialize distances
    distance = {}
    nodes = G.nodes()
    for x in nodes:
        #distance to all other nodes is infinity
        distance[x] = {'node_name': x, 'distance': float("inf"), 'next_hop': None}

    #distance to source node is 0
    distance[source_node] = {'node_name': source_node, 'distance': 0, 'next_hop': source_node};

    visited_nodes = {}
    unvisited_nodes = copy.copy(distance)

    while(len(unvisited_nodes) > 0):
        min_dist = float('inf')
        min_node = None

        #get node with minimum distance
        for x in unvisited_nodes.values():
            if(x['distance'] == float('inf')):
                continue
            if x['distance'] <= min_dist:
                min_dist = x['distance']
                min_node = x['node_name']
                next_hop = x['next_hop']

        if (min_node not in visited_nodes):
            visited_nodes[min_node] = [next_hop]
            for x in G.neighbors(min_node):
                if(x in visited_nodes):
                    continue
                if(min_dist + G[min_node][x]['weight'] < unvisited_nodes[x]['distance']):
                    unvisited_nodes[x]['distance'] = min_dist + G[min_node][x]['weight']
                    unvisited_nodes[x]['node_name'] = x
                    unvisited_nodes[x]['next_hop'] = x if min_node == source_node else next_hop

        distance[min_node] = unvisited_nodes[min_node]
        del unvisited_nodes[min_node]

```

```

return distance

if __name__ == '__main__':

    G = nx.Graph()
    G.add_nodes_from([1,2,3,4,5,6])

    G.add_edges_from([(1,2,{ 'weight':3}), (1,3,{ 'weight':5}), (2,3,{ 'weight':1}), (2,4,{ 'weight':4}), (2,6,{ 'weight':5}), (3,4,{ 'weight':3}), (3,5,{ 'weight':7}), (4,5,{ 'weight':2}), (4,6,{ 'weight':8})])
    d = dijkstras(G, 1)
    print(str(d))

```

2_dijkstra.py

```

from heapq import heappop, heappush
import networkx as nx

def dijkstra(G, source_node):
    routes = {}
    unvisited = [(0, source_node, source_node)]
    visited = set()

    while unvisited:
        (cost, dst, nh) = heappop(unvisited)
        if dst not in visited or cost == routes[dst][0][1]:
            visited.add(dst)
            if nh == source_node:
                nh = dst

            if dst in routes:
                routes[dst].append((nh, cost))
            else:
                routes[dst] = [(nh, cost)]

            for n in G.neighbors(dst):
                heappush(unvisited, (cost+G[dst][n]['cost'], n, nh))

    return routes

```



```

def mk_topo(pods, bw):
    num_hosts      = (pods ** 3)/4
    num_agg_switches = pods * pods
    num_core_switches = (pods * pods)/4

    hosts = [('h' + str(i), {'type':'host'})
              for i in range(1, num_hosts + 1)]

    core_switches = [('s' + str(i), {'type':'switch','id':i})
                     for i in range(1,num_core_switches + 1)]

    agg_switches = [('s' + str(i), {'type':'switch','id':i})
                    for i in range(num_core_switches + 1,num_core_switches + num_agg_switches+ 1)]

    g = nx.DiGraph()
    g.add_nodes_from(hosts)
    g.add_nodes_from(core_switches)
    g.add_nodes_from(agg_switches)

    host_offset = 0
    for pod in range(pods):
        core_offset = 0
        for sw in range(pods/2):
            switch = agg_switches[(pod*pods) + sw][0]
            # Connect to core switches
            for port in range(pods/2):
                core_switch = core_switches[core_offset][0]
                g.add_edge(switch,core_switch,
                          {'src_port':port,'dst_port':pod,'capacity':bw,'cost':1})
                g.add_edge(core_switch,switch,
                          {'src_port':pod,'dst_port':port,'capacity':bw,'cost':1})
                core_offset += 1

            # Connect to aggregate switches in same pod
            for port in range(pods/2,pods):
                lower_switch = agg_switches[(pod*pods) + port][0]
                g.add_edge(switch,lower_switch,

```

```

        {'src_port':port,'dst_port':sw,'capacity':bw,'cost':1})
    g.add_edge(lower_switch,switch,
        {'src_port':sw,'dst_port':port,'capacity':bw,'cost':1})

    for sw in range(pods/2,pods):
        switch = agg_switches[(pod*pods) + sw][0]
        # Connect to hosts
        for port in range(pods/2,pods): # First k/2 pods connect to upper layer
            host = hosts[host_offset][0]
            # All hosts connect on port 0
            g.add_edge(switch,host,
                {'src_port':port,'dst_port':0,'capacity':bw,'cost':1})
            g.add_edge(host,switch,
                {'src_port':0,'dst_port':port,'capacity':bw,'cost':1})
            host_offset += 1

    return g

if __name__ == "__main__":

    G = mk_topo(4,1)
    # edges = [
    #     ("A","B", {'cost':7}),
    #     ("A","D", {'cost':5}),
    #     ("B","C", {'cost':8}),
    #     ("B","D", {'cost':9}),
    #     #("B","E", {'cost':7}),
    #     ("C","E", {'cost':5}),
    #     ("D","E", {'cost':15}),
    #     ("D","F", {'cost':6}),
    #     ("E","F", {'cost':8}),
    #     ("E","G", {'cost':9}),
    #     ("F","G", {'cost':11})
    # ]

    # edges = [
    #     ("A", "C", {'cost': 1}),
    #     ("A", "G", {'cost': 1}),
    #     ("A", "E", {'cost': 1}),

```

```

# ("B", "D", {'cost': 1}),
# ("B", "H", {'cost': 1}),
# ("B", "F", {'cost': 1}),
# ("D", "C", {'cost': 1}),
# ("H", "G", {'cost': 1}),
# ("F", "E", {'cost': 1}),
# ("F", "C", {'cost': 1}),
# ("D", "G", {'cost': 1}),
# ("H", "E", {'cost': 1}),
# ]

# G.add_edges_from(edges)
# print edges

for x in G:
    for y in G.neighbors(x):
        print x,y, G[x][y]['cost']

print "DIJSKTRA-----"

r = dijkstra(G, "h1")

print r

```

Multipath Routing with Load Balancing

Running Custom Controller in Raspberry Pi:

```

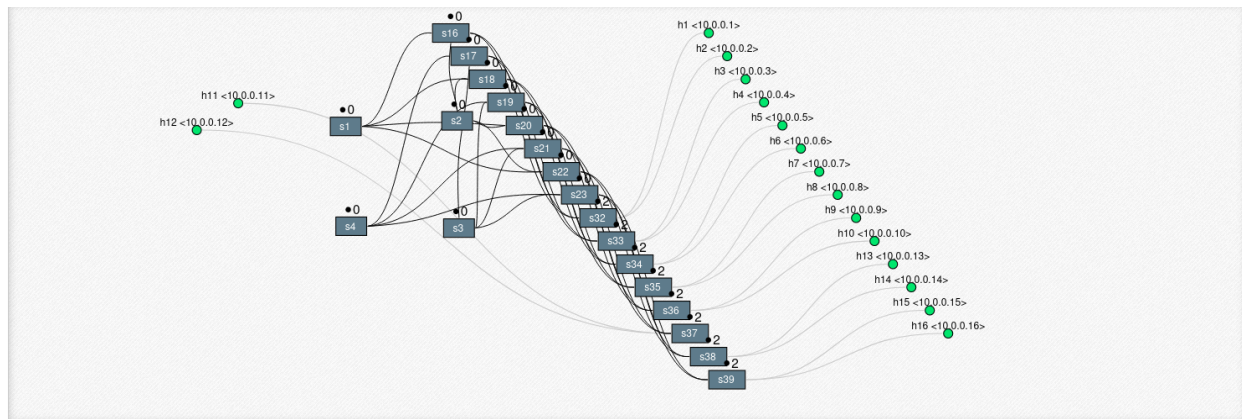
pi@raspberrypi: ~/Documents/ACN_Project/multipath-RYU
File Edit Tabs Help

pi@raspberr... x pi@raspberr... x

pi@raspberrypi:~/Documents/ACN_Project/multipath-RYU $ bash run-controller.sh
loading app ./controllers/controller.py
loading app ryu.app.ofctl_rest
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ./controllers/controller.py of SimpleSwitch13
instantiating app ryu.app.ofctl_rest of RestStatsApi
instantiating app ryu.controller.ofp_handler of OFPHandler
(22472) wsgi starting up on http://0.0.0.0:8080

```

FAT TREE TOPOLOGY



EXECUTION OF TOPOLOGY IN VIRTUAL MACHINE

```

ameen@ameen:~/Documents/ACN_Project/multipath-RYU$ bash build-topo.sh
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.80.19:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
*** Adding switches:
s1 s2 s3 s4 s16 s17 s18 s19 s20 s21 s22 s23 s32 s33 s34 s35 s36 s37 s38 s39
*** Adding links:
(h1, s32) (h2, s32) (h3, s33) (h4, s33) (h5, s34) (h6, s34) (h7, s35) (h8, s35) (h9, s36) (h10, s36) (h11, s37) (h12, s37) (h13, s38) (h14, s38)
(h15, s39) (h16, s39) (s16, s1) (s16, s2) (s17, s3) (s17, s4) (s18, s1) (s18, s2) (s19, s3) (s19, s4) (s20, s1) (s20, s2) (s21, s3) (s21, s4) (
s22, s1) (s22, s2) (s23, s3) (s23, s4) (s32, s16) (s32, s17) (s33, s16) (s33, s17) (s34, s18) (s34, s19) (s35, s18) (s35, s19) (s36, s20) (s36,
s21) (s37, s20) (s37, s21) (s38, s22) (s38, s23) (s39, s22) (s39, s23)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
*** Starting controller
c0
*** Starting 20 switches
s1 s2 s3 s4 s16 s17 s18 s19 s20 s21 s22 s23 s32 s33 s34 s35 s36 s37 s38 s39 ...
*** Starting CLI:

```

```

*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h10 h11 h12 h13 h14 h15 h16
h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h10 h11 h12 h13 h14 h15 h16
h9 -> h1 h2 h3 h4 h5 h6 h7 h8 h10 h11 h12 h13 h14 h15 h16
h10 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h11 h12 h13 h14 h15 h16
h11 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h12 h13 h14 h15 h16
h12 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h13 h14 h15 h16
h13 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h14 h15 h16
h14 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h15 h16
h15 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h16
h16 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
*** Results: 0% dropped (240/240 received)
mininet>

```

Successfully pinged each hosts

Check if packets are load-balanced between the two paths

- Setting 'h1' and as server

```

"Node: h1"
root@ghaayathri-HP-Laptop-14s-er0xxx:/home/ghaayathri/Downloads/multipath-RYU-master# iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)

[ 80] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 57774
[ ID] Interval      Transfer    Bandwidth
[ 80] 0.0-11.4 sec  2.75 MBytes  2.02 Mbits/sec
[ 80] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 57926
[ 81] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 57918
[ 83] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 57924
[ 84] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 57940
[ 81] 0.0-18.8 sec  1.75 MBytes  781 Kbits/sec
[ 84] 0.0-19.2 sec   768 KBytes  328 Kbits/sec
[ 80] 0.0-19.5 sec   640 KBytes  269 Kbits/sec
[ 83] 0.0-21.8 sec  1.38 MBytes  529 Kbits/sec
[SUM] 0.0-21.8 sec  4.50 MBytes  1.73 Mbits/sec
root@ghaayathri-HP-Laptop-14s-er0xxx:/home/ghaayathri/Downloads/multipath-RYU-master#

```

- Setting 'h2' and as client

```

"Node: h2"
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 79] local 10.0.0.2 port 57774 connected with 10.0.0.1 port 5001
[ 10] Interval      Transfer      Bandwidth
[ 79] 0.0-10.0 sec  2.75 MBytes  2.31 Mbits/sec
root@ghaayathri-HP-Laptop-14s-er0xxx:/home/ghaayathri/Downloads/multipath-RYU-master# iperf -c 10.0.0.1 -P 4
-----
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 82] local 10.0.0.2 port 57940 connected with 10.0.0.1 port 5001
[ 79] local 10.0.0.2 port 57918 connected with 10.0.0.1 port 5001
[ 80] local 10.0.0.2 port 57924 connected with 10.0.0.1 port 5001
[ 81] local 10.0.0.2 port 57926 connected with 10.0.0.1 port 5001
[ 10] Interval      Transfer      Bandwidth
[ 79] 0.0-10.0 sec  1.75 MBytes  1.47 Mbits/sec
[ 82] 0.0-10.2 sec   768 KBytes   617 Kbits/sec
[ 81] 0.0-12.2 sec   640 KBytes   431 Kbits/sec
[ 80] 0.0-12.5 sec   1.38 MBytes   922 Kbits/sec
[SUM] 0.0-12.5 sec  4.50 MBytes   3.02 Mbits/sec
root@ghaayathri-HP-Laptop-14s-er0xxx:/home/ghaayathri/Downloads/multipath-RYU-master#

```

- Check the total packets sent from the outgoing ports of s4

```

ameen@Ameen: ~/Documents/ACN Project/multipath-RYU
ameen@Ameen: ~/Documents/ACN Project/multipath-RYU x ameen@Ameen: ~/Documents/ACN Project/multipath-RYU x
ameen@Ameen:~/Documents/ACN Project/multipath-RYU$ sudo ovs-ofctl -O OpenFlow13 dump-ports s4
[sudo] password for ameen:
OFPST_PORT reply (OF1.3) (xid=0x2): 5 ports
port LOCAL: rx pkts=0, bytes=0, drop=1806, errs=0, frame=0, over=0, crc=0
tx pkts=0, bytes=0, drop=0, errs=0, coll=0
duration=3358.421s
port "s4-eth1": rx pkts=1616, bytes=523592, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=917, bytes=61114, drop=0, errs=0, coll=0
duration=3358.437s
port "s4-eth4": rx pkts=1695, bytes=528590, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=1533, bytes=518426, drop=0, errs=0, coll=0
duration=3358.437s
port "s4-eth2": rx pkts=1612, bytes=523424, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=1543, bytes=518846, drop=0, errs=0, coll=0
duration=3358.436s
port "s4-eth3": rx pkts=1542, bytes=518804, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=1557, bytes=519434, drop=0, errs=0, coll=0
duration=3358.437s
ameen@Ameen:~/Documents/ACN Project/multipath-RYU$

```

- Add the number of parallel clients created at h2

```

"Node: h2"
root@Ameen:/home/ameen/Documents/ACN Project/multipath-RYU# iperf -c 10.0.0.1 -P 4
-----
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 82] local 10.0.0.2 port 44686 connected with 10.0.0.1 port 5001
[ 79] local 10.0.0.2 port 44680 connected with 10.0.0.1 port 5001
[ 81] local 10.0.0.2 port 44684 connected with 10.0.0.1 port 5001
[ 80] local 10.0.0.2 port 44682 connected with 10.0.0.1 port 5001
[ 10] Interval      Transfer      Bandwidth
[ 81] 0.0-12.8 sec   400 KBytes   256 Kbits/sec
[ 82] 0.0-13.5 sec   324 KBytes   197 Kbits/sec
[ 79] 0.0-13.7 sec   290 KBytes   173 Kbits/sec
[ 80] 0.0-15.0 sec   314 KBytes   171 Kbits/sec
[SUM] 0.0-15.0 sec  1.30 MBytes   723 Kbits/sec
root@Ameen:/home/ameen/Documents/ACN Project/multipath-RYU#

```

```

"Node: h1"
root@Ameen:/home/ameen/Documents/ACN Project/multipath-RYU# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 80] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 44678
[ ID] Interval      Transfer      Bandwidth
[ 80] 0.0- 0.0 sec   404 KBytes   -nan bits/sec
S[ 80] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 44680
[ 83] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 44686
[ 82] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 44684
[ 81] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 44682
[ 83] 0.0-44.2 sec   324 KBytes   60.1 Kbits/sec
[ 80] 0.0-45.8 sec   290 KBytes   51.9 Kbits/sec
[ 81] 0.0-50.7 sec   314 KBytes   50.7 Kbits/sec
[ 82] 0.0-58.2 sec   400 KBytes   56.2 Kbits/sec
[SUM] 0.0-58.2 sec  1.30 MBytes   187 Kbits/sec

```

- Check the total packets sent from the outgoing ports of s4

```

ameen@Ameen: ~/Documents/ACN Project/multipath-RYU
ameen@Ameen: ~/Documents/ACN Project/multipath-RYU x ameen@Ameen: ~/Documents/ACN Project/multipath-RYU x
ameen@Ameen:~/Documents/ACN Project/multipath-RYU$ sudo ovs-ofctl -O OpenFlow12 dump-ports s4
[sudo] password for ameen:
OFPST_PORT reply (OF1.2) (xid=0x2): 5 ports
  port LOCAL: rx pkts=0, bytes=0, drop=3696, errs=0, frame=0, over=0, crc=0
              tx pkts=0, bytes=0, drop=0, errs=0, coll=0
  port "s4-eth1": rx pkts=3507, bytes=2013373, drop=0, errs=0, frame=0, over=0, crc=0
                  tx pkts=918, bytes=61239, drop=0, errs=0, coll=0
  port "s4-eth4": rx pkts=3586, bytes=2018371, drop=0, errs=0, frame=0, over=0, crc=0
                  tx pkts=3424, bytes=2008207, drop=0, errs=0, coll=0
  port "s4-eth2": rx pkts=3504, bytes=2013275, drop=0, errs=0, frame=0, over=0, crc=0
                  tx pkts=3434, bytes=2008627, drop=0, errs=0, coll=0
  port "s4-eth3": rx pkts=3433, bytes=2008585, drop=0, errs=0, frame=0, over=0, crc=0
                  tx pkts=3448, bytes=2009215, drop=0, errs=0, coll=0

```

STATELESS Load Balancing

```

vishnu@vishnu-0368:~$ sudo mn -c
[sudo] password for vishnu:
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs-testcontroller udpbwtest mnexec lvs ryu-manager z> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs-testcontroller udpbwtest mnexec lvs ryu-manager z> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
*** Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --if-exists del-br s1
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([_[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.

```

```
vishnu@vishnu-0308:~$ cd ryu/loadbalancing-sdn-master/
vishnu@vishnu-0308:~/ryu/loadbalancing-sdn-master$ ryu-manager lb_stateless.py
loading app lb_stateless.py
loading app ryu.controller.ofp_handler
instantiating app lb_stateless.py of loadbalancer
instantiating app ryu.controller.ofp_handler of OFPHandler
```

```
Every 2.0s: ovs-ofctl -O OpenFlow13 dump-flows s1 vishnu-0308: Mon Jan 23 22:20:12 2023
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0xf1265dea3a169fd8, duration=376.281s, table=0, n_packets=100, n_bytes=7400, tcp,in_port=4,dl_src=00:00:00:00:00:04,dl_dst=ab:bc:cd:ef:ab:bc,nw_src=10.0.0.4,nw_dst=10.0.0.100 actions=set_field:00:00:00:00:00:01->eth_dst,set_field:10.0.0.1->ip_dst,output:1
 cookie=0x9942e0e6cfb7122c, duration=376.282s, table=0, n_packets=100, n_bytes=5400, tcp,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04,nw_src=10.0.0.1,nw_dst=10.0.0.4 actions=set_field:ab:bc:cd:ef:ab:bc->eth_src,set_field:10.0.0.100->ip_src,output:4
 cookie=0xa450cdda936eeb0, duration=317.649s, table=0, n_packets=100, n_bytes=7400, tcp,in_port=5,dl_src=00:00:00:00:00:05,dl_dst=ab:bc:cd:ef:ab:bc,nw_src=10.0.0.5,nw_dst=10.0.0.100 actions=set_field:00:00:00:00:00:02->eth_dst,set_field:10.0.0.2->ip_dst,output:2
 cookie=0x45765a1c5726828f, duration=317.649s, table=0, n_packets=100, n_bytes=5400, tcp,in_port=2,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:05,nw_src=10.0.0.2,nw_dst=10.0.0.5 actions=set_field:ab:bc:cd:ef:ab:bc->eth_src,set_field:10.0.0.100->ip_src,output:5
 cookie=0xf23f4ed51205c9d, duration=291.473s, table=0, n_packets=100, n_bytes=7400, tcp,in_port=6,dl_src=00:00:00:00:00:06,dl_dst=ab:bc:cd:ef:ab:bc,nw_src=10.0.0.6,nw_dst=10.0.0.100 actions=set_field:00:00:00:00:00:03->eth_dst,set_field:10.0.0.3->ip_dst,output:3
 cookie=0x4ea25a7668ac9e63, duration=291.473s, table=0, n_packets=100, n_bytes=5400, tcp,in_port=3,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:06,nw_src=10.0.0.3,nw_dst=10.0.0.6 actions=set_field:ab:bc:cd:ef:ab:bc->eth_src,set_field:10.0.0.100->ip_src,output:6
 cookie=0xa0e4b0a0f80265d, duration=269.364s, table=0, n_packets=100, n_bytes=7400, tcp,in_port=7,dl_src=00:00:00:00:00:07,dl_dst=ab:bc:cd:ef:ab:bc,nw_src=10.0.0.7,nw_dst=10.0.0.100 actions=set_field:00:00:00:00:00:01->eth_dst,set_field:10.0.0.1->ip_dst,output:1
 cookie=0x1f4c2b2c0db802d, duration=269.364s, table=0, n_packets=100, n_bytes=5400, tcp,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:07,nw_src=10.0.0.1,nw_dst=10.0.0.7 actions=set_field:ab:bc:cd:ef:ab:bc->eth_src,set_field:10.0.0.100->ip_src,output:7
 cookie=0x0, duration=376.234s, table=0, n_packets=0, n_bytes=0, priority=1,in_port=4,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=317.681s, table=0, n_packets=0, n_bytes=0, priority=1,in_port=5,dl_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x0, duration=291.424s, table=0, n_packets=0, n_bytes=0, priority=1,in_port=6,dl_dst=00:00:00:00:00:03 actions=output:3
 cookie=0x0, duration=269.315s, table=0, n_packets=0, n_bytes=0, priority=1,in_port=7,dl_dst=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=1694.489s, table=0, n_packets=117, n_bytes=8126, priority=0 actions=CONTROLLER:65535
```

```
vishnu@vishnu-0308:~$ sudo mn --topo single,7 --mac --controller=remote --switch ovs,protocols=OpenFlow13
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1) (h7, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> xterm h1 h2 h3 h4 h5 h6 h7 s1
mininet> xterm h1 h2 h3 h4 h5 h6 h7 s1
mininet> □
```

```
"Node: h2"
root@vishnu-0308:/home/vishnu# python -m http.server 90
Serving HTTP on 0.0.0.0 port 90 (http://0.0.0.0:90/) ...
□

"Node: h1"
root@vishnu-0308:/home/vishnu# python -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
^C
Keyboard interrupt received, exiting.
root@vishnu-0308:/home/vishnu# python -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
^C
Keyboard interrupt received, exiting.
root@vishnu-0308:/home/vishnu# python -m http.server 90
Serving HTTP on 0.0.0.0 port 90 (http://0.0.0.0:90/) ...
□

"Node: h3"
root@vishnu-0308:/home/vishnu# python -m http.server 90
Serving HTTP on 0.0.0.0 port 90 (http://0.0.0.0:90/) ...
□
```


[illegible]

[illegible]

```
vishnu@vishnu-0308:~$ cd ryu/loadbalancing-sdn-master/
vishnu@vishnu-0308:~/ryu/loadbalancing-sdn-master$ ryu-manager lb_stateful.py
loading app lb_stateful.py
loading app ryu.controller.ofp_handler
instantiating app lb_stateful.py of loadbalancer
instantiating app ryu.controller.ofp_handler of OFPHandler
loadbalancer: Exception occurred during handler processing. Backtrace from offending handler
_in_handler] servicing event [EventOFPPacketIn] follows.
Traceback (most recent call last):
  File "/home/vishnu/.local/lib/python3.8/site-packages/ryu/base/app_manager.py", line 290
    _loop
      handler(ev)
  File "/home/vishnu/ryu/loadbalancing-sdn-master/lb_stateful.py", line 156, in _packet_in
    ip_header = pkt.get_protocols(ipv4.ipv4)[0]
IndexError: list index out of range
loadbalancer: Exception occurred during handler processing. Backtrace from offending handler
_in_handler] servicing event [EventOFPPacketIn] follows.
```



```
"Node: h3"
root@vishnu-0308:/home/vishnu# python -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.0.0.4 - - [23/Jan/2023 22:47:25] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:25] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:25] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:25] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:25] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:25] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:25] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:26] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:26] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:26] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:26] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:26] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:27] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:27] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:27] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:27] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:27] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:27] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:27] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:28] "GET / HTTP/1.1" 200 -
10.0.0.4 - - [23/Jan/2023 22:47:28] "GET / HTTP/1.1" 200 -
```

```
"Node: h4"
root@vishnu-0308:/home/vishnu# httpperf --server=10.0.0.100 --uri=/ --num-conns=
100 --rate=20
httpperf --client=0/1 --server=10.0.0.100 --port=80 --uri=/ --rate=20 --send-buff
er=4096 --recv-buffer=16384 --num-conns=100 --num-calls=1
httpperf; warning: open file limit > FD_SETSIZE; limiting max. # of open files to
FD_SETSIZE
Maximum connect burst length: 1

Total: connections 100 requests 100 replies 100 test-duration 5.959 s

Connection rate: 16.8 conn/s (59.6 ms/conn, <=21 concurrent connections)
Connection time [ms]: min 1002.0 avg 1019.4 max 1036.2 median 1018.5 stddev 9.6
Connection time [ms]: connect 1018.1
Connection length [replies/conn]: 1.000

Request rate: 16.8 req/s (59.6 ms/req)
Request size [B]: 63.0

Reply rate [replies/s]: min 16.0 avg 16.0 max 16.0 stddev 0.0 (1 samples)
Reply time [ms]: response 1.3 transfer 0.0
Reply size [B]: header 156.0 content 3966.0 footer 0.0 (total 4122.0)
Reply status: 1xx=0 2xx=100 3xx=0 4xx=0 5xx=0

CPU time [s]: user 0.85 system 5.11 (user 14.2% system 85.8% total 100.0%)
Net I/O: 68.6 KB/s (0.6*10^6 bps)

Errors: total 0 client-time 0 socket-time 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
root@vishnu-0308:/home/vishnu#
```

```
"Node: h5"
root@vishnu-0308:/home/vishnu# httpperf --server=10.0.0.100 --uri=/ --num-conns=
100 --rate=20
httpperf --client=0/1 --server=10.0.0.100 --port=80 --uri=/ --rate=20 --send-buff
er=4096 --recv-buffer=16384 --num-conns=100 --num-calls=1
httpperf: warning: open file limit > FD_SETSIZE; limiting max. # of open files to
FD_SETSIZE
Maximum connect burst length: 1

Total: connections 100 requests 100 replies 100 test-duration 5.977 s

Connection rate: 16.7 conn/s (59.8 ms/conn, <=21 concurrent connections)
Connection time [ms]: min 1002.5 avg 1017.9 max 1032.8 median 1018.5 stddev 9.3
Connection time [ms]: connect 1016.6
Connection length [replies/conn]: 1.000

Request rate: 16.7 req/s (59.8 ms/req)
Request size [B]: 63.0

Reply rate [replies/s]: min 16.0 avg 16.0 max 16.0 stddev 0.0 (1 samples)
Reply time [ms]: response 1.2 transfer 0.0
Reply size [B]: header 156.0 content 3966.0 footer 0.0 (total 4122.0)
Reply status: 1xx=0 2xx=100 3xx=0 4xx=0 5xx=0

CPU time [s]: user 0.74 system 5.23 (user 12.4% system 87.5% total 99.9%)
Net I/O: 68.4 KB/s (0.6*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
root@vishnu-0308:/home/vishnu#
```

```
"Node: h6"
root@vishnu-0308:/home/vishnu# httpperf --server=10.0.0.100 --uri=/ --num-con
100 --rate=20
httpperf --client=0/1 --server=10.0.0.100 --port=80 --uri=/ --rate=20 --send-
er=4096 --recv-buffer=16384 --num-conns=100 --num-calls=1
httpperf: warning: open file limit > FD_SETSIZE; limiting max. # of open
FD_SETSIZE
Maximum connect burst length: 1

Total: connections 100 requests 100 replies 100 test-duration 5.960 s

Connection rate: 16.8 conn/s (59.6 ms/conn, <=21 concurrent connections)
Connection time [ms]: min 1001.4 avg 1017.8 max 1033.6 median 1017.5 std
Connection time [ms]: connect 1016.5
Connection length [replies/conn]: 1.000

Request rate: 16.8 req/s (59.6 ms/req)
Request size [B]: 63.0

Reply rate [replies/s]: min 16.0 avg 16.0 max 16.0 stddev 0.0 (1 samples)
Reply time [ms]: response 1.2 transfer 0.0
Reply size [B]: header 156.0 content 3966.0 footer 0.0 (total 4122.0)
Reply status: 1xx=0 2xx=100 3xx=0 4xx=0 5xx=0

CPU time [s]: user 0.83 system 5.12 (user 14.0% system 86.0% total 100.0)
Net I/O: 68.6 KB/s (0.6*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
root@vishnu-0308:/home/vishnu#
```



```
"Node: h7"
root@vishnu-0308:/home/vishnu# httpperf --server=10.0.0.100 --uri=/ --num-conns=
100 --rate=20
httpperf --client=0/1 --server=10.0.0.100 --port=80 --uri=/ --rate=20 --send-buff
er=4096 --recv-buffer=16384 --num-conns=100 --num-calls=1
httpperf: warning: open file limit > FD_SETSIZE; limiting max. # of open files to
FD_SETSIZE
maximum connect burst length: 1

total: connections 100 requests 100 replies 100 test-duration 5.967 s

Connection rate: 16.8 conn/s (59.7 ms/conn, <=21 concurrent connections)
Connection time [ms]: min 1002.5 avg 1018.2 max 1037.0 median 1018.5 stddev 9.6
Connection time [ms]: connect 1016.9
Connection length [replies/conn]: 1.000

Request rate: 16.8 req/s (59.7 ms/req)
Request size [B]: 63.0

Reply rate [replies/s]: min 16.0 avg 16.0 max 16.0 stddev 0.0 (1 samples)
Reply time [ms]: response 1.2 transfer 0.0
Reply size [B]: header 196.0 content 3966.0 footer 0.0 (total 4122.0)
Reply status: 1xx=0 2xx=100 3xx=0 4xx=0 5xx=0

CPU time [s]: user 0.80 system 5.17 (user 13.4% system 86.6% total 100.0%)
Net I/O: 68.5 KB/s (0.6*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
root@vishnu-0308:/home/vishnu#
```

References:

- [1] <https://wildanmsyah.wordpress.com/2018/01/21/testing-ryu-multipath-routing-with-load-balancing-on-mininet/>
- [2] <https://users.cs.fiu.edu/~pand/publications/13icc-yu.pdf>
- [3] <https://www.ijert.org/research/a-survey-on-load-balancing-in-cloud-computing-IJERTV1IS9150.pdf>
- [4] <https://blogchinmaya.blogspot.com/2017/04/what-is-fat-tree-and-how-to-construct.html>