Improved Dijkstra's Algorithm based load balancing SDN application

SAN JOSÉ STATE UNIVERSITY

Group ID 9

Submitted by:

Tejaswi goel(Class ID-6, SJSU ID-010698623)
Nirbhay Kumar Singh(Class ID-26, SJSU ID-010693917)
Gunveet Singh Arora(Class ID-2, SJSU ID-010641904)

Submitted to:

Prof. Younghee Park

Talking about Mininet

Network simulation system used for interpretation. Mininet controls a lot of things including various switches and routers. Mininet is known for its features and advantages some of which are:

- Speed
- Compatibility
- Open Source
- Customization

GRE Tunnel and VxLAN Tunnel

- GRE tunnel is basically a tunneling protocol used for enclosing a lot of protocols that come under a point to point link network. This protocol is mostly deployed in between gateways or in some cases or from a gateway to an end station
- VxLAN Tunnel has uplink and a downlink

Floodlight

- Floodlight is an SDN Controller, to put it simply. It is an open source version available to everyone.

- One of the main Purposes of Floodlight controller are making instructions and controlling the rules on how the traffic has to be controlled

- The main advantages of using a floodlight controller are:

- Scalability
- Versatility
- Open Source

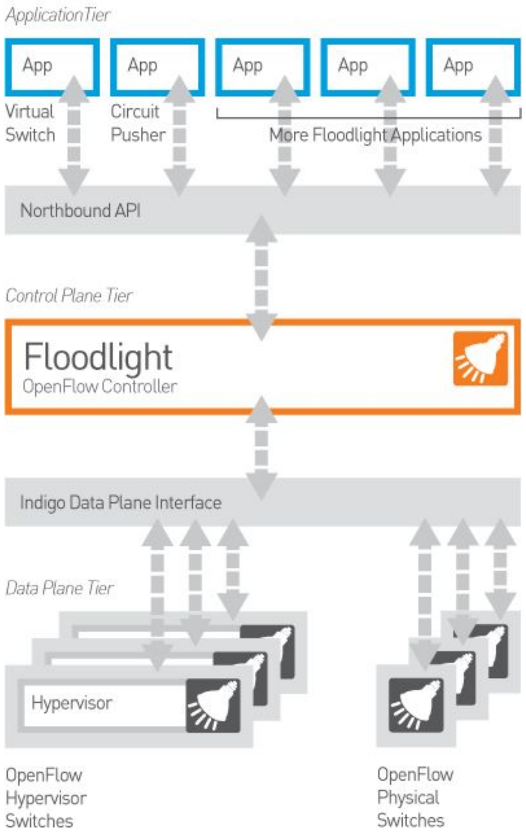


Fig 1: Architecture for Floodlight Controller

IPerf

- A simple networking tool designed to test the UDP and the TCP protocol so that we can get into the details about Network Bandwidth, delay and data loss.
- We take a look at the characteristics of the TCP and UDP and change them in such a manner as to run tests and know more about these protocols.

Dijkstra’s Algorithm

- In simple words, Dijkstra Algorithm is finding the shortest path between two nodes, which in context of networks can be two hosts connected through a mesh of Switches and links.
- In our case we extend the Dijkstra’s algorithm and compare it with the original to give the analysis in the form of graphs and figures.

Introduction to Load Balancing

- Load Balancing refers to the dividing of the workload among multiple computing Platforms to optimize the throughput

- This helps in reducing the redundancy and increasing the reliability

- Load Balancing can occur from client side as well as from the server side

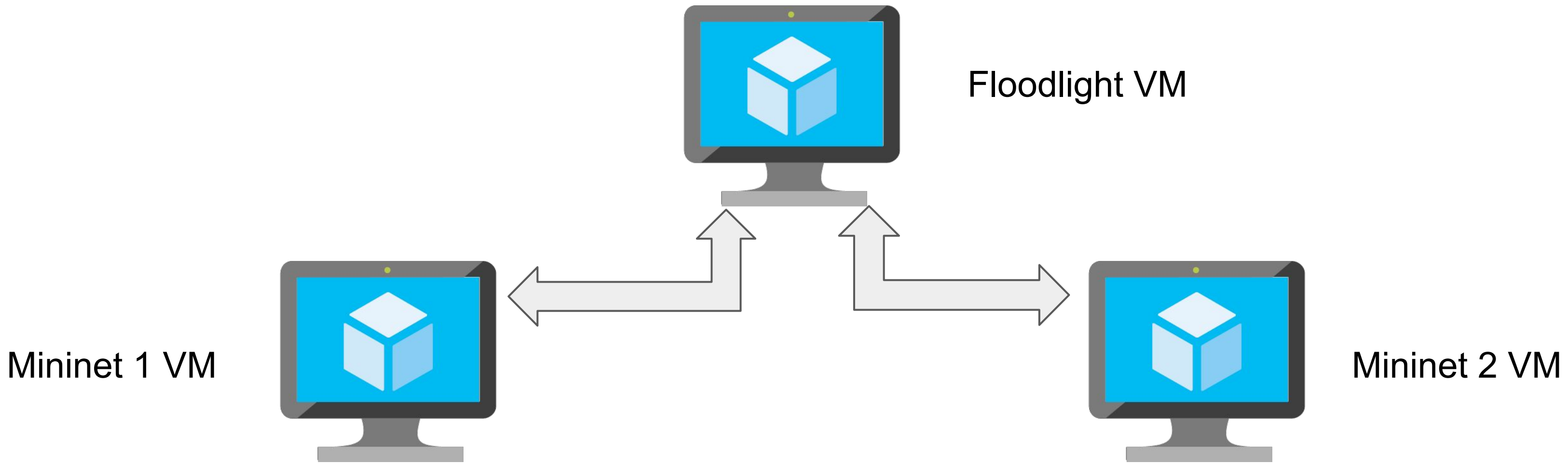
Motivation

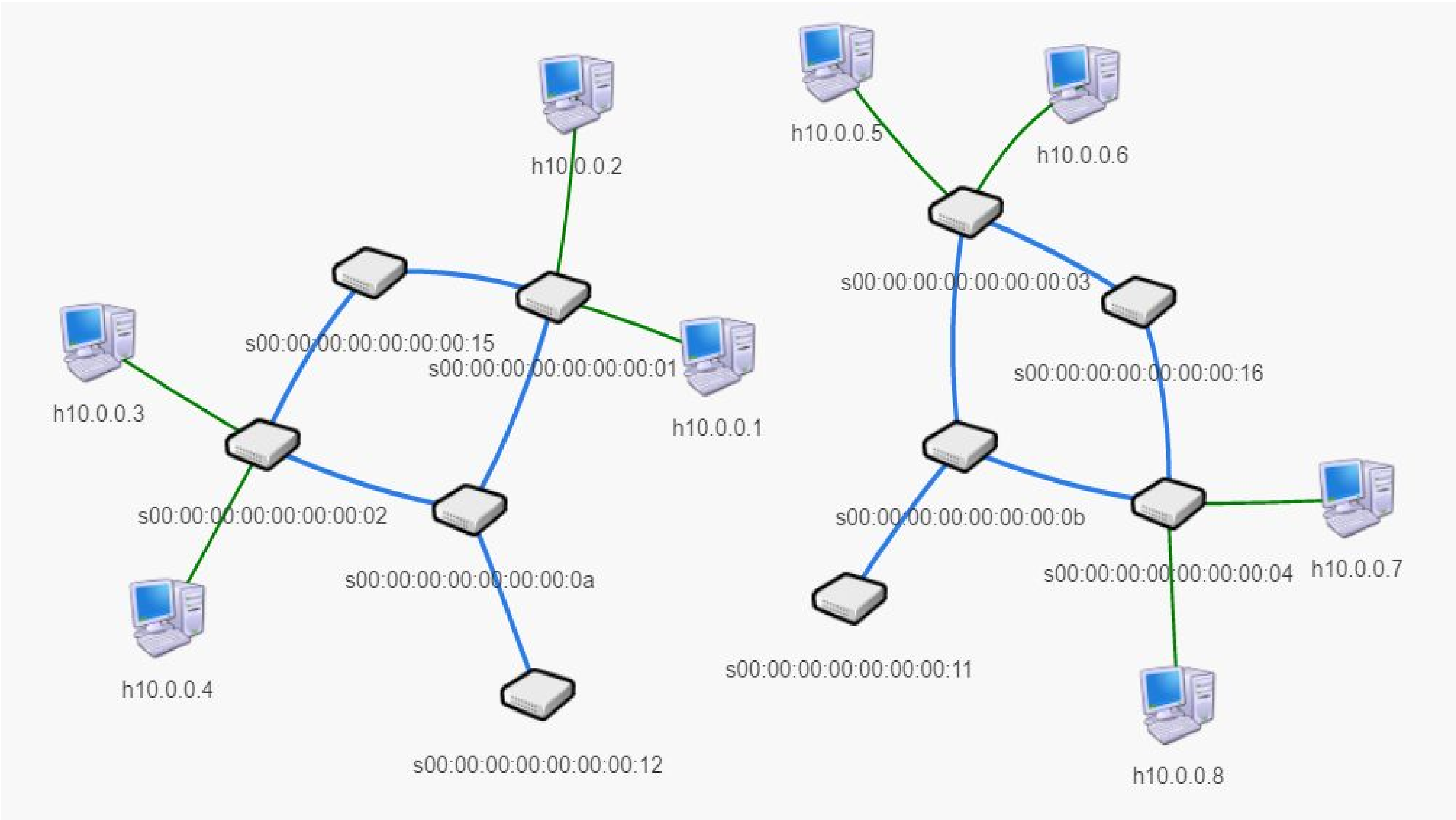
- At present, network traffic is growing fast and complex as enterprises need to purchase more equipment to handle this complex network. So, we wanted to develop the load balancer meeting the current need and outperform the current load balancer of floodlight.

- The online services like e-commerce, websites, and social networks frequently use multiple servers to get high reliability and availability. So, we focussed adding the control plane HA feature to floodlight controllers cluster.

Infrastructure or Topology setup (1)

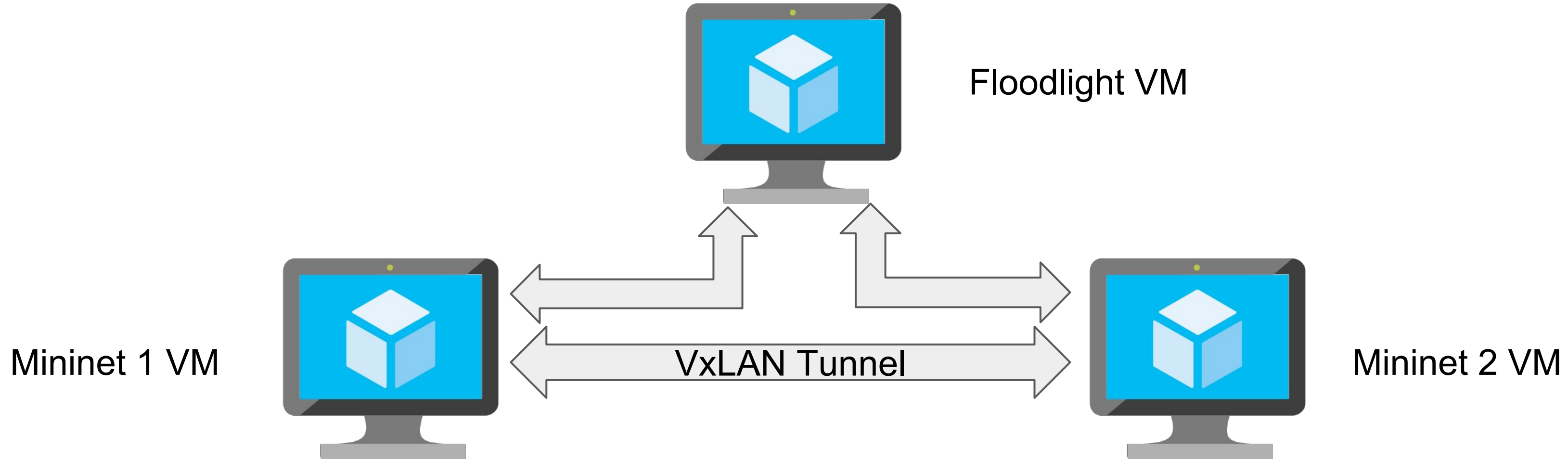
- We developed a WAN setup by connecting two topologies created in two mininet VMs with one GRE tunnel and one VxLAN tunnel
- Step 1: Topology without GRE or VxLAN tunnel

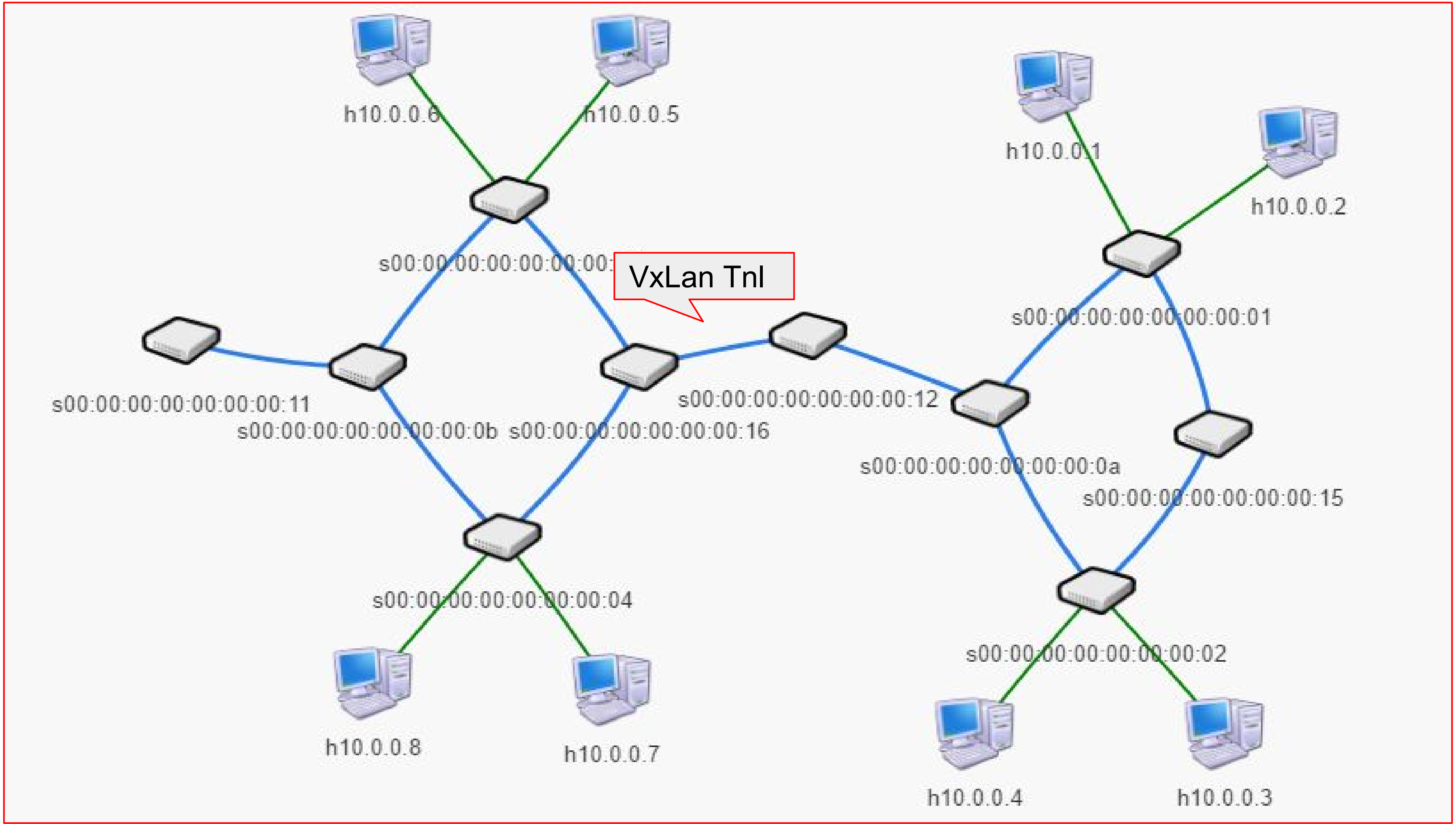



h10.0.0.2
h10.0.0.5
h10.0.0.6
s00:00:00:00:00:00:00:15
s00:00:00:00:00:00:00:01
h10.0.0.3
h10.0.0.1
s00:00:00:00:00:00:00:03
s00:00:00:00:00:00:00:16
s00:00:00:00:00:00:00:02
s00:00:00:00:00:00:00:0a
s00:00:00:00:00:00:00:0b
s00:00:00:00:00:00:00:04
h10.0.0.7
h10.0.0.4
s00:00:00:00:00:00:00:12
s00:00:00:00:00:00:00:11
h10.0.0.8

Infrastructure or Topology setup (2)

- Step 2: Create a VxLAN tunnel between S18 and S22.

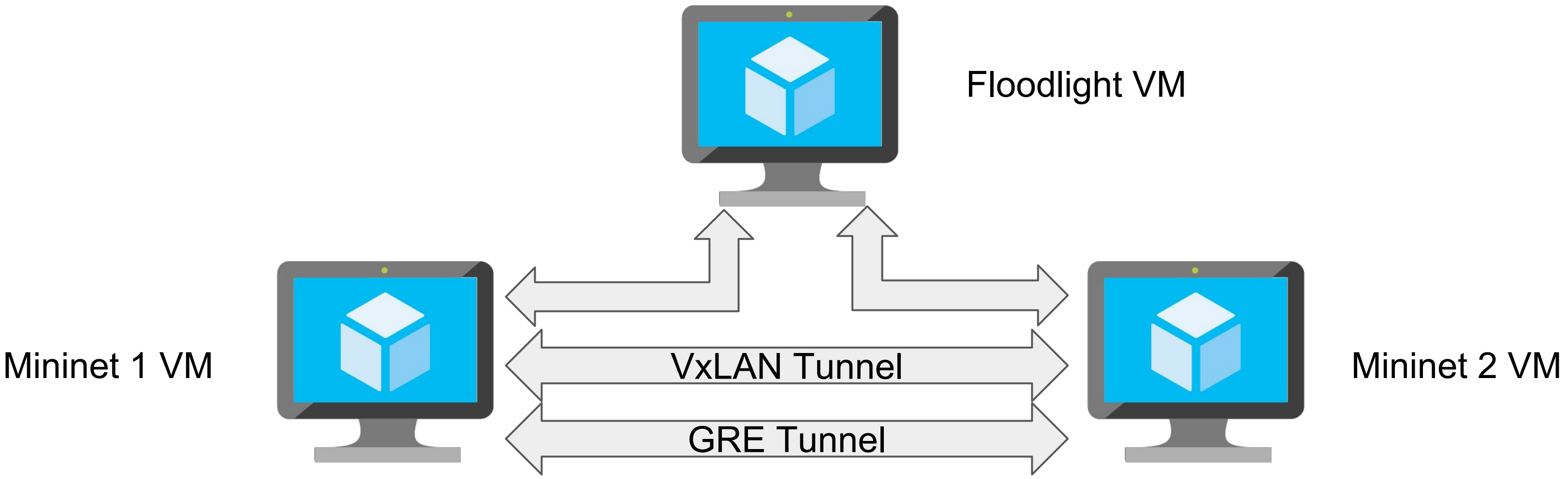


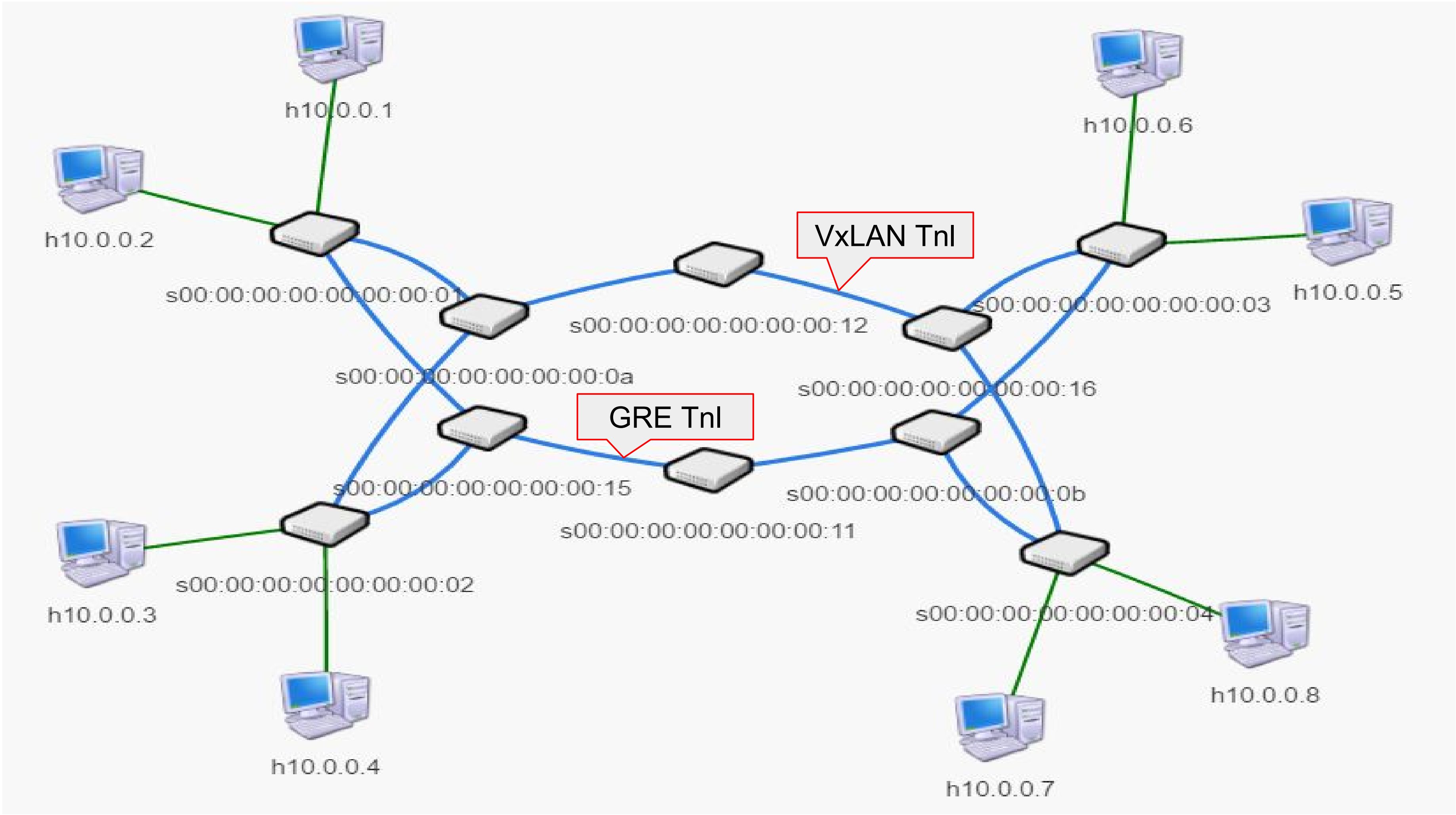


h10.0.0.6
h10.0.0.5
h10.0.0.1
h10.0.0.2
VxLan Tnl
s00:00:00:00:00:00:00:01
s00:00:00:00:00:00:00:11
s00:00:00:00:00:00:00:0b
s00:00:00:00:00:00:00:16
s00:00:00:00:00:00:00:12
s00:00:00:00:00:00:00:0a
s00:00:00:00:00:00:00:15
s00:00:00:00:00:00:00:04
s00:00:00:00:00:00:00:02
h10.0.0.8
h10.0.0.7
h10.0.0.4
h10.0.0.3

Infrastructure or Topology setup (3)

- Step 3: Create a GRE tunnel between S21 and S17




h10.0.0.1
h10.0.0.2
h10.0.0.3
h10.0.0.4
h10.0.0.5
h10.0.0.6
h10.0.0.7
h10.0.0.8
s00:00:00:00:00:00:00:01
s00:00:00:00:00:00:00:02
s00:00:00:00:00:00:00:03
s00:00:00:00:00:00:00:04
s00:00:00:00:00:00:00:0a
s00:00:00:00:00:00:00:0b
s00:00:00:00:00:00:00:11
s00:00:00:00:00:00:00:12
s00:00:00:00:00:00:00:15
s00:00:00:00:00:00:00:16
VxLAN Tnl
GRE Tnl

Infrastructure or Topology setup (4)

Learnings from the Infrastructure setup:

1. We could not connect two networks in two different mininet VMs with two GRE tunnels or two VxLAN tunnel.
2. So, we had to connect the two networks by creating one GRE tunnel and one VxLAN tunnel.

Implementation details

Start

API interface to collect extract real time meaningful data about the topology

The meaningful data from JSON data pool

Routing module to compute list of all possible shortest routes from one host to another

List of all possible shortest routes

Load balancing module to take the list and find the route with least transmission cost

The best route possible

Then the flow rule with the least cost is pushed into switches

Stop

Implemented methods of Load Balancing (1)

The default load balancing module in Floodlight uses Dijkstra's algorithm to calculate a list of shortest paths and chooses any one of them randomly

In extended Dijkstra's algorithm, we have implemented the load balancing module which chooses shortest paths using the following two attributes:

1. Bandwidth (Transmission Rate)
2. Node weight (NW) and Edge weight (EW)

Implemented methods of Load Balancing (2)

1. Implementation of extended Dijkstra's algorithm using bandwidth (Transmission Rate)


```
[
  {
    "dpid": "00:00:00:00:00:00:00:0b",
    "port": "2",
    "updated": "Fri Dec 02 01:15:14 PST 2016",
    "link-speed-bits-per-second": "10000000",
    "bits-per-second-rx": "60",
    "bits-per-second-tx": "60"
  }
]
```

Implemented methods of Load Balancing (3)

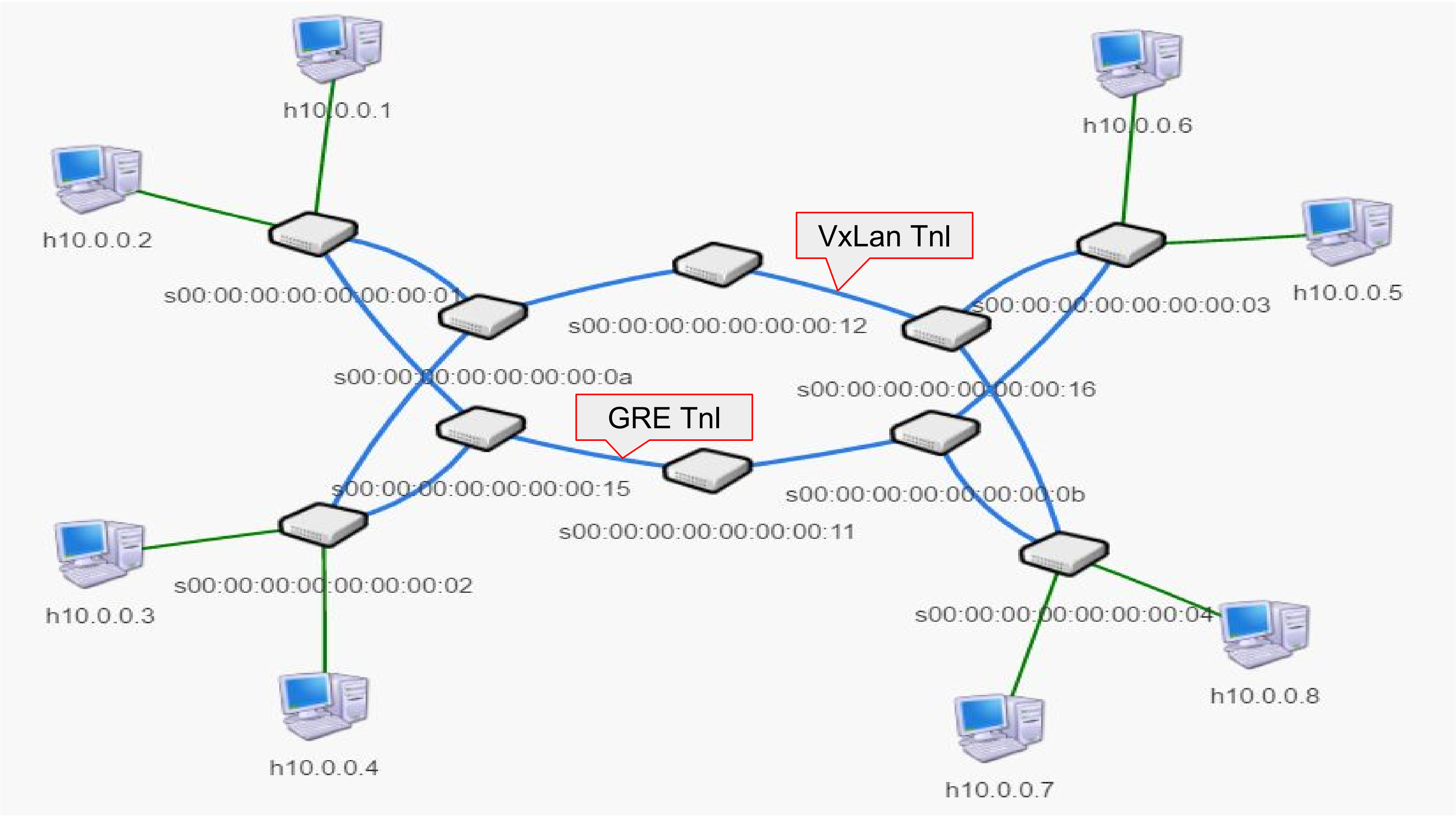
1. Node weight (NW) and Edge weight (EW): The definition of node weight and the edge weight
    a. Node weight - The amount of time a switch takes to process a certain number of packets (i.e. Latency of switch)
    b. Edge weight - The amount of time a link takes to transmit a certain number of packets (i.e. Latency of links)

2. Therefore,

Node weight

Edge weight

End to End latency = Sum of all <u>Switch Latencies</u> + Sum of all <u>Link Latencies</u>



h10.0.0.1
h10.0.0.2
h10.0.0.3
h10.0.0.4
h10.0.0.5
h10.0.0.6
h10.0.0.7
h10.0.0.8
s00:00:00:00:00:00:00:01
s00:00:00:00:00:00:00:02
s00:00:00:00:00:00:00:03
s00:00:00:00:00:00:00:04
s00:00:00:00:00:00:00:0a
s00:00:00:00:00:00:00:0b
s00:00:00:00:00:00:00:11
s00:00:00:00:00:00:00:12
s00:00:00:00:00:00:00:15
s00:00:00:00:00:00:00:16
VxLan Tnl
GRE Tnl

Implemented methods of Load Balancing (4)

/wm/core/switch/all/flow/json

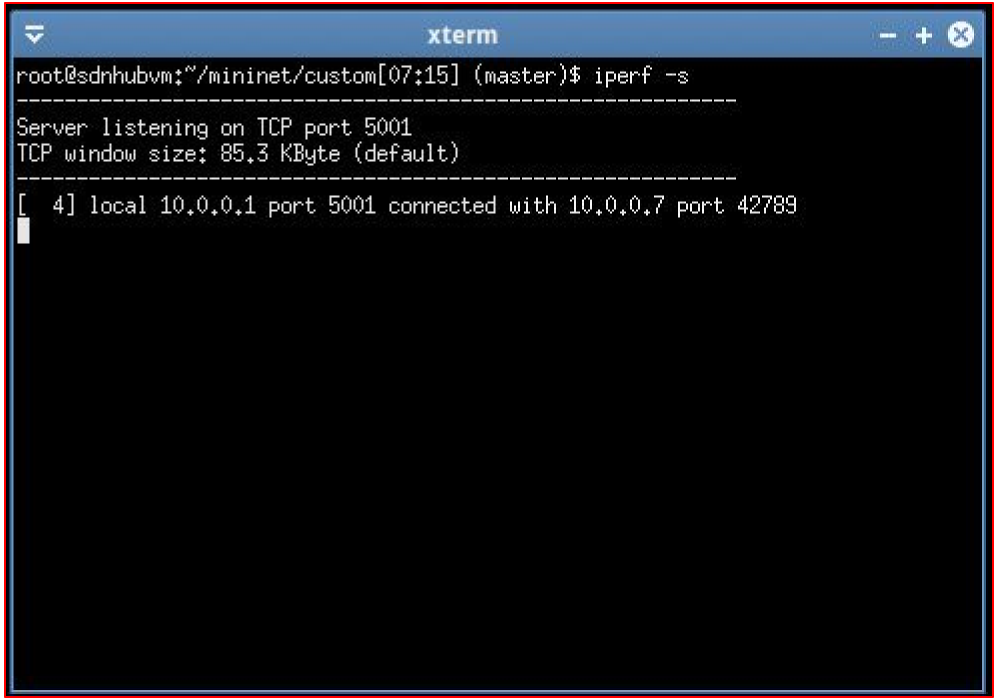

```
"00:00:00:00:00:00:00:0b": {
  "flows": [
    {
      "version": "OF_13",
      "cookie": "0",
      "table_id": "0x0",
      "packet_count": "1128",
      "byte_count": "84588",
      "duration_sec": "5314",
      "duration_nsec": "629000000",
      "priority": "0",
      "idle_timeout_s": "0",
      "hard_timeout_s": "0",
      "flags": [],
      "match": {},
      "instructions": {
        "instruction_apply_actions": {
          "actions": "output=controller"
        }
      }
    }
  ]
},
```

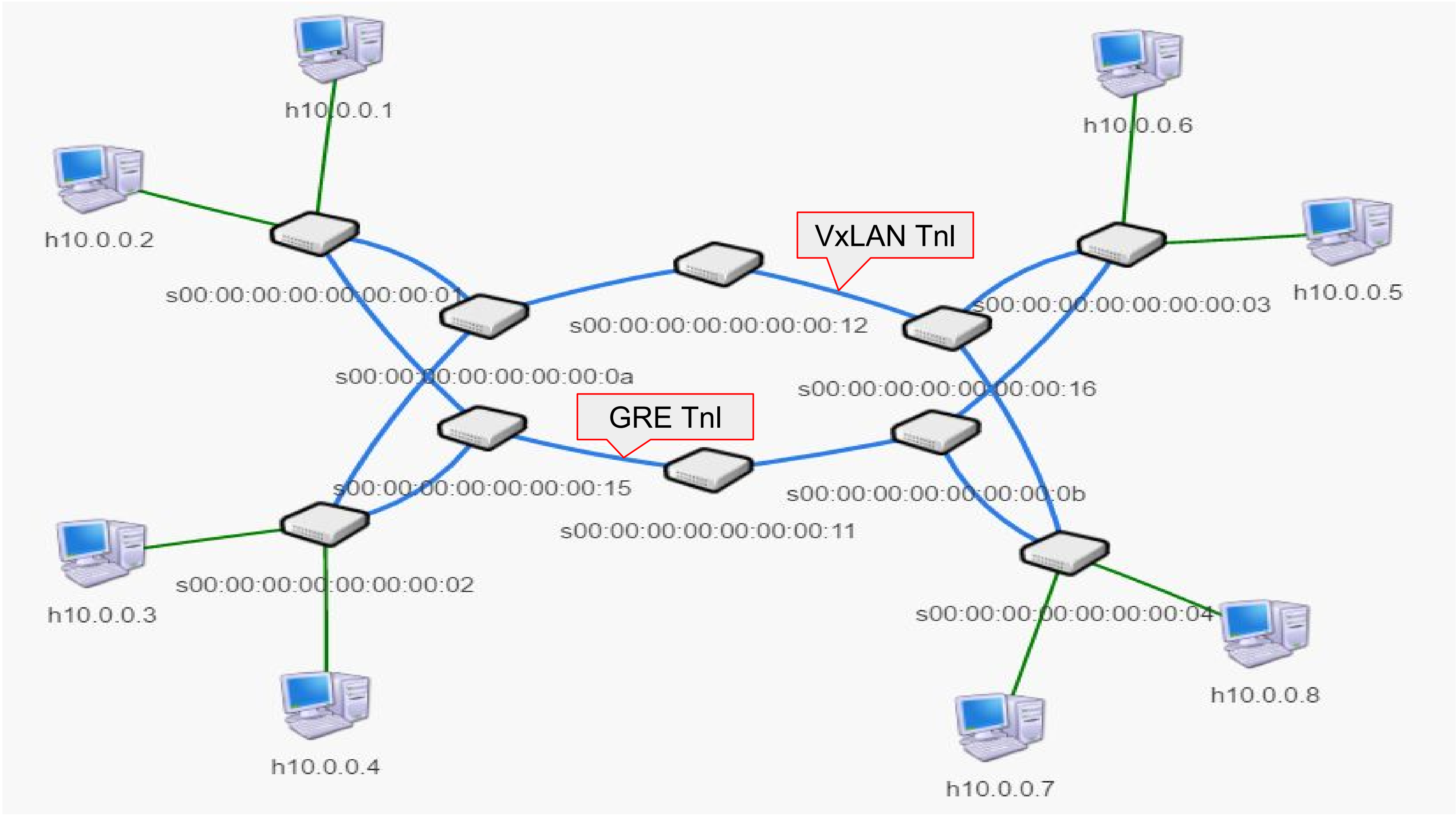

/wm/topology/links/json


```
{
    "src-switch": "00:00:00:00:00:00:00:04",
    "src-port": 3,
    "dst-switch": "00:00:00:00:00:00:00:0b",
    "dst-port": 2,
    "type": "internal",
    "direction": "bidirectional",
    "latency": 15
},
```

Iperf Test Results (1)

- Iperf server is created at the host 10.0.0.1 i.e. h1.
- When a client request comes, then the server responds to the request




h10.0.0.1
h10.0.0.2
h10.0.0.6
h10.0.0.5
VxLAN Tnl
s00:00:00:00:00:00:00:01
s00:00:00:00:00:00:00:12
s00:00:00:00:00:00:00:03
s00:00:00:00:00:00:00:0a
s00:00:00:00:00:00:00:16
GRE Tnl
s00:00:00:00:00:00:00:15
s00:00:00:00:00:00:00:0b
s00:00:00:00:00:00:00:11
s00:00:00:00:00:00:00:02
h10.0.0.3
s00:00:00:00:00:00:00:04
h10.0.0.8
h10.0.0.4
h10.0.0.7

Iperf Test Results (2)

Iperf results default load balancing:

```
root@sdnhubvm:~/mininet/custom[02:40] (master)$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.7 port 45076 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-960.0 sec  63.6 KBytes   543 bits/sec
root@sdnhubvm:~/mininet/custom[02:57] (master)$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.7 port 45077 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-939.4 sec  63.6 KBytes   555 bits/sec
root@sdnhubvm:~/mininet/custom[03:14] (master)$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.7 port 45078 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-1029.0 sec  66.5 KBytes   529 bits/sec
root@sdnhubvm:~/mininet/custom[03:36] (master)$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.7 port 45559 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-939.4 sec  63.6 KBytes   555 bits/sec
root@sdnhubvm:~/mininet/custom[11:51] (master)$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.7 port 45560 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-1028.9 sec  66.5 KBytes   529 bits/sec
root@sdnhubvm:~/mininet/custom[12:23] (master)$
```

S/No	Bandwidth Readings
1	543 bits/sec
2	555 bits/sec
3	529 bits/sec
4	555 bits/sec
5	529 bits/sec
	Average = 542.2 bits/sec

Average: 542.2 bits/sec

Iperf Test Results (3)

Iperf results with load balancing using Bandwidth (Tx) scheme:

```
root@sdnhubvm:~/mininet/custom[12:23] (master)$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.7 port 45561 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-1028.7 sec  66.5 KBytes   529 bits/sec
root@sdnhubvm:~/mininet/custom[12:57] (master)$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.7 port 45562 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-939.5 sec  63.6 KBytes   555 bits/sec
root@sdnhubvm:~/mininet/custom[13:22] (master)$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.7 port 45563 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-935.1 sec  63.6 KBytes   557 bits/sec
root@sdnhubvm:~/mininet/custom[13:50] (master)$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.7 port 45564 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-935.1 sec  63.6 KBytes   557 bits/sec
root@sdnhubvm:~/mininet/custom[14:20] (master)$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.7 port 45565 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-939.1 sec  63.6 KBytes   555 bits/sec
root@sdnhubvm:~/mininet/custom[15:51] (master)$
```

S/No	Bandwidth Readings
1	529 bits/sec
2	555 bits/sec
3	557 bits/sec
4	557 bits/sec
5	555 bits/sec
	Average = 550.6 bits/sec

Average:
550.6 bits/sec

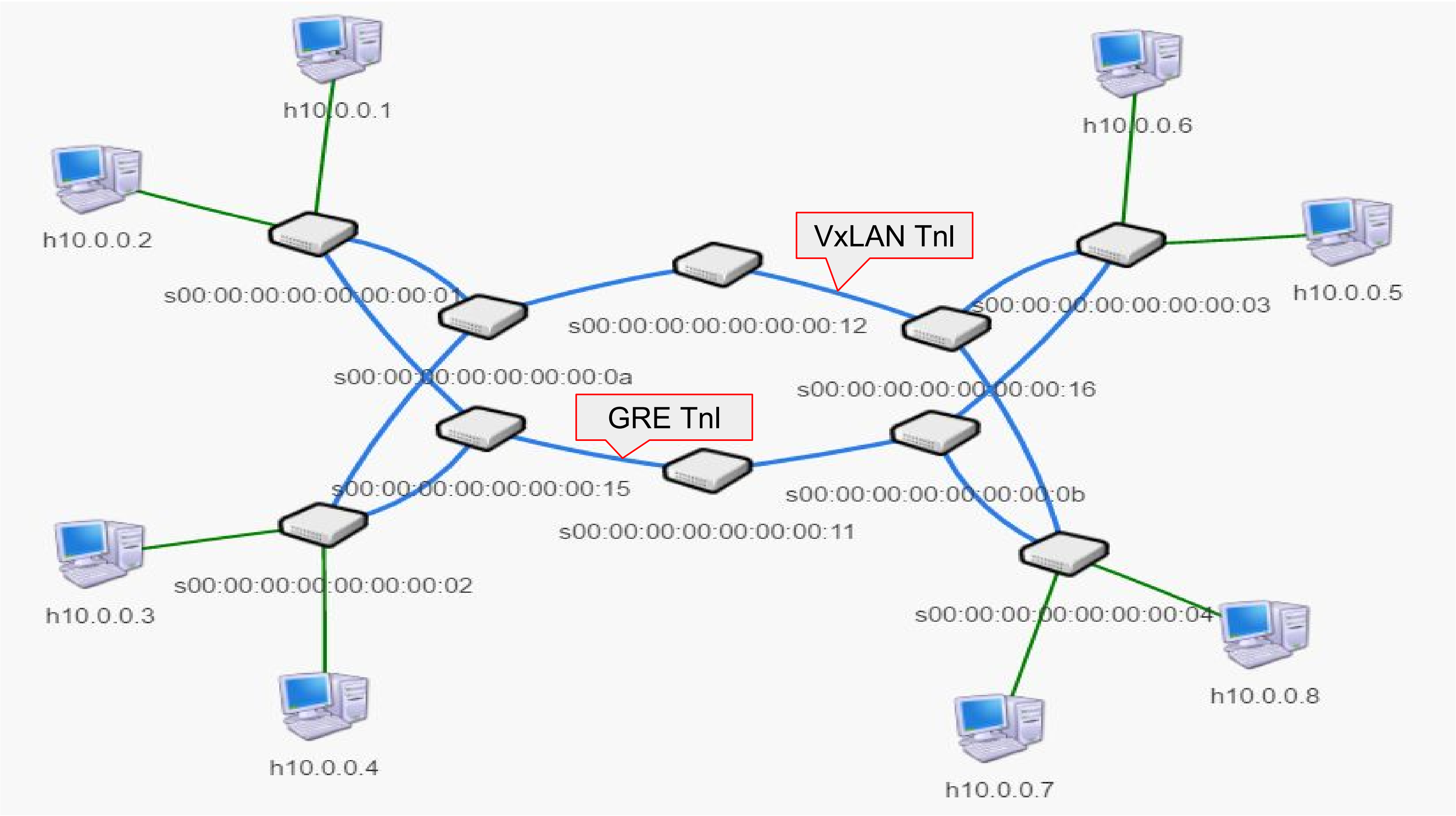
Iperf Test Results (4)

Iperf results with load balancing using NW/EW scheme:

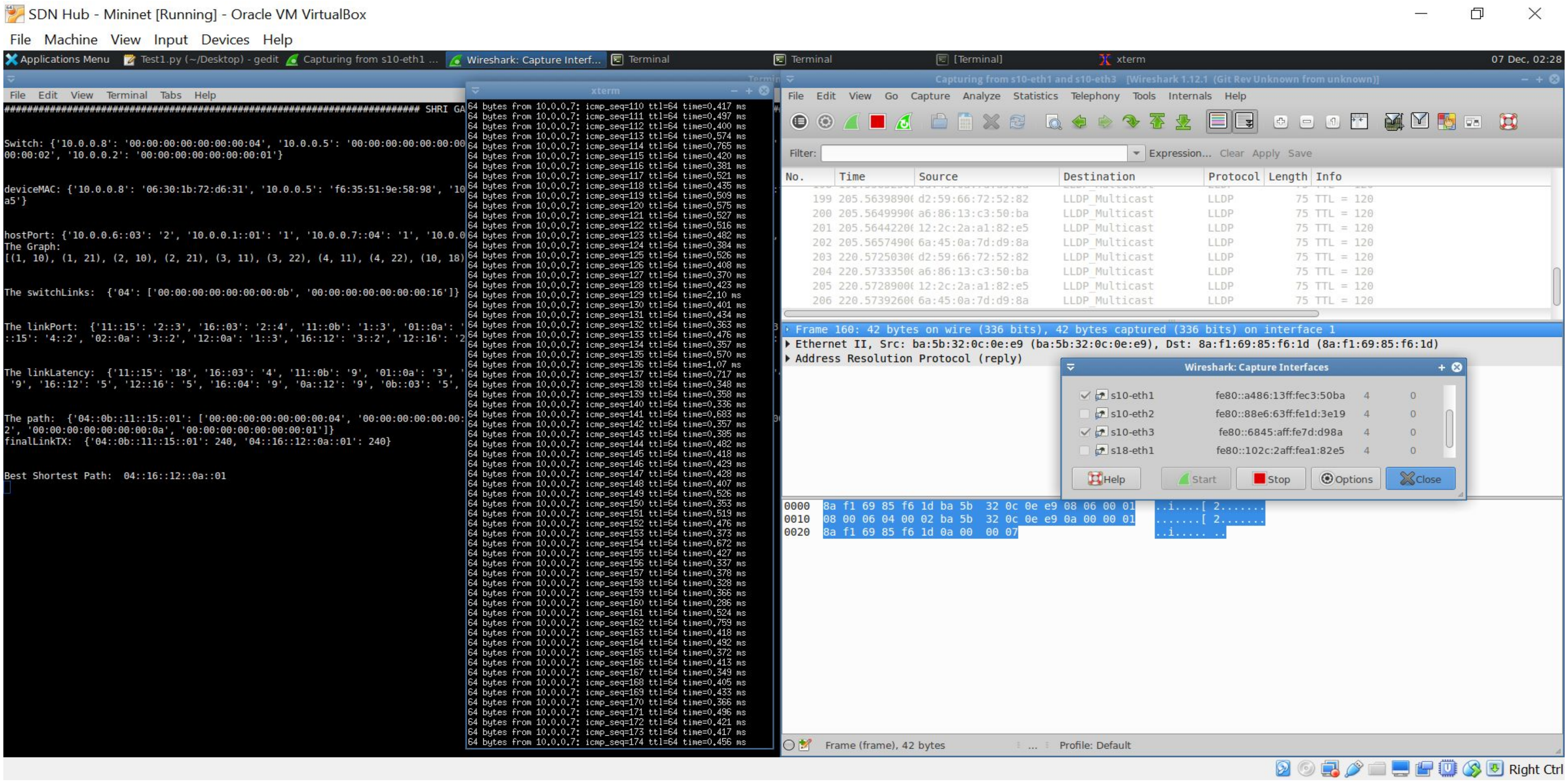
```
root@sdnhubvm:~/mininet/custom[15:51] (master)$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.7 port 50618 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-935.2 sec  63.6 KBytes   557 bits/sec
root@sdnhubvm:~/mininet/custom[16:12] (master)$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.7 port 50619 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-943.2 sec  63.6 KBytes   553 bits/sec
root@sdnhubvm:~/mininet/custom[16:50] (master)$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.7 port 50620 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-939.4 sec  63.6 KBytes   555 bits/sec
root@sdnhubvm:~/mininet/custom[17:24] (master)$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.7 port 50621 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-939.1 sec  63.6 KBytes   555 bits/sec
root@sdnhubvm:~/mininet/custom[17:41] (master)$ iperf -c 10.0.0.1
connect failed: No route to host
root@sdnhubvm:~/mininet/custom[20:09] (master)$ iperf -c 10.0.0.1
------------------------------------------------------------
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.7 port 58314 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-931.3 sec  63.6 KBytes   560 bits/sec
root@sdnhubvm:~/mininet/custom[20:36] (master)$
```

S/No	Bandwidth Readings
1	557 bits/sec
2	553 bits/sec
3	555 bits/sec
4	555 bits/sec
5	560 bits/sec
	Average = 556 bits/sec

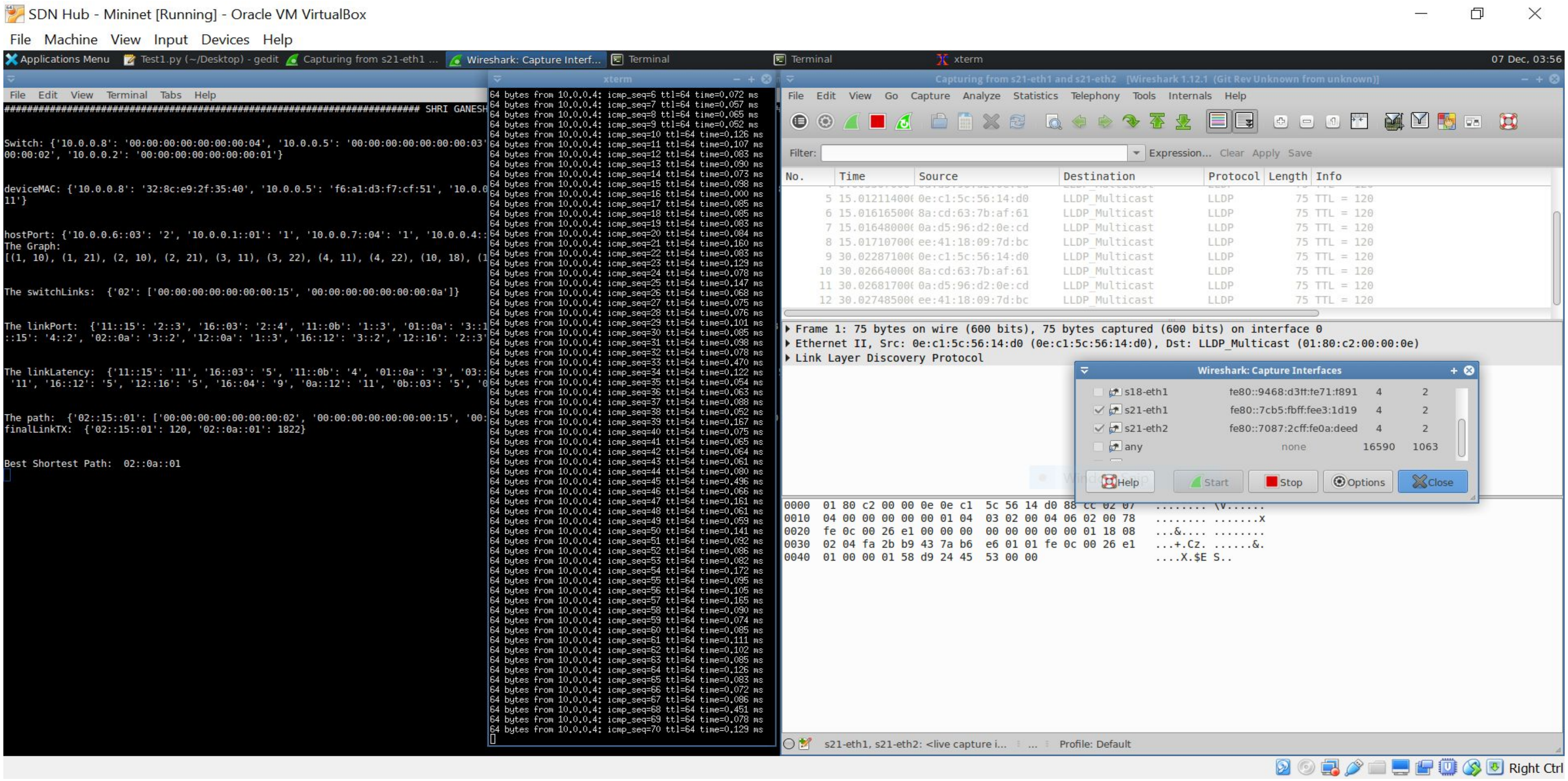
Average:
556 bits/sec


h10.0.0.1
h10.0.0.2
h10.0.0.6
h10.0.0.5
VxLAN Tnl
s00:00:00:00:00:00:00:01
s00:00:00:00:00:00:00:12
s00:00:00:00:00:00:00:03
s00:00:00:00:00:00:00:0a
s00:00:00:00:00:00:00:16
GRE Tnl
s00:00:00:00:00:00:00:15
s00:00:00:00:00:00:00:0b
s00:00:00:00:00:00:00:11
s00:00:00:00:00:00:00:02
h10.0.0.3
s00:00:00:00:00:00:00:04
h10.0.0.8
h10.0.0.4
h10.0.0.7

Wireshark test results (1)



Wireshark test results (2)



High Availability Cluster

A controller failure can quickly paralyze the entire network.

Many HA tools available in market: Pacemaker, Heartbeat, Corosync.

Heartbeat:

1. Automatically create **VIP** for our cluster of hosts/nodes.
2. **VIP** should at all times answer to requests, directing the traffic to **primary node** normally, but in case master is down, **secondary node** should take over automatically and answer to requests to the VIP

Heartbeat has three main configuration files defined for both Nodes:

1. /etc/ha.d/ha.cf
2. /etc/ha.d/authkeys
3. /etc/ha.d/haresources

Heartbeat configuration files (on both nodes)

```
Terminal - root@node2: /etc/ha.d
File   Edit   View   Terminal   Tabs   Help
root@node2:/etc/ha.d# ls
authkeys  ha.cf  harc  haresources  rc.d  README.config  resource.d  shellfuncs
root@node2:/etc/ha.d# cat ha.cf
keepalive 2
warntime 3
deadtime 5
initdead 10
udpport 694
auto_failback on
ucast eth1 192.168.56.102
ucast eth1 192.168.56.103
logfile /var/log/ha-log
debugfile /var/log/ha-debug
node node1 node2
autojoin none

root@node2:/etc/ha.d# cat haresources
node1 IPaddr::192.168.56.105/24/eth1:0 failback.sh
root@node2:/etc/ha.d# cat authkeys
auth 1
1 sha1 tg
root@node2:/etc/ha.d#
```

Demo

1. https://youtu.be/OcqasJ_0EpM
2. https://youtu.be/xS3qukzEzCg
3. https://youtu.be/KCJB9Oz-IOc

Conclusion

1. We were successful in designing the advanced Load balancer with multiple elements to find the best path for routing and it outperforms the existing load balancer of floodlight.

2. The working of HA in the floodlight controllers cluster has widened our scope of research to the scaling of control plane and replication of data plane.

Learning outcomes:

1. Mininet
2. SDN Application development
3. Floodlight modules.
4. GRE, VxLAN tunnels
5. RestFul APIs
6. Load balancer coding in Python.
7. Iperf, wireshark, ping for testing
8. HA and it’s tools in market.
9. VIP
10. Hypervisor Network types, NAT/Host-only/Bridged Network.

Any Questions?