

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

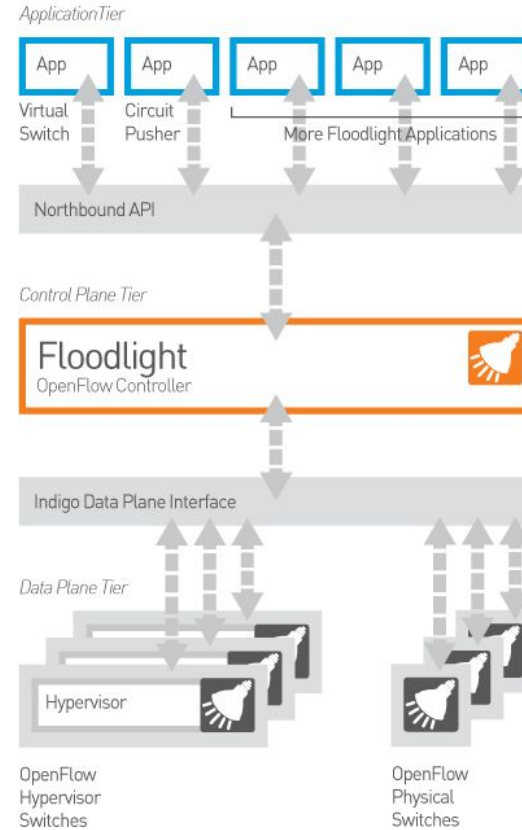


Fig 1: Architecture for Floodlight Controller

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

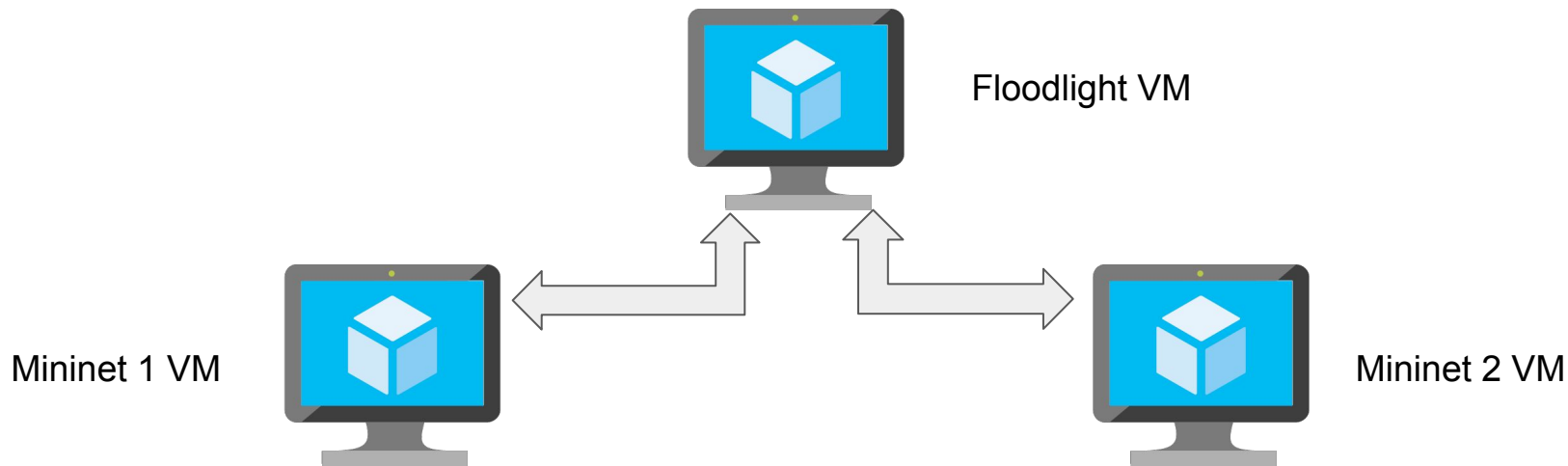
- Load Balancing refers to the dividing of the workload among multiple computing Platforms to optimize the throughput
- This helps in reducing the redundancy and increasing the reliability
- Load Balancing can occur from client side as well as from the server side

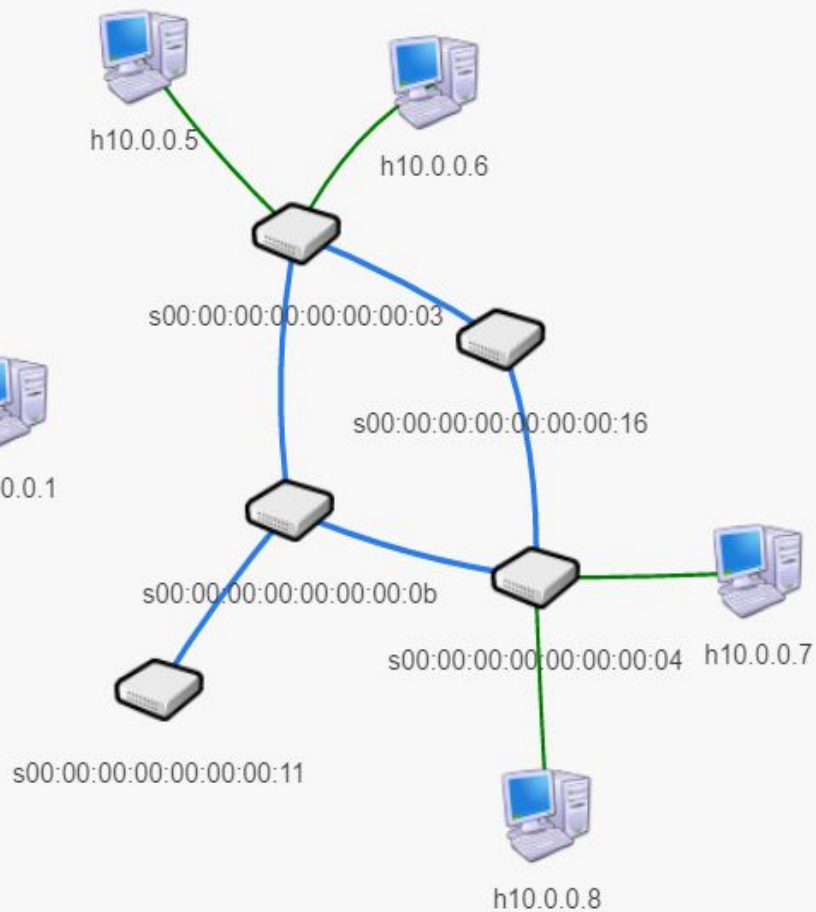
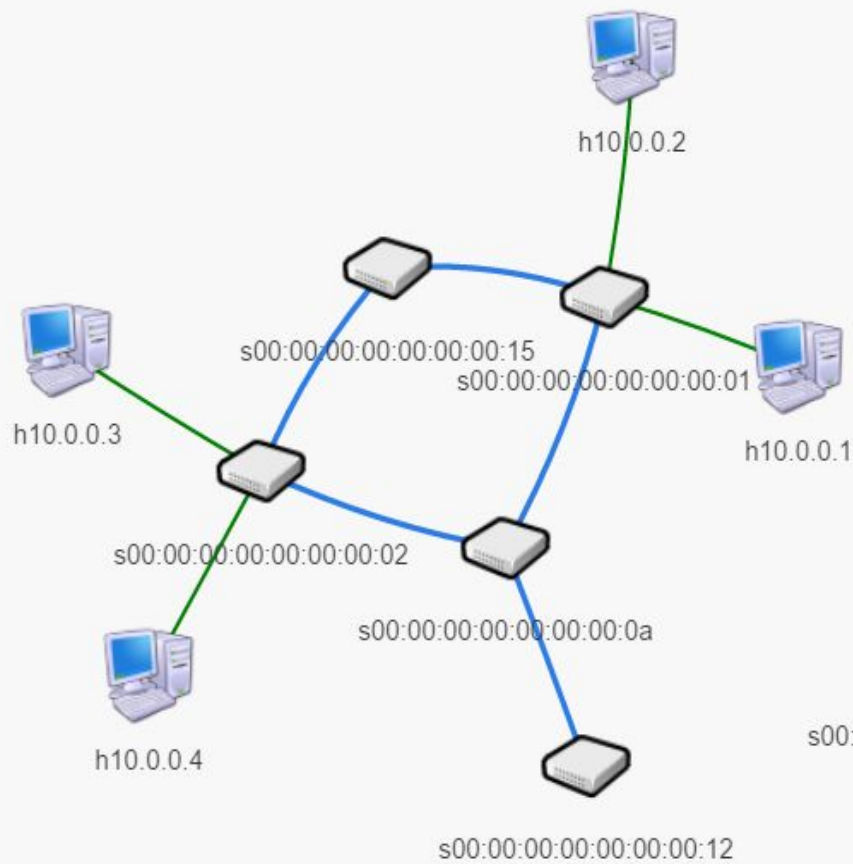
Motivation

- 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)

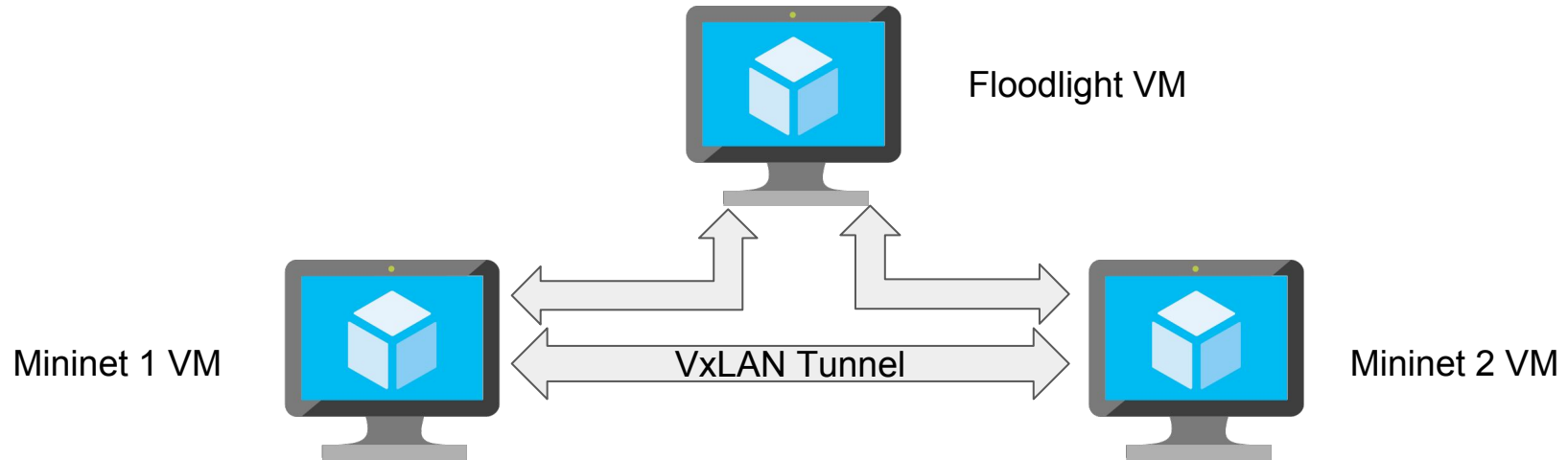
- We developed a WAN setup by connecting two topologies created in two mininet VMs with one GRE tunnel and one VxLAN tunnel
- Step 1: Topology without GRE or VxLAN tunnel

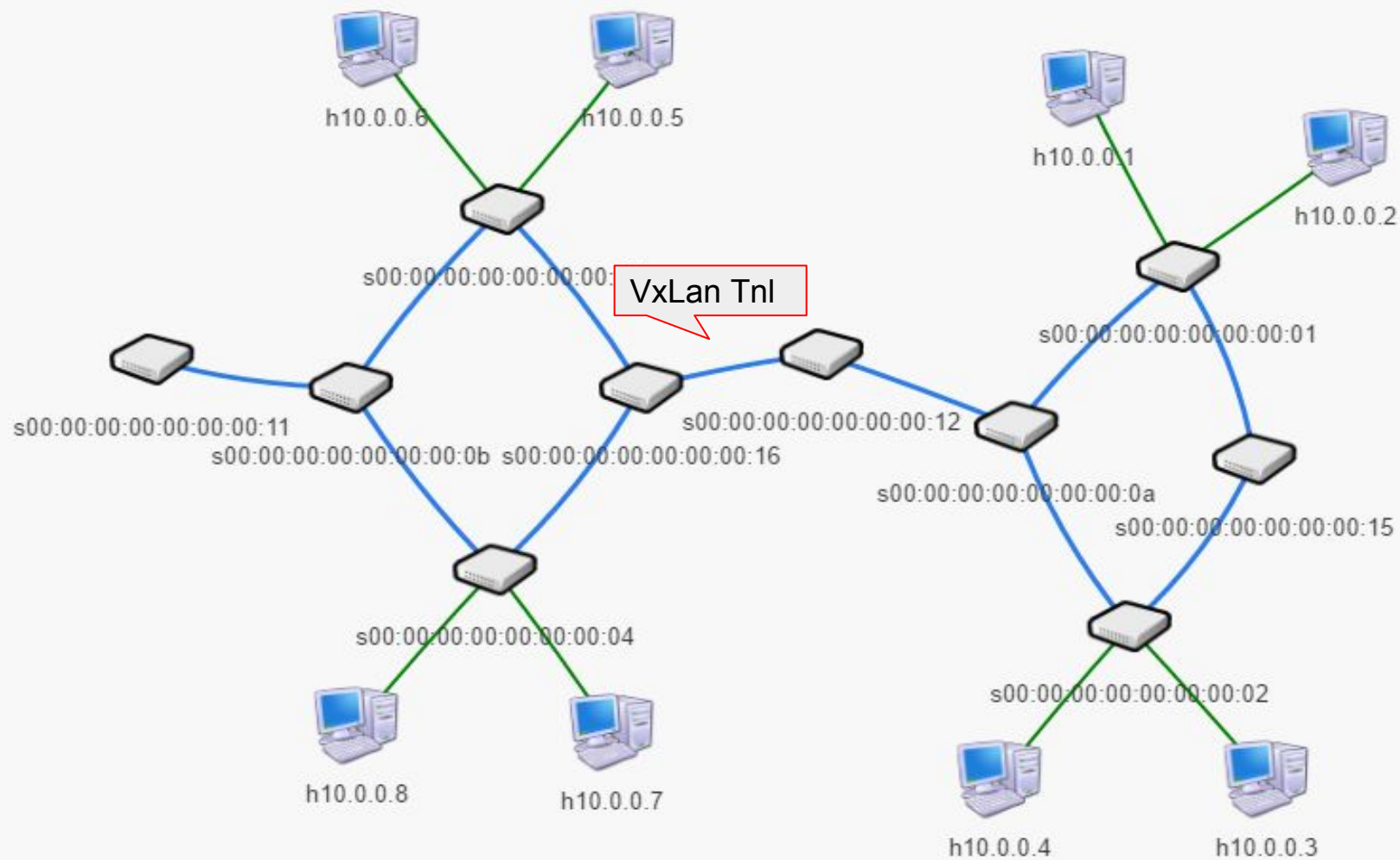




Infrastructure or Topology setup (2)

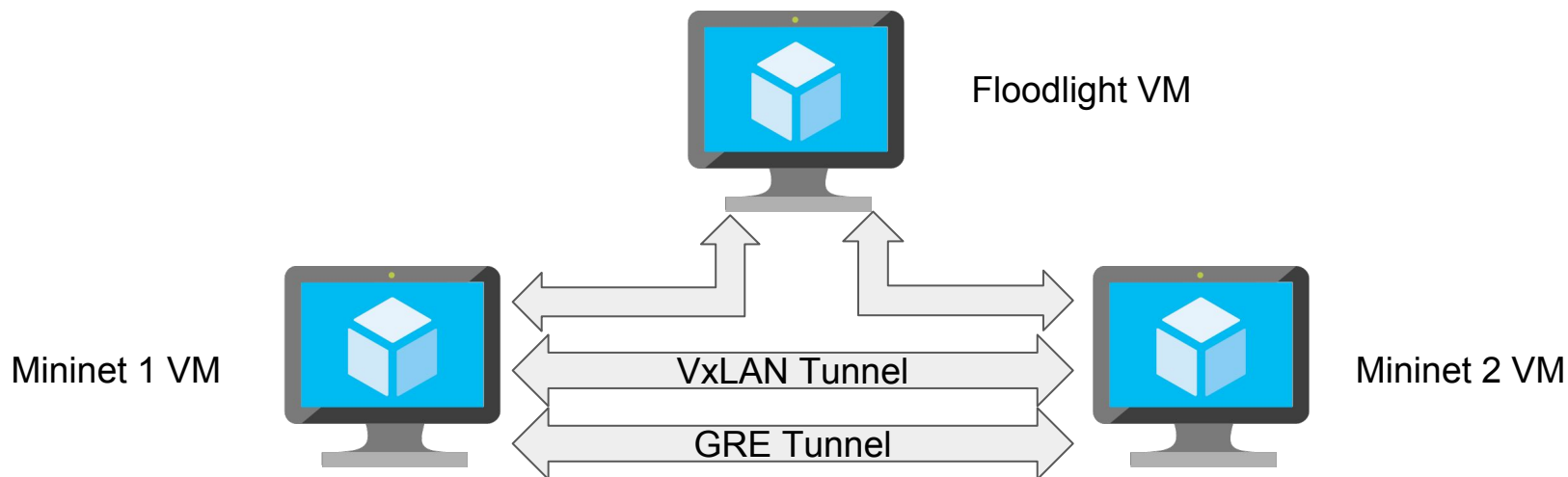
- Step 2: Create a VxLAN tunnel between S18 and S22.

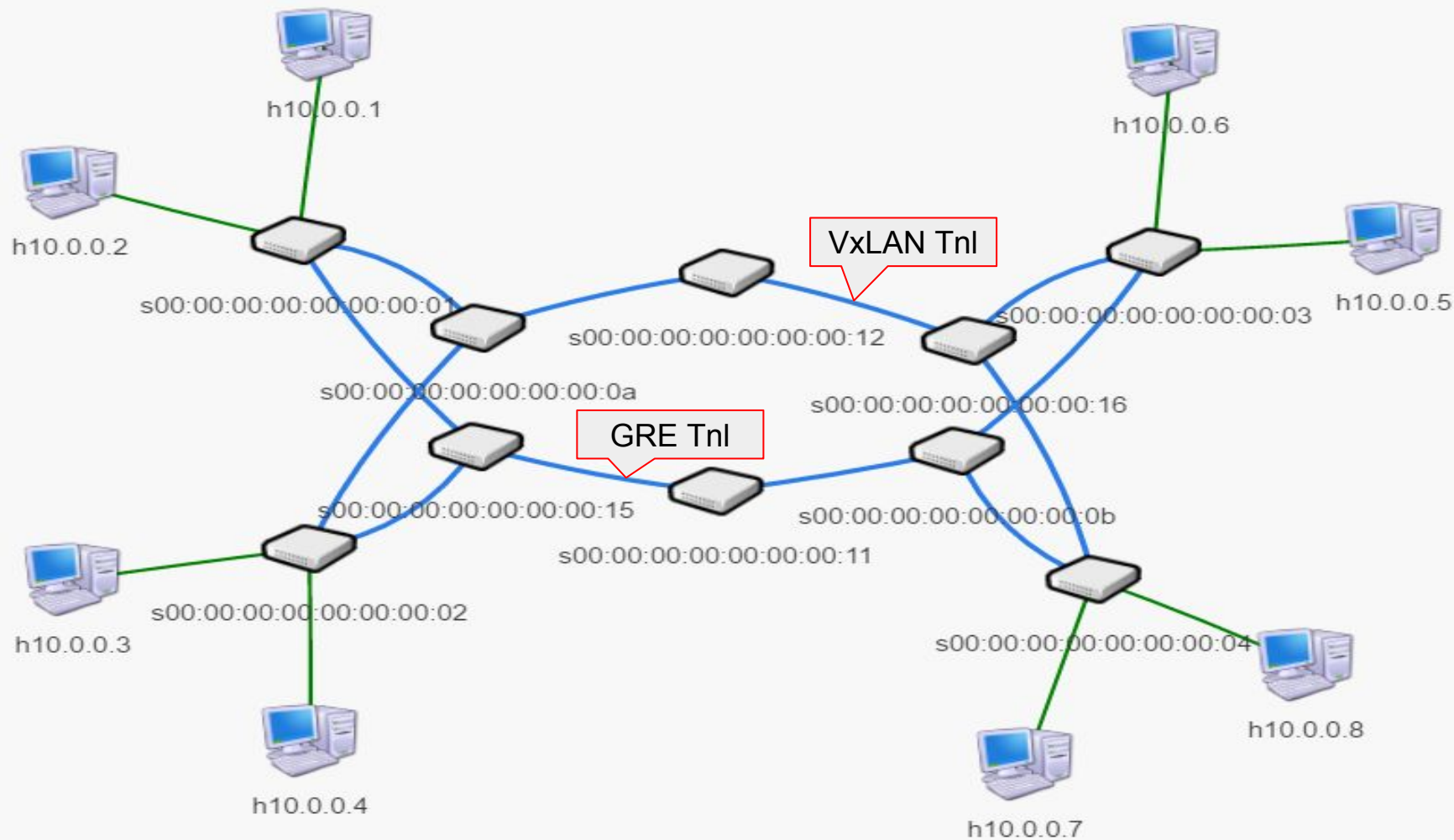




Infrastructure or Topology setup (3)

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



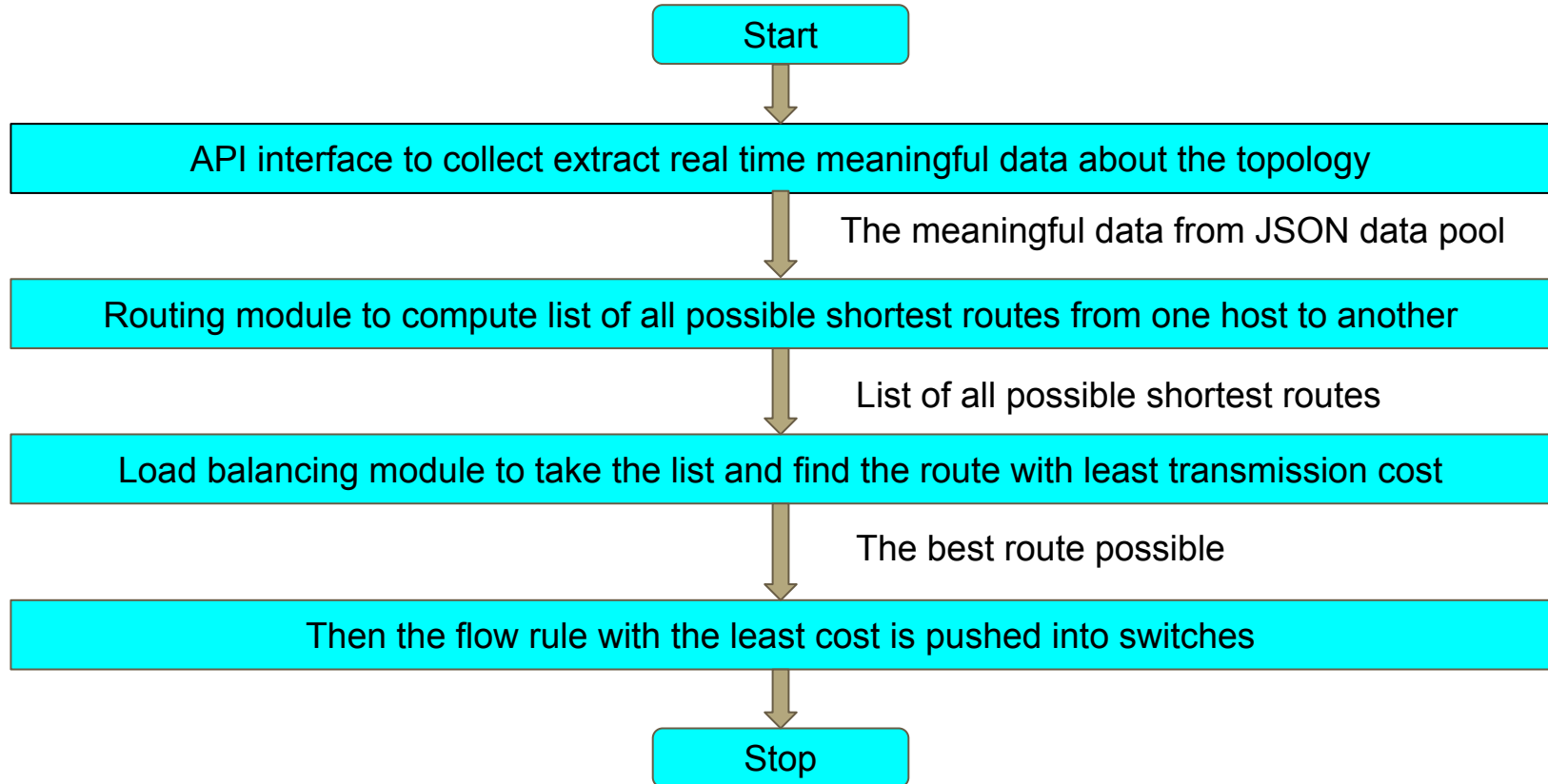


Infrastructure or Topology setup (4)

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



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,

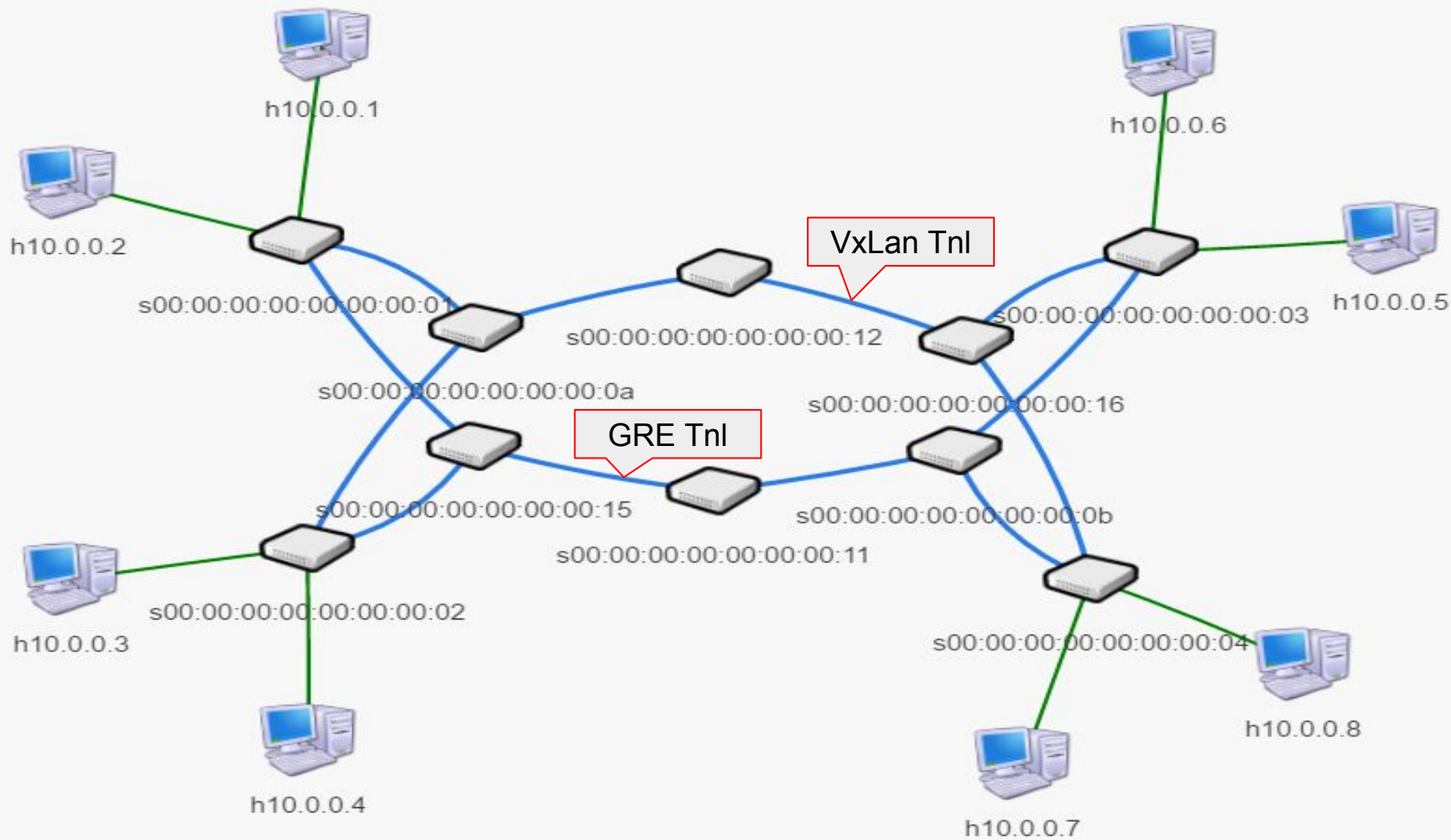
Node weight



Edge weight



End to End latency = Sum of all Switch Latencies + Sum of all Link Latencies



Implemented methods of Load Balancing (4)

```
"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/core/switch/all/flow/json

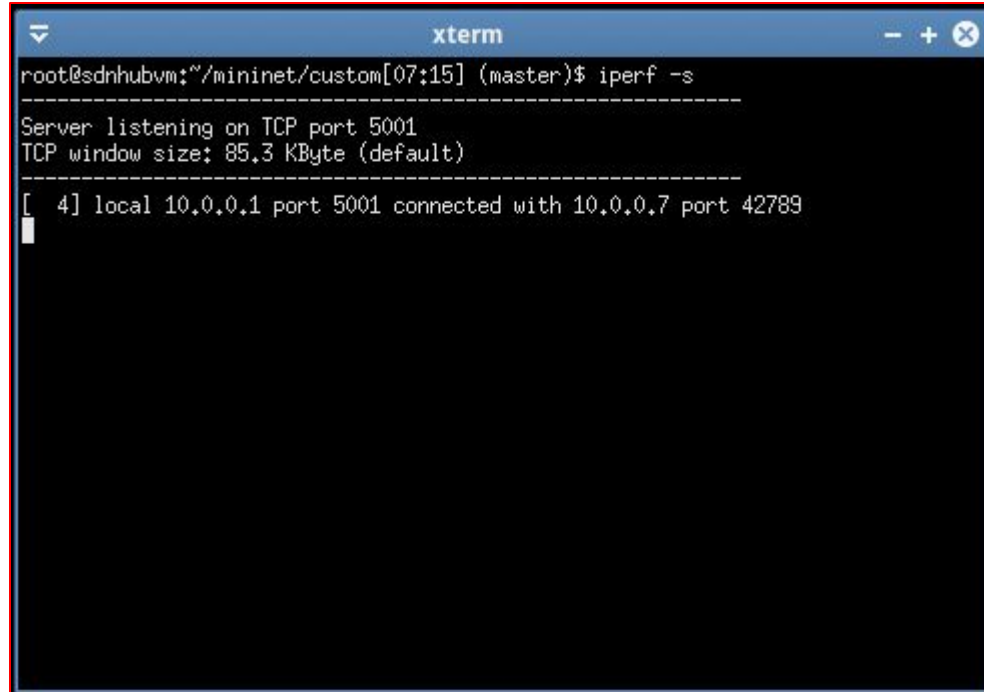
```
▼ {  
  "src-switch": "00:00:00:00:00:00:00:04",  
  "src-port": 3,  
  "dst-switch": "00:00:00:00:00:00:00:0b",  
  "dst-port": 2,  
  "type": "internal",  
  "direction": "bidirectional",  
  "latency": 15  
},
```

/wm/topology/links/json

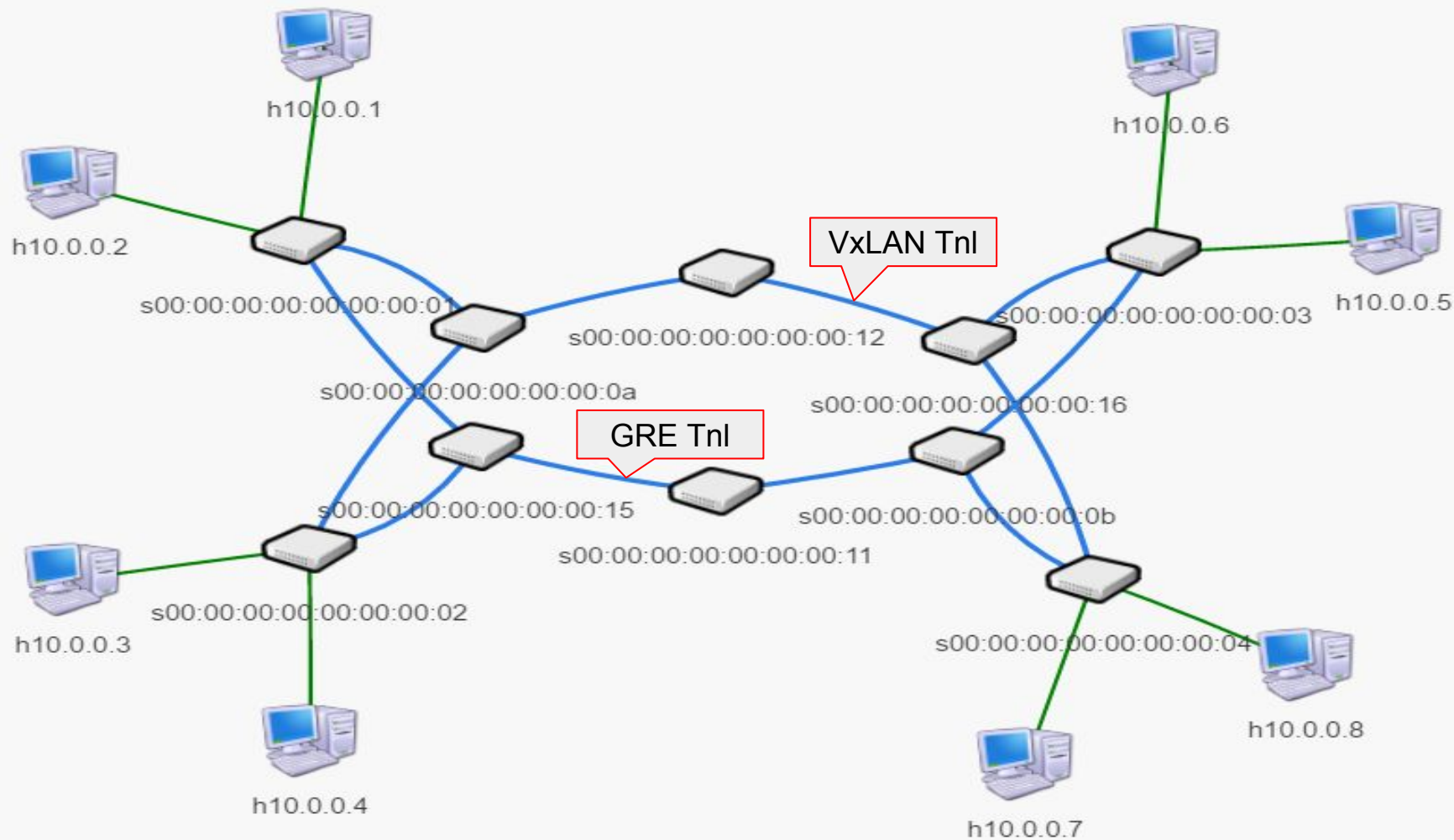


Iperf Test Results (1)

- 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

A screenshot of an xterm terminal window. The title bar is blue with the text 'xterm' and standard window controls. The terminal content shows the execution of 'iperf -s' on a host named 'sdnhubvm'. The output indicates the server is listening on TCP port 5001 with a window size of 85.3 KByte. A connection is then established from 10.0.0.1 port 5001 to 10.0.0.7 port 42789.

```
root@sdnhubvm:~/mininet/custom[07:15] (master)$ iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 4] local 10.0.0.1 port 5001 connected with 10.0.0.7 port 42789
```



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)

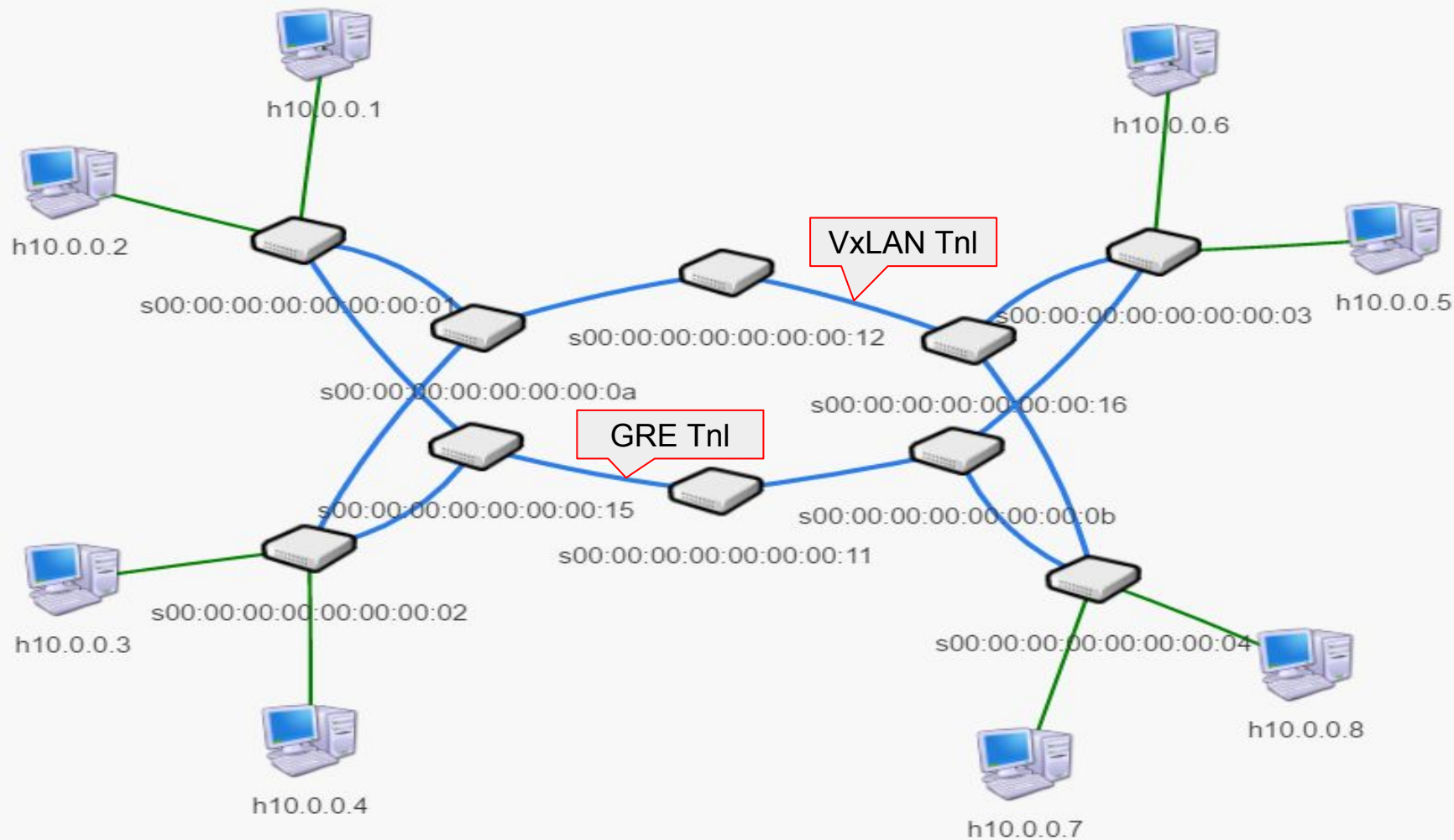
Iperf results with load balancing using NW/EW scheme:

```
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**



SDN Hub - Mininet [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Applications Menu Test1.py (~/Desktop) - gedit Capturing from s21-eth1 ...

Wireshark: Capture Interf... Terminal

 Terminal Terminal xterm

07 Dec, 03:56

File Edit View Terminal Tabs Help

SHRI GANE

```
Switch: {'10.0.0.8': '00:00:00:00:00:00:00:04', '10.0.0.5': '00:00:00:00:00:00:00:00:  
00:00:02', '10.0.0.2': '00:00:00:00:00:00:00:01'}
```

```
deviceMAC: {'10.0.0.8': '32:8c:e9:2f:35:40', '10.0.0.5': 'f6:a1:d3:f7:cf:51', '10.0.0.11'}
```

```
hostPort: {'10.0.0.6::03': '2', '10.0.0.1::01': '1', '10.0.0.7::04': '1', '10.0.0.4
```

```
The switchLinks: {'02': ['00:00:00:00:00:00:00:15', '00:00:00:00:00:00:00:0a']}
```

```
The linkPort: {'11:15': '2:3', '16:03': '2:4', '11:0b': '1:3', '01:0a': '3:
:15': '4:2', '02:0a': '3:2', '12:0a': '1:3', '16:12': '3:2', '12:16': '2:
```

```
The linkLatency: { '11::15': '11', '16::03': '5', '11::0b': '4', '01::0a': '3', '03  
'11', '16::12': '5', '12::16': '5', '16::04': '9', '0a::12': '11', '0b::03': '5',
```

```
The path: {'02::15::01': ['00:00:00:00:00:00:00:02', '00:00:00:00:00:00:00:15', '00:00:00:00:00:00:00:00'],  
finalLinkTX: {'02::15::01': 120, '02::0a::01': 1822}}
```

```
Best Shortest Path: 02::0a::01
```

64 bytes	from 10.0.0.4	icmp_seq=5	ttl=64	time=0.072	0.05
64 bytes	from 10.0.0.4	icmp_seq=6	ttl=64	time=0.057	na
64 bytes	from 10.0.0.4	icmp_seq=8	ttl=64	time=0.055	na
64 bytes	from 10.0.0.4	icmp_seq=9	ttl=64	time=0.055	na
64 bytes	from 10.0.0.4	icmp_seq=10	ttl=64	time=0.126	na
64 bytes	from 10.0.0.4	icmp_seq=11	ttl=64	time=0.107	na
64 bytes	from 10.0.0.4	icmp_seq=12	ttl=64	time=0.083	na
64 bytes	from 10.0.0.4	icmp_seq=13	ttl=64	time=0.083	na
64 bytes	from 10.0.0.4	icmp_seq=14	ttl=64	time=0.073	0.03
64 bytes	from 10.0.0.4	icmp_seq=15	ttl=64	time=0.098	na
64 bytes	from 10.0.0.4	icmp_seq=16	ttl=64	time=0.085	na
64 bytes	from 10.0.0.4	icmp_seq=17	ttl=64	time=0.085	na
64 bytes	from 10.0.0.4	icmp_seq=18	ttl=64	time=0.085	na
64 bytes	from 10.0.0.4	icmp_seq=19	ttl=64	time=0.083	na
64 bytes	from 10.0.0.4	icmp_seq=20	ttl=64	time=0.083	na
64 bytes	from 10.0.0.4	icmp_seq=21	ttl=64	time=0.160	na
64 bytes	from 10.0.0.4	icmp_seq=22	ttl=64	time=0.083	0.03
64 bytes	from 10.0.0.4	icmp_seq=23	ttl=64	time=0.123	na
64 bytes	from 10.0.0.4	icmp_seq=24	ttl=64	time=0.123	na
64 bytes	from 10.0.0.4	icmp_seq=25	ttl=64	time=0.147	na
64 bytes	from 10.0.0.4	icmp_seq=26	ttl=64	time=0.068	na
64 bytes	from 10.0.0.4	icmp_seq=27	ttl=64	time=0.075	na
64 bytes	from 10.0.0.4	icmp_seq=28	ttl=64	time=0.075	na
64 bytes	from 10.0.0.4	icmp_seq=29	ttl=64	time=0.101	na
64 bytes	from 10.0.0.4	icmp_seq=30	ttl=64	time=0.085	na
64 bytes	from 10.0.0.4	icmp_seq=31	ttl=64	time=0.085	na
64 bytes	from 10.0.0.4	icmp_seq=32	ttl=64	time=0.078	na
64 bytes	from 10.0.0.4	icmp_seq=33	ttl=64	time=0.470	na
64 bytes	from 10.0.0.4	icmp_seq=34	ttl=64	time=0.122	na
64 bytes	from 10.0.0.4	icmp_seq=35	ttl=64	time=0.083	na
64 bytes	from 10.0.0.4	icmp_seq=36	ttl=64	time=0.063	na
64 bytes	from 10.0.0.4	icmp_seq=37	ttl=64	time=0.088	na
64 bytes	from 10.0.0.4	icmp_seq=38	ttl=64	time=0.052	na
64 bytes	from 10.0.0.4	icmp_seq=39	ttl=64	time=0.052	na
64 bytes	from 10.0.0.4	icmp_seq=40	ttl=64	time=0.075	na
64 bytes	from 10.0.0.4	icmp_seq=41	ttl=64	time=0.065	na
64 bytes	from 10.0.0.4	icmp_seq=42	ttl=64	time=0.064	na
64 bytes	from 10.0.0.4	icmp_seq=43	ttl=64	time=0.064	na
64 bytes	from 10.0.0.4	icmp_seq=44	ttl=64	time=0.080	na
64 bytes	from 10.0.0.4	icmp_seq=45	ttl=64	time=0.496	na
64 bytes	from 10.0.0.4	icmp_seq=46	ttl=64	time=0.161	na
64 bytes	from 10.0.0.4	icmp_seq=47	ttl=64	time=0.161	na
64 bytes	from 10.0.0.4	icmp_seq=48	ttl=64	time=0.061	na
64 bytes	from 10.0.0.4	icmp_seq=49	ttl=64	time=0.053	na
64 bytes	from 10.0.0.4	icmp_seq=50	ttl=64	time=0.053	na
64 bytes	from 10.0.0.4	icmp_seq=51	ttl=64	time=0.092	na
64 bytes	from 10.0.0.4	icmp_seq=52	ttl=64	time=0.086	na
64 bytes	from 10.0.0.4	icmp_seq=53	ttl=64	time=0.082	na
64 bytes	from 10.0.0.4	icmp_seq=54	ttl=64	time=0.082	na
64 bytes	from 10.0.0.4	icmp_seq=55	ttl=64	time=0.095	na
64 bytes	from 10.0.0.4	icmp_seq=56	ttl=64	time=0.105	na
64 bytes	from 10.0.0.4	icmp_seq=57	ttl=64	time=0.165	na
64 bytes	from 10.0.0.4	icmp_seq=58	ttl=64	time=0.165	na
64 bytes	from 10.0.0.4	icmp_seq=59	ttl=64	time=0.074	na
64 bytes	from 10.0.0.4	icmp_seq=60	ttl=64	time=0.085	na
64 bytes	from 10.0.0.4	icmp_seq=61	ttl=64	time=0.085	na
64 bytes	from 10.0.0.4	icmp_seq=62	ttl=64	time=0.102	na
64 bytes	from 10.0.0.4	icmp_seq=63	ttl=64	time=0.085	na
64 bytes	from 10.0.0.4	icmp_seq=64	ttl=64	time=0.126	na
64 bytes	from 10.0.0.4	icmp_seq=65	ttl=64	time=0.126	na
64 bytes	from 10.0.0.4	icmp_seq=66	ttl=64	time=0.072	0.02
64 bytes	from 10.0.0.4	icmp_seq=67	ttl=64	time=0.086	na
64 bytes	from 10.0.0.4	icmp_seq=68	ttl=64	time=0	

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

© 2014 Pearson Education, Inc. or its affiliate(s). All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or by any information storage or retrieval system, without prior written permission from the publisher.

Filter: <input type="text"/> Expression... Clear Apply Save						
No.	Time	Source	Destination	Protocol	Length	Info
5	15.012114000	0e:c1:5c:56:14:d0	LLDP Multicast	LLDP	75	TTL = 120
6	15.016165000	8a:cd:63:7b:af:61	LLDP Multicast	LLDP	75	TTL = 120
7	15.016480000	0a:d5:96:d2:0e:cd	LLDP Multicast	LLDP	75	TTL = 120
8	15.017107000	ee:41:18:09:7d:bc	LLDP Multicast	LLDP	75	TTL = 120
9	30.022871000	0e:c1:5c:56:14:d0	LLDP Multicast	LLDP	75	TTL = 120
10	30.026640000	8a:cd:63:7b:af:61	LLDP Multicast	LLDP	75	TTL = 120
11	30.026817000	0a:d5:96:d2:0e:cd	LLDP Multicast	LLDP	75	TTL = 120
12	30.027485000	ee:41:18:09:7d:bc	LLDP Multicast	LLDP	75	TTL = 120

```
▶ Frame 1: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface 0
```

```
▶ Ethernet II, Src: 0e:c1:5c:56:14:d0 (0e:c1:5c:56:14:d0), Dst: LLDP Multicast (01:80:c2:00:00:0e)
```

- ▶ Link Layer Discovery Protocol

The screenshot shows the 'Wireshark: Capture Interfaces' window. It lists two interfaces: 's18-eth1' and 's21-eth2', both of which are checked. The interface 's18-eth1' has a MAC address of 'fe80::9468:d3ff:te71:891' and a speed of '4 2'. The interface 's21-eth2' has a MAC address of 'fe80::7b51:bf:ff:bc3:d19' and a speed of '4 2'. At the bottom, there are buttons for 'Help', 'Start', 'Stop', 'Options', and 'Close'.

0000	01 80	c2 00	00 0e	0e c1	5c 56	14 d0	88 cc	02 07	IV.....
0010	04 00	00 00	00 01	04 03	02 00	04 06	02 00	78
0020	fe 0c	00 26	e1 00	00 00	00 00	00 00	00 01	18 08	&.....
0030	02 04	fa 2b	19 43	7a b6	e6 01	01 fe	0c 00	26 e1	+CZ.....&
0040	01 00	00 01	58 d9	24 45	53 00				X-SE S...

s21-eth1, s21-eth2: <live capture i... Profile: Default


Right Ctrl

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?

