CSC-792-038 HomeWork 2

Solution 1:

Reading datasheets:

For 500 Mbps throughput in CSR 1000v, we would need at least **1 vCPU and a memory of 4GB.**
For 1000 Mbps throughput in CSR 1000v, we would need at least **2 vCPUs and a memory of 4GB**.
For 5000 Mbps throughput in CSR 1000v, we would need at least **8 vCPUs and a memory of 4GB**.

(The above values are for the AX case where the CSR 1000v instance supports all features.)

Networking features:

1. CSR 1000v provides routing features using BGP, OSPF, EIGRP, Policy-Based Routing (PBR), IPv6, VRF-Lite, Multicast, LISP, GRE, and Connectionless Network Services (CLNS)
2. Network Function Virtualization is also supported using vBNG, vISG and vRR.
3. Application visibility, performance monitoring and control (QoS, AVC) are also provided by CSR 1000v

Security features:

1. Virtual Private Networks can be implemented using IPsec VPN, DMVPN, Easy VPN, FlexVPN, and GetVPN
2. Zone Based FireWall (ZBFW) provides the firewall support.
3. Access control lists can be configured using AAA, RADIUS, and TACACS+

Management Interface features:

1. Management interfaces can be used with IOS XE CLI, SSH, Telnet, Cisco Prime Infrastructure, Cisco Prime Network Services Controller, and OpenStack Configdrive
2. REST APIs are supported.(License installation and Smart Licensing, interfaces and subinterfaces, routing protocols, IPsec and Easy VPN, firewall, ACL, NAT, configuration import and export, reports (CPU usage, interface statistics, routing table, VPN and firewall sessions, etc.), VRF, Network Time Protocol (NTP), DNS, DHCP, SNMP, TACACS, LISP, VXLAN, and HSRP)
3. Cisco vManage in the IOS SD-WAN Software can be used to manage the CSR 1000v instance from cloud.

Cost:

- A single CSR 1000V instance providing AX Pkg max performance for 1000Mbps throughput will cost **$3723 per year.** If the EC2 instance needs to be used it would cost $876 per year too.
- So the total cost per CSR 1000v instance (with EC2) would be **$4599**. (For two instances, it would be **$9198.** Without EC2, the cost would be $7446).

Note: There are multiple options of CSR 1000v instances to choose from. **The one above** is chosen as it is not obsolete and it provides all features.

Solution 2:

Creating Virtual Machines:

1. IP address and MAC address of:

```
>
 virsh domifaddr ssujalVM1
>
```

a. VM Network Interface Card:

```
>
 Name       MAC address          Protocol     Address
-------------------------------------------------------------------------------
 vnet0      52:54:00:88:d7:a8    ipv4         192.168.123.17/24
>
```

b. Bridge L2 & SVI:

```
>
MAC: 52:54:00:6e:fc:b4
IP: 192.168.123.1/24
>
```

c. Hypervisor NIC used to go to the internet:

First we find out interface it uses:

```
>
ece792@t11_vm4:~$ ip route get 8.8.8.8
8.8.8.8 via 192.168.126.1 dev ens6 src 192.168.126.110 uid 1000
    cache
>
```

ens6 :
MAC: 52:54:00:52:8f:b1
IP: 192.168.126.110/24

Note: The NIC used to reach the internet changes. The case for ens6 is shown above.

2. Packet Capture:

It is expected that the Destination IP will only be the common field in both the captures.

VM's output interface	Hypervisor's output Interface
Source MAC: 52:54:00:88:d7:a8	Source MAC: 52:54:00:52:8f:b1
Destination MAC: 52:54:00:6e:fc:b4	Destination MAC: 52:54:00:6a:d9:14
Source IP: 192.168.123.17	Source IP: 192.168.126.110
Destination IP: 172.217.13.78	Destination IP: 172.217.13.78

As expected, only the Destination IP is common in both the captures. **The tuples are different**. This is because the hypervisor's outgoing interface is at least one hop ahead of that of the virtual bridge. (i.e The starting point of the ping is different) . Hence, the source IP & MAC values are obviously different. The VM is using virbr0 as the next hop. virbr0 will in turn use the hypervisor's outgoing interfaces for further hops. So the VM and the hypervisor's captures will have different Source IP & MAC, Destination MAC. Only the Destination IP of google.com will remain the same between them.

Solution 3:

1. **To add a network in BRIDGE mode:**

First, we need to create a new network in the same location i.e. "/etc/libvirt/qemu/network"
You can start by copying the default network:

```
>
cd /etc/libvirt/qemu/network
cp default.xml net2.xml
>
```

Now, we need to edit the file "net1.xml" as follows:

```
>
<network>
    <name>net2</name>
    <forward mode='bridge'/>
    <bridge name='sw2'/>
</network>
>
```

After creating the file, we need to define it via virsh as the following:

```
>
```

```
sudo virsh net-define net2.xml
sudo virsh net-define net2
>
```

2. **To add an interface to your VM to connect to net2**

For this you need to edit the configuration xml file for the respective VM (ssujalVM1 in this case). Add the interface tags as shown in the snippet below from our lab:

Adding 3 interfaces:

```
>
    <interface type='network'>
      <mac address='52:54:00:88:d7:a8'/>
      <source network='default'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>

    <interface type='network'>
      <mac address='92:6f:20:59:7b:dd'/>
      <source network='net1'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x09' function='0x0'/>
    </interface>

    <interface type='network'>
      <mac address='0a:0f:dd:4c:cb:4e'/>
      <source network='net2'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x0a' function='0x0'/>
    </interface>
>
```

3. **To clone your VM**

Command:

```
>
 sudo virt-clone --original ssujalVM1 --name ssujalVM2 --auto-clone
>
```

Eg:

```
>
ece792@t11_vm4:/etc/libvirt/qemu/networks$ sudo virt-clone --original ssujalVM1
--name ssujalVM2 --auto-clone
Allocating 'VM1-clone-1.img'                                | 2.0 GB  00:00:03

Clone 'ssujalVM2' created successfully.
>
```

To verify, you should try pinging one VM from the other VM and they should be connected.

4. **List MAC and IP addresses of all interfaces of each VM.**

Command:

```
>
 virsh domiflist <VM name>
>
```

Eg:

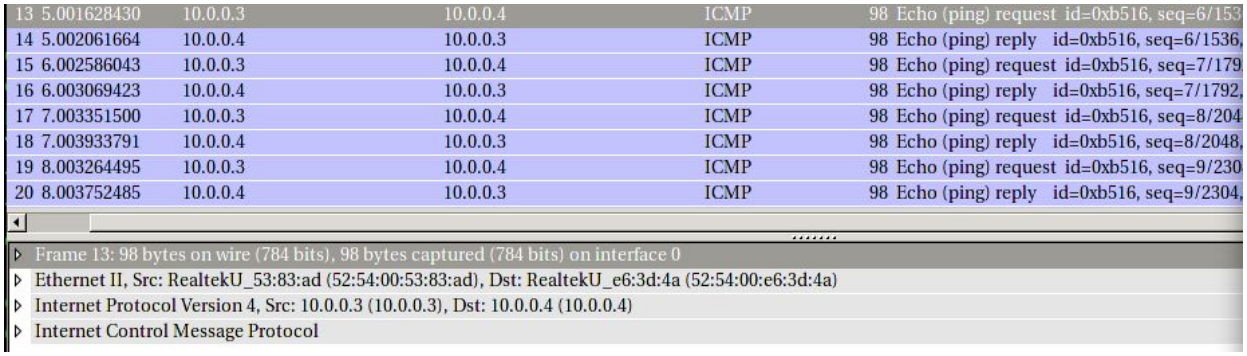
```
>
ece792@t11_vm4:/etc/libvirt/qemu$ virsh domiflist ssujalVM1
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet0      network    default    virtio      52:54:00:88:d7:a8
vnet1      bridge     net1       virtio      92:6f:20:59:7b:dd
vnet2      bridge     net2       virtio      0a:0f:dd:4c:cb:4e
vnet10     bridge     net1       virtio      52:54:00:53:83:ad

ece792@t11_vm4:/etc/libvirt/qemu$ virsh domiflist ssujalVM2
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet3      network    default    virtio      52:54:00:15:2e:bc
vnet4      bridge     net1       virtio      52:54:00:f8:a3:03
vnet5      bridge     net2       virtio      52:54:00:19:cf:d3
vnet11     bridge     net1       virtio      52:54:00:e6:3d:4a
>
```

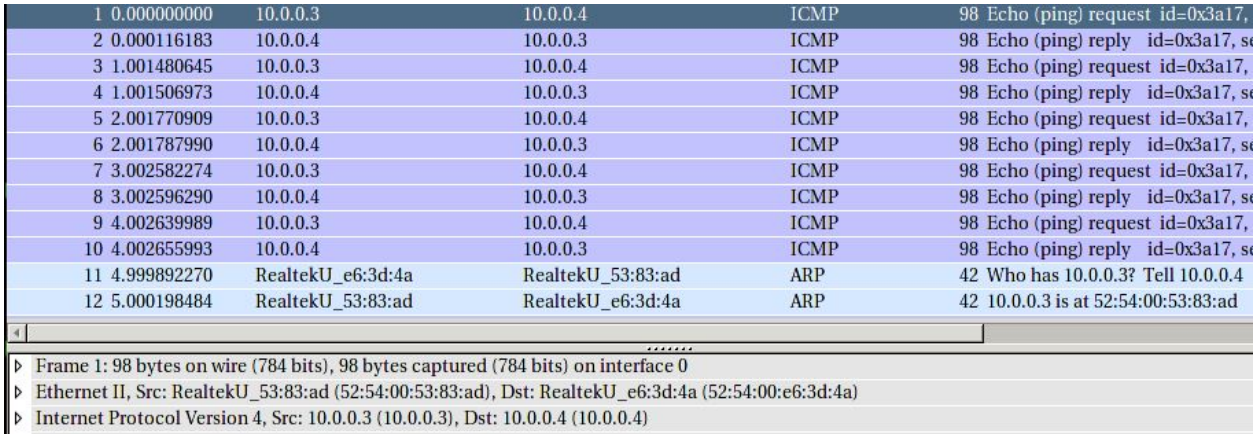
5.

ssujalVM1's (request)	ssujalVM2's (request)
Source MAC: 52:54:00:53:83:ad	Source MAC: 52:54:00:53:83:ad
Destination MAC: 52:54:00:e6:3d:4a	Destination MAC: 52:54:00:e6:3d:4a
Source IP: 10.0.0.3	Source IP: 10.0.0.3
Destination IP: 10.0.0.4	Destination IP: 10.0.0.4

Request packet monitored at VM1 :



Request packet monitored at VM2:



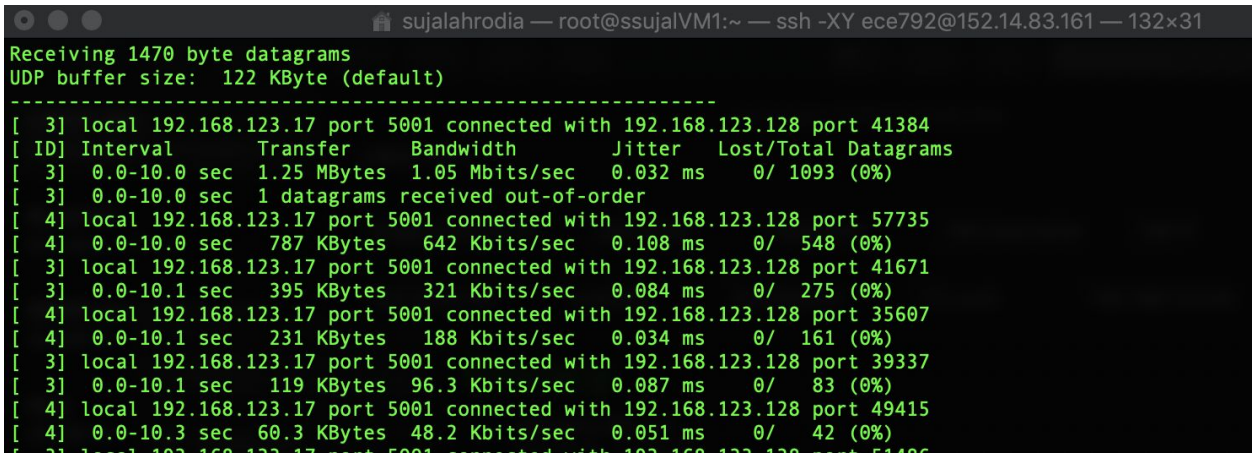
It can be observed that the tuple is the same at both ends. This makes sense because we are monitoring the same flow from two different points (i.e. from VM1 and VM2)

6.

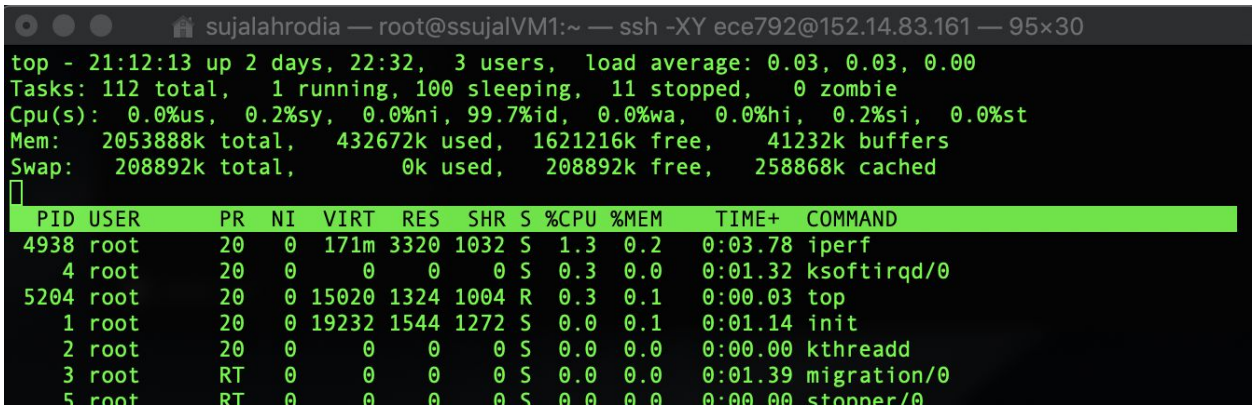
IPERF

UDP Traffic:

Without any packet size given, the maximum throughput measured is **1.05 Mbits/sec.** This tends to decrease when the packet size is increased above the default packet size. For any packet size <= default size, the throughput is constant at 1.05Mbps.



The bottleneck resource is identified to be the CPU. Iperf traffic hardly involves I/O utilization (**using iotop)** and when we compare between CPU and Memory (using **top),** it is found that CPU utilization is more significant than memory.



TCP TRAFFIC:

Without any packet size given, the maximum throughput measured is **8.9 Gbits/sec.** This tends to decrease when the packet size is increased above the default packet size. For any packet size <= default size, the throughput is constant at 8.9Gbps. The packet sizes affect the bandwidth for the respective traffic in the following way :

- UDP : Packet size is inversely proportional to bandwidth
- TCP : Packet size is directly proportional to bandwidth

The packet size is tuned to observe changes in bandwidth (from HW1)

```
[root@ssujalVM1 ~]# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.123.17 port 5001 connected with 192.168.123.128 port 38910
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  10.4 GBytes  8.90 Gbits/sec
[  5] local 192.168.123.17 port 5001 connected with 192.168.123.128 port 38912
[  5]  0.0-10.0 sec   244 MBytes   205 Mbits/sec
[  4] local 192.168.123.17 port 5001 connected with 192.168.123.128 port 38914
[  4]  0.0-10.0 sec   444 MBytes   372 Mbits/sec
[  5] local 192.168.123.17 port 5001 connected with 192.168.123.128 port 38916
[  5]  0.0-10.0 sec  1.08 GBytes   927 Mbits/sec
[  4] local 192.168.123.17 port 5001 connected with 192.168.123.128 port 38918
[  4]  0.0-10.0 sec  1.48 GBytes  1.27 Gbits/sec
[  5] local 192.168.123.17 port 5001 connected with 192.168.123.128 port 38920
[  5]  0.0-10.0 sec  9.68 GBytes  8.30 Gbits/sec
```

PACKET SIZE: 1200

```
sujalahrodia — root@ssujalVM1:~ — ssh -XY ece792@152.14.83.161 — 95×30
top - 21:26:42 up 2 days, 22:46,  3 users,  load average: 0.08, 0.08, 0.02
Tasks: 106 total,   1 running, 100 sleeping,   5 stopped,   0 zombie
Cpu(s):  0.2%us, 15.8%sy,  0.0%ni, 69.4%id,  0.0%wa,  1.1%hi,  5.8%si,  7.7%st
Mem:   2053888k total,   431184k used,  1622704k free,    41240k buffers
Swap:   208892k total,        0k used,   208892k free,   258880k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 5275 root      20   0  171m 3420 1036 S 71.2  0.2   1:00.51 iperf
    4 root      20   0     0    0    0 S  0.3  0.0   0:01.48 ksoftirqd/0
   11 root      20   0     0    0    0 S  0.3  0.0   4:30.27 events/0
 5204 root      20   0 15020 1324 1004 R  0.3  0.1   0:01.24 top
    1 root      20   0 19232 1544 1272 S  0.0  0.1   0:01.14 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:01.40 migration/0
```

PACKET SIZE: 128000

```
sujalahrodia — root@ssujalVM1:~ — ssh -XY ece792@152.14.83.161 — 95×30
top - 21:27:03 up 2 days, 22:46,  3 users,  load average: 0.14, 0.09, 0.02
Tasks: 106 total,   1 running, 100 sleeping,   5 stopped,   0 zombie
Cpu(s):  0.9%us, 32.9%sy,  0.0%ni, 53.1%id,  0.0%wa,  0.0%hi, 10.9%si,  2.2%st
Mem:   2053888k total,   431496k used,  1622392k free,    41240k buffers
Swap:   208892k total,        0k used,   208892k free,   258880k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 5275 root      20   0  171m 3420 1036 S 95.1  0.2   1:08.31 iperf
 5204 root      20   0 15020 1324 1004 R  0.7  0.1   0:01.29 top
   11 root      20   0     0    0    0 S  0.3  0.0   4:30.30 events/0
    1 root      20   0 19232 1544 1272 S  0.0  0.1   0:01.14 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:01.40 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:01.50 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 stopper/0
```

It is observed that the CPU utilization is increased much as well when the packet size is increased. With maximum throughput (8.9Gbps), CPU utilization is almost 100%. To achieve maximum throughput, we had to increase the packet size significantly (i.e. from 1200 to 128000 bytes). Although it did boost up

the CPU utilization at the server end (from 71% to 95%), it wasn't proportional to the change in the packet size.

Solution 4:

For Ansible:

OVS L3 bridge, create network and add interfaces :

```
ece792@t11_vm4:~$ virsh domifaddr arangar2VM3
 Name       MAC address          Protocol     Address
-------------------------------------------------------------------------------

ece792@t11_vm4:~$ sudo ovs-vsctl show
73a989b9-1635-476c-a3fb-c17b9dfa400b
    Bridge "arangar2OVS"
        Port "vnet6"
            Interface "vnet6"
        Port "arangar2OVS"
            Interface "arangar2OVS"
                type: internal
        Port "vnet7"
            Interface "vnet7"
    Bridge "sw2"
        Port "sw2"
            Interface "sw2"
                type: internal
        Port "vnet5"
            Interface "vnet5"
        Port "vnet2"
            Interface "vnet2"
    ovs_version: "2.9.2"
ece792@t11_vm4:~$
ece792@t11_vm4:~$ virsh domiflist arangar2VM3
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet6      bridge     arangar2-netL3 virtio      52:54:00:bd:64:4a

ece792@t11_vm4:~$ virsh domiflist arangar2VM4
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet7      bridge     arangar2-netL3 virtio      52:54:00:ab:2c:74
```

Solution 6:

1. **Two VMs connected to same bridge:**

```
[root@ssujalVM1 ~]# ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
From 10.0.0.3 icmp_seq=2 Destination Host Unreachable
From 10.0.0.3 icmp_seq=3 Destination Host Unreachable
From 10.0.0.3 icmp_seq=4 Destination Host Unreachable
^Z
[1]+  Stopped                 ping 10.0.0.4
```

SETUP:

The setup has two VMs (VM1 and VM2). VM2's MAC has been changed to that of VM1. It can be seen that VM1 and VM2 have different IPs, but same MAC.

```
[root@ssujalVM1 ~]# ip addr show eth3
5: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 52:54:00:53:83:ad brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.3/24 scope global eth3
    inet6 fe80::5054:ff:fe53:83ad/64 scope link
       valid_lft forever preferred_lft forever
[root@ssujalVM1 ~]#
[root@ssujalVM1 ~]#
```

```
[root@ssujalVM2 ~]# sudo ifconfig eth6 hw ether 52:54:00:53:83:ad
[root@ssujalVM2 ~]# sudo ifconfig eth6 up
[root@ssujalVM2 ~]#
[root@ssujalVM2 ~]#
[root@ssujalVM2 ~]# ip addr show eth6
5: eth6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 52:54:00:53:83:ad brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.4/24 scope global eth6
    inet6 fe80::5054:ff:fe53:83ad/64 scope link tentative dadfailed
       valid_lft forever preferred_lft forever
```

We ping from VM1 to VM2 :

Observation at destination:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	RealtekU_53:83:ad	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
2	0.000018426	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad
3	1.000008243	RealtekU_53:83:ad	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
4	1.000021813	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad
5	1.999961828	RealtekU_53:83:ad	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
6	1.999973204	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad
7	4.000903943	RealtekU_53:83:ad	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
8	4.000916786	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad
9	5.000919576	RealtekU_53:83:ad	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
10	5.000933395	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad
11	6.001270922	RealtekU_53:83:ad	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
12	6.001288976	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad

▸ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▸ Ethernet II, Src: RealtekU_53:83:ad (52:54:00:53:83:ad), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Address Resolution Protocol (request)

ARP is received at destination, from source and it sends out that it has the destination IP. It replies with its own MAC(which is the same as SRC MAC).

At interface between bridge and VM2:

No.	Time	Source	Destination	Protocol	Length	Info
6	2.000283963	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad
7	4.001060234	RealtekU_53:83:ad	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
8	4.001354495	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad
9	5.000980294	RealtekU_53:83:ad	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
10	5.001310344	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad
11	6.001192108	RealtekU_53:83:ad	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
12	6.001581318	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad
13	8.002308598	RealtekU_53:83:ad	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
14	8.002651245	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad
15	9.002049395	RealtekU_53:83:ad	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
16	9.002397907	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad
17	10.002056968	RealtekU_53:83:ad	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
18	10.002361745	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad
19	12.002906510	RealtekU_53:83:ad	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
20	12.003137659	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad
21	13.003010349	RealtekU_53:83:ad	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
22	13.003370212	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad
23	14.002987330	RealtekU_53:83:ad	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.3
24	14.003340925	RealtekU_53:83:ad	RealtekU_53:83:ad	ARP	42	10.0.0.4 is at 52:54:00:53:83:ad

▸ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▸ Ethernet II, Src: RealtekU_53:83:ad (52:54:00:53:83:ad), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Address Resolution Protocol (request)

At this interface, the bridge forwards the ARP request from SRC to DEST and also receives the ARP reply by VM2.

At interface between VM1 and SW1:

5	6.001970186	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
6	7.001982089	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
7	8.001979328	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
8	10.003004512	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
9	11.002983213	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
10	12.002922752	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
11	14.004002675	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
12	15.004033152	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
13	16.003984168	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
14	18.004933022	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
15	19.004931459	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
16	20.004960006	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
17	22.006301822	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
18	23.005960360	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
19	24.005955231	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
20	26.007187841	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
21	27.006963012	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
22	28.007047646	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
23	30.008030027	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3

```
▸ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▾ Ethernet II, Src: RealtekU_53:83:ad (52:54:00:53:83:ad), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Source: RealtekU_53:83:ad (52:54:00:53:83:ad)
    Type: ARP (0x0806)
▸ Address Resolution Protocol (request)
```

Bridge now learns about the new MAC of VM2 interface and updates it. It also gets the ARP requests from VM1 where it learns about the same MAC belonging to VM1. So the MAC address entry keeps flipping inside.

So VM1 will not be able to know about the destination IP at all, but it sends out the requests constantly like below :

1	0.000000000	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
2	1.000006496	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
3	3.001012806	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
4	4.001037202	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
5	5.001045282	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
6	7.002016276	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
7	8.002009829	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
8	9.002004747	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
9	11.003010118	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
10	12.003004719	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
11	13.003025815	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3
12	15.004015261	RealtekU_53:83:ad	Broadcast	ARP	42 Who has 10.0.0.4? Tell 10.0.0.3

```
› Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
› Ethernet II, Src: RealtekU_53:83:ad (52:54:00:53:83:ad), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
› Address Resolution Protocol (request)
```

Same IP:

SETUP:

```
[root@ssujalVM1 ~]# ip addr show eth3
5: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 52:54:00:53:83:ad brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.3/24 scope global eth3
    inet6 fe80::5054:ff:fe53:83ad/64 scope link
       valid_lft forever preferred_lft forever
[root@ssujalVM1 ~]#
[root@ssujalVM1 ~]#
[root@ssujalVM1 ~]# _
```

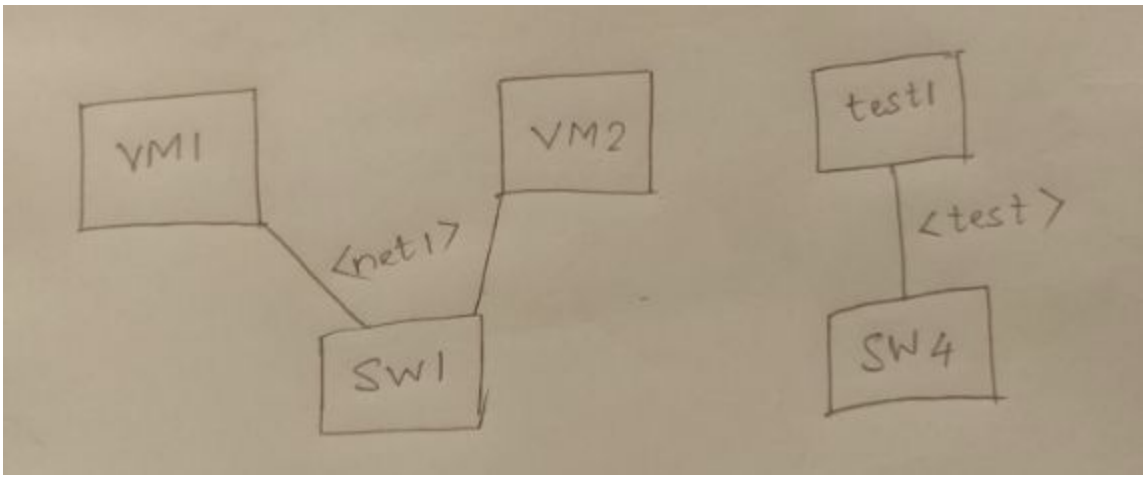
```
    link/ether 52:54:00:e6:3d:4a brd ff:ff:ff:ff:ff:ff
[root@ssujalVM2 ~]#
[root@ssujalVM2 ~]# ip addr show eth6
5: eth6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 52:54:00:e6:3d:4a brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.3/24 brd 10.0.0.255 scope global eth6
    inet6 fe80::5054:ff:fee6:3d4a/64 scope link
       valid_lft forever preferred_lft forever
[root@ssujalVM2 ~]#
```

No observation in any interface at source, destination and bridge interfaces. There can be no wireshark captures seen. In fact, the **source pings its own loopback** because it will first check if there is any entry for Dest IP inside itself. As the dest IP is the same, it pings itself and packet does not go out.

```
[root@ssujalVM1 ~]# tcpdump -i eth3
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth3, link-type EN10MB (Ethernet), capture size 65535 bytes
^Z
[13]+  Stopped                 tcpdump -i eth3
[root@ssujalVM1 ~]# tcpdump -i lo
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 65535 bytes
15:58:08.209272 IP 10.0.0.3 > 10.0.0.3: ICMP echo request, id 49178, seq 11, length 64
15:58:08.209290 IP 10.0.0.3 > 10.0.0.3: ICMP echo reply, id 49178, seq 11, length 64
15:58:09.209549 IP 10.0.0.3 > 10.0.0.3: ICMP echo request, id 49178, seq 12, length 64
```

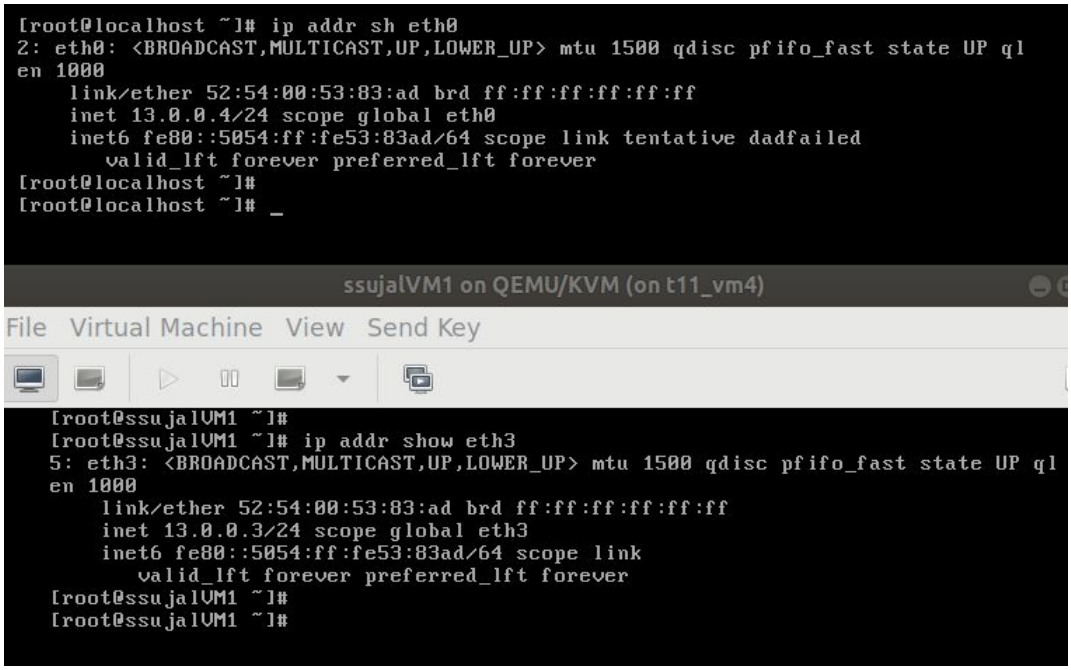
2. **Two VMs connected to different bridge in bridge mode:**

We create a new linux bridge **sw4** that has the network **test** in bridge mode. A VM - **test1** is created and attached to this network.



It is assumed that both bridges are in **different networks and not connected. So traffic from one bridge does not get to the other.**

Setup:
Anyway, the MAC addresses of different interfaces in different bridges are made to have the same values as below.



Ping does not work successfully.

```
       valid_lft forever preferred_lft forever
[root@ssujalVM1 ~]#
[root@ssujalVM1 ~]# ping 13.0.0.4
PING 13.0.0.4 (13.0.0.4) 56(84) bytes of data.
From 13.0.0.3 icmp_seq=2 Destination Host Unreachable
From 13.0.0.3 icmp_seq=3 Destination Host Unreachable
From 13.0.0.3 icmp_seq=4 Destination Host Unreachable
^Z
[3]+  Stopped                 ping 13.0.0.4
[root@ssujalVM1 ~]# _
```

Same IP:

Ping works here as the it is pinging itself (similar to case above)

```
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# ip addr sh eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether f2:54:12:43:7a:23 brd ff:ff:ff:ff:ff:ff
    inet 13.0.0.3/24 scope global eth0
    inet6 fe80::f054:12ff:fe43:7a23/64 scope link
       valid_lft forever preferred_lft forever
```

ssujalVM1 on QEMU/KVM (on t11_vm4)

e Virtual Machine View Send Key

```
[root@ssujalVM1 ~]#
[root@ssujalVM1 ~]#
[root@ssujalVM1 ~]# ip addr show eth3
5: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 52:54:00:53:83:ad brd ff:ff:ff:ff:ff:ff
    inet 13.0.0.3/24 scope global eth3
    inet6 fe80::5054:ff:fe53:83ad/64 scope link
       valid_lft forever preferred_lft forever
[root@ssujalVM1 ~]#
[root@ssujalVM1 ~]# ping 13.0.0.3
PING 13.0.0.3 (13.0.0.3) 56(84) bytes of data.
64 bytes from 13.0.0.3: icmp_seq=1 ttl=64 time=0.020 ms
64 bytes from 13.0.0.3: icmp_seq=2 ttl=64 time=0.040 ms
^Z
[5]+  Stopped                 ping 13.0.0.3
```

3. **Two VMs connected to different bridge in routed mode**

Setup:
Two bridges s5 and s6 are created in routed mode as shown below. VMs are added to it to test. It is assumed that both bridges are in **different networks and not connected. So traffic from one bridge does not get to the other.**

For this experiment, two bridges in routed mode (sw5, sw6) are created.

```
97: s5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group defaul
t qlen 1000
    link/ether 52:54:00:1d:6e:1a brd ff:ff:ff:ff:ff:ff
    inet 13.0.0.1/24 brd 13.0.0.255 scope global s5
       valid_lft forever preferred_lft forever
98: s5-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master s5 state DOWN group d
efault qlen 1000
    link/ether 52:54:00:1d:6e:1a brd ff:ff:ff:ff:ff:ff
99: s6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group defaul
t qlen 1000
    link/ether 52:54:00:14:a2:26 brd ff:ff:ff:ff:ff:ff
    inet 16.0.0.1/24 brd 16.0.0.255 scope global s6
       valid_lft forever preferred_lft forever
100: s6-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master s6 state DOWN group
default qlen 1000
    link/ether 52:54:00:14:a2:26 brd ff:ff:ff:ff:ff:ff
```

```
ece792@tll_vm4:~$ brctl show
bridge name     bridge id               STP enabled     interfaces
s5              8000.5254001d6e1a       yes             s5-nic
                                                        vnet8
s6              8000.52540014a226       yes             s6-nic
                                                        vnet9
```

For this part too (like previous part), it is assumed that the devices are in different networks as the bridges are not connected. Having the same MAC will make no difference and the ping will fail, as it cannot find the destination. Having the same IP will ping but to itself.