Team:

|  |  |
| --- | --- |
| Khantil Choksi - khchoksi | Shubhankar Reddy - skatta2 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Staging Server IP |  |  | VM IP |  |  |
| 152.14.83.156 | ece792 | EcE792net! | 192.168.124.15 | ece792 | EcE792net! |

**PROBLEM 1:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 500 Mbps | 1000 Mbps | 5000 Mbps |
| CPU | 1 vCPU | 1 vCPU (2 for the AX tech stack) | 1/2/8 vCPU (for IP base, Security and AppX) |
| Memory | 4 GB | 4 GB | 4 GB |

1. Networking: BGP, OSPF, EIGRP, Routing Information Protocol (RIP), Intermediate System-to-Intermediate System (IS-IS), IPv6, GRE, VRF-Lite, NTP
2. Security: ZBFW, IPsec VPN, Easy VPN, DMVPN, FlexVPN
3. Management: Cisco IOS XE CLI, SSH, Flexible NetFlow, SNMP, EEM, and NETCONF

1 year annual cost of CSR 1000V (AX stack) on aws = $ 3,723.00 (Software cost)

1 year of c4.large EC2 instance cost = $515.00 (C4.large, yearly cost)

Cost of running two 1000 Mbps CSR in AWS = (3,723 + 515.00) \* 2 = $8,476 USD

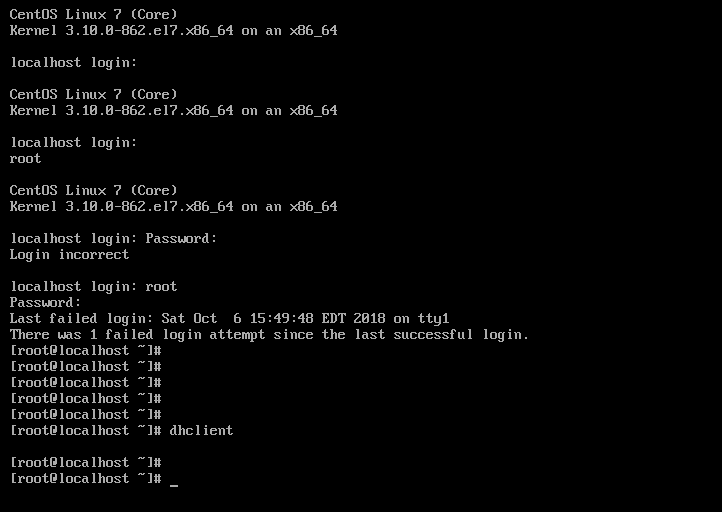
<https://aws.amazon.com/marketplace/pp/B00OCG4OAA?qid=1539727240910&sr=0-1&ref_=srh_res_product_title&cl_spe=T1>

<https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/>

**PROBLEM 2:**

|  |
| --- |
| sudo virt-install -n khchoksi -r 2048 --vcpu=4 --cpu host --disk path=/var/lib/libvirt/images/khchoksi.img,size=10 --network network=khchoksivm -c /home/ece792/iso/CentOS-7-x86\_64-Minimal-1804.iso -v |



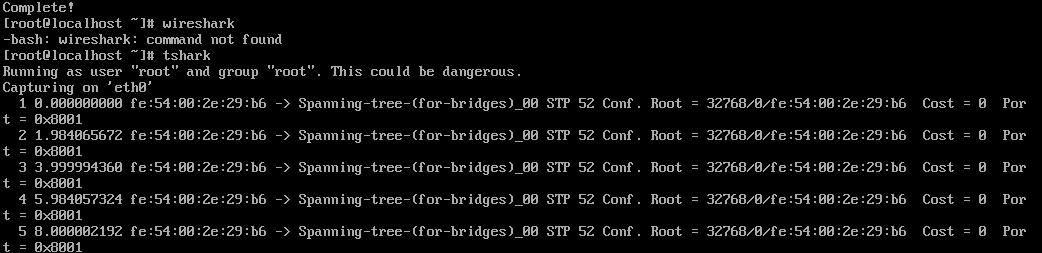


To install required applications:

|  |
| --- |
| $ yum install iperf3  $ yum install wireshark |

(i)

|  |  |  |
| --- | --- | --- |
|  | VM’s NIC | hypervisor NIC |
| IP Address | 192.168.122.92/24 | 192.168.124.15 |
| MAC Address | 52:54:00:2e:29:b6 | 52:54:00:2f:dd:ba |



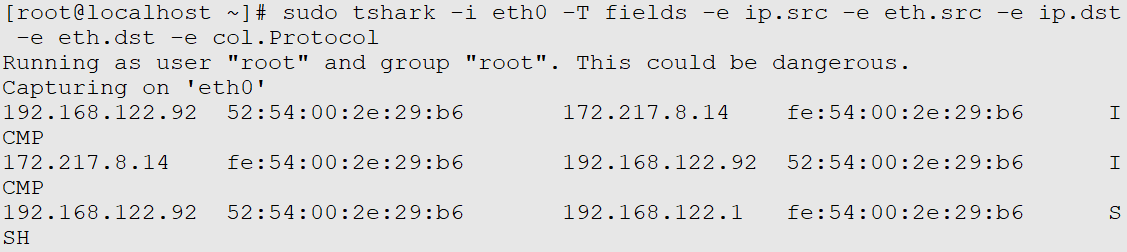
(ii)

|  |  |
| --- | --- |
|  | output interface of VM |
| srcIP | 192.168.122.92 |
| destIP | 172.217.8.14 |
| srcMAC | 52:54:00:2e:29:b6 |
| destMAC | fe:54:00:2e:29:b6 |

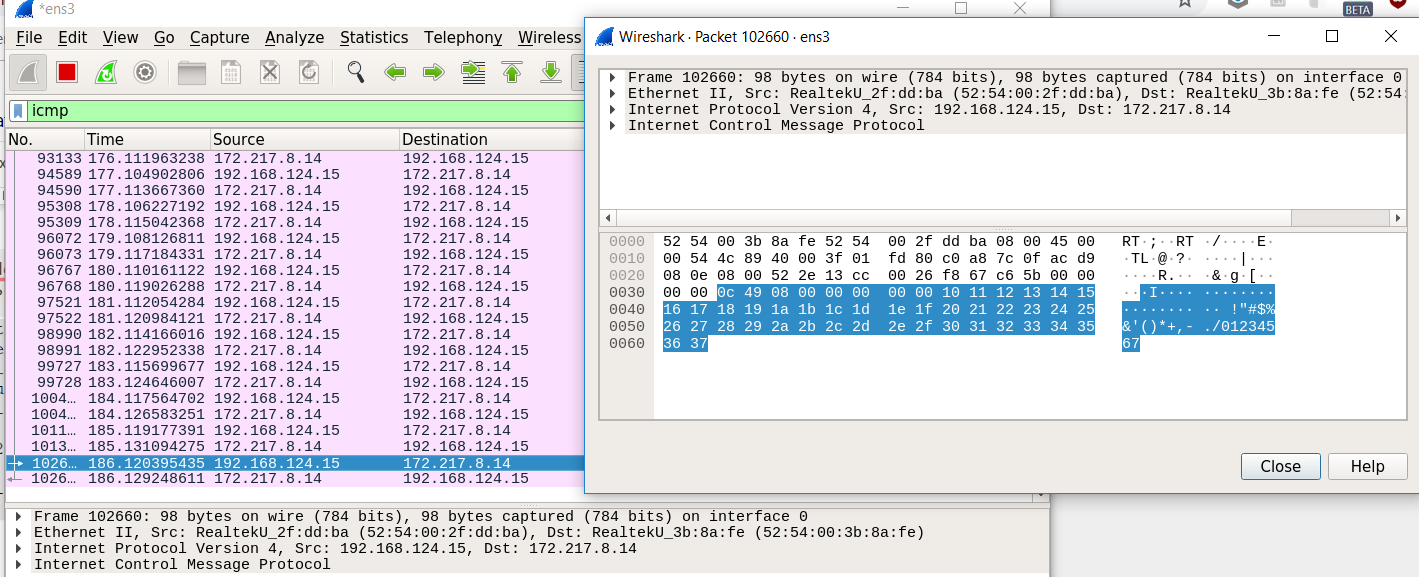
|  |  |
| --- | --- |
|  | output interface of hypervisor |
| srcIP | 192.168.122.15 |
| destIP | 172.217.8.14 |
| srcMAC | 52:54:00:2f:dd:aa |
| destMAC | 52:54:00:3b:8a:fe |

* Packet going out of the VM

|  |
| --- |
| $ sudo tshark -i eth0 -T fields -e ip.src -e eth.src -e ip.dst -e eth.dst -e col.Protocol |



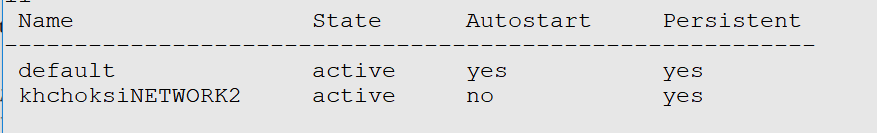
* Packet going out of the hypervisor using wireshark



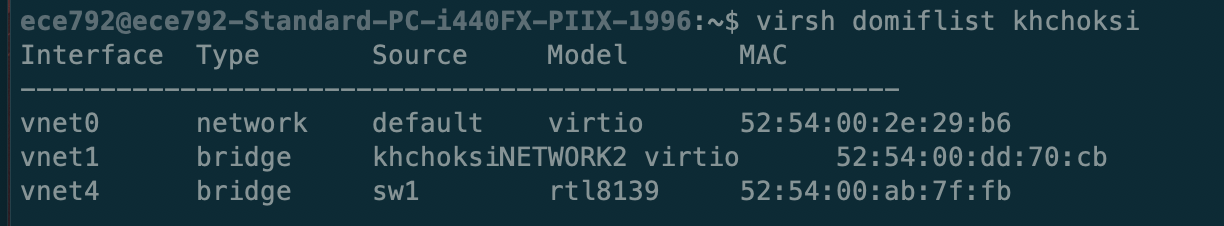
These tuples are in different network. According to datapath, the interface ens3 will do NAT and encap, decap operation and changes the src IP to ens3 IP. And it will be sent out the request over ens3.

**PROBLEM 3:**

1. khchoksiNETWORK2.xml
   1. $ virsh net-define khchoksiNETWORK2.xml
   2. $ brctl addbr sw1
   3. virsh net- start khchoksiNETWORK2



* 1. virsh attach-interface --domain khchoksi --type bridge --source sw1
  2. deactivate VM: $ virsh shutdown khchoksi
  3. Add interface to xml file
  4. $ virsh define /etc/libvirt/qemu/khchoksi.xml
  5. $ virsh start khchoksi (restart VM)

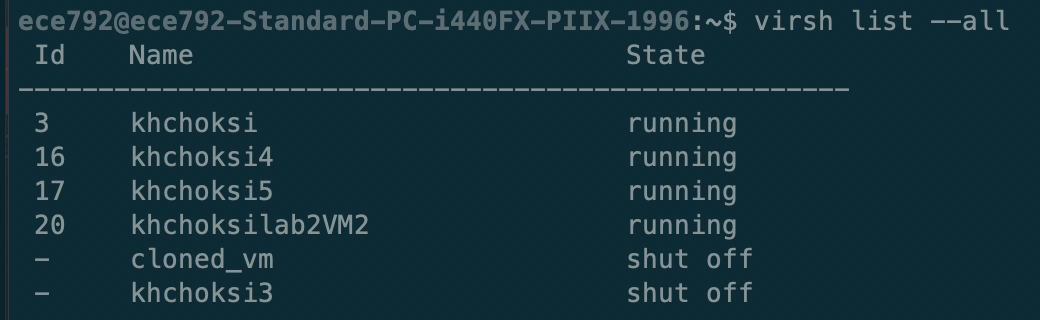


1. $ virsh suspend khchoksi

$ virt clone --original khchoksi --name khchoksilab2VM2 --auto-clone

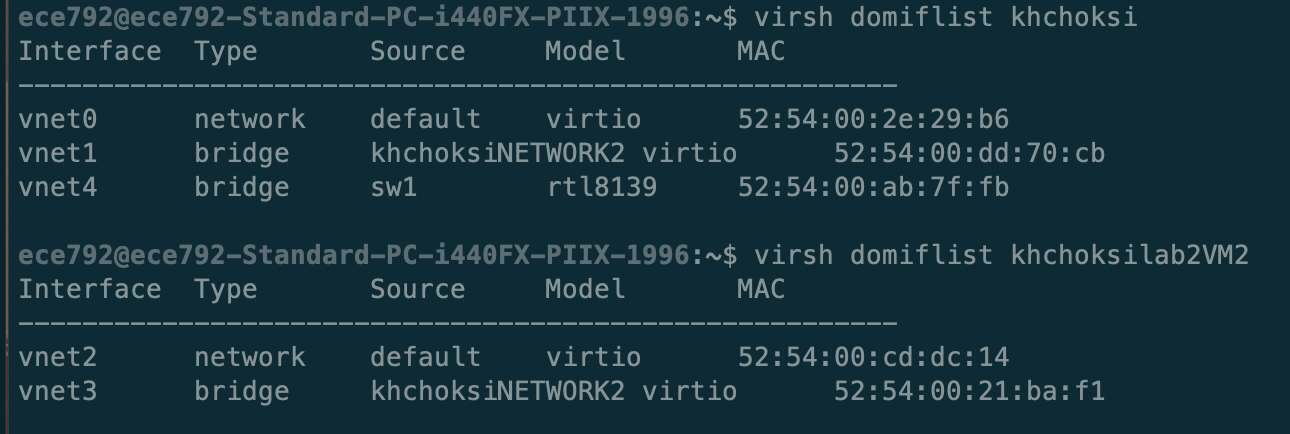
$ virsh resume khchoksi

$ virsh start khchoksilab2VM2

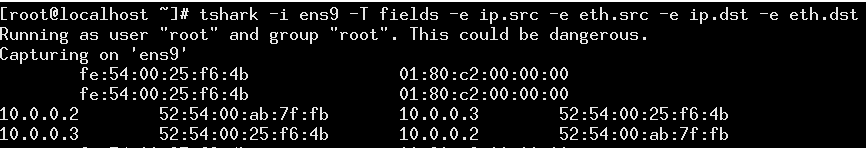


1. Assign ip to eth1 : ifconfig ens9 10.0.0.1/24

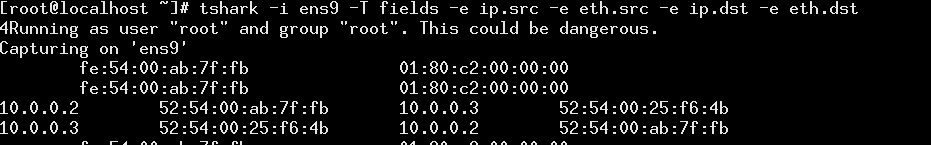
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| VM | interface | network | MAC Address | IP Addresses |
| khchoksi | vnet0 | default | 52:54:00:2e:29:b6 | 192.168.122.92 |
|  | ens9 | khchoksiNETWORK2 | 52:54:00:ab:7f:fb | 10.0.0.2 |
|  |  |  |  |  |
| khchoksilab2VM2 | vnet2 | default | 52:54:00:cd:dc:14 | 192.168.122.51 |
|  | ens9 | khchoksiNETWORK2 | 52:54:00:25:f6:4b | 10.0.0.3 |



1. Tuple fields on 10.0.0.2 are:

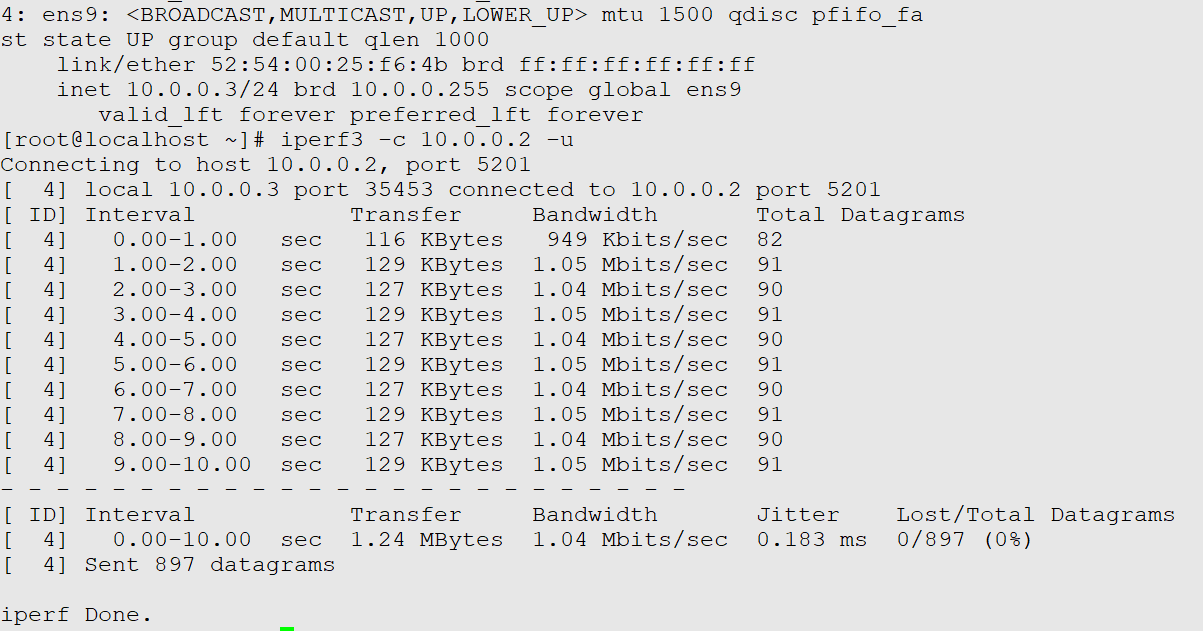


Tuple fields on 10.0.0.3 are:

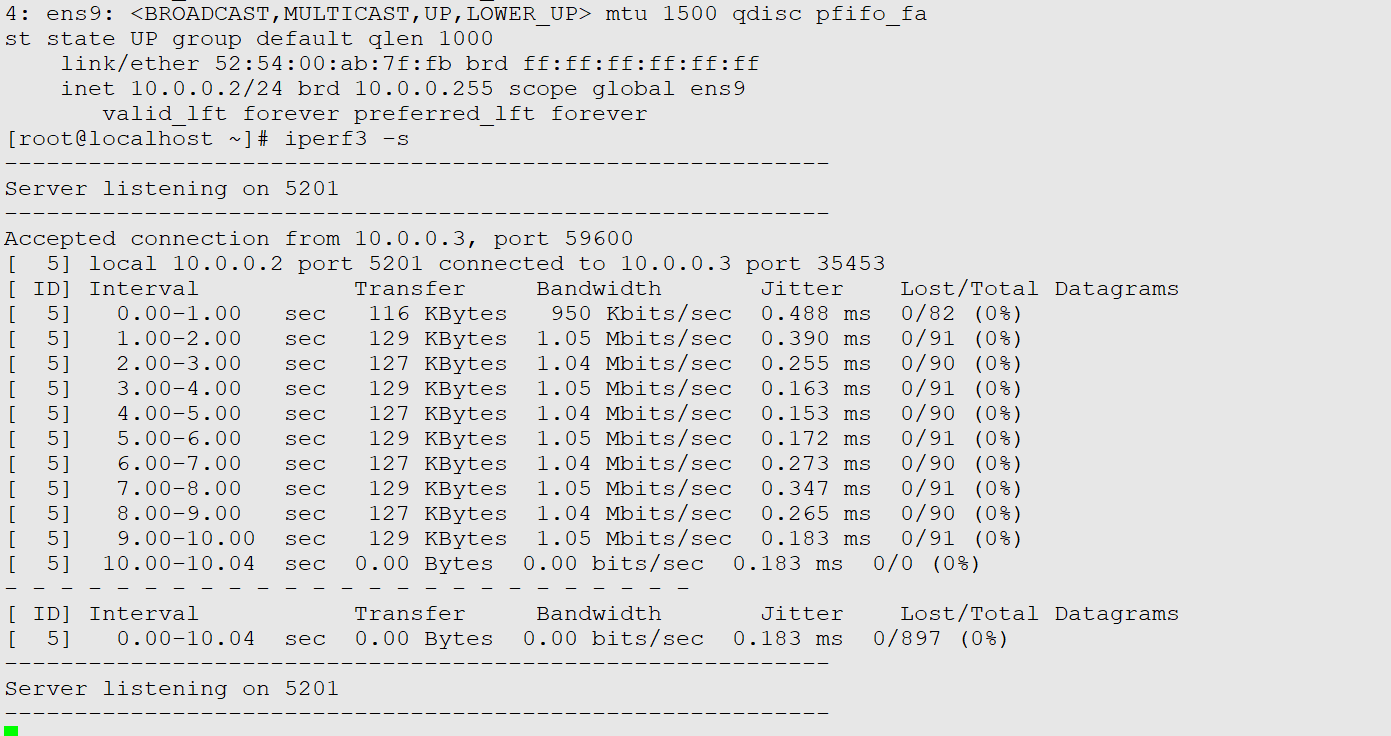


The tuple fields do not change as the packets are forwarded over L2 and no encap/decap takes place.

**6.** On khchoksilab2VM2

****

On khchoksi

****

The maximum throughput achieved is 1.04 Mbits/sec.

From the top command we noticed that CPU usage is not consumed and CPU is idle and I/O is also being affected. So by elimination, we can think that it is the memory.

We can also deduce that, both the VMs will have virtual memory from host and will have memory(RAM) crunch at the time of transferring packets. So, we think that that memory could be the bottleneck.

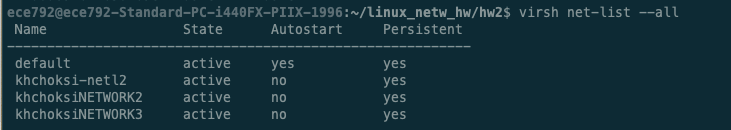
**PROBLEM 4:**

(Note: All the code and output is store in q4 folder submitted in zip)

README  
Prerequisite: Install ansible on host machine

|  |
| --- |
| $ sudo apt-add-repository ppa:ansible/ansible $ sudo apt-get update $ sudo apt-get install ansible $ ansible --version  ansible 2.7.0 |

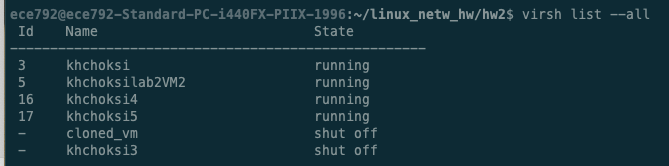
1. **Ansible Playbook: q4\_1.yml**
   1. Create pure L2 network and named khchoksi-netl2

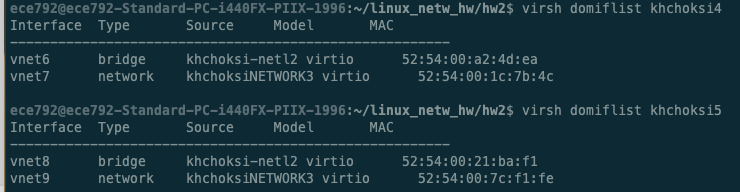


* 1. Create two VMs (khchoksi4, khchoksi5) and connect with this L2 Network

Please make sure to run the ansible script with X11 forwarding.

- The script will wait for configuration of VM using GUI (which can't be automated)





Run playbook using following command

|  |
| --- |
| $ sudo ansible-playbook q4\_1.yml --extra-vars="/home/ece792/iso/CentOS-7-x86\_64-Minimal-1804.iso" |

Playbook: q4\_1.yml:

|  |
| --- |
| --- - hosts: localhost  gather\_facts: no  vars:  network\_name: khchoksi-netl2  bridge\_name: sw2  packages:  - python-libvirt  - python-lxml   guests:  - name: khchoksi4  mem: 512  vcpu: 1  network: "{{ network\_name }}"  - name: khchoksi5  mem: 1024  vcpu: 2  network: "{{ network\_name }}"    vm\_disk\_location: /var/lib/libvirt/images/  iso\_file\_path: {{ iso\_file | default('/home/ece792/iso/CentOS-7-x86\_64-Minimal-1804.iso') }}   tasks:  # Install required packages  - name: Install required packages for libvirt, lxml  apt:  name: "{{packages}}"  become: yes    # Define a new network  - name: Define Virtual Network  virt\_net:  command: define  name: '{{ network\_name }}'  xml: '{{ lookup("template", "templates/bridge\_template.xml.j2") }}'   # Create and start a network  - name: Create Virtual Network if not created  virt\_net:  command: create  name: "{{ network\_name }}"  ignore\_errors: true   # Stop a network  # - name: Stop Virtual Network if running  # virt\_net:  # command: stop  # name: "{{ network\_name }}"  # ignore\_errors: true # To make task idempotent   # List available networks  - name: List available networks  virt\_net:  command: list\_nets    # Create New VM and will pop up UI  - name: Create VM instance  command: >  virt-install -n {{ item.name }} -r {{ item.mem }} --vcpu={{ item.vcpu }} --cpu host --disk path={{ vm\_disk\_location }}{{ item.name }}.img,size=5 --network network={{ network\_name }} -c {{ iso\_file\_path }} -v  become: yes  with\_items: "{{ guests }}" |

**Intermediate steps to setup ssh:**

1. Create a new NAT bridge virbr1 so that both the newly created vms can have ips.
2. Create new network: khchoksiNETWORK3.xml

|  |
| --- |
| <**network**>  <**name**>khchoksiNETWORK3</**name**>  <**uuid**>eadcd6b7-c89a-43b5-9fe0-407eb0034038</**uuid**>  <**forward** mode='nat'/>  <**bridge** name='virbr1' stp='on' delay='0'/>  <**mac** address='52:54:00:9f:f8:b6'/>  <**ip** address='192.168.119.1' netmask='255.255.255.0'>  <**dhcp**>  <**range** start='192.168.119.2' end='192.168.119.254'/>  </**dhcp**>  </**ip**> </**network**> |

1. $ brctl addbr virbr1

$ virsh net- start khchoksiNETWORK3

Add this interface to both the VMs and restart them.

|  |
| --- |
| <**interface** type='network'>  <**source** network='khchoksiNETWORK3'/>  <**model** type='virtio'/>  <**address** type='pci' domain='0x0000' bus='0x00' slot='0x0a' function='0x0'/> </**interface**> |

1. Get the ips of both the vms (if not assigned, do dhcpclient).
2. Create SSH key pairs on host machine

$ ssh-keygen

and then follow the command

1. Copy public keys to both guest machines

$ ssh-copy-id -i ./keys/vm\_rsa.pub root@192.168.119.58

$ ssh-copy-id -i ./keys/vm\_rsa.pub root@192.168.119.252

VII. Create new inventory file as follows:

|  |
| --- |
| [vms] localhost ansible\_connection=local 192.168.119.58 ansible\_ssh\_user=root ansible\_ssh\_private\_key\_file=./keys/vm\_rsa 192.168.119.252 ansible\_ssh\_user=root ansible\_ssh\_private\_key\_file=./keys/vm\_rsa |

1. **Ansible playbook to collect logs:**
   1. Make sure inventory file and keys are created as mentioned above.
   2. Run the playbook with total time(in minutes) as parameter given below. If time not defined, it will take default 5 minutes. Granularity is set as 1 minutes as mentioned in the description.

|  |
| --- |
| $ sudo ansible-playbook q4\_2.yml -i ./inventory --extra-vars "time=7" |

Logs will be generated at: /var/customlogs/logs.csv (Attached within zip:-> q4\_2\_logs.csv)

Note: Ansible playbook will create ‘customlogs’ directory if not present.

Playbook: q4\_2.yml

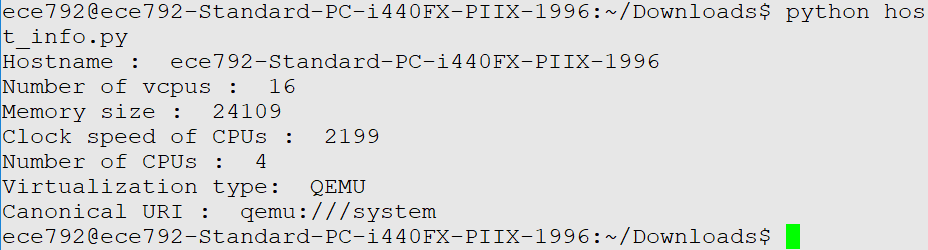
|  |
| --- |
| --- - hosts: vms  gather\_facts: no  vars:  total\_time: "{{ time | default(5) }}" # Defined total time if not passed from command line  granularity: 60 #in seconds  log\_file\_directory: /var/customlogs  log\_file\_path: "{{ log\_file\_directory }}/logs.csv"  tasks:  - name: Create logs directory if not present  file:  path: "{{ log\_file\_directory }}"  state: directory  mode: 0777  owner: ece792  group: ece792  become: yes  delegate\_to: localhost  run\_once: true   - name: Generate Log CSV File Header  shell: echo "hostname, timestamp, cpu1min, cpu5min, cpu15min" >> "{{ log\_file\_path }}"  delegate\_to: localhost  run\_once: true  become: yes   - name: Generate loop sequence based on input total time parameter  set\_fact:  loop\_sequence: "{{ loop\_sequence | default([]) + [item | int] }}"  with\_sequence: start=1 end={{ total\_time }}   # - name: debug\_list # de**bug:** # msg: "{{ loop\_sequence }} "    - name: Fetch cpu usages from host and guests, store it in output variable  shell: "echo -n '{{hostvars[inventory\_hostname]['inventory\_hostname']}},' && date +%X | awk -F, '{printf \" %s, \", $1}' && uptime | sed 's/.\*load average: //' | awk -F\\, '{ printf \"%s, %s, %s\", $1, $2, $3}'"  register: output  loop: "{{ loop\_sequence }}"  loop\_control:  pause: "{{ granularity }}"   # - name: debugging # de**bug:** # msg: "{{ item.stdout }}" # with\_items: "{{ output.results }}"    - name: Writing logs to csv file  shell: |  echo "{{ item.stdout }}" >> "{{ log\_file\_path }}"  with\_items: "{{ output.results }}"  delegate\_to: localhost  become: yes |

**Problem 5:**

(Note: All the code and output is store in q5 folder submitted in zip)

1. **Obtaining host information:**

|  |
| --- |
| **import** sys **import** libvirt **import** random  conn = libvirt.open('qemu:///system') **if** **not** conn:  **print** "Connection failed"  exit(1)  domainIDs = conn.listDomainsID() **if** len(domainIDs) == 0:  **print** "No active domains" randomid = random.sample(domainIDs, 1)  rdom = conn.lookupByID(randomid[0])  state, maxmem, mem, cpus, cput = rdom.info() **print** "UUID of the guest vm : ", rdom.UUIDString() **print** "OS type of the guest vm : ", rdom.OSType() **print** "Max vcpus of the guest vm : ", str(rdom.maxVcpus()) **print** "State of the guest vm : ", str(state) **print** "Name of the guest vm : ", rdom.name() **print** "Max memory of the guest vm : ", str(maxmem) **print** "Number of cpus in the guest vm : ", str(cpus)  conn.close() exit(1) |

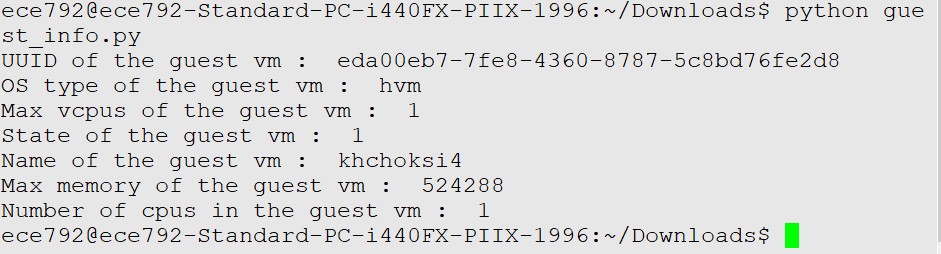
****

(README) To run this code, simply run following:

|  |
| --- |
| $ python host\_info.py |

**2) Obtaining Guest Information**

|  |
| --- |
| **import** sys **import** libvirt  conn = libvirt.open('qemu:///system')  **if** conn == **None**:  print('Connection failed')  exit(1)  node\_info = conn.getInfo()  **print** "Hostname : ", conn.getHostname() **print** "Number of vcpus : ", conn.getMaxVcpus(**None**) **print** "Memory size : ", node\_info[1] **print** "Clock speed of CPUs : ", node\_info[3] **print** "Number of CPUs : ", node\_info[2] **print** "Virtualization type: ", conn.getType() **print** "Canonical URI : ", conn.getURI()  conn.close() exit(1) |

****

(README) To run this code, simply run following:

|  |
| --- |
| $ python guest\_info.py |

**3) Performance Monitoring:**

We calculate cpu utilization using : 100\*(cpu\_timet2 - cpu\_timet1)/10^9\*(t2 - t1)

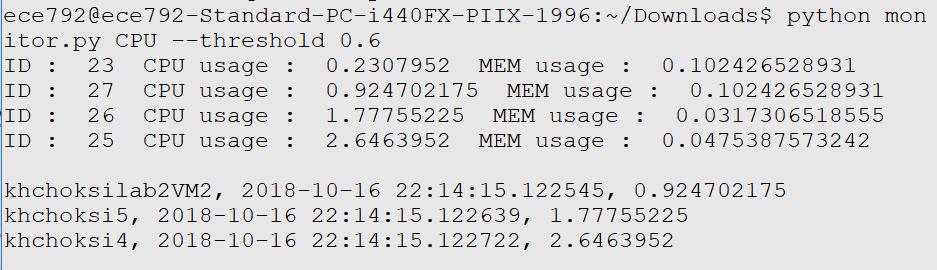
Where t1 and t2 are in seconds, and cpu\_time is in nanoseconds

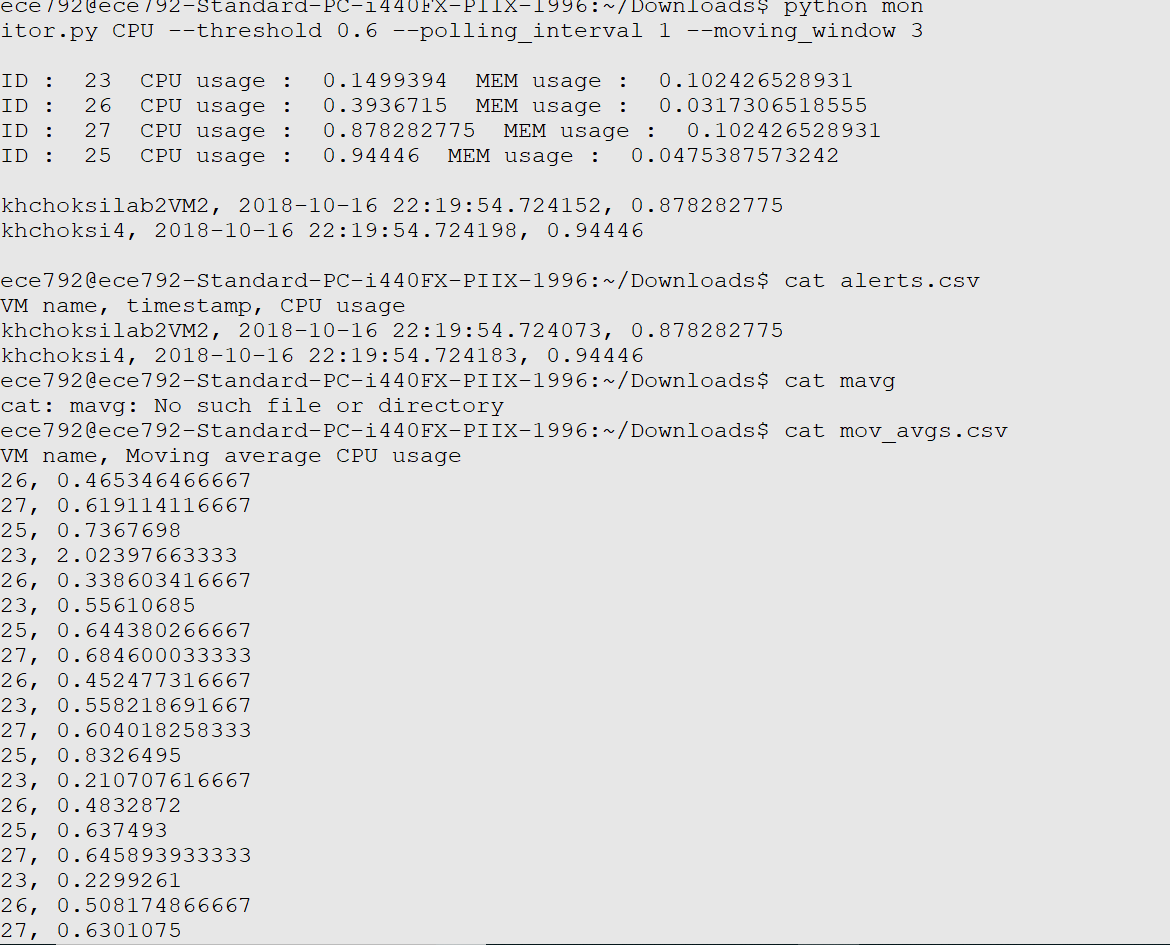
In cases where we have more than 1 vcpu, we divide the utilization by the number of vcpus to obtain the aggregate CPU utilization.

For memory, we calculate the utilization as : 1 - memory\_available/memory\_actual

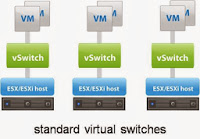
This gives us the amount of memory used, which we believe is a measure of the memory utilization

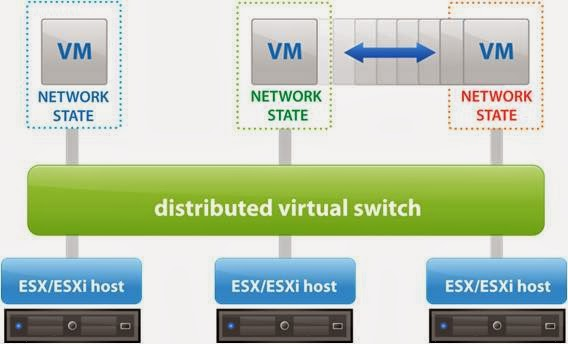
|  |
| --- |
| **import** sys **import** libvirt **import** random **import** os **import** datetime **import** argparse **import** collections **from** time **import** sleep  py\_parser = argparse.ArgumentParser(description='Monitor script')  py\_parser.add\_argument('order', nargs=1, choices = ["CPU","MEM"], help="order to sort by") py\_parser.add\_argument('--threshold', nargs=1, type = float, help="threshold CPU value") py\_parser.add\_argument('--polling\_interval', nargs=1, type = float, help="polling interval value") py\_parser.add\_argument('--moving\_window', nargs=1, type = int, help="moving window value")  py\_args = py\_parser.parse\_args()  order = py\_args.order[0]  **if** **not** py\_args.threshold:  threshold = 0 **else**:  threshold = py\_args.threshold[0]  conn = libvirt.open('qemu:///system') **if** **not** conn:  **print** "Connection failed"  exit(1)  domainIDs = conn.listDomainsID() **if** len(domainIDs) == 0:  **print** "No active domains"  vm\_id\_list = [] + domainIDs stats = [] **for** vm\_id **in** vm\_id\_list:  vm = conn.lookupByID(vm\_id)  cpu\_stats = vm.getCPUStats(**True**)[0]  mem\_stats = vm.memoryStats()  vcpus = vm.maxVcpus()  stats.append([vm\_id, cpu\_stats['cpu\_time']\*1.0,\  1 - mem\_stats['available']\*1.0/mem\_stats['actual'], vcpus, vm])  sleep(1)  **for** indx, stat **in** enumerate(stats):  vm = stat[-1]  cpu\_stats = vm.getCPUStats(**True**)[0]  stats[indx][1] = (cpu\_stats['cpu\_time'] - stats[indx][1])/10\*\*9  stats[indx][1] = (stats[indx][1]\*100)/stats[indx][3]  **if** stats[indx][1] > 100:  stats[indx][1] = 100  #Sort by CPU or MEM **if** order == "CPU":  stats.sort(key=**lambda** x: x[1]) **else**:  stats.sort(key=**lambda** x: x[2])  #Logging **if** **not** os.path.isfile("alerts.csv"):  log\_file = open("alerts.csv",'w')  log\_file.write("VM name, timestamp, CPU usage\n") **else**:  log\_file = open("alerts.csv",'a')  log\_op = "" print\_op = "" **for** vm\_stat **in** stats:  #Printing sorted list  **print** "ID : ", vm\_stat[0], " CPU usage : ", vm\_stat[1]\  , " MEM usage : ", vm\_stat[2]  #If cpu > threshold, log and print  **if** vm\_stat[1] > threshold:  log\_op += vm\_stat[-1].name()+", "+str(datetime.datetime.now())+", "+str(vm\_stat[1])+"\n"  print\_op += vm\_stat[-1].name()+", "+str(datetime.datetime.now())+", "+str(vm\_stat[1])+"\n"  log\_file.write(log\_op) **print** "\n",print\_op  #Bonus part poll\_int = py\_args.polling\_interval[0] mov\_wind = py\_args.moving\_window[0]  **if** poll\_int==**None** **or** mov\_wind==**None**:  exit(1) poll\_int = py\_args.polling\_interval[0] mov\_wind = py\_args.moving\_window[0]  prev\_poll\_time = {} curr\_poll\_time = {} polled\_values = collections.defaultdict(list)  **for** indx, stat **in** enumerate(stats):  vm = stat[-1]  cpu\_stats = vm.getCPUStats(**True**)[0]  prev\_poll\_time[stat[0]] = cpu\_stats['cpu\_time']  **if** **not** os.path.isfile("mov\_avgs.csv"):  mavgs = open("mov\_avgs.csv",'w')  mavgs.write("VM ID, Moving average CPU usage\n") **else**:  mavgs = open("mov\_avgs.csv",'a')  poll\_timer = 0  **try**:  **while** **True**:  sleep(poll\_int)   **if** poll\_timer >= mov\_wind\*poll\_int:  #Log the moving window averages   unsorted\_list = []  **for** v **in** polled\_values:  unsorted\_list.append([v, sum(polled\_values[v])/len(polled\_values[v])])  polled\_values[v].pop(0)  unsorted\_list.sort(key = **lambda** x: x[1])  **for** v **in** unsorted\_list:  mavgs.write(str(v[0])+", "+str(v[1])+"\n")  #print unsorted\_list   **for** indx, stat **in** enumerate(stats):  vm = stat[-1]  cpu\_stats = vm.getCPUStats(**True**)[0]  curr\_poll\_time[stat[0]] = cpu\_stats['cpu\_time']  polled\_values[stat[0]].append(100\*(curr\_poll\_time[stat[0]] - \  prev\_poll\_time[stat[0]])/(10\*\*9 \* poll\_int \* stats[indx][3]))  prev\_poll\_time[stat[0]] = curr\_poll\_time[stat[0]]  #print polled\_values  poll\_timer += poll\_int **except**:  mavgs.close()  exit(0) |

****

****

**PROBLEM 6:**



****

Distributed vSwitch: Distributed vSwitches allow different hosts to use the switch as long as they exist within the same host cluster. A distributed vSwitch extends its ports and management across all the servers in a cluster, supporting up to 500 hosts per distributed switch. Instead of making virtual networks more complicated with its additional options, the distributed vSwitch simplifies operations and helps catch configuration errors and increase network visibility.

Standard vSwitch: A standard vSwitch works within one ESX/ESXi host only. Standard switch is created in host level i.e. we can create and manage vSphere standard switch independently on an ESXi host. Inbound traffic shaping is not available as a part in the standard switch.

In cases where we want to extend the L2 layer across VMs in different hosts, instead of creating GRE/VXLAN tunnels between each host pair, we could use the distributed vSwitch.

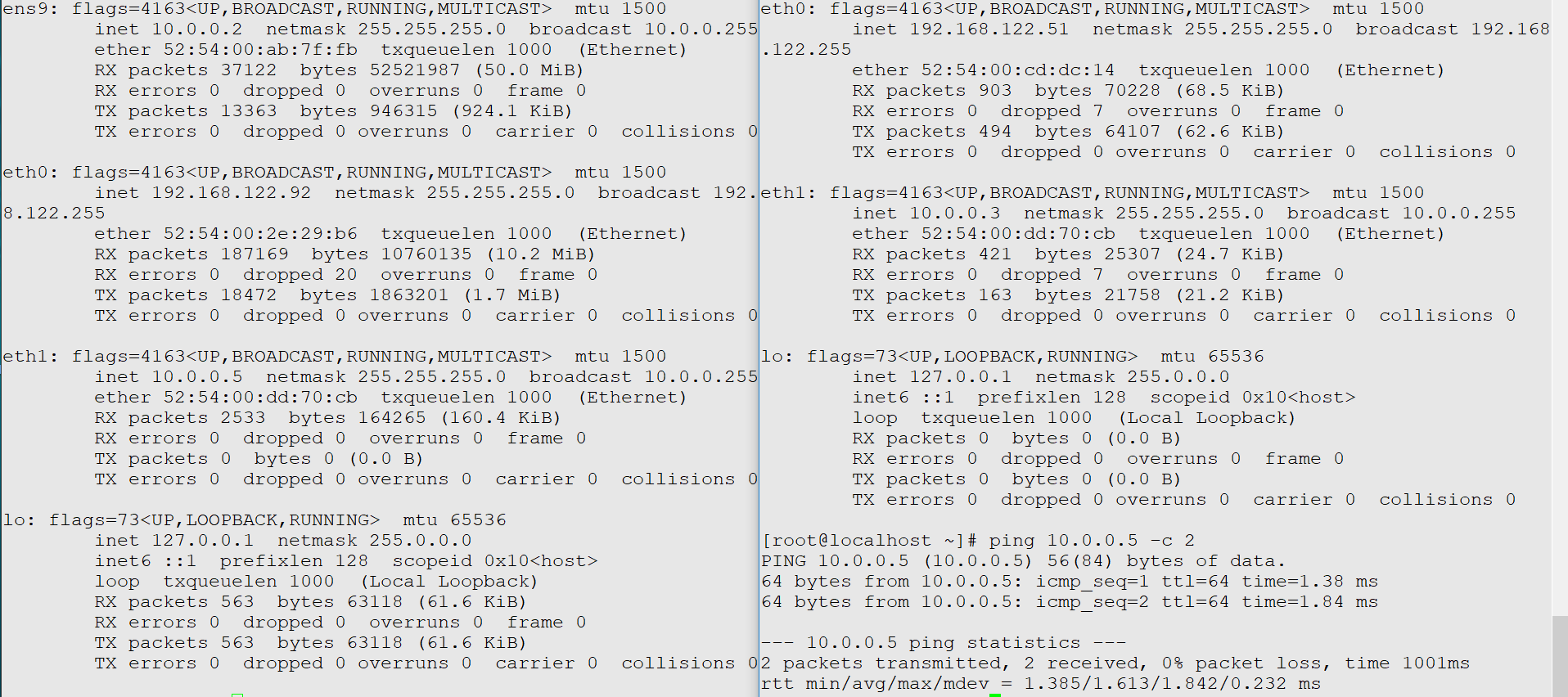
Ref: <https://searchvmware.techtarget.com/photostory/2240185944/Getting-VMware-terminology-straight/9/How-do-switches-vSwitches-and-distributed-vSwitches-differ>

**PROBLEM 7:**

1. If 2 VMs connected to same bridge in bridge mode:
   1. **Same MAC address**:

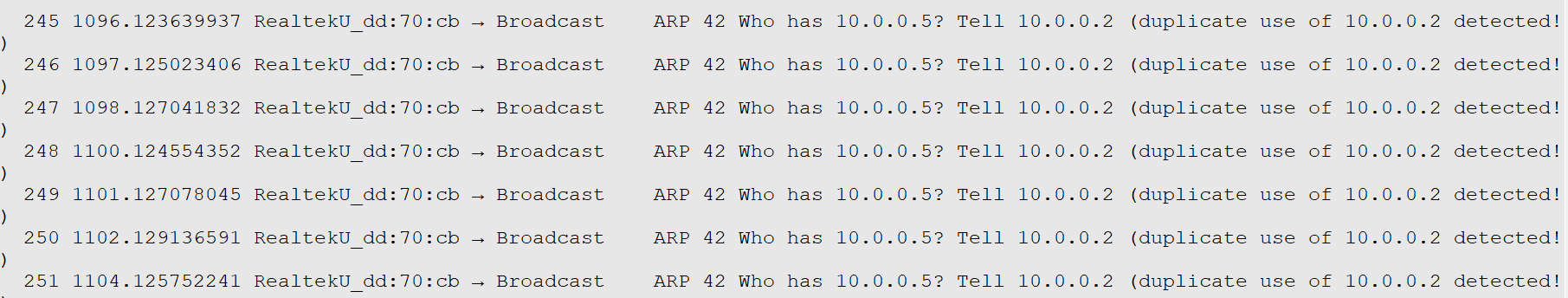
It will overwrite the entries for same MAC address when ARP will be done. The 2VMs should be able to ping each other.

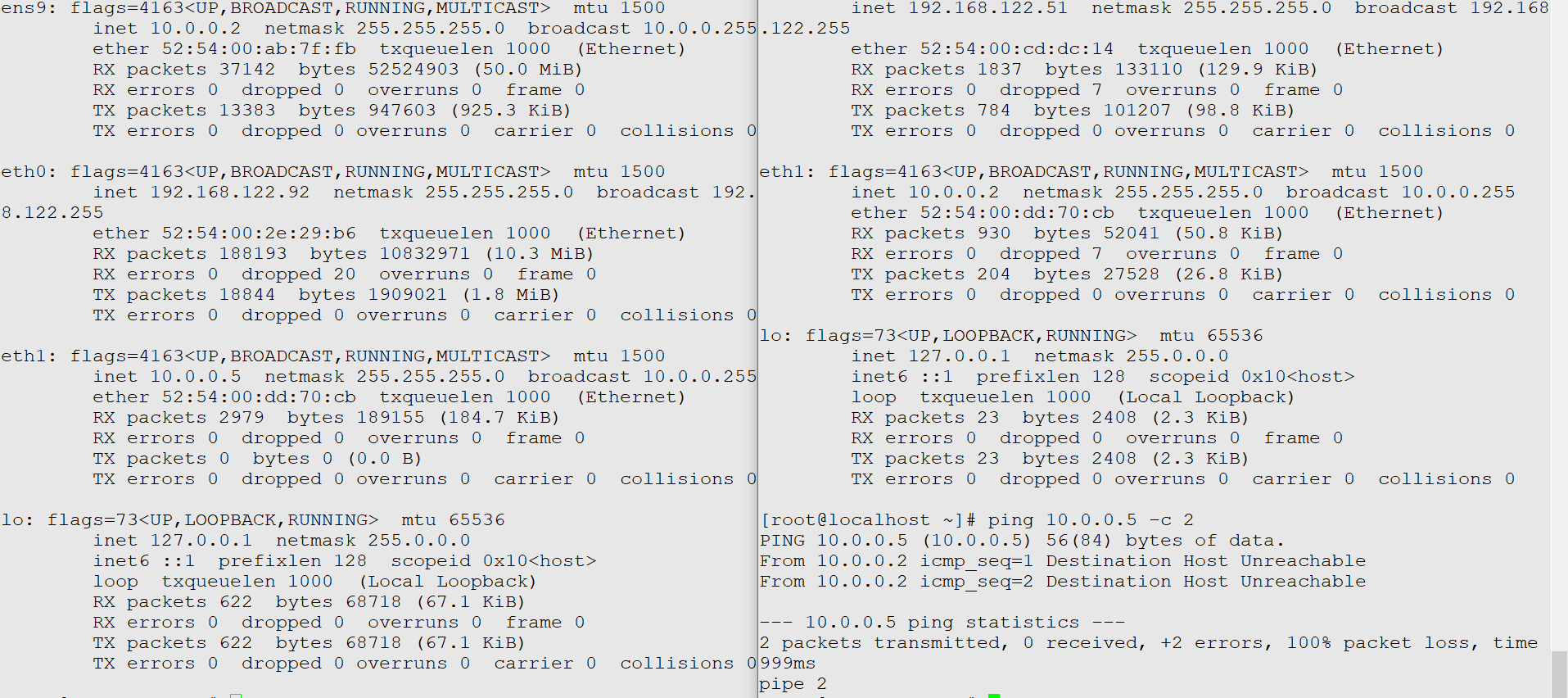
As they have different IP address, the ARP will be resolved and ping will be successful between two VMs as well as bridge will work fine.



* 1. **Same IP address:**

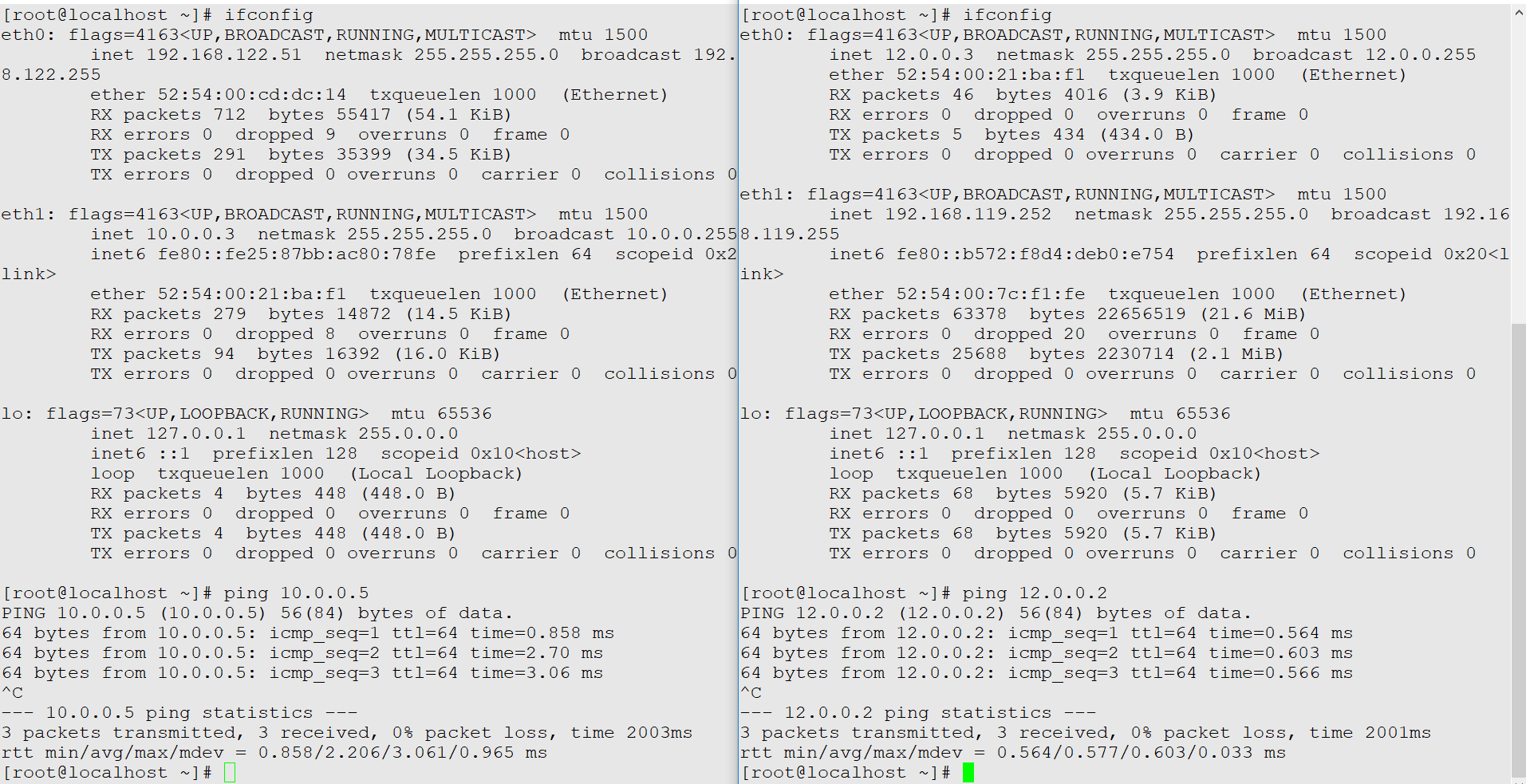
Switch will throw an error that, duplicate use of ip detected.





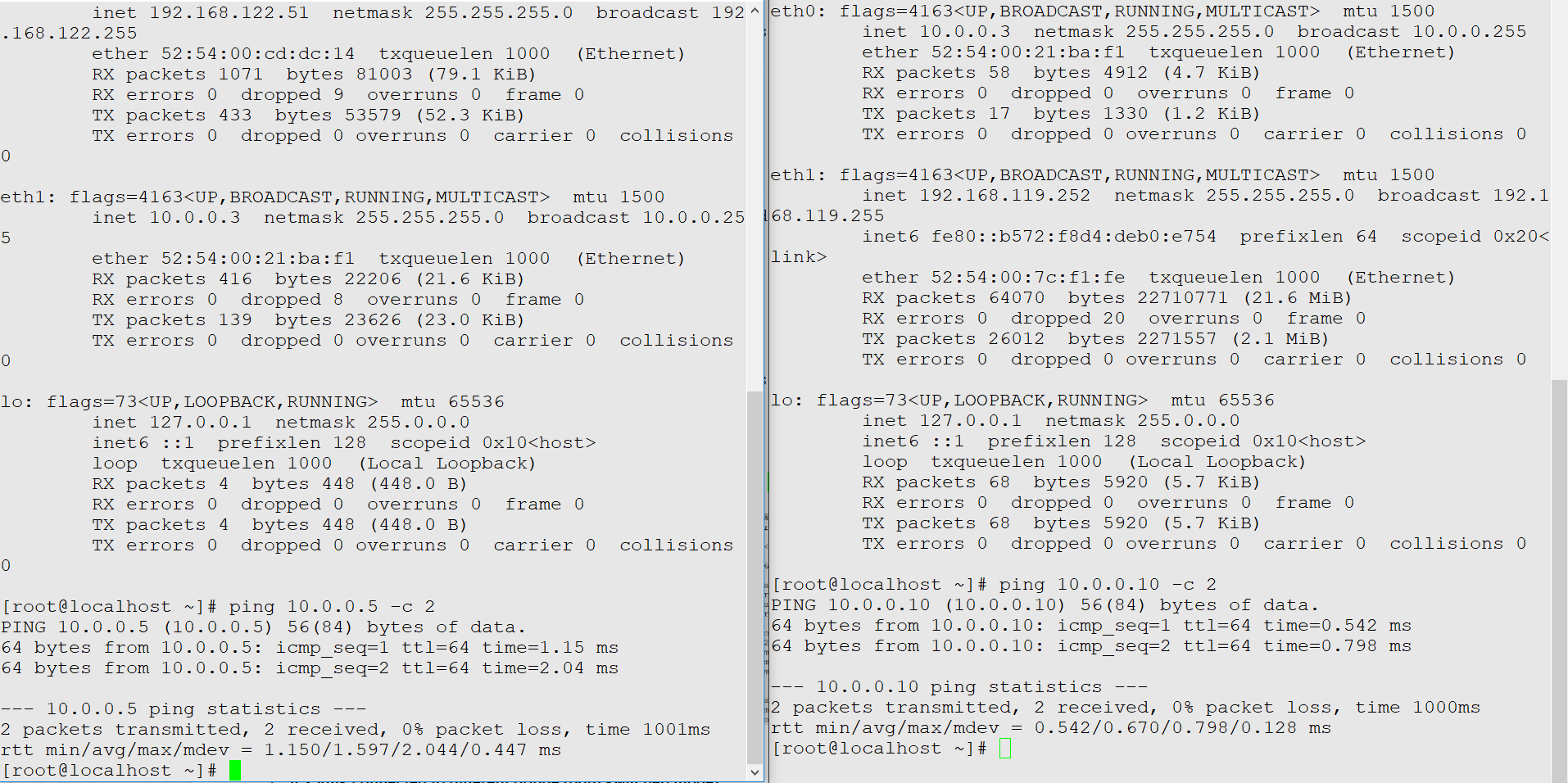
1. **If 2 Vms connected to different bridge (both bridge mode)**
   1. Same MAC address:

As both VMs are in different networks, it won’t affect the bridge.



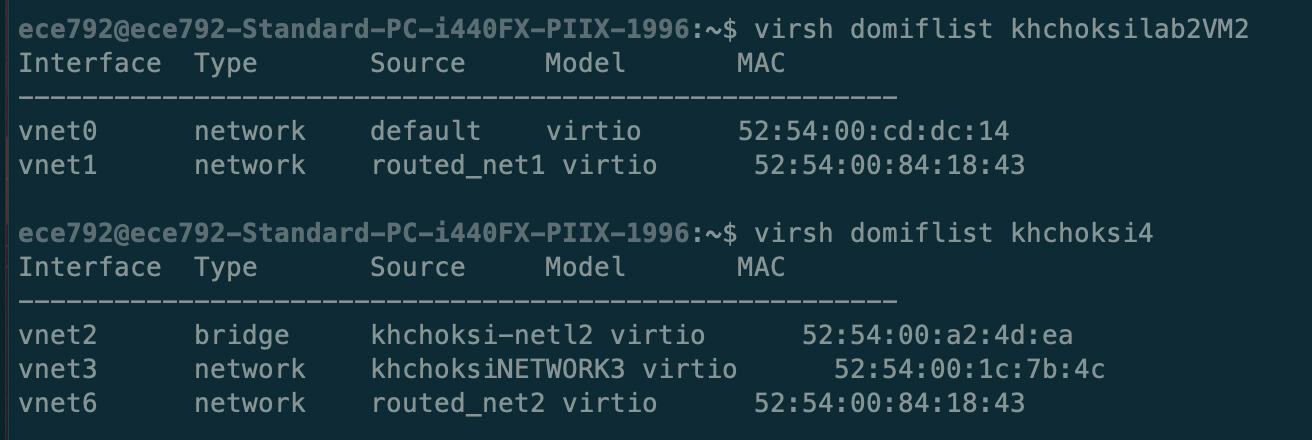
* 1. Same IP address:

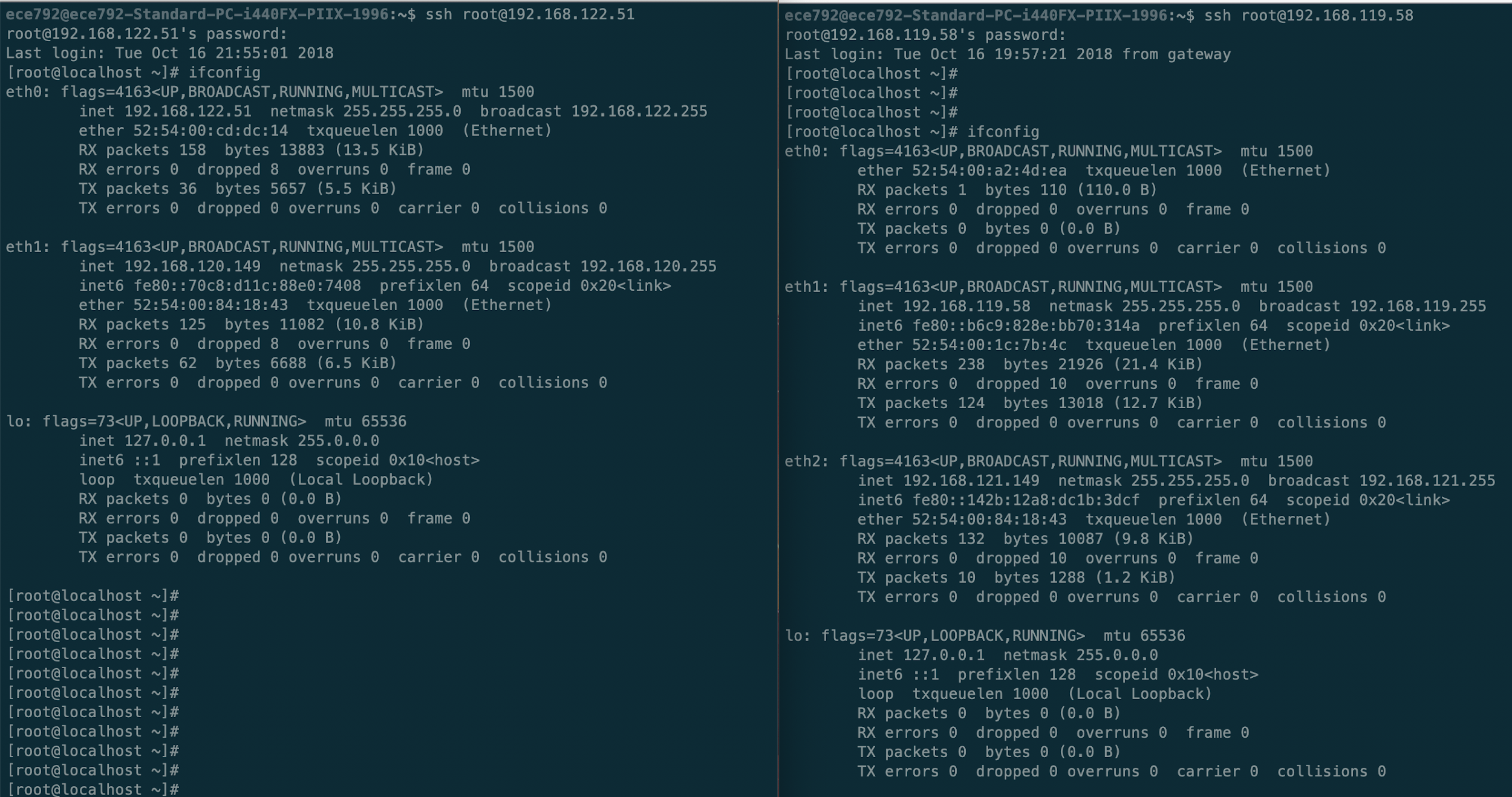
As both the VMs are in different network, it won’t allow to change the ip of second VM and network won’t be affected.



1. **If 2 Vms connected to different bridge (both routed mode)**
   1. **Same MAC address:** If the two VMs are connected to different bridges (over here sw3 and sw4) then everything will work correctly even if we provide same MAC address to both the VMs.

As they both are on different networks (routed\_subnet1) and (routed\_subnet2), it won’t hinder each other.



****

* 1. **Same IP address:**

If they have the same IP address, the VM which has its IP address from the other subnet will lose connectivity with the subnet it is in. In our case, the VM khchoksilab2VM2 will lose connectivity from the 192.168.120.0/24 network as it has been moved to the 192.168.121.0/24 network IP address.

