Live VM Migration and Analysis

A Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Technology
in
Computer Science & Engineering

by

Muskan Agarwal(20144046)
Sukanya Gupta(20144024)
Pooja Vishnoi(20144124)
Prashant Vissa(20144018)
Sunil Kumar(20134152)

to the
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
MOTILAL NEHRU NATIONAL INSTITUTE OF TECHNOLOGY
ALLAHABAD
May, 2018

UNDERTAKING

We declare that the work presented in this report titled "*Live VM Migration and Analysis*", submitted to the Computer Science and Engineering Department, Motilal Nehru National Institute of Technology, Allahabad, for the award of the ***Bachelor of Technology*** degree in ***Computer Science & Engineering***, is our original work. We have not plagiarized or submitted the same work for the award of any other degree. In case this undertaking is found incorrect, we accept that our degree may be unconditionally withdrawn.

May, 2018
Allahabad

Muskan Agarwal(20144046)
Sukanya Gupta(20144024)
Pooja Vishnoi(20144124)
Prashant Vissa(20144018)
Sunil Kumar(20134152)

CERTIFICATE

Certified that the work contained in the report titled "*Live VM Migration and Analysis*", by Muskan Agarwal,Sukanya Gupta, Pooja Vishnoi,Prashant Vissa, Sunil Kumar, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Prof. Neeraj Tyagi)
Computer Science and Engineering Dept.
M.N.N.I.T, Allahabad

May, 2018

Preface

We hereby present our project on the topic Live VM Migration and its Analysis. The project phases into sections describing VM migration and its types, the common techniques followed for VM migration and its issues pertaining to the same. Further on, we introduce the various tools mandatory for Live VM migration. These phase out into KVM, QEMU, Libvirt, VXLAN and Open vSwitch. Finally we present our practical on various projects carried out during different semesters. We began with a Live VM Migration across the same subnet. Then moved on to live VM migration across different subnets using NAT and finally live VM migration across different subnets using bridged mode. Towards the end we provide a comparative analysis on the same. This project provides a good pedestal to readers who wish to understand the concepts of VM migration specific to its applicability in data centers and also illustrates a detailed step by step procedure to conduct a Live VM migration across subnets using personal computers. We also declare that we have taken help from the Internet whose references have been stated in references. We have also taken immense help and guidance from our mentors Prof. Neeraj Tyagi and Dr. Mayank Pandey. We hope this thesis proves helpful to its readers.

Acknowledgements

We take this opportunity to express the gratitude to all the people who extended their cooperation and played their important role in the successful completion of this project. We express our sincere thanks to Prof. Neeraj Tyagi and Dr. Mayank Pandey, for their guidance and encouragement which led to completion of this project. Without their constant appraisals and efforts this task could not have been possible. We would also like to thank our friends and others who imparted great help at every step to make it a success.

Contents

Chapter 1

Introduction

A virtual machine (VM) is an emulation of a computer system. Virtual machines are based on computer architectures and provide the functionality of a physical computer. Their implementations may involve specialized hardware, software, or a combination of both. Another concept essential to our thesis is live migration. It refers to the process of moving a running virtual machine or application between separate physical machines without disconnecting the client or application. This thesis introduces itself by describing VM migration and its types, the common techniques followed for VM migration and its issues pertaining to the same. Further on, we introduce the various tools mandatory for intra and inter subnet VM migration. These phase out into KVM, QEMU, Libvirt, VXLAN and Open vSwitch. Finally we present our practical on various projects carried out during different semesters. We began with a Live VM Migration across the same subnet. Then moved on to live VM migration across different subnets using NAT and finally live VM migration across different subnets using bridged mode. Towards the end we provide a comparative analysis on the same.

1.1 Motivation

In todays world, the term Cloud has become a buzzing sensation. The ease of use and access are only to name a few of its reasons. The increase in demand for IaaS

(Infrastructure as a service), PaaS (Platform as a service) and SaaS (Software as a service) form the crux of cloud computing. Virtualization is a key piece of modern data center design. Virtualization occurs on many devices within the data center, where multiple logical devices run on a physical device. The virtual machines are migrated from one physical host to another for efficient use of physical resources, maintenance purposes, etc. Hence we sought to emulate this VM migration within and across subnets and get a deep understanding of its behavior and provide a final analysis on the same. Although intra and inter subnet migration can occur easily through various softwares, we were enthusiastic to integrate our own tools and resources to perform this migration. Once we had the understanding of how things were happening our next phase was why they were happening (i.e. the analysis part of our project) which so far has surfaced very little in the research world. In doing so, we were to understand the depths of VM migration and the requirements to be fulfilled to execute a successful VM migration.

1.2 Proposed Work

We propose to implement Live VM migration. It includes migration of VM across both intra-subnet and inter-subnet topologies. In inter subnet migration, we propose to carry out the migration with the help of VXLAN using bridge mode and provide a comparative analysis on the same. Also, we propose to create a live deployment of our project in the Computer Science and Engineering Department and all analysis results shall be on the sole basis of the live deployment of our project.

Chapter 2

Hypothesis

2.1 Virtualization

Virtualization refers to creation of virtual environments, which can be used to run different operating systems and applications on a single physical machine. Virtualization also helps in the ease of migration of resources. It also helps in providing software isolation to applications in shared environments as well as ensures better utilization of server resources.

2.2 VM Migration

A virtual machine running on one physical host can be migrated to another physical host. It consists of

- Transfer of the persistent state of the VM(i.e.file system)
- Transfer of the volatile state of the VM (i.e. RAM contents and CPU state)
- Redirection of network traffic

Once the state transfer is complete, the VM continues to run in the new physical machine.

2.2.1 Types of VM Migration

- Cold Migration: During cold migration, the service is paused or terminated while the state of the VM is transferred. This method has low complexity but results in large service downtime.

- Live Migration: During live migration, the service continues to run on the source host. The state on the target host is then updated iteratively until the state difference is small. Service downtime is very less. It is energy efficient, results in load balancing and high availability of physical servers in cloud data center.

Live migration is of two types:

- **Pre-copy Live Migration:**

  The bulk of the VMs memory state is migrated to a target node even as the VM continues to execute at a source node. If a transmitted page is dirtied, it is re-sent to the target in the next round. This iterative copying of dirtied pages continues until either a small, writable working set (WWS) has been identified, or a preset number of iterations is reached, whichever comes first. This constitutes the end of the memory transfer phase and the beginning of service downtime. The VM is then suspended and its processor state plus any remaining dirty pages are sent to a target node. Finally, the VM is restarted and the copy at source is deleted.

  Pre-copy consists of three phases:

  i. **Pre-Copy phase:** At this stage, the VM continues to run, while its memory is iteratively copied page wise from the source to the target host. Iteratively means, the algorithm works in several rounds. It starts with transferring all active memory pages. As each round takes some time and in the mean time the VM is still running on the source host, some pages may be dirtied and have to be re-sent in an additional round to ensure memory consistency.
  ii. **Pre-Copy Termination Phase:** Without any stop condition, the iterative pre-copy phase may carry on indefinitely. Stop conditions depend highly

on the design of the used hypervisor, but typically take one of the following thresholds into account: (1) the number of performed iterations exceeds a predefined threshold, (2) the total amount of memory that has already been transmitted, exceeds a predefined threshold or (3) the number of pages dirtied in the previous round falls below a predefined threshold.

iii. **Stop-and-Copy Phase:** At this stage the hypervisor suspends the VM to stop page dirtying and copies the remaining dirty pages as well as the state of the CPU registers to the destination host. After the migration process is completed, the hypervisor on the target host resumes the VM.

- **Postcopy:**

  In the basic approach, post-copy first suspends the migrating VM at the source node, copies minimal processor state to the target node, resumes the virtual machine, and begins fetching memory pages over the network from the source. The manner in which pages are fetched gives rise to different variants of post-copy, each of which provides incremental improvements. Post-copy ensures that each page is sent over the network only once, unlike in pre-copy where repeatedly dirtied pages could be resent multiple times.

  Variants of Post Copy

  i. **Post-copy via Demand Paging:** The demand paging variant of post-copy is the simplest and slowest option. Once the VM resumes at the target, its memory accesses result in page faults that can be serviced by requesting the referenced page over the network from the source node. However, servicing each fault will significantly slow down the VM due to the networks round trip latency. Consequently, even though each page is transferred only once, this approach considerably lengthens the resume time and leaves long term residual dependencies in the form of unfetched pages, possibly for an indeterminate duration. Thus, post-copy performance for this variant by itself would be unacceptable both from the viewpoint of total migration time and application degradation.

  ii. **Post-Copy via Active Pushing:** One way to reduce the duration of residual dependencies on the source node is to pro actively push the VMs pages from the source to the target even as the VM continues executing at the target. Any major faults incurred by the VM can be serviced concurrently over the network via demand paging. Active push avoids transferring pages that have already been faulted in by the target VM. Thus, each page is transferred only once, either by demand paging or by an active push.

iii. **Post-Copy via Pre-paging:** The goal of post-copy via pre-paging is to anticipate the occurrence of major faults in advance and adapt the page pushing sequence to better reject the VMs memory access pattern. While it is impossible to predict the VMs exact faulting behavior, some approaches work by using the faulting addresses as hints to estimate the spatial locality of the VMs memory access pattern. The pre-paging component then shifts the transmission window of the pages to be pushed such that the current page fault location falls within the window. This increases the probability that the pushed pages would be the ones accessed by the VM in the near future, reducing the number of major faults.

Post-copy is preferred over pre-copy because

- In post copy live migration, downtime is less as compared to pre-copy live migration.
- Post-copy live migration ensures that each page is transferred only once whereas in pre-copy, pages are migrated iteratively till the number of dirtied pages remains very less. So, the network traffic is reduced to a large extent in case of post-copy as compared to pre-copy.

2.2.2 Techniques of Live VM Migration across subnet

1. **VXLAN**:

   VXLAN **(Virtual eXtensible Local Area Network)** is a proposed encapsulation protocol for running an Layer 2 overlay network on existing Layer 3 infrastructure. Overlay network can be defined as any logical network that is created on top of the existing physical networks. VXLAN creates **Layer 2 logical networks** on top of the IP network.

The intention of using VXLAN is to convert the problem of migrating VMs across subnets to the problem of migrating VMs within a subnet. In other words, the problem of layer-3 migration is converted into a problem of layer-2 migration with the help of VXLAN.

2. **GRE Tunneling**

   Generic Routing Encapsulation (GRE), is a simple IP packet encapsulation protocol. A GRE tunnel is used when IP packets need to be sent from one network to another, without being parsed or treated like IP packets by any intervening routers. For example, in Mobile IP, a mobile node registers with a Home Agent. When the mobile node roams to a new network, it registers with a Foreign Agent there. Whenever IP packets addressed to the mobile node are received by the Home Agent, they can be relayed over a GRE tunnel to the Foreign Agent for delivery. It does not matter how the Home Agent and Foreign Agent communicate with each other – hops in between just pass along the GRE packet. Only the GRE tunnel endpoints – the two Agents – actually route the encapsulated IP packet.

2.2.3 Network Recovery

When VM migration takes place within subnet, the IP address of VM does not change and the existing TCP connections are maintained. In the case of VM migration across subnets, the IP address of VMs after migration are not compatible in the subnet. With the help of VXLAN an overlay layer 2 network is created above the Layer 3 network and hence the IP address of the VM becomes compatible in the new subnet.

2.3 Hypervisor

Virtualization, in cloud based solutions, is the key enabler where multiple applications are co-hosted on a single machine. On such virtualized servers, physical hardware is shared among multiple virtual machines by a layer called hypervisor. A hypervisor or virtual machine monitor (VMM) is computer software, firmware or hardware that creates and runs virtual machines. A computer on which a hypervisor runs one or more virtual machines is called a host machine, and each virtual machine is called a guest machine. The hypervisor presents the guest operating systems with a virtual operating platform and manages the execution of the guest operating systems. Applications hosted in virtual machine access the shared hardware through the hypervisor. Hypervisor manages access to the physical resources, maintaining isolation between VMs at all times. The decoupling between physical and virtual resources provided by the hypervisors enable flexibility of resource provisioning for the VMs. There are 2 types of hypervisors currently existing,

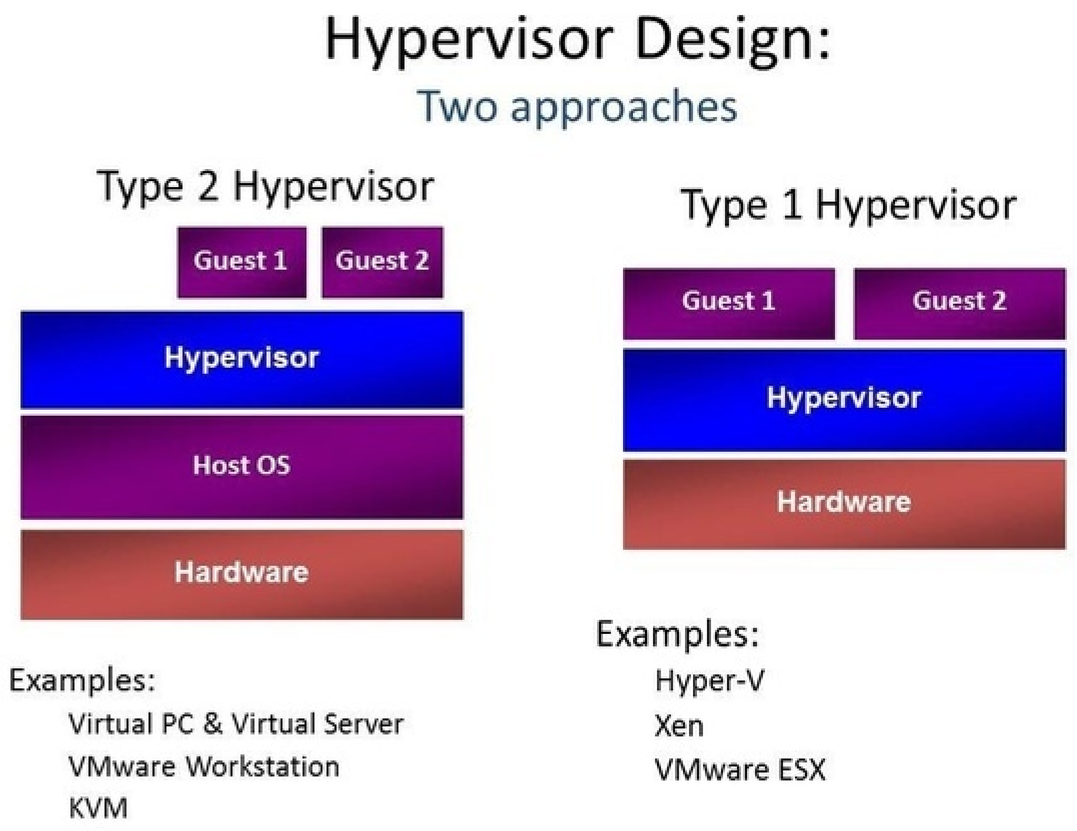
- Type 1 hypervisor, also called bare-metal hypervisor, runs directly on top of the hardware. There is no software/OS layer in between the hardware and the hypervisor.

- Type 2 hypervisor, which operates as an application on top of an existing operating system just as other computer programs.

  Because they run directly on the hardware, Type 1 hypervisors support hardware virtualization. Because they run as an application on top of an operating system, Type 2 hypervisors perform software virtualization.

  A Type 1 hypervisor provides better performance and greater flexibility because it operates as a thin layer designed to expose hardware resources to virtual machines (VMs), reducing the overhead required to run the hypervisor itself.

  Typically, a Type 1 hypervisor is more efficient than a Type 2 hypervisor, yet in many ways they both provide the same type of functionality because they both run the same kind of VMs. In fact, you can usually move a VM from a host server running a Type 1 hypervisor to one running a Type 2 hypervisor and vice versa.



2.4 Kernel Virtual Machine (KVM)

KVM (Kernel Virtual Machine) is a Linux kernel module that allows a user space program to utilize the hardware virtualization features of various processors. Today, KVM supports recent Intel and AMD processors. By itself, KVM does not perform any emulation. Instead, it exposes the /dev/kvm interface, which a userspace host can then use to:

- Set up the guest VM's address space. The host must also supply a firmware image (usually a custom BIOS when emulating PCs) that the guest can use to bootstrap into its main OS.

- Feed the guest simulated I/O.
- Map the guest's video display back onto the host.

2.5 QEMU

QEMU (short for Quick Emulator) is a free and open-source hosted hypervisor of type 2, which performs hardware virtualization. QEMU can save and restore the state of the virtual machine with all programs running. To emulate more than just the processor, QEMU includes a long list of peripheral emulators: disk, network, VGA, PCI, USB, serial/parallel ports, etc.

KVM by itself cannot provide the complete virtualization solution. It needs QEMU to provide full hypervisor functionality. By itself, KVM is more of a virtualization infrastructure provider. Hence, QEMU is used along with KVM, called QEMU-KVM in the hosts to provide virtualization. KVM supports hardware virtualization to provide near native performance to the guest operating systems. On the other hand, QEMU emulates the target operating system.

2.6 Libvirt

Libvirt provides a hypervisor-agnostic API to securely manage guest operating systems running on a host. Libvirt isn't a tool per se but an API to build tools to manage guest operating systems. Libvirt itself is built on the idea of abstraction. It provides a common API for common functionality that the supported hypervisors implement. Libvirt was originally designed as a management API for Xen, but it has since been extended to support a number of hypervisors. Some of the examples of the use of libvirt virtualization API include virt-manager and virsh(virtualization shell).

- **Virtual Machine Manager:**

The virt-manager application is a user interface for managing virtual machines through libvirt. It primarily targets KVM VMs, but also manages Xen and LXC(linux containers). It presents a summary view of running domains(guest operating system),, their live performances and resource utilization statistics. Wizards enable creation of new domains and configuration and adjustment of a domains resource allocation and virtual hardware.

- **Virsh:**

  Virsh or virtualization shell is built on top of libvirt and it permits the use of much of the libvirt functionality but in an interactive(shell based) fashion. Libvirt and Virsh not only allow us to manage VMs running on our own system, but help us control VMs running on remote systems or a cluster of physical machines. When we use virsh we need to specify some sort of URI to tell libvirt which sets of virtual machines we want to control.

  For example, we want to control a XEN virtual machine running on a remote server called myserver.com. When using virsh, we can refer to that VM by providing an URI like xen+ssh://root@myserver.com, indicating that we want to use ssh to connect as root to the server myserver.com, and control xen virtual machines running there.

  With libvirt, you have two distinct means of control. The first is where the management application and domains exist on the same node. In this case, the management application works through libvirt to control the local domains. The other means of control exist when the management application and the domains are on separate nodes. In this case, remote communication is required. This mode uses a special daemon called libvirtd that runs on remote nodes. This daemon is started automatically when libvirt is installed on a new node and can automatically determine the local hypervisors and set up drivers for them. The management application communicates through the local libvirt to the remote libvirtd through a custom protocol. For QEMU, the protocol ends at the QEMU monitor. QEMU includes a monitor console that allows you to inspect a running guest operating system as well as control various aspects of the virtual machine (VM).

2.7 Network File System (NFS)

The Network File System (NFS) is a client/server application that lets a computer user view and optionally store and update files on a remote computer as though they were on the users own computer. The NFS protocol is one of several distributed file system standards for network-attached storage (NAS). NFS allows the user to mount (designate as accessible) all or a portion of a file system on a server. The portion of the file system that is mounted can be accessed by clients with whatever privileges are assigned to each file (read-only or read-write). NFS uses Remote Procedure Calls (RPC) to route requests between clients and servers. In our project, we have used NFS as a shared storage between two hosts which is used to store the disk image file, the memory states of the OS etc. Therefore after VM migration the host retrieves the information related to VM from the disk image to boot the VM, memory state information etc. from the remote computer used for NFS storage. Therefore, with the help of NFS, the information related to VM is accessible to both the hosts independently.

2.8 VXLAN

A VLAN (Virtual Local Area Network) is a group of devices on one or more LANs that are configured to communicate as if they were attached to the same wire, when in fact they are located on a number of different LAN segments. VLANs define broadcast domains in a Layer 2 network. As its name indicates, VXLAN (Virtual eXtensible Local Area Network) is designed to provide the same Ethernet Layer 2 network services as VLAN does today, but with greater extensibility and flexibility. VXLAN technology allows you to segment your networks (as VLANs do), but it provides benefits that VLANs cannot. The advantages of VXLAN are:

- Flexible placement of multitenant segments throughout the data center: It provides a solution to extend Layer 2 segments over the underlying shared network infrastructure so that tenant workload can be placed across physical pods in the data center.

- Higher scalability to address more Layer 2 segments: VLANs use a 12-bit VLAN ID to address Layer 2 segments, which results in limiting scalability of only 4094 VLANs. VXLAN uses a 24-bit segment ID known as the VXLAN network identifier (VNID), which enables up to 16 million VXLAN segments to coexist in the same administrative domain.

- Better utilization of available network paths in the underlying infrastructure: VLAN uses the Spanning Tree Protocol for loop prevention, which ends up not using half of the network links in a network by blocking redundant paths. In contrast, VXLAN packets are transferred through the underlying network based on its Layer 3 header and can take complete advantage of Layer 3 routing, equal-cost multipath (ECMP) routing, and link aggregation protocols to use all available paths.

2.8.1 VXLAN Terminology

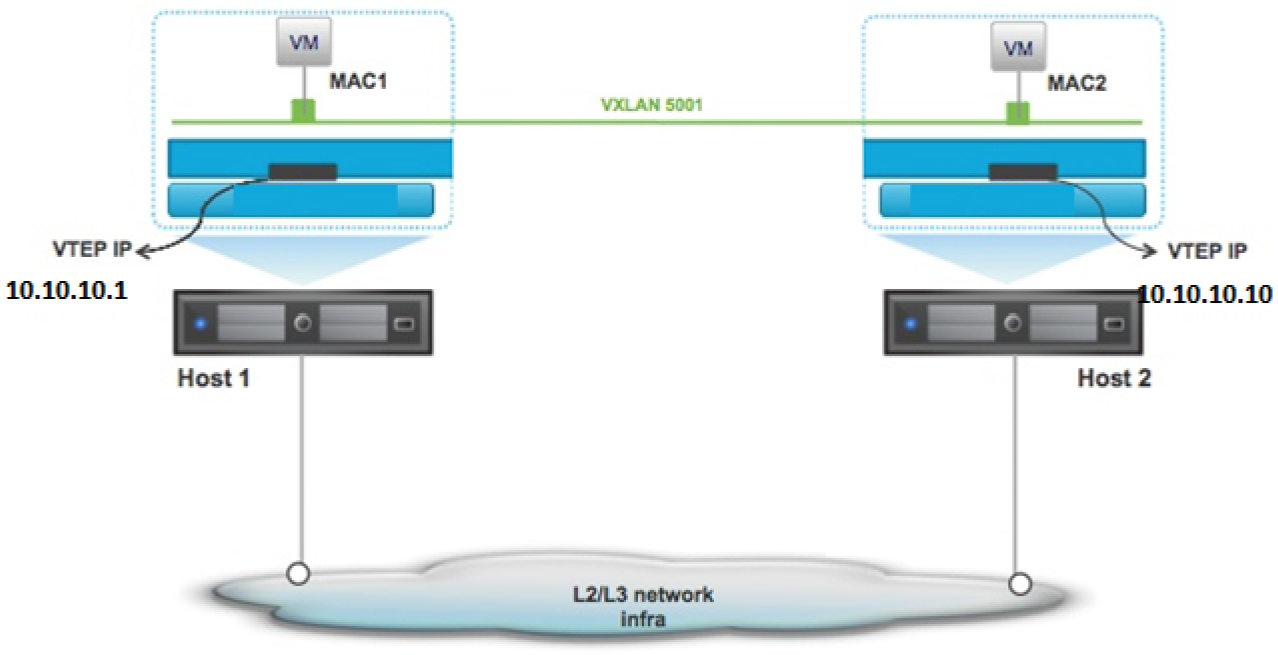
- **VXLAN gateway:** A VXLAN gateway bridges traffic between VXLAN and non-VXLAN environments. The BIG-IP system uses a VXLAN gateway to bridge a traditional VLAN and a VXLAN network, by becoming a virtual network endpoint.

- **VXLAN segment:** A VXLAN segment is a Layer 2 overlay network over which VMs communicate. Only VMs within the same VXLAN segment can communicate with each other.

- **VNI:** The Virtual Network Identifier (VNI) is also called the VXLAN segment ID. The system uses the VNI to identify the appropiate tunnel.

- **VTEP:** The VXLAN Tunnel Endpoint (VTEP) originates or terminates a VXLAN tunnel. The same local IP address can be used for multiple tunnels.

- **VXLAN header:** In addition to the UDP header, encapsulated packets include a VXLAN header, which carries a 24-bit VNI to uniquely identify Layer 2 segments within the overlay.

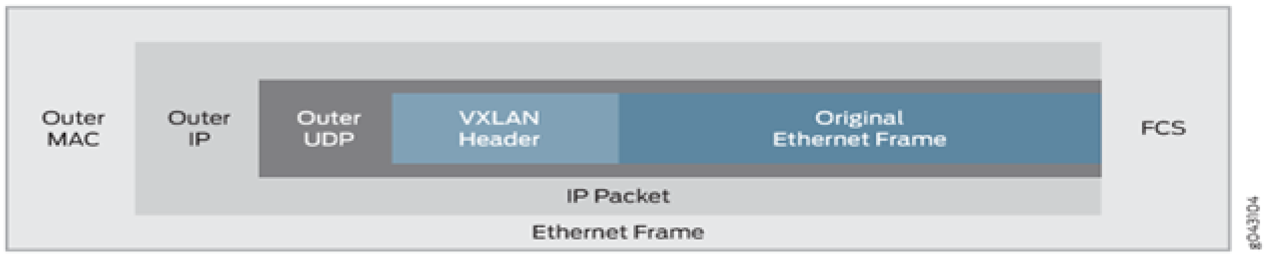


2.8.2 How does VXLAN work?

VXLAN is often described as an overlay technology because it allows you to stretch Layer 2 connections over an intervening Layer 3 network by encapsulating (tunneling) Ethernet frames in a VXLAN packet that includes IP addresses. Devices that support VXLANs are called virtual tunnel endpoints (VTEPs). They can be end hosts or network switches or routers. VTEPs encapsulate VXLAN traffic and decapsulate that traffic when it leaves the VXLAN tunnel. To encapsulate an Ethernet frame, VTEPs add a number of fields, including the following fields:

1. Outer media access control (MAC) destination address (MAC address of the tunnel endpoint VTEP)

2. Outer MAC source address (MAC address of the tunnel source VTEP)
3. Outer IP destination address (IP address of the tunnel endpoint VTEP)
4. Outer IP source address (IP address of the tunnel source VTEP)
5. Outer UDP header
6. Frame Check Sequence



2.8.3 VXLAN Encapsulation and Packet Format

VXLAN defines a MAC-in-UDP encapsulation scheme where the original Layer 2 frame has a VXLAN header added and is then placed in a UDP-IP packet. With this MAC-in-UDP encapsulation, VXLAN tunnels Layer 2 network over Layer 3 network. VXLAN introduces an 8-byte VXLAN header that consists of a 24-bit VNI and a few reserved bits. The VXLAN header together with the original Ethernet frame goes in the UDP payload. The 24-bit VNI is used to identify Layer 2 segments and to maintain Layer 2 isolation between the segments.

Frame encapsulation and decapsulation is performed by a VXLAN tunnel endpoint (VTEP). A VTEP originates and terminates VXLAN tunnels. A Layer 2 VXLAN gateway is a VTEP endpoint which provides encapsulation of VLAN traffic from a physical network to a VXLAN based virtual network, and decapsulation of VXLAN traffic from the virtual network to VLAN traffic in the physical network.

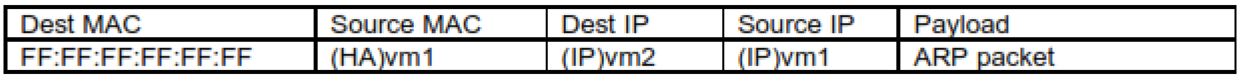
A VTEP has two logical interfaces: an uplink and a downlink. The uplink is responsible for receiving VXLAN frames and acts as a tunnel endpoint with an IP address used for routing VXLAN encapsulated frames. VTEP functionality can be implemented in software such as a virtual switch or in the form a physical switch.

VXLAN frames are sent to the IP address assigned to the destination VTEP; this IP is placed in the Outer IP Destination Address (DA). The IP of the VTEP sending the frame resides in the Outer IP Source Address (SA). Packets received on the uplink are mapped from the VXLAN ID to a VLAN and the Ethernet frame payload is sent as an 802.1Q Ethernet frame on the downlink. During this process the inner MAC SA and VXLAN ID is learned in a local table. Packets received on the downlink are mapped to a VXLAN ID using the VLAN of the frame. A lookup is then performed within the VTEP L2 table using the VXLAN ID and destination MAC; this lookup provides the IP address of the destination VTEP. The frame is then encapsulated and sent out the uplink interface.

2.8.4 VM to VM Communication with VXLAN

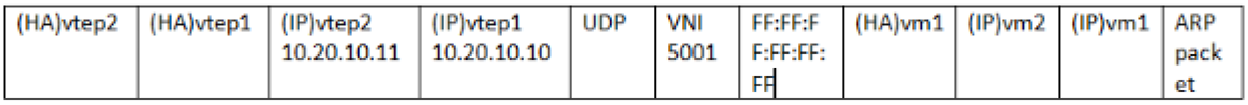
Suppose VM1 wants to communicate with VM2.

1. VM1 would first send ARP request to know MAC address of VM2.



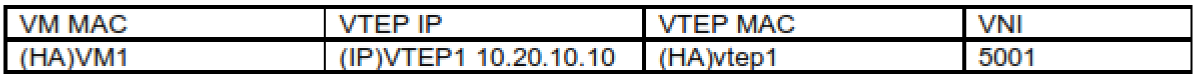
Dest MAC	Source MAC	Dest IP	Source IP	Payload
FF:FF:FF:FF:FF:FF	(HA)vm1	(IP)vm2	(IP)vm1	ARP packet

2. This packet will be received by VTEP1. VTEP1 will add VXLAN headers. VTEP will check if it has the MAC address of VM2. Since the VTEP table is currently empty, it will forward the packet to all the machine to which VXLAN tunnel has been created.We have created a tunnel to machine VTEP2 (IP 10.20.10.11). It will put the destination ip of VTEP2 10.20.10.11 and the MAC address of VTEP2 will be known by simple ARP (Address Resolution Protocol) packet which we have studied in Networking. After knowing the MAC address of VTEP2, the packet sent by VTEP1 will be as shown below:



(HA)vtep2	(HA)vtep1	(IP)vtep2 10.20.10.11	(IP)vtep1 10.20.10.10	UDP	VNI 5001	FF:FF:FF:FF:FF	(HA)vm1	(IP)vm2	(IP)vm1	ARP packet

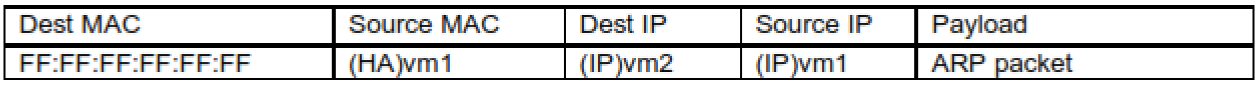
3. When VTEP2 receives this packet, it will first learn from the packet received and populate its forwarding table as shown below :



VM MAC	VTEP IP	VTEP MAC	VNI
(HA)VM1	(IP)VTEP1 10.20.10.10	(HA)vtep1	5001

Since VTEP2 contains VMs in VNI 5001, it will decapsulate the VXLAN headers and send the ARP packet from VM1 to all the VMs having VNI as 5001.

4. VM2 will receive the ARP request shown in Figure 1 from VM1.



Dest MAC	Source MAC	Dest IP	Source IP	Payload
FF:FF:FF:FF:FF:FF	(HA)vm1	(IP)vm2	(IP)vm1	ARP packet

It will send ARP reply packet to VM1. The packet from VM2 will look like below:

Dest MAC	Source MAC	Dest IP	Source IP	Payload
(HA)vm1	(HA)vm2	(IP)vm1	(IP)vm2	ARP Packet

5. This packet will be received by vtep2 and encapsulated into vxlan header as shown below:

Dest MAC	Source MAC	Dest IP	Source IP		VXLAN	Dest VM MAC	Source VM MAC	Dest VM IP	Source VM IP	Payload
(HA)vtep1	(HA)vtep2	(IP)vtep1	(IP)vtep2	UDP	VNI 5001	(HA)vm1	(HA)vm2	(IP)vm1	(IP)vm2	ARP Packet

6. This packet will be received by VTEP1 .VTEP1 will learn the details about VM2 and populate its forwarding table. The packet will again be decapsulated as shown in step3.
   The remaining packet will be forwarded to all the VMs in VNI=5001. The packet received by VM1 will be as shown below and it will learn the MAC address of VM2((HA) vm2).

Dest MAC	Source MAC	Dest Ip	Source IP	Payload
(HA)vm1	(HA)vm2	(IP)vm1	(IP)vm2	ARP Packet

   Now VM1 can communicate with VM2 since it knows the MAC address of VM2.

2.8.5 Flow of packets in case of Migration of VM1 from VTEP1 to VTEP2

Now VM1 is being migrated from VTEP1 to VTEP2.

1. Suppose VM1 pings another machine on same Layer 2 domain say, VM3. VM1 will learn the MAC address of VM3 as explained in previous section.
   Packet sent by VM1 will be :

Dest MAC	Source MAC	Dest IP	Source IP	Payload
(HA)VM3	(HA)VM1	(HA)VM3	(HA)VM1	Ping packet

2. This packet will be encapsulated by VTEP1 as below and sent into network:

Dest MAC	Source MAC	Dest IP	Source IP		VXLAN	Dest VM MAC	Source VM MAC	Dest VM IP	Source VM IP	Payload
(HA)vtep3	(HA)vtep1	(IP)vtep3	(IP)vtep1	UDP	VNI 5001	(HA)vm3	(HA)vm1	(IP)vm3	(IP)vm1	ICMP request

3. Suppose after sending this packet, VM1 has migrated to VTEP2. After reaching to VTEP2, the hypervisor in VTEP2 sends RARP packet to update the tables of VTEPs connected to this host through tunnel. Therefore the RARP packet will be sent to all the machines connected to this VTEP2 through VXLAN tunnel. The RARP packet is required for updating the forwarding tables of VTEPs.
   The packet sent by VTEP2 will be as shown below:

Dest MAC	Source MAC	Dest IP	Source IP		VXLAN	Dest VM MAC	Source VM MAC	Payload
(HA)vtep1	(HA)vtep2	(IP)vtep1	(IP)vtep2	UDP	VNI 5001	FF:FF:FF:FF:FF:FF	(HA)VM1	RARP packet

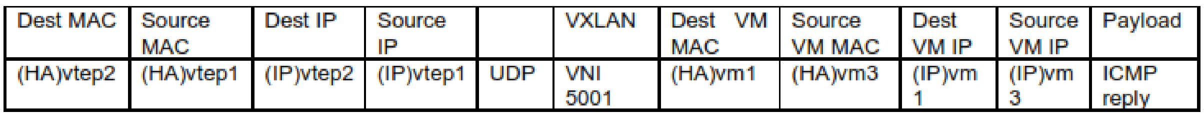
4. On receiving this packet, VTEP1 will update its table by including below entry :

VM Mac	VTEP IP	VTEP MAC	VNI
(HA)VM1	(IP)VTEP2 10.20.10.11	(HA)VTEP2	5001

5. Whe ping reply comes for the previous ping request sent on the VTEP1(old host), it will receive packet as shown below:

Dest MAC	Source MAC	Dest IP	Source IP		VXLAN	Dest VM MAC	Source VM MAC	Dest VM IP	Source VM IP	Payload
(HA)vtep1	(HA)vtep3	(IP)vtep 1	(IP)vtep 3	UDP	VNI 5001	(HA)vm 1	(HA)vm3	(IP)vm1	(IP)vm 3	ICMP reply

   VTEP1 will process this packet as the Dest MAC in the received packet is MAC of VTEP1. Now it will see that it has an entry for this VM1 in its forwarding table and hence it will add new VXLAN headers to send the packet to new host machine (VTEP2).VTEP1 will get the IP and MAC of VTEP2 from its forwarding table. The packet sent by VTEP1 in the network will be as shown below:



Dest MAC	Source MAC	Dest IP	Source IP		VXLAN	Dest VM MAC	Source VM MAC	Dest VM IP	Source VM IP	Payload
(HA)vtep2	(HA)vtep1	(IP)vtep2	(IP)vtep1	UDP	VNI 5001	(HA)vm1	(HA)vm3	(IP)vm 1	(IP)vm 3	ICMP reply

6. VTEP2 will receive the above packet and it will decapsulate it and send it to VM1 in VNI 5001.
   Hence the replies of ping request which were sent from old host (VTEP1) when received on VTEP1 are being forwarded to VTEP2.
   And after the migration , when the ping requests are being sent from new host(VTEP2), the reply will be sent back to VTEP2 as the source IP in the ping request packet in that case would be of VTEP2.

2.9 Open vSwitch

When a VM is created, it is hosted on a hypervisor, which connects the guest OS to the host OS. Applications on the VM write to a vNIC (they think it is a real NIC). Packets go through the TCP/IP stack on the guest OS first. The hypervisor creates a ”tap” device to receive all these packets. The hypervisor gets these L2 frames and transmits them via a software switch / bridge.

Open vSwitch (OVS) is an open-source project that allows hypervisors to virtualize the networking layer. This caters for the large number of virtual machines running on one or more physical nodes. The virtual machines connect to virtual ports on virtual bridges (inside the virtualized network layer.)

A software switch functions similar to a hardware switch, but works as a kernel module. For example, the Linux Bridge or the more recent Open vSwitch (OVS). Each of the vNICs of the VMs plug into one of the virtual ports on the software switch / bridge. Typically, one port also connects to a real physical NIC to send packets out. The software switch works much like a real switch, and forwards packets to other ports or via the physical interface. Packets then undergo L2 processing at the physical NIC / device driver.

This is very similar to a physical server connecting to physical ports on a Layer 2 networking switch. These virtual bridges then allow the virtual machines to communicate with each other on the same physical node. These bridges also connect these virtual machines to the physical network for communication outside the hypervisor node. Open vSwitch supports integration into virtual environments and allows for switch distribution, its interfaces are exported for manipulating the forwarding state and managing configuration state at runtime.

Open vSwitch is configured inside both the hosts in KVM hypervisor with virtual bridges to allow the VM traffic as well as the host traffic to reach the outside world. It operates like a basic L2 switch in its standalone configuration. Open vSwitch also allows the migration of tunneling rules that can be used to support seamless migration between different IP subnets. Therefore, Open vSwitch along with VXLAN tunnel allows us to maintain the network connectivity across subnets while the VM is migrating.

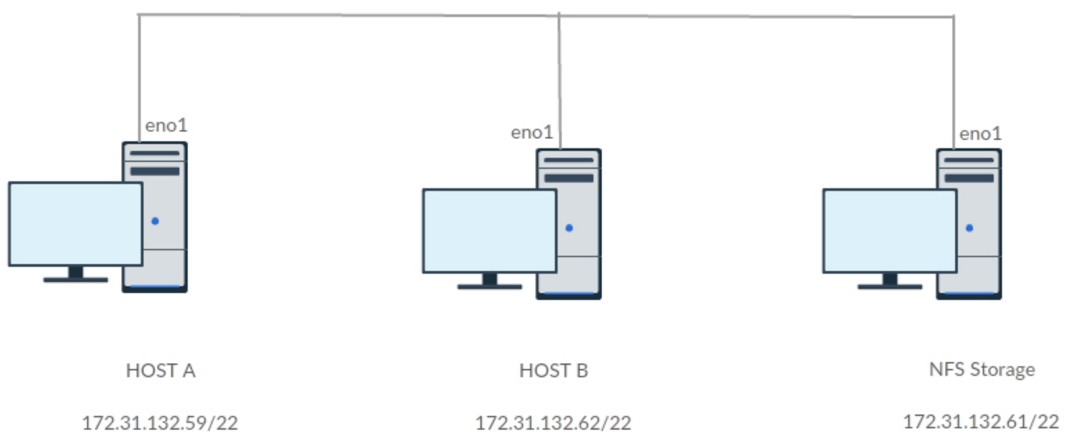
We have created the following bridge in Host A and Host B to support migration through VXLAN tunnel:

- br-phy
  - br-phy:Internal Port
  - vx1:Port for VXLAN bidirectional tunnel with remote IP as the IP address of the second host (172.31.131.123, in our case). All the VM traffic passes through this port.
  - vnet0:Port for virtual machine network interface
  - eno1

Chapter 3

Experimental Setup and Results Analysis

Initially, for our mini project we decided to simulate the live VM migration across the same subnet. This was made successful by the integration of various tools. We decided to use Libvirt as our Virtual Machine manager. Along with this, we also utilized the KVM hypervisor and QEMU emulator.

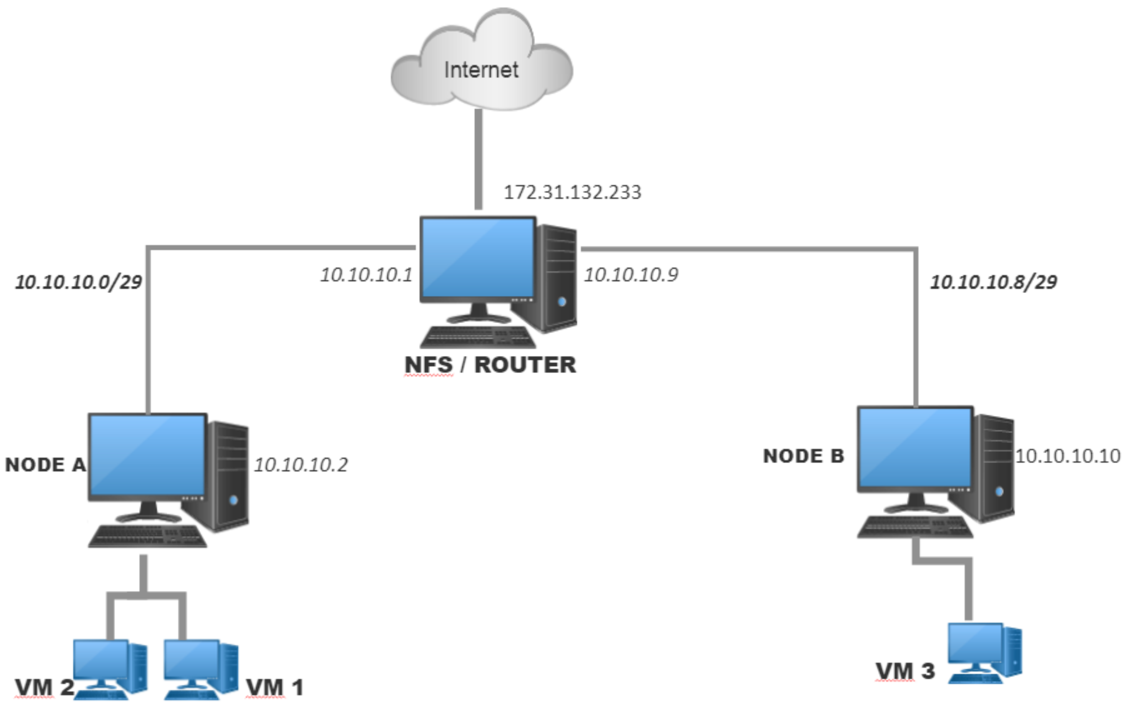


Live VM migration within same subnet

As seen in the diagram above, VM1 resides on host one, and VM2 resides on host two. On VM1, we ping the college proxy, 172.31.100.14. Now, when we migrate

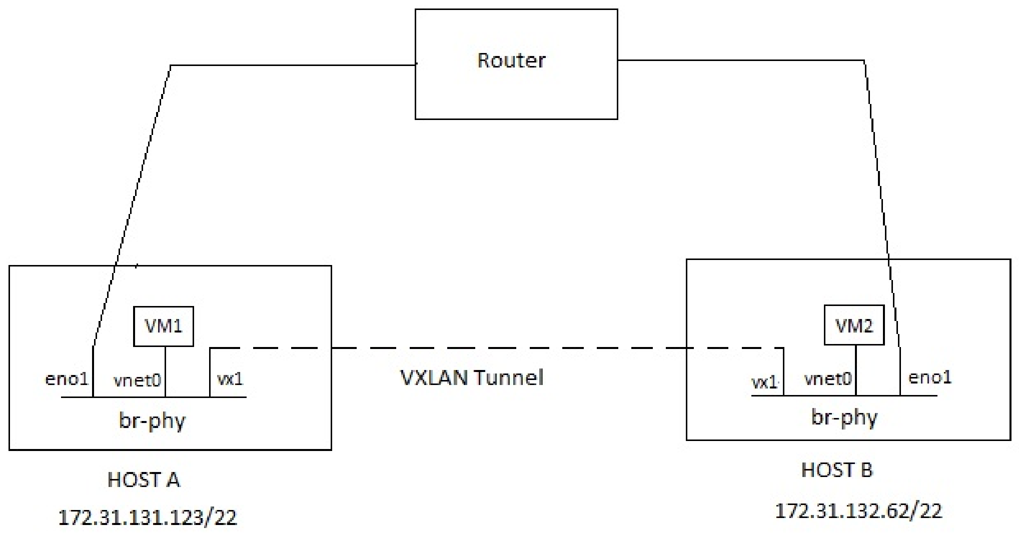
VM1 from node one to two, keeping the ping alive, we notice that VM1 detaches itself from node one and migrates to node two. Once VM1 boots on node two, all future pings are carried out from node two.

Next, we decided to take this one step further and conduct live VM migration across different subnets. This involved the integration of the above used tools used in the NAT configuration mode. There is also the addition of router which we configured one of our systems to mimic.



Live VM Migration across different subnets using NAT Mode

As we see in the diagram above, VM1 and VM2 reside on node A and VM3 resides on node B. Another node is configured as the NFS, which also happens to be our router. In the NAT configuration mode, when on VM1, we ping the college proxy, 172.31.100.14 and migrate VM1 from node A to node B, we notice that VM1 detaches itself from node one and migrates to node two. Once VM1 boots on node two, all future pings are carried out from node two. A thing to note here is that node A and node B replace the header information with their own credentials which was originally the VMs.



Live VM Migration across different subnets using Bridge mode

Finally, we now conduct the same live VM migration across different subnets but only this time in bridged mode. As we see in the diagram above, VM1 reside on node A and VM2 resides on node B. Another node is configured as the NFS. vnet0 (Virtual Network Interface of VM) is connected to br-phy which connects to the physical NIC eno1. A VXLAN tunnel is created between 2 nodes namely node A and node B with terminal end points vx1. The port eno1 connects the node to the Internet. All VM-VM communication is done through ports vx1 (VXLAN tunnel). Thus when on VM1, we ping the college proxy, 172.31.100.14 and migrate VM1 from node A to node B, we notice that VM1 detaches itself from node one and migrates to node two. Once VM1 boots on node two, all previous pending ping replies are redirected from node A to node B via the VXLAN tunnel.

3.1 Prerequisites

1. Make sure that the system has the hardware virtualization extensions: For Intel-based hosts, verify the CPU virtualization extension [vmx] are available using following command:

   - grep -e vmx /proc/cpuinfo.

2. Verify that KVM modules are loaded in the kernel by following command:

   - lsmod — grep kvm

   The output should contain kvm intel for intel based hosts and kvm amd for amd-based hosts

3. Make sure that the system is up-to-date.

   - yum update

4. Set Selinux in Permissive mode through this command:

   - setenforce 0

3.2 Installation and Deployment of KVM

1. We will install qemu-kvm and qemu-img packages at

   rst. These packages provide the user-level KVM and disk image manager.

   - yum install qemu-kvm qemu-img.

2. Install virt-manager, libvirt-client, virt-install, libvirt using this command:

   - yum install virt-manager libvirtd libvirt-python libvirt-client

3. For Centos7 ,install additional package groups such as: Virtualization Client, Virtualization Platform and Virtualization Tools.

- yum groupinstall virtualization-client virtualization-platform virtualizationtools

4. Restart the virtualization daemon ”libvirtd” which manage all of the platform.

    - systemctl restart libvirtd

5. After restarting the daemon, then check its status by running following command:

    - systemctl status libvirtd

    It should show the status as active.

3.3 Installation of NFS Server

1. Configure the firewall.Open the SSH and NFS ports to ensure that the client will be able to connect to nfs server

    - firewall-cmd –permanent –add-service=nfs
    - firewall-cmd –permanent –add-service=mountd
    - firewall-cmd –permanent –add-service=rpc-bind
    - firewall-cmd –reload

2. Install NFS pacakages.

    - yum -y install nfs-utils

3. Uncomment the following line in /etc/sysconfig/nfs

    - SECURE NFS=”yes”

4. Open /etc/idmapd.conf in a text editor and uncomment the below line and change the domain to match the one used within the rest of your network.

    - Domain= localdomain

5. Now we export our NFS shares.Open /etc/exports in a text editor and add this

    - /home 10.10.10.0/29(rw_sync,no_root_squash,no_subtree_check)
    - /home 10.10.10.8/29(rw_sync,no_root_squash,no_subtree_check)

    Note that 10.10.10.0/29 is the network of Host1 server and 10.10.10.8/29 is the network of Host2.

6. Run this command to make the changes effective.

    - exportfs -a

7. Activate the NFS services at boot

    - systemctl start rpcbind nfs-server
    - systemctl enable rpcbind nfs-server

3.4 Mounting the NFS Share on the Client

1. Create the NFS directory mount point as follows

    - mkdir -p /mnt/nfs/home

2. Mount the NFS shared content in the client machine

    - mount -t nfs 172.31.132.62:/home /mnt/nfs/home/

    It will mount /home of NFS server.

3. To check whether the system is connected to NFS server, run this command

    - df-kh

3.5 Configuration of Open vSwitch

The environment assumes the use of two hosts, named host1 and host2. We only detail the configuration of Host1 but a similar configuration can be used for Host2. Both hosts should be configured with Open vSwitch, QEMU/KVM and suitable VM images. Open vSwitch should be running before proceeding. Perform the folowing configuration on host1:

1. Put the following xml into a file (e.g. /tmp/ovsnet.xml):

   ```
   ⟨ network ⟩
   ⟨ name ⟩ ovs-br0 ⟨ / name ⟩
   ⟨ forward mode='bridge'/ ⟩
   ⟨ bridge name='br-phy'/ ⟩
   ⟨ virtualport type='openvswitch'/ ⟩
   ⟨ /network ⟩
   ```

2. Create a br-phy bridge:
   - ovs-vsctl add-br br-phy
3. Add the ethernet port to this bridge. In our case it is eno1.
   - ovs-vsctl add-port br-phy eno1
4. Disable eno1 and configure the bridge br-phy created with the IP address of eno1.
   - ifconfig eno1 0 && ifconfig br-phy 172.31.132.62 netmask 255.255.252.0
5. Add route entry:
   - route add default gw 172.31.132.1 br-phy
6. Add vxlan port on bridge br-phy.
   - ovs-vsctl add-port br-phy vx1 – set interface vx1 type=vxlan options:remote_ip=172.31.131.123

7. Define and start the network:

    - virsh net-define /tmp/ovsnet.xml
    - virsh net-start ovs-br0
    - virsh net-autostart ovs-br0

3.6 Creation of Virtual Machines

1. Open virt-manager by writing the following command in terminal

    - virt-manager

2. From "File" tab, select "Add Connection".
3. Check "Connect to remote host" option and then provide Hostname/IP of the remote server.
4. Now create a Volume Disk for the virtual machine.
5. Right click on the connection and select "Details" and then select "Storage" tab.
6. Add a new netfs type Storage Pool by clicking on + sign on bottom-left corner. Make sure the name of storage pool should be same on the client machines on which live migration has to take place.
7. Enter the target path for the device. The target path must be identical on all host physical machines for migration.
8. Enter the hostname or IP address of the NFS server.
9. Enter the NFS path. In our case it will be /home.
10. Now we can add virtual machine to our connections. Click on "VM" button in the main window.

11. Select the installation method for creating the virtual machine.We have used Local media install.

12. Now its time to specify which Local install media to be used, we have two options:

    - From physical [CDROM/DVD].
    - From ISO image.

    We have used ISO image method.

13. Select the virtual netfs directory which we had created to store the VM's image.

14. Finally enter the name of virtual machine and click Finish.

15. Select the Network mode to be the network created locally by you.In our case it is ovs-br0.

16. Click on Finish Installation

17. Open the VM and ping the proxy 172.31.100.14 from the terminal.

3.7 Migration of virtual machine

1. Make sure the virtual machine is running.

2. Type this command on terminal and hit Enter.

    - virsh migrate –live (guest vm name) qemu+ssh://(destination ip address)/system

3. The running virtual machine will be migrated to destination host.

3.8 Configuring Wireshark to decode VXLAN encapsulated packets

Wireshark on its own, is incapable of decoding and understanding VXLAN packets and its headers. Thus, in order to decode VXLAN packets, it is crucial for us to add the following Lua script.

- Create a file named vxlan.lua and add this to the file do
  ```
  localp_vxlan = Proto (“My”, “VirtualeXtendedLAN”)
  local f_flags = ProtoField.uint8(“My.flags”, “Flags”, base.HEX)
  localf_flag_i = ProtoField.bool (“My.flags.i”, “IFlag”, 8, {“ValidVNITagpresent”, “ValidV
  localf_rsvd1 = ProtoField.uint24 (“My.rsvd1”, “Reserved”, base.HEX)
  localf_vni = ProtoField.uint24 (“My.vni”, “VNI”, base.HEX)
  localf_rsvd2 = ProtoField.uint8 (“My.rsvd2”, “Reserved”, base.HEX)
  p_vxlan.fields = {f_flags, f_flag_i, f_rsvd1, f_vni, f_rsvd2}
  functionp_vxlan.dissector (buf, pinfo, root)
  localt = root : add (p_vxlan, buf (0, 8))
  localf = t : add (f_flags, buf (0, 1))
  f : add (f_flag_i, buf (0, 1))
  t : add (f_rsvd1, buf (1, 3))
  t : add (f_vni, buf (4, 3))
  t : add (f_rsvd2, buf (7, 1))
  t : append_text (“, VNI : 0x”..string.format (“buf (4, 3) : uint ()))
  localeth_dis = Dissector.get (“eth”)
  eth_dis : call (buf (8) : tvb () , pinfo, root)
  end
  localudp_encap_table = DissectorTable.get (“udp.port”)
  udp_encap_table : add (4789, p_vxlan)
  end
  ```

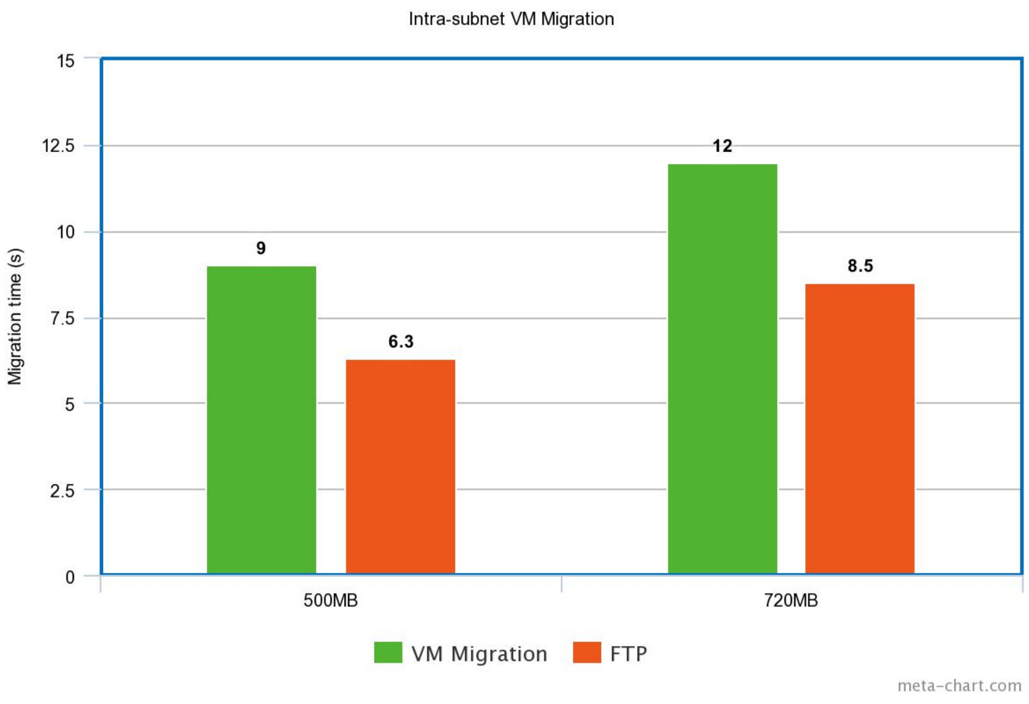
- To enable this dissector we need to do the following.

  1. Open Wireshark

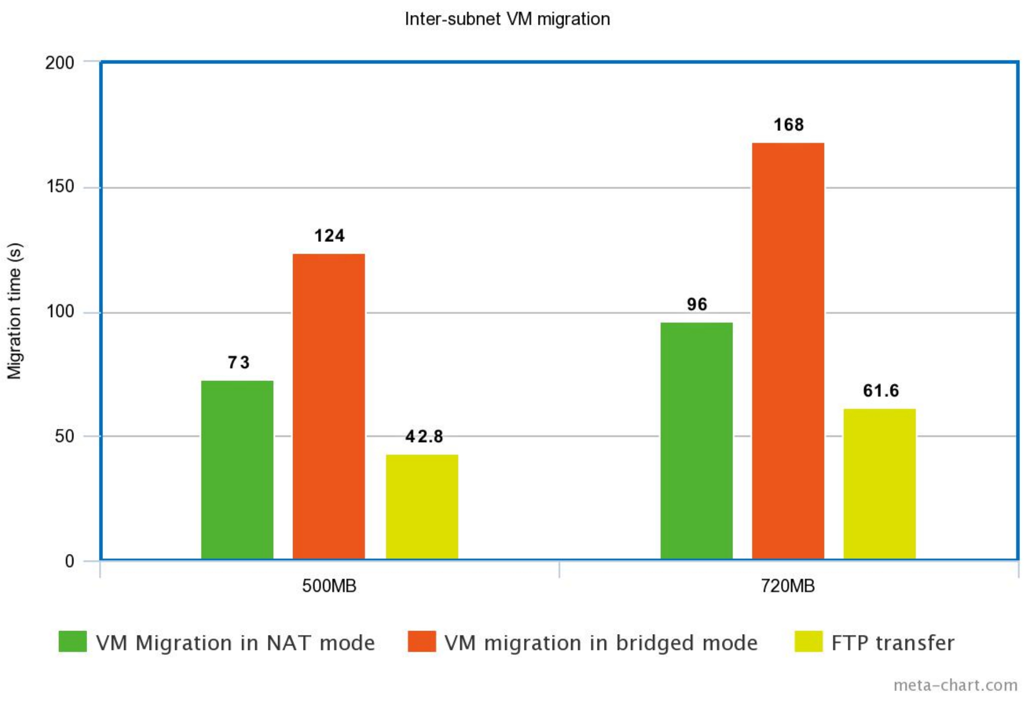
2. Go to Help—About—Folders
3. Look for "Personal Plugins" and "Global Plugins"
4. Create a file named vxlan.lua in either of those two locations
5. Restart Wireshark
6. Finally, go to Help—About—Plugins and verify vxlan.lua is listed.

3.9 Bandwidth Calculation and Analysis

The following graphs were obtained while performing analysis of live migrations in the cases mentioned. Migration time is the total time required by the VM to transfer all the pages and states and resume at the new host. The downtime is calculated in terms of the number of packets lost as observed by the ping sequence numbers.



Intra-subnet Live VM migration



Inter-subnet Live VM migration

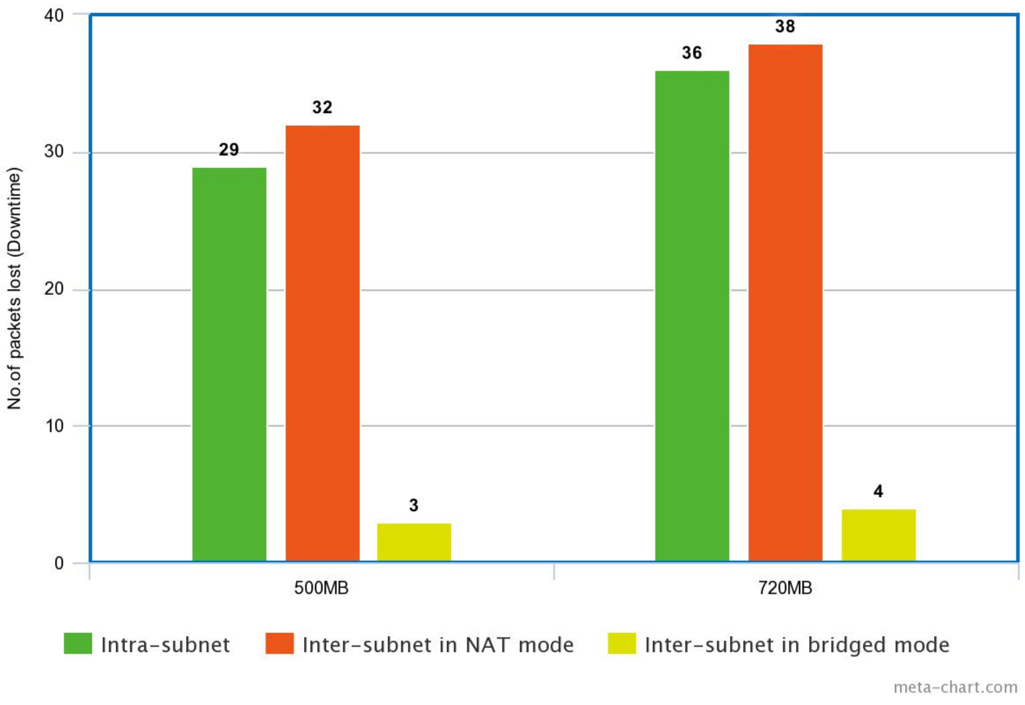
We calculated the network bandwidth using the migration time of our VM in case of intra and inter-subnet migration. It came out as follows:

1. Intra-subnet VM migration
   Bw= 500/9 = 55.56 MBps
   Bw= 720/12= 60.0 MBps
2. Inter-subnet VM migration in NAT mode
   Bw= 500/73= 6.85 MBps
   Bw= 720/96= 7.5 MBps
3. Inter-subnet VM migration in bridged mode
   Bw= 500/124= 4.03 MBps
   Bw= 720/168= 4.28 MBps

We also calculated the bandwidth using FTP transfer of a file of same size as the VM across the same two hosts. The results were as follows:

1. Intra-subnet FTP transfer
   Bw= 500/6.3 = 79.36 MBps
   Bw= 720/8.5 = 84.71 MBps
2. Inter-subnet FTP tranfer
   Bw= 500/42.8 = 11.68 MBps
   Bw= 720/61.6 = 11.69 MBps

The network bandwidth in case of VM migration comes out to be less because while the transfer is happening, the VM is still in running state on the source host, so the dirtied pages are to be repeatedly copied to the destination host, which adds the extra overhead. Moreover, in case of migration with VXLAN in bridged mode, it comes out to be even less as there is an additional overhead because of the addition of VXLAN headers at the source host and their removal at the destination.



Packet Loss during Live VM Migration

The number of packet lost in case of Intra VM Migration and Inter VM Migration using NAT Mode is greater as compared to Inter VM Migration in Bridged mode due to shutting down of the VM, hence the pending replies are received at the original host which are not sent to new host after the migration. In case of Inter VM Migration using Bridged Mode, the pending replies are sent through the VXLAN Tunnel from source host to destination host, hence it results in almost negligible packet loss.

Chapter 4

Conclusion

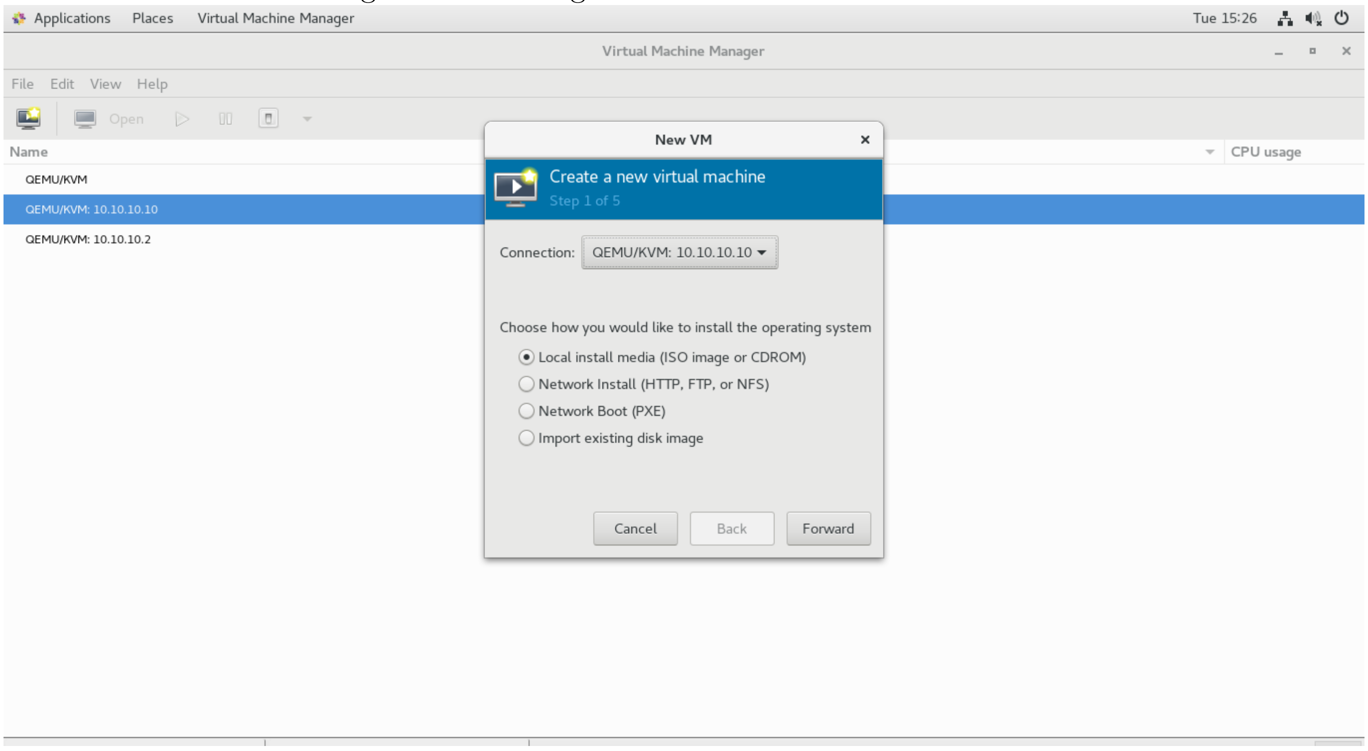
The motive of this project was to mimic live VM migration just as it occurs in data centers. We ventured onto how VM migration can be done efficiently and the network related issues pertaining to the same. We have analysed the migration time and the downtime for different images of VM, both for intra-subnet and inter-subnet. The packet loss in case of VM Migration in Bridge mode was very less as compared to the other two as the pending replies were sent through VXLAN Tunnel to the new host.This, in turn, results in increased migration time in case of Bridged mode as compared to NAT and Intra-subnet VM Migration due to retransmission of dirty pages and the added overhead of VXLAN headers.

Appendix A

Snapshots of Intra Subnet VM Migration

Following snapshots shows the live virtual machine migration between two hosts in same subnets using a third host as NFS server.

Figure 1: Creating virtual machine



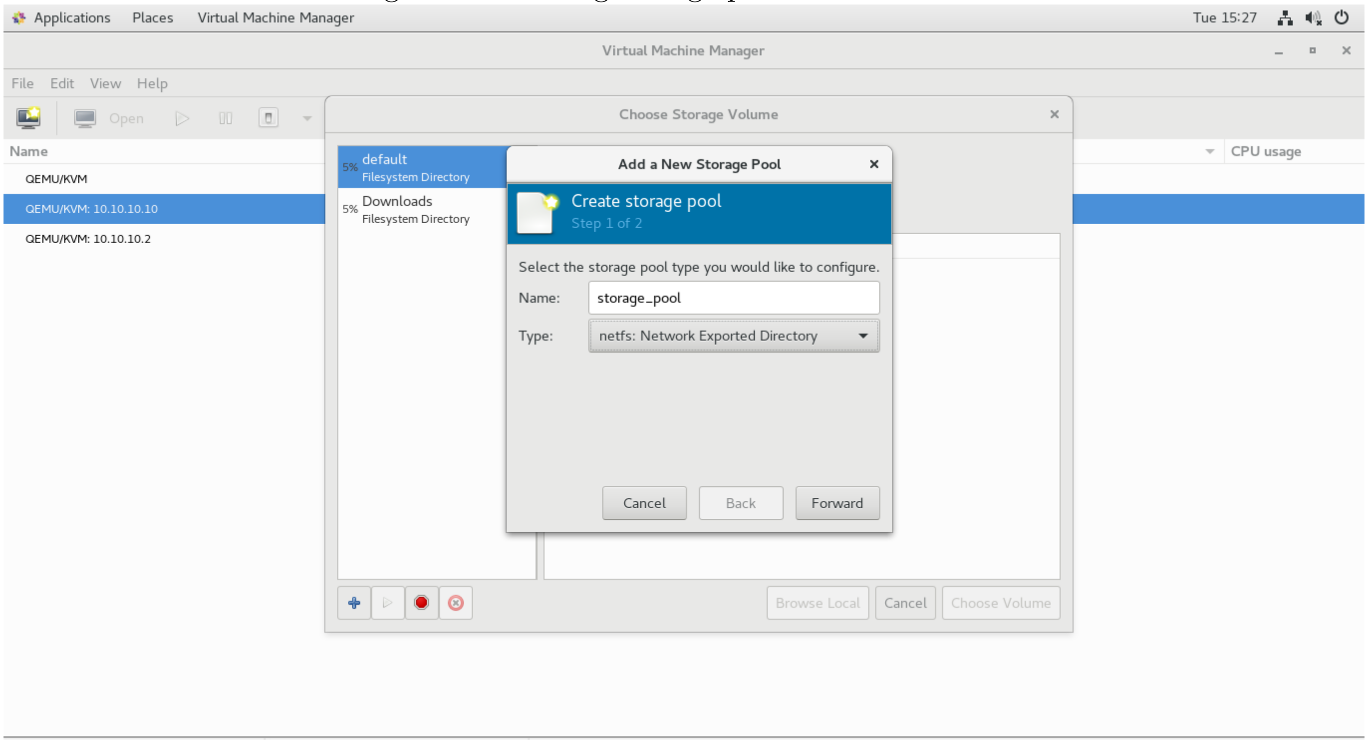
Create a new virtual machine
Step 1 of 5

Connection: QEMU/KVM: 10.10.10.10

Choose how you would like to install the operating system

- [✓ Local install media (ISO image or CDROM)]
- Network Install (HTTP, FTP, or NFS)
- Network Boot (PXE)
- Import existing disk image

Figure 2: Creating storage pool



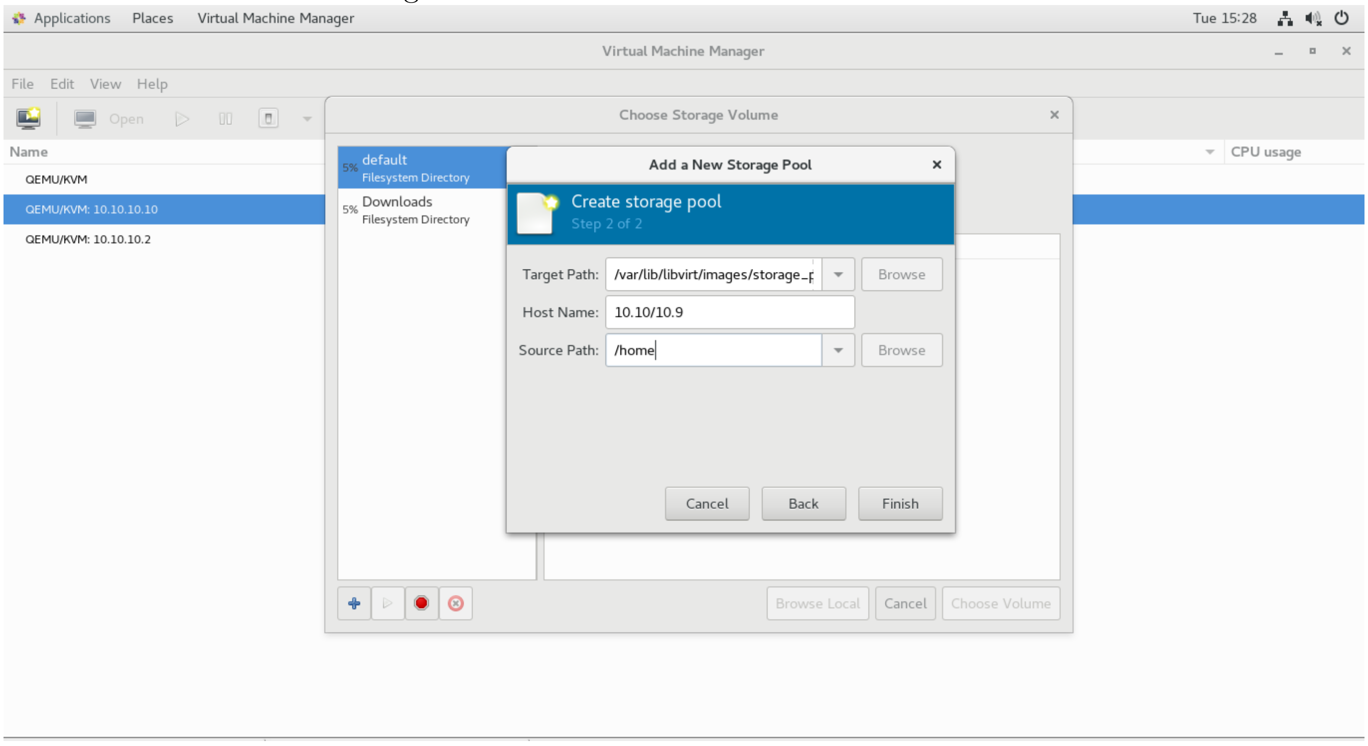
Create storage pool
Step 1 of 2

Select the storage pool type you would like to configure.

Name: storage_pool

Type: netfs: Network Exported Directory

Figure 3: NFS Server details



Add a New Storage Pool

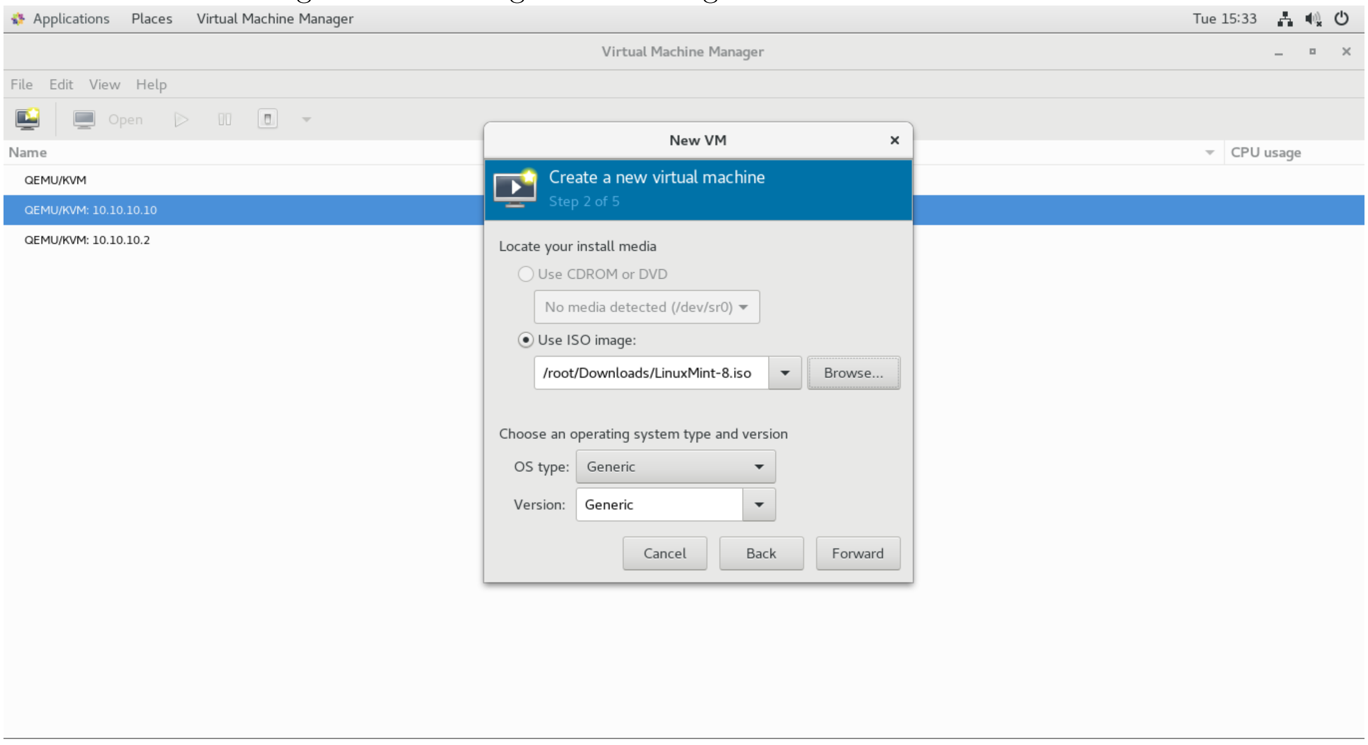
Create storage pool
Step 2 of 2

Target Path: /var/lib/libvirt/images/storage_p

Host Name: 10.10/10.9

Source Path: /home

Figure 4: Selecting the ISO image of the OS



New VM

Create a new virtual machine
Step 2 of 5

Locate your install media

( ) Use CDROM or DVD
No media detected (/dev/sr0)

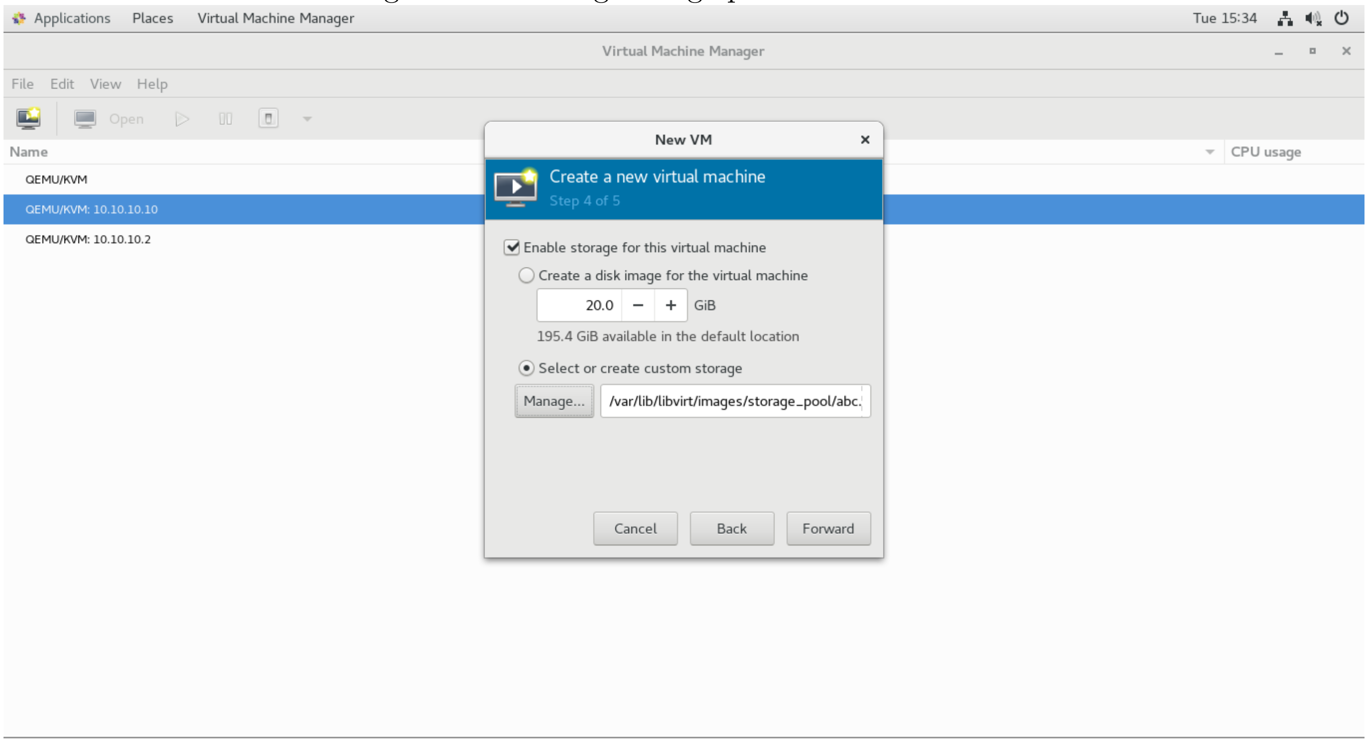
(•) Use ISO image:
/root/Downloads/LinuxMint-8.iso

Choose an operating system type and version

OS type: Generic

Version: Generic

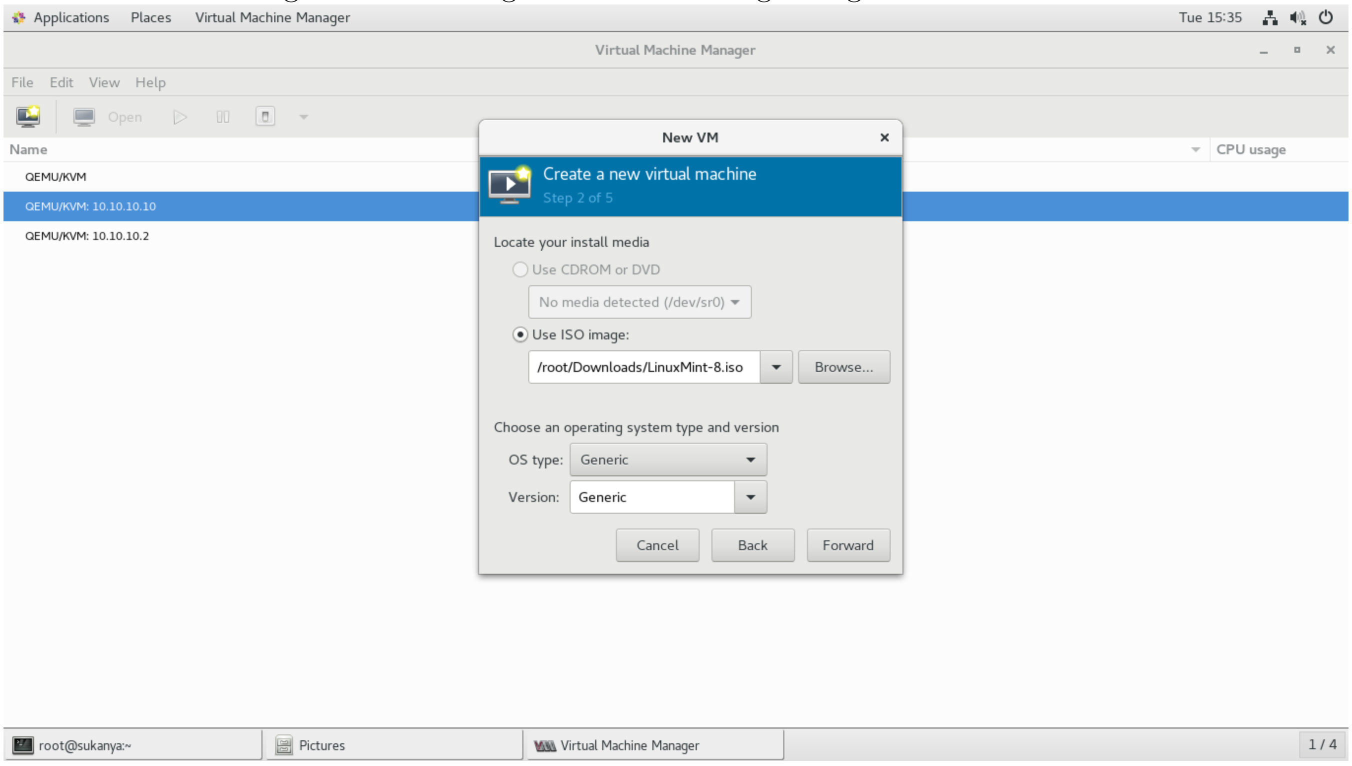
Figure 5: Creating storage pool



Create a new virtual machine
Step 4 of 5

☑ Enable storage for this virtual machine
○ Create a disk image for the virtual machine
20.0 GiB
195.4 GiB available in the default location
● Select or create custom storage
Manage... /var/lib/libvirt/images/storage_pool/abc.

Figure 6: Choosing the custom storage image

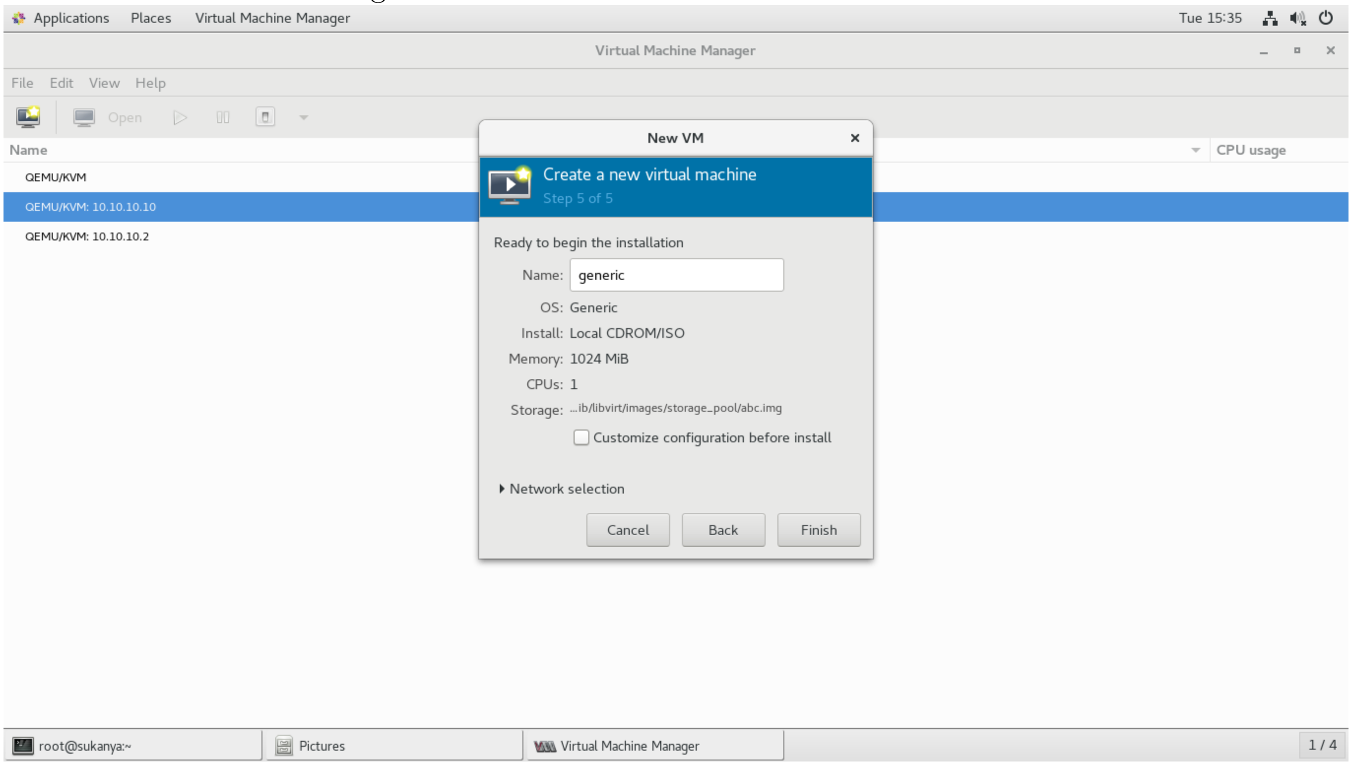


Create a new virtual machine
Step 2 of 5

Locate your install media
○ Use CDROM or DVD
No media detected (/dev/sr0)
● Use ISO image:
/root/Downloads/LinuxMint-8.iso Browse...

Choose an operating system type and version
OS type: Generic
Version: Generic

Figure 7: Virtual Machine details



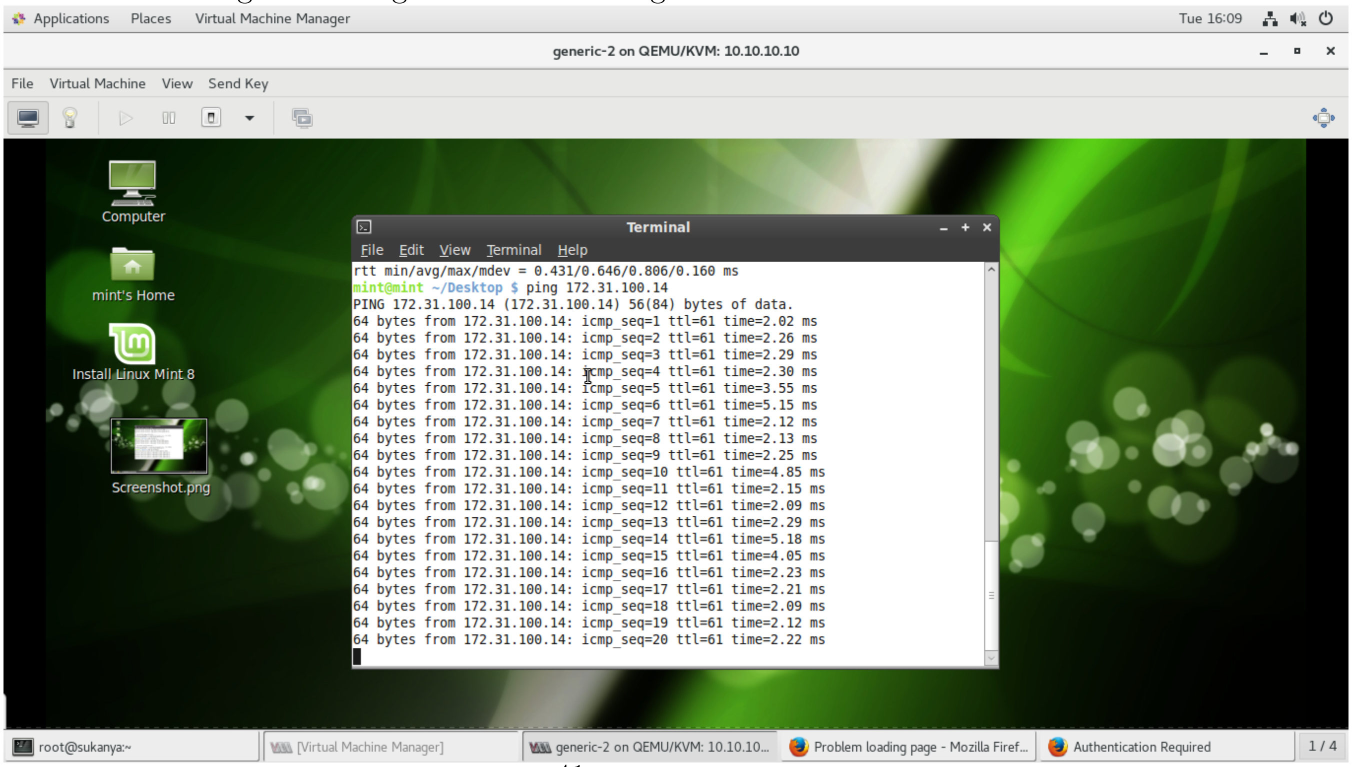
Ready to begin the installation

Name: generic
OS: Generic
Install: Local CDROM/ISO
Memory: 1024 MiB
CPUs: 1
Storage: ...ib/libvirt/images/storage_pool/abc.img

[ ] Customize configuration before install

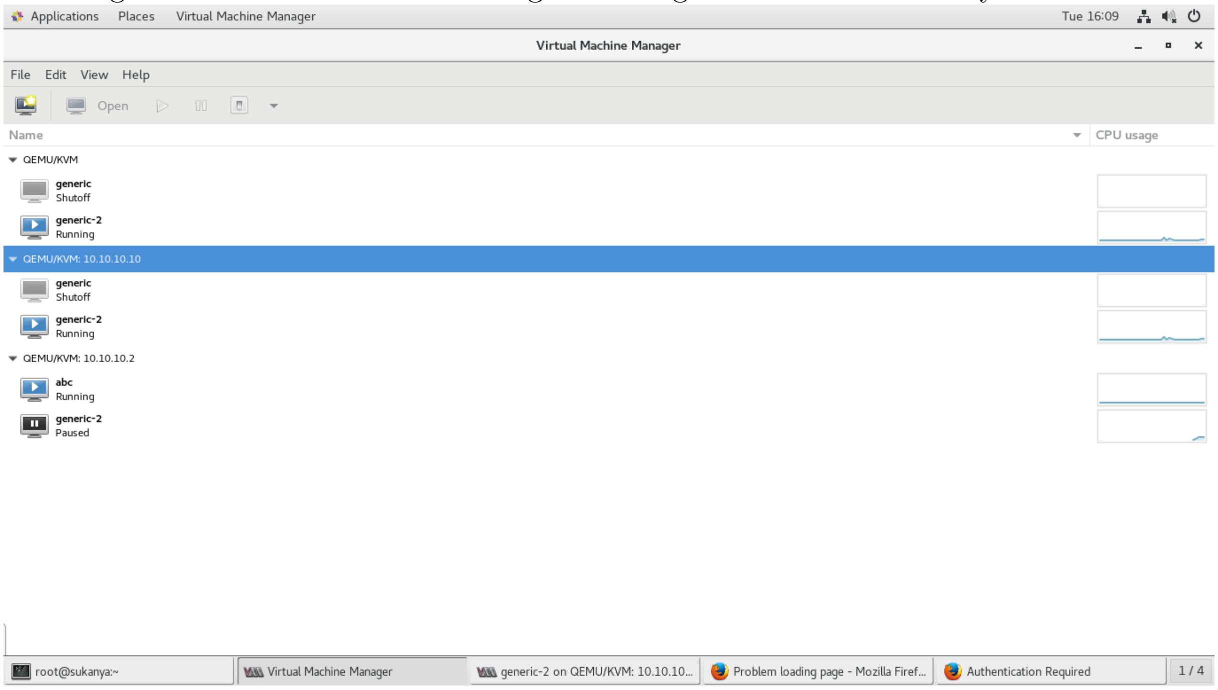
Network selection

Figure 8: Ping command running on virtual machine



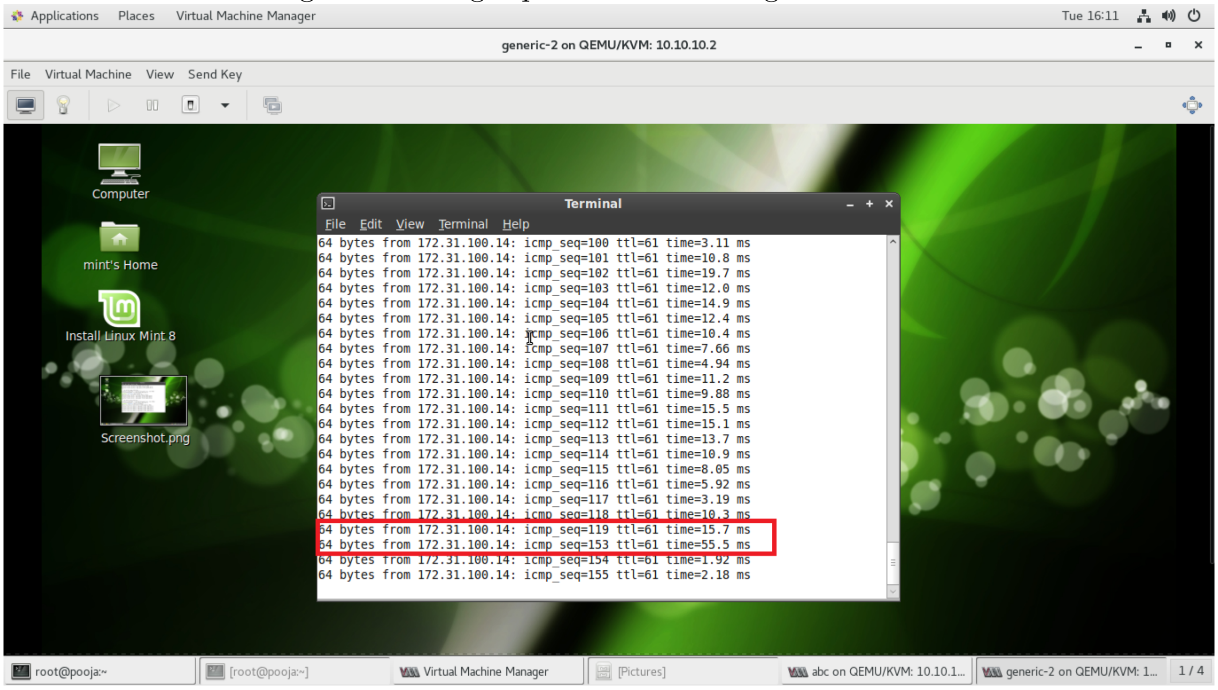
```
rtt min/avg/max/mdev = 0.431/0.646/0.806/0.160 ms
mint@mint ~/Desktop $ ping 172.31.100.14
PING 172.31.100.14 (172.31.100.14) 56(84) bytes of data.
64 bytes from 172.31.100.14: icmp_seq=1 ttl=61 time=2.02 ms
64 bytes from 172.31.100.14: icmp_seq=2 ttl=61 time=2.26 ms
64 bytes from 172.31.100.14: icmp_seq=3 ttl=61 time=2.29 ms
64 bytes from 172.31.100.14: icmp_seq=4 ttl=61 time=2.30 ms
64 bytes from 172.31.100.14: icmp_seq=5 ttl=61 time=3.55 ms
64 bytes from 172.31.100.14: icmp_seq=6 ttl=61 time=5.15 ms
64 bytes from 172.31.100.14: icmp_seq=7 ttl=61 time=2.12 ms
64 bytes from 172.31.100.14: icmp_seq=8 ttl=61 time=2.13 ms
64 bytes from 172.31.100.14: icmp_seq=9 ttl=61 time=2.25 ms
64 bytes from 172.31.100.14: icmp_seq=10 ttl=61 time=4.85 ms
64 bytes from 172.31.100.14: icmp_seq=11 ttl=61 time=2.15 ms
64 bytes from 172.31.100.14: icmp_seq=12 ttl=61 time=2.09 ms
64 bytes from 172.31.100.14: icmp_seq=13 ttl=61 time=2.29 ms
64 bytes from 172.31.100.14: icmp_seq=14 ttl=61 time=5.18 ms
64 bytes from 172.31.100.14: icmp_seq=15 ttl=61 time=4.05 ms
64 bytes from 172.31.100.14: icmp_seq=16 ttl=61 time=2.23 ms
64 bytes from 172.31.100.14: icmp_seq=17 ttl=61 time=2.21 ms
64 bytes from 172.31.100.14: icmp_seq=18 ttl=61 time=2.09 ms
64 bytes from 172.31.100.14: icmp_seq=19 ttl=61 time=2.12 ms
64 bytes from 172.31.100.14: icmp_seq=20 ttl=61 time=2.22 ms
```

Figure 9: Virtual Machine Manager showing all the VMs on the system



```
QEMU/KVM
    generic
    Shutoff
    generic-2
    Running
QEMU/KVM: 10.10.10.10
    generic
    Shutoff
    generic-2
    Running
QEMU/KVM: 10.10.10.2
    abc
    Running
    generic-2
    Paused
```

Figure 10: Ping replies after VM migration



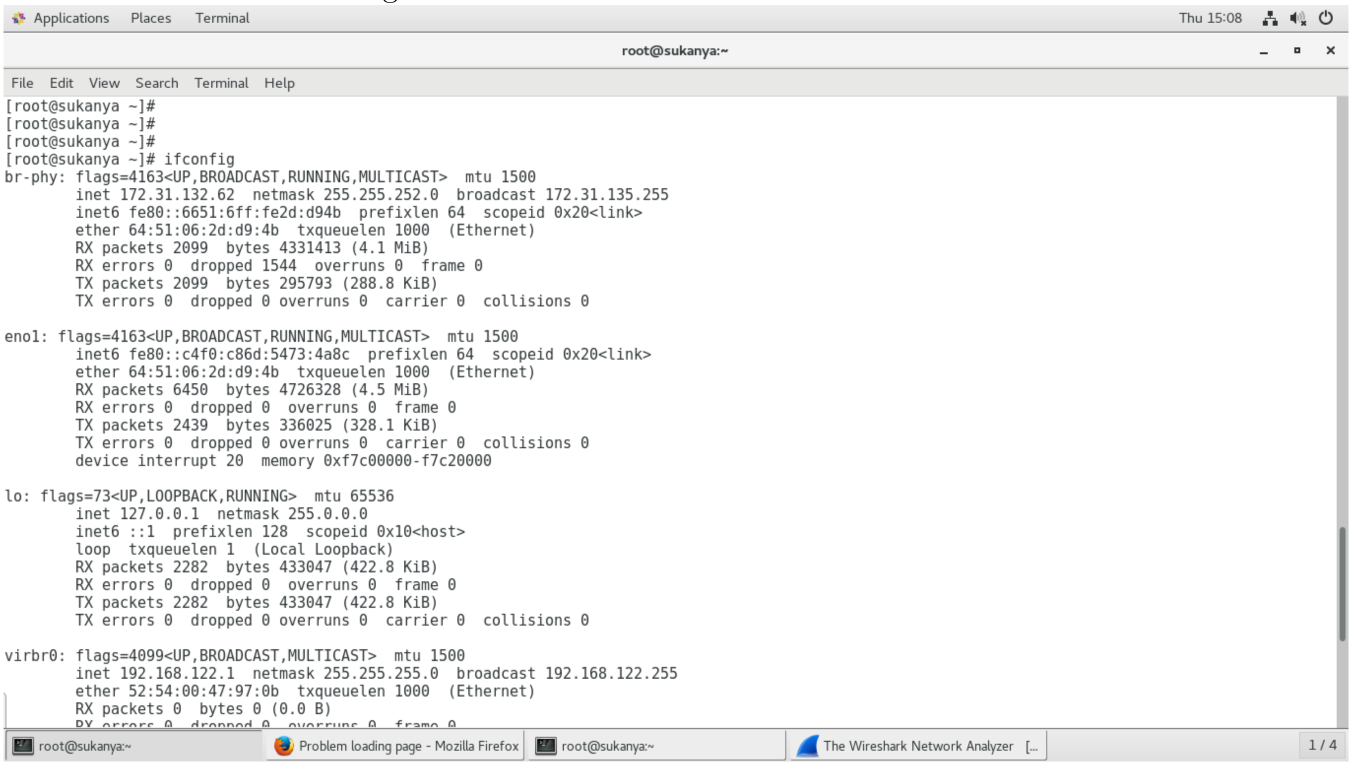
```
64 bytes from 172.31.100.14: icmp_seq=100 ttl=61 time=3.11 ms
64 bytes from 172.31.100.14: icmp_seq=101 ttl=61 time=10.8 ms
64 bytes from 172.31.100.14: icmp_seq=102 ttl=61 time=19.7 ms
64 bytes from 172.31.100.14: icmp_seq=103 ttl=61 time=12.0 ms
64 bytes from 172.31.100.14: icmp_seq=104 ttl=61 time=14.9 ms
64 bytes from 172.31.100.14: icmp_seq=105 ttl=61 time=12.4 ms
64 bytes from 172.31.100.14: icmp_seq=106 ttl=61 time=10.4 ms
64 bytes from 172.31.100.14: icmp_seq=107 ttl=61 time=7.66 ms
64 bytes from 172.31.100.14: icmp_seq=108 ttl=61 time=4.94 ms
64 bytes from 172.31.100.14: icmp_seq=109 ttl=61 time=11.2 ms
64 bytes from 172.31.100.14: icmp_seq=110 ttl=61 time=9.88 ms
64 bytes from 172.31.100.14: icmp_seq=111 ttl=61 time=15.5 ms
64 bytes from 172.31.100.14: icmp_seq=112 ttl=61 time=15.1 ms
64 bytes from 172.31.100.14: icmp_seq=113 ttl=61 time=13.7 ms
64 bytes from 172.31.100.14: icmp_seq=114 ttl=61 time=10.9 ms
64 bytes from 172.31.100.14: icmp_seq=115 ttl=61 time=8.05 ms
64 bytes from 172.31.100.14: icmp_seq=116 ttl=61 time=5.92 ms
64 bytes from 172.31.100.14: icmp_seq=117 ttl=61 time=3.19 ms
64 bytes from 172.31.100.14: icmp_seq=118 ttl=61 time=10.3 ms
64 bytes from 172.31.100.14: icmp_seq=119 ttl=61 time=15.7 ms
64 bytes from 172.31.100.14: icmp_seq=153 ttl=61 time=55.5 ms
64 bytes from 172.31.100.14: icmp_seq=154 ttl=61 time=1.92 ms
64 bytes from 172.31.100.14: icmp_seq=155 ttl=61 time=2.18 ms
```

Appendix B

Snapshots of Inter Subnet VM Migration using VXLAN

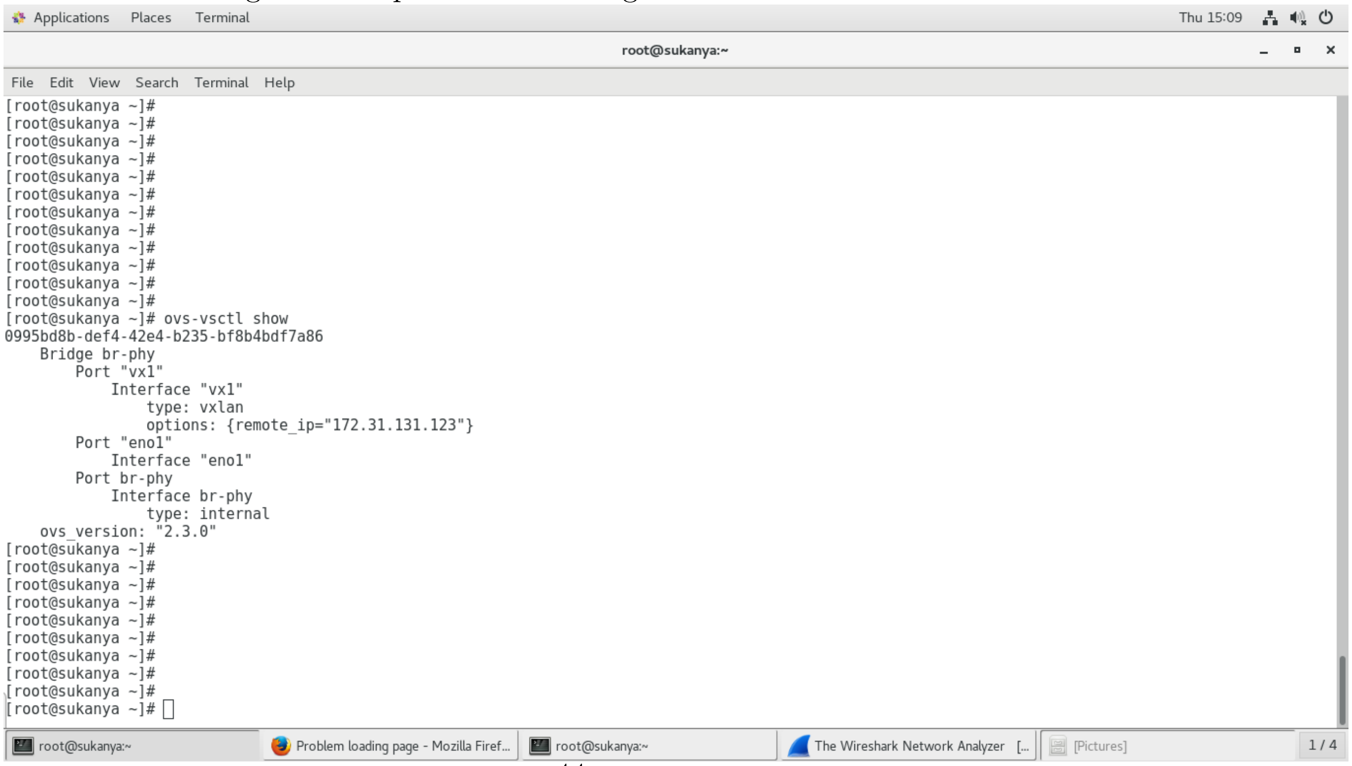
Following snapshots shows the live virtual machine migration between two hosts in different subnets using a third host as NFS server and VXLAN technology.

Figure 11: Interfaces on Host A


Applications Places Terminal
Thu 15:08
root@sukanya:~
File Edit View Search Terminal Help
root@sukanya:~
Problem loading page - Mozilla Firefox
root@sukanya:~
The Wireshark Network Analyzer [...
1 / 4

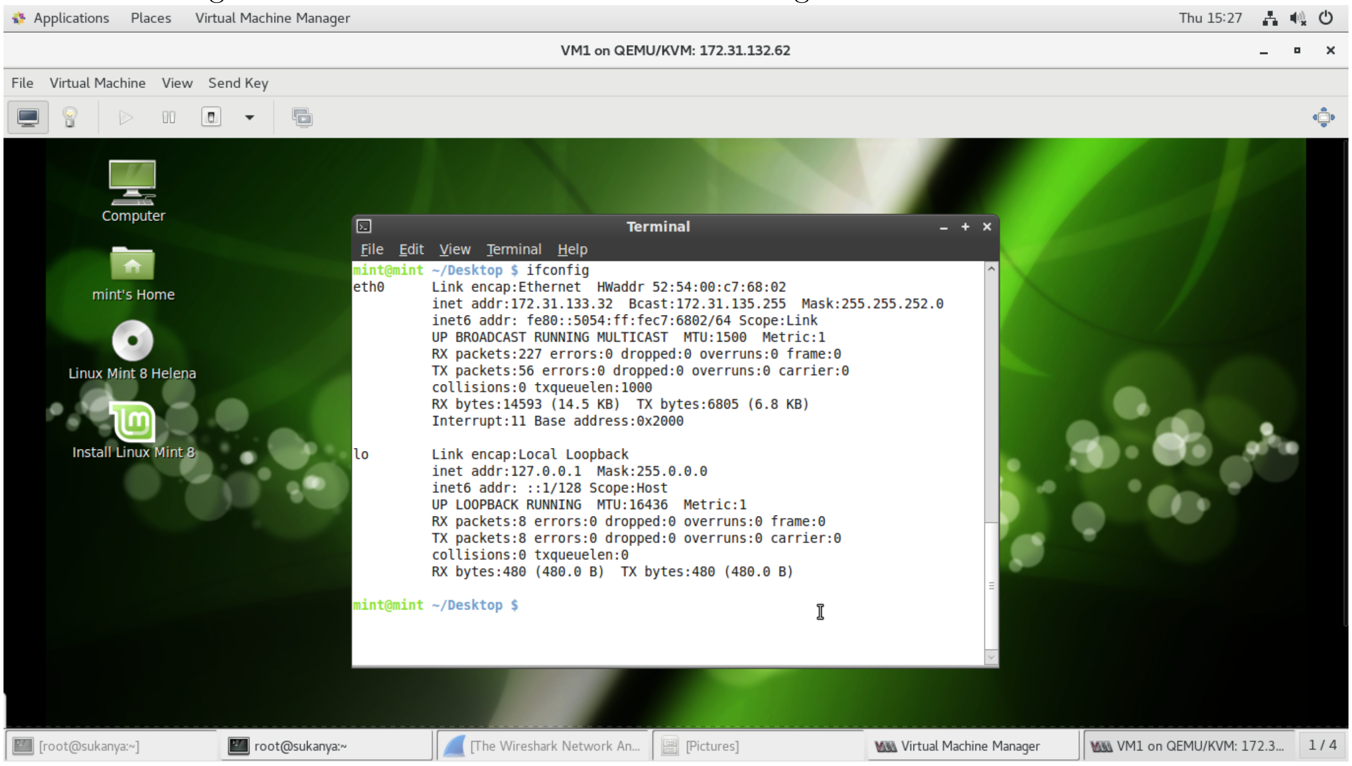
```
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]# ifconfig
br-phy: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.31.132.62  netmask 255.255.252.0  broadcast 172.31.135.255
        inet6 fe80::6651:6ff:fe2d:d94b  prefixlen 64  scopeid 0x20<link>
        ether 64:51:06:2d:d9:4b  txqueuelen 1000  (Ethernet)
        RX packets 2099  bytes 4331413 (4.1 MiB)
        RX errors 0  dropped 1544  overruns 0  frame 0
        TX packets 2099  bytes 295793 (288.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::c4f0:c86d:5473:4a8c  prefixlen 64  scopeid 0x20<link>
        ether 64:51:06:2d:d9:4b  txqueuelen 1000  (Ethernet)
        RX packets 6450  bytes 4726328 (4.5 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2439  bytes 336025 (328.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 20  memory 0xf7c00000-f7c20000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 2282  bytes 433047 (422.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2282  bytes 433047 (422.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

virbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
        ether 52:54:00:47:97:0b  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
```

Figure 12: OpenVSwitch bridges created on Host A


Applications Places Terminal
Thu 15:09
root@sukanya:~
File Edit View Search Terminal Help
root@sukanya:~
Problem loading page - Mozilla Firef...
root@sukanya:~
The Wireshark Network Analyzer [...
[Pictures]
1 / 4

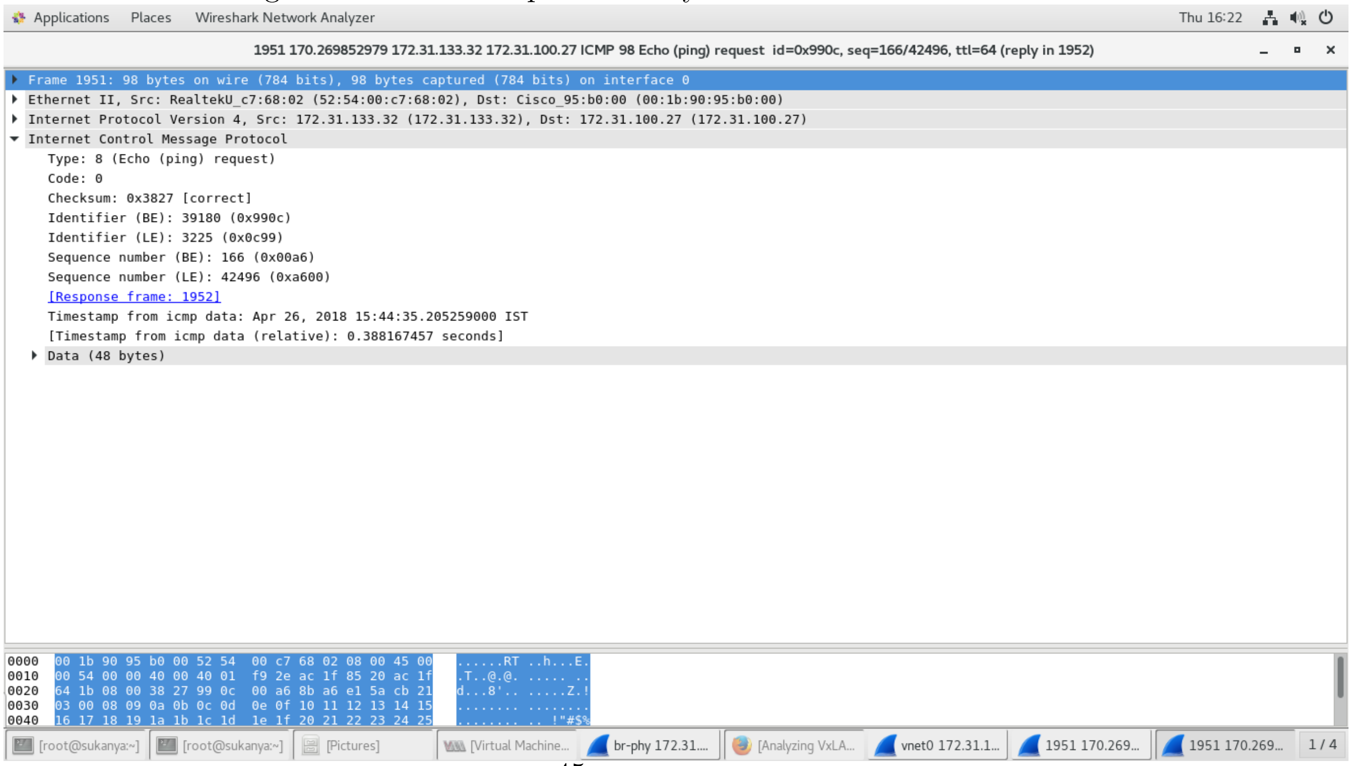
```
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]# ovs-vsctl show
0995bd8b-def4-42e4-b235-bf8b4bdf7a86
    Bridge br-phy
        Port "vx1"
            Interface "vx1"
                type: vxlan
                options: {remote_ip="172.31.131.123"}
        Port "eno1"
            Interface "eno1"
        Port br-phy
            Interface br-phy
                type: internal
    ovs_version: "2.3.0"
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
[root@sukanya ~]#
```

Figure 13: Network Interfaces on VM running on Host A



```
mint@mint ~/Desktop $ ifconfig
eth0      Link encap:Ethernet  HWaddr 52:54:00:c7:68:02
          inet addr:172.31.133.32  Bcast:172.31.135.255  Mask:255.255.252.0
          inet6 addr: fe80::5054:ff:fec7:6802/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:227 errors:0 dropped:0 overruns:0 frame:0
          TX packets:56 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:14593 (14.5 KB)  TX bytes:6805 (6.8 KB)
          Interrupt:11 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:480 (480.0 B)  TX bytes:480 (480.0 B)

mint@mint ~/Desktop $
```

Figure 14: ICMP Request sent by VM in Host A

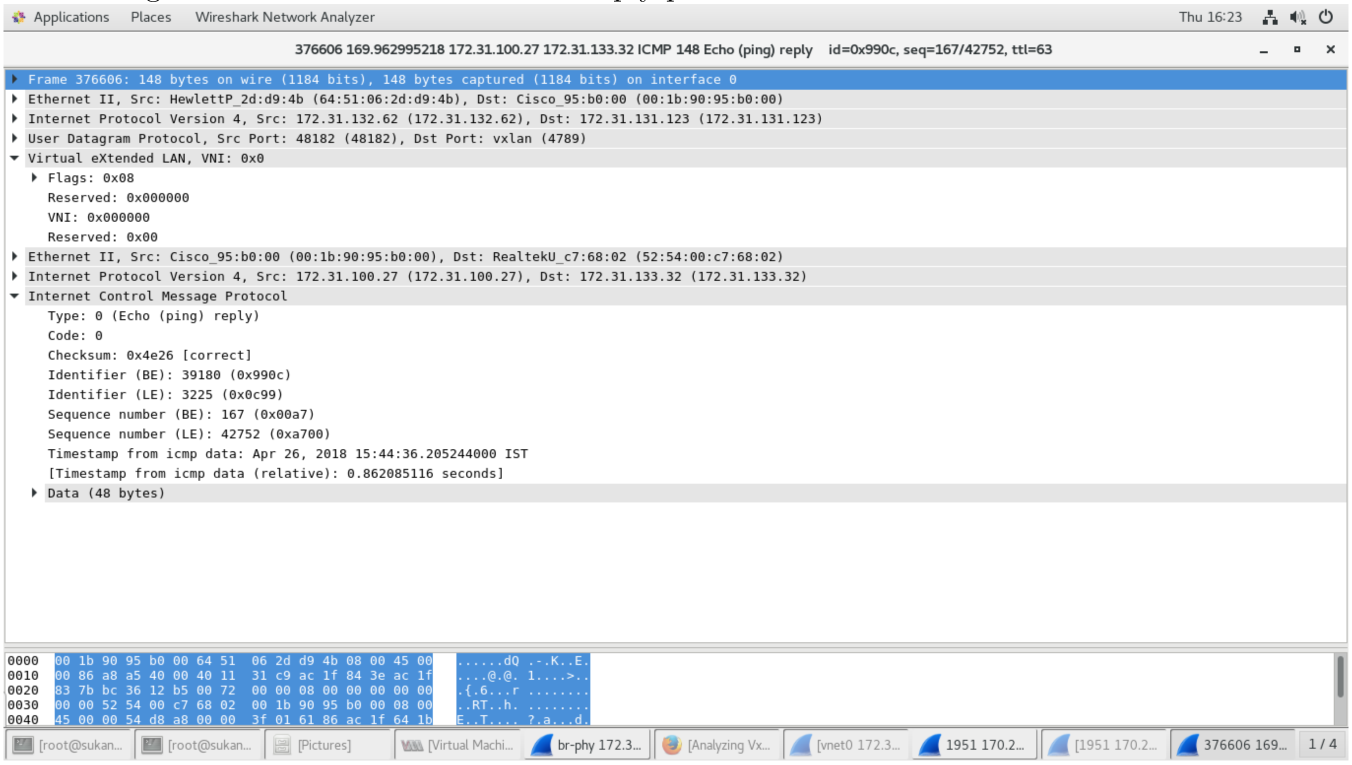


1951 170.269852979 172.31.133.32 172.31.100.27 ICMP 98 Echo (ping) request id=0x990c, seq=166/42496, ttl=64 (reply in 1952)

```
▸ Frame 1951: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▸ Ethernet II, Src: RealtekU_c7:68:02 (52:54:00:c7:68:02), Dst: Cisco_95:b0:00 (00:1b:90:95:b0:00)
▸ Internet Protocol Version 4, Src: 172.31.133.32 (172.31.133.32), Dst: 172.31.100.27 (172.31.100.27)
▾ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x3827 [correct]
    Identifier (BE): 39180 (0x990c)
    Identifier (LE): 3225 (0x0c99)
    Sequence number (BE): 166 (0x00a6)
    Sequence number (LE): 42496 (0xa600)
    [Response frame: 1952]
    Timestamp from icmp data: Apr 26, 2018 15:44:35.205259000 IST
    [Timestamp from icmp data (relative): 0.388167457 seconds]
  ▸ Data (48 bytes)
```

```
0000  00 1b 90 95 b0 00 52 54  00 c7 68 02 08 00 45 00   ......RT ..h...E.
0010  00 54 00 00 40 00 40 01  f9 2e ac 1f 85 20 ac 1f   .T..@.@. ...... ..
0020  64 1b 08 00 38 27 99 0c  00 a6 8b a6 e1 5a cb 21   d...8'.. .....Z.!
0030  03 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
```

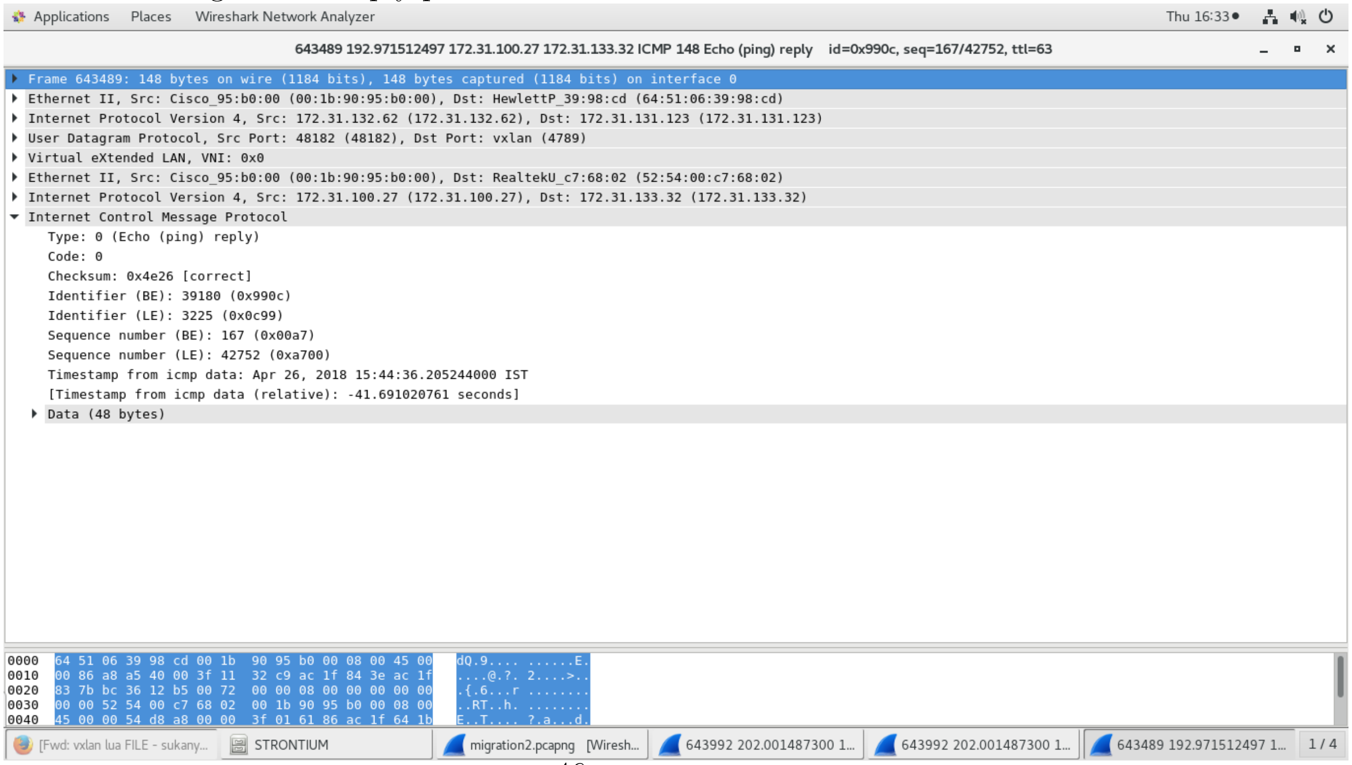
Figure 15: VXLan Headers in the reply packet recieved on Host A


Applications
Places
Wireshark Network Analyzer
Thu 16:23
376606 169.962995218 172.31.100.27 172.31.133.32 ICMP 148 Echo (ping) reply   id=0x990c, seq=167/42752, ttl=63

```
▸ Frame 376606: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0
▸ Ethernet II, Src: HewlettP_2d:d9:4b (64:51:06:2d:d9:4b), Dst: Cisco_95:b0:00 (00:1b:90:95:b0:00)
▸ Internet Protocol Version 4, Src: 172.31.132.62 (172.31.132.62), Dst: 172.31.131.123 (172.31.131.123)
▸ User Datagram Protocol, Src Port: 48182 (48182), Dst Port: vxlan (4789)
▾ Virtual eXtended LAN, VNI: 0x0
  ▸ Flags: 0x08
    Reserved: 0x000000
    VNI: 0x000000
    Reserved: 0x00
▸ Ethernet II, Src: Cisco_95:b0:00 (00:1b:90:95:b0:00), Dst: RealtekU_c7:68:02 (52:54:00:c7:68:02)
▸ Internet Protocol Version 4, Src: 172.31.100.27 (172.31.100.27), Dst: 172.31.133.32 (172.31.133.32)
▾ Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x4e26 [correct]
    Identifier (BE): 39180 (0x990c)
    Identifier (LE): 3225 (0x0c99)
    Sequence number (BE): 167 (0x00a7)
    Sequence number (LE): 42752 (0xa700)
    Timestamp from icmp data: Apr 26, 2018 15:44:36.205244000 IST
    [Timestamp from icmp data (relative): 0.862085116 seconds]
  ▸ Data (48 bytes)
```

```
0000  00 1b 90 95 b0 00 64 51  06 2d d9 4b 08 00 45 00   ......dQ .-.K..E.
0010  00 86 a8 a5 40 00 40 11  31 c9 ac 1f 84 3e ac 1f   ....@.@. 1....>..
0020  83 7b bc 36 12 b5 00 72  00 00 08 00 00 00 00 00   .{.6...r ........
0030  00 00 52 54 00 c7 68 02  00 1b 90 95 b0 00 08 00   ..RT..h. ........
0040  45 00 00 54 d8 a8 00 00  3f 01 61 86 ac 1f 64 1b   E..T.... ?.a...d.
```

Figure 16: Reply packet recieved on Host B from Host A


Applications
Places
Wireshark Network Analyzer
Thu 16:33
643489 192.971512497 172.31.100.27 172.31.133.32 ICMP 148 Echo (ping) reply   id=0x990c, seq=167/42752, ttl=63

```
▸ Frame 643489: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0
▸ Ethernet II, Src: Cisco_95:b0:00 (00:1b:90:95:b0:00), Dst: HewlettP_39:98:cd (64:51:06:39:98:cd)
▸ Internet Protocol Version 4, Src: 172.31.132.62 (172.31.132.62), Dst: 172.31.131.123 (172.31.131.123)
▸ User Datagram Protocol, Src Port: 48182 (48182), Dst Port: vxlan (4789)
▸ Virtual eXtended LAN, VNI: 0x0
▸ Ethernet II, Src: Cisco_95:b0:00 (00:1b:90:95:b0:00), Dst: RealtekU_c7:68:02 (52:54:00:c7:68:02)
▸ Internet Protocol Version 4, Src: 172.31.100.27 (172.31.100.27), Dst: 172.31.133.32 (172.31.133.32)
▾ Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x4e26 [correct]
    Identifier (BE): 39180 (0x990c)
    Identifier (LE): 3225 (0x0c99)
    Sequence number (BE): 167 (0x00a7)
    Sequence number (LE): 42752 (0xa700)
    Timestamp from icmp data: Apr 26, 2018 15:44:36.205244000 IST
    [Timestamp from icmp data (relative): -41.691020761 seconds]
  ▸ Data (48 bytes)
```

```
0000  64 51 06 39 98 cd 00 1b  90 95 b0 00 08 00 45 00   dQ.9.... ......E.
0010  00 86 a8 a5 40 00 3f 11  32 c9 ac 1f 84 3e ac 1f   ....@.?. 2....>..
0020  83 7b bc 36 12 b5 00 72  00 00 08 00 00 00 00 00   .{.6...r ........
0030  00 00 52 54 00 c7 68 02  00 1b 90 95 b0 00 08 00   ..RT..h. ........
0040  45 00 00 54 d8 a8 00 00  3f 01 61 86 ac 1f 64 1b   E..T.... ?.a...d.
```

References

[1] Connecting VMs Using Tunnels *http://docs.openvswitch.org/en/latest/howto/userspace-tunneling/*

[2] Configuration of VXLAN Tunnels on OpenvSwitch *networkstatic.net/configuring-vxlan-and-gre-tunnels-on-openvswitch/*

[3] Installation of KVM Hypervisor on CentOS *https://www.linuxtechi.com/install-kvm-hypervisor-on-centos-7-and-rhel-7/*

[4] Installation of OpenvSwitch *https://github.com/ebiken/doc-network/wiki/How-To:-Install-OVS-/28Kernel-Module/29-from-Source-Code*

[5] IP Mobility to Support Live Migration of Virtual Machines Across Subnets *Ezra Silvera ,Gilad Sharaby, Dean Lorenz, Inbar Shapira, 2009*