Efficient Live Wide Area VM Migration with IP Address Change Using Type II Hypervisor

by

Omhenimhen Iyamu

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science in

Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Faculty of Engineering

Carleton University

Ottawa, Ontario, Canada

May 2013

Your file Votre référence
ISBN: 978-0-494-94640-4

Our file Notre référence
ISBN: 978-0-494-94640-4

ABSTRACT

Live wide area virtual machine migration is the transfer of a virtual machine currently running and connected to clients from one host server to another host server over a Wide Area Network. In this thesis we have designed and implemented an approach for live migration while allowing the IP address of the VM to change. Our approach is efficient and with little disruption in services. The solution is divided into three key parts, runtime state migration, persistent state migration, and network connection handling. The solution uses the type 2 hypervisor Virtual Box. The runtime state migration is executed using Virtual Box's teleportation function. The persistent state migration is split into stages executed at different point of the migration. The network connections are done using packet redirection and Dynamic DNS. Testing revealed a 358 times better in total disruption time compared to freeze and copy migration.

ACKNOWLEDGEMENTS

My appreciation goes to Professor Samuel Ajila for his support and guidance throughout this thesis.

My love and thanks to my parents Isaac and Catherine Iyamu for their support and sponsorship of my Master of Applied Science (MASc) degree.

I thank everyone who has ever supported and encouraged me in my academic endeavors.

Finally, I thank the Almighty God for everything that He has done for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

API	Application Programming Interface
DNS	Domain Name Service
CPU	Central Processing Unit
GUI	Graphical User Interface
I/O	Input/output
IaaS	Infrastructure-as-a-Service
iSCSI	Internet Small Computer System Interface
JVM	Java Virtual Machine
KVM	Kernel Virtual Machine
LAN	Local Area Network
MAC	Media Access Control
OS	Operating System
PaaS	Platform-as-a-Service
PE	Provider Edge
QoE	Quality of Experience
QoS	Quality of Service
SaaS	Software-as-a-Service
SLA	Service Level Agreement
VLAN	Virtual Local Area Network
VPC	Virtual Private Cloud
VPLS	Virtual Private LAN Service

VPN	Virtual Private Network
VXLAN	Virtual Extendable Local Area Network
VM	Virtual Machine
WAN	Wide Area Network

Chapter 1 : Introduction

The National Institute of Standards and Technology (NIST) [26] defines cloud computing as any computing model with a shared pool of configurable computing resources such as Networks, servers, storage, applications, and services that can be accessed over the network in a convenient manner, and provisioned and released on-demand with minimal management or service provider effort [26]. Cloud Datacenters are the facilities that house the computing resources. Resources could be spread across multiple datacenter facilities for reasons such as power constraints and closeness to customers. Different datacenters networks are connected to each other over the Internet to form one large interconnected cloud network [27].

There are numerous advantages to cloud computing for both providers and customers, two main examples are:

- Cloud computing relieves customers of IT responsibilities such as upgrades and maintenance of IT infrastructure. Non-IT companies can focus on their core business and rent their computing requirements from the cloud provider. Providers can have a solid business model on just providing computing and keeping up to date with advancements of such resources. [28].

- Cloud computing provides flexibility and economy of scale. Providers can increase productivity by improving on resources as needed and provisioning only what is requested. Customers can request for more or less computing resources whenever needed and only be charged for what they use. This prevents the customer from investing in more infrastructures if more computing resources are needed. [28]

Examples of some challenges and limitations to cloud computing are defining a suitable pay structure for customers, synchronization of data both within and across datacenters, live virtual machine migration, and resource management in the datacenters [28]. Live virtual machine migration is the transfer of a virtual machine from a source host to a target host while the machine remains running and with as little downtime as possible [1]. This thesis reports on the work we did on the problem of live virtual machine migration with IP address change. We included a new dynamic to the live virtual machine problem, which is allowing for the IP address of the VM to change when it is migrated. This adds another layer of difficulty to the already challenging problem.

1.1 Problem Motivation

Three main problem scenarios are considered for our problem motivation: Hardware maintenance, scheduled power outage, and server relocation. Assuming a situation arises in a cloud datacenter whereby one of the host servers needs to undergo maintenance, power needs to be shut down, or that a resource need to be physically moved

to a different site. In any such a case, the problem that arises is how to handle clients who are currently running virtual machines (VMs) on the subject host server. In a cloud environment these clients are paying for their services, hold service level agreements (SLA) with the cloud provider, and may have very sensitive processes currently running on the subject host. Because all the three scenarios that we focus on are known ahead of time, a solution could be to also inform the client ahead of time to make alternate arrangements. A better solution is to move the clients' virtual machines running on the subject host to a different host whether permanently or temporarily and transparent to the client. This move is defined as migration [1].

Migrating a client's virtual machine presents its own challenges with respect to how exactly it should be done. The current state of the virtual machine adds a layer of difficulty to the problem because a running VM may not be able to be migrated the same way as a powered down VM. Another problem is the location of the target host at which the VM is to be migrated to. This is a problem because migrating to a target host on the same local network cannot be treated the same way as migrating to a host on another network. There are two main challenges with this: live migration and migration with IP change. Live migration refers to a migration that is performed while the Virtual Machine (VM) is still connected to the client or clients, and still processing request and sending responses. The downtime (length of service disruption) of the client-VM connection is crucial in this process. Migration with IP change refers to a migration where the VM after migration will have a different IP address than the source VM.

A simple solution to live migration is known as Freeze and Copy. In Freeze and Copy, the VM is "frozen", moved to the target, and then "unfrozen" [23]. When a VM is "frozen" the client cannot use it until it is "unfrozen". This means the client will experience a large disruption in services. Current solutions to this problem mostly involve flattening layer 2, or migration along a local area network [13][16][12]. This enables migrations to take place but keeping the Virtual Machine's IP address the same. The problem with this is that flattening layer 2 is an expensive and complex process because it requires special routers and network configurations [13]. For large cloud networks reconfiguration costs plus the potential cost of new equipment makes this solution very costly.

The goal of this thesis is to provide a solution to virtual machine migration on layer 3, so as to not require flattening layer 2. The significance of this research is a simpler and more cost effective way to move Virtual Machines across hosts while allowing the IP address to change and not require much or any reconfiguration, and with little disruption on the client's connection.

1.2 Overview of Our Solution

In order to perform live virtual machine migration on layer 3, which is allowing for IP address change, three main areas of concern need to be addressed. They are moving the runtime state, moving the persistent state, and network connections. The virtualization tool used in this solution is Virtual Box, which is an open source, too that uses a Type 2 hypervisor. This was chosen because research has already started using Type 1 hypervisor

in [23]. A type 1 hypervisor sits directly on to the physical hardware and manages all operating systems on the machine, whereas a Type 2 hypervisor sits on to of an operating system. The Virtual Box teleportation function is used to tackle the runtime state migration. The persistent state migration is tackled in three steps. First a snapshot of the VM is taken in order to save the current state of the persistent state storage and change deltas are saved in a differencing image. Second step is transferring the base persistent state storage to the target host. The last step is transferring the differencing image and applying the changes in the differencing image. Moving the network connections involve first redirecting packets by the gateway routers on the source VM's network, then using Dynamic DNS to update the IP address of the VM's domain name. Testing of our implementation reveled that this solution works as a live migration solution. Although the design of the solution works on WAN, the tests were performed on LAN in order to compare wireless and wired host connections. Comparing the migration time of an online migration with freeze and copy revealed that the total length of time of disruption in services to the client was about 358 times better.

This solution does not consider security concerns such as packet encryption because it is out of the scope of this work. This is because any security threats will arise in the area of packet transmission or database access, which have both have been well researched. As such this work does not introduce any new security threats. Security issues can be explored as a future work for this thesis.

This solution can work with any of the tree cloud services. In infrastructure as a service, the client's custom VM can be migrated to a host having similar infrastructure offerings. In platform as a service, the VM offered to the client is migrated. In software as a service, the VM holding the application(s) offered to the client is migrated.

1.3 Contributions of this Thesis

The main contribution of this thesis is a live migration solution that allows for the IP address to change. The contributions of our solution include:

- The solution was designed using a type 2 hypervisor.
- Packet redirection was done at the network level freeing the source of any packet forwarding responsibilities.
- Packet redirection achieved by IP table manipulation.
- The development and use of a watchdog to monitor stop packet redirection when redirection is no longer needed.
- Software was developed that implements our design.

1.4 Thesis Outline

The rest of this thesis report is organized as follows. Chapter 2 gives a background of some of the main concept of this thesis. It then goes on to give a description of some of the prominent live migration solutions and research. Chapter 3 discusses the design of our

migration system. Chapter 4 presents the implementation of our design solution and the different choices made. Chapter 5 talks about how we evaluated our implementation. It gives the experimental setup, testing, results, and evaluation of the results. Chapter 6 gives our conclusions.

Chapter 2 : Background and Related Work

This section provides a brief summary of some key concepts of this thesis, a state of the art review of how VM migration is currently performed in industry, and a review of some research in VM Migration.

2.1 Virtualization

Virtualization is the building of a layer of abstraction of physical hardware on top of the physical hardware for use by an operating system simultaneously [4][7]. This abstraction layer allows for the multiplexing of the physical hardware for use by multiple operating systems simultaneously [4][7][11][19]. The software responsible for handling the multiplexing and other virtualization abstractions is known a Hypervisor (Section 2.1.2).

Figure 2-1 shows a typical computer's ring model. These rings represent different levels of control and privilege. The inner rings have lower numbers and represent greater privilege levels [7][24]. The rings are numbered 0 to 3 with 0 as the innermost ring (most privileged), and 3 the outermost (least privileged). The Operating System (OS) kernel typically sits in ring 0, and user applications in ring 3 [7][24].

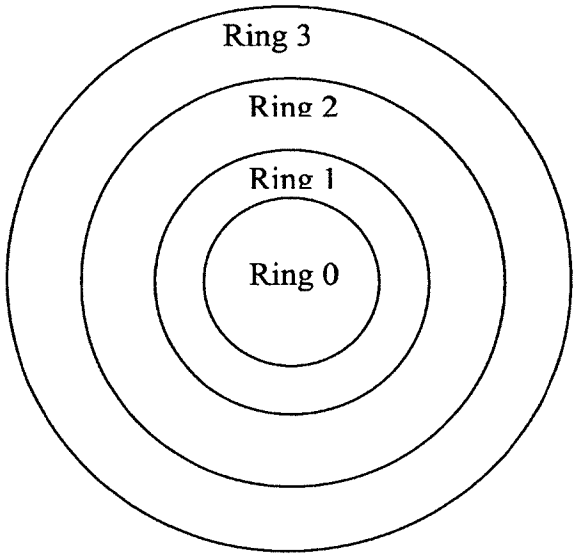

Figure 2-1: Ring Model [24]

There are two types of virtualization techniques, software virtualization and hardware virtualization. Software virtualization involves running the operating system kernel in a lower privileged ring rather than ring 0 [11]. This allows the hypervisor to run in ring 0. Some OS features were designed to only be accessible by the kernel in ring 0 (e.g. Page tables). As such these features were emulated in software [11]. This lead to some performance drawbacks and as such hardware-assisted virtualization was introduced [11].

In the time of introduction of hardware-assisted virtualization, there were some x86 instructions that were very hard to virtualize in software and caused great performance degradation [3]. Paravertualization is an approach that involves not implementing those x86 instructions and allowing the VMs direct access to them [3].

Around 2005 and 2006, both AMD and Intel introduced a new series of processors that would include some modifications and instructions to help facilitate virtualization [11]. These processors are Intel VT-x (Vanderpool) and AMD-V (Pacifica). Some of the modifications include improved segmentation of memory, a greater privileged level (ring -1), and support for better virtualize I/O and networking [3][7].

2.1.1 Virtual Machines

A virtual machine (VM) is a guest Operating System running on top of a host Operating system, which in turn runs on a physical machine [4][7]. In a typical cloud datacenter, each physical server will have a host operating system that manages and runs multiple guest operating systems (Virtual Machines) [4][7].

2.1.2 Hypervisors

A Hypervisor, also known as a Virtual Machine Manager, coordinates operations between a VM and the physical hardware through virtual hardware [5][7]. The hypervisor can multiplex access to the physical hardware, giving the perception of multiple copies of a single hardware [7]. This allows multiple virtual machines running simultaneously [3][7]. There are two types of hypervisors, type 1 or Bare Metal, and type 2 or Hosted hypervisors [3][7].

2.1.2.1 Bare Metal Hypervisor (Type 1)

This is the most common type of hypervisor [3] [5]. This hypervisor is installed at the highest privilege level and governs all the VMs above it in lower privilege levels [3][5]. The architecture is illustrated in figure 2-2 below.

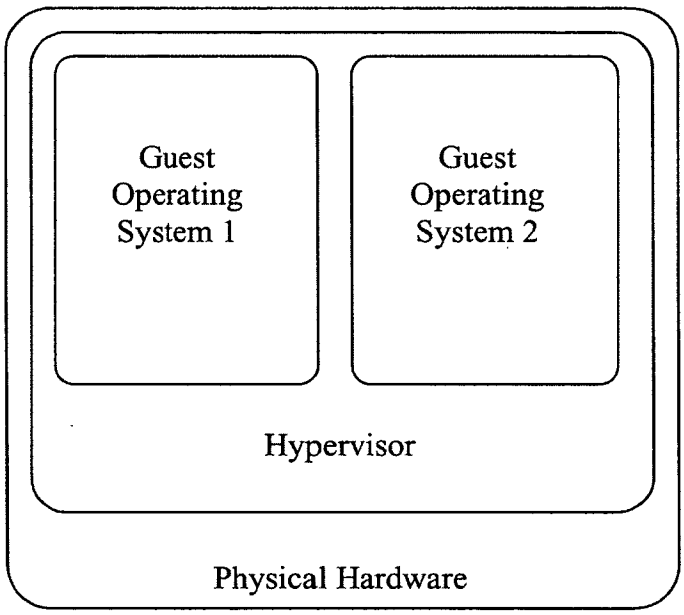

Figure 2-2: Type 1 Hypervisor [3]

Examples of applications of this approach are Xen Hypervisor [19], VMWare ESX [4], and KVM. In Xen, operating systems run in separate domains [19]. The host operating system runs in domain 0. Domain 0 contains all device drivers and controls management and networking for all other domains [19]. VMware's ESX on the other hand performs all duties of management, networking, within the hypervisor itself [4]. KVM uses a Linux kernel as its hypervisor [12].

2.1.2.2 Hosted Hypervisor (Type 2)

Type 2 hypervisors are installed on a privilege level above the host operating system [3][5]. This architecture is illustrated in figure 2-3 below.

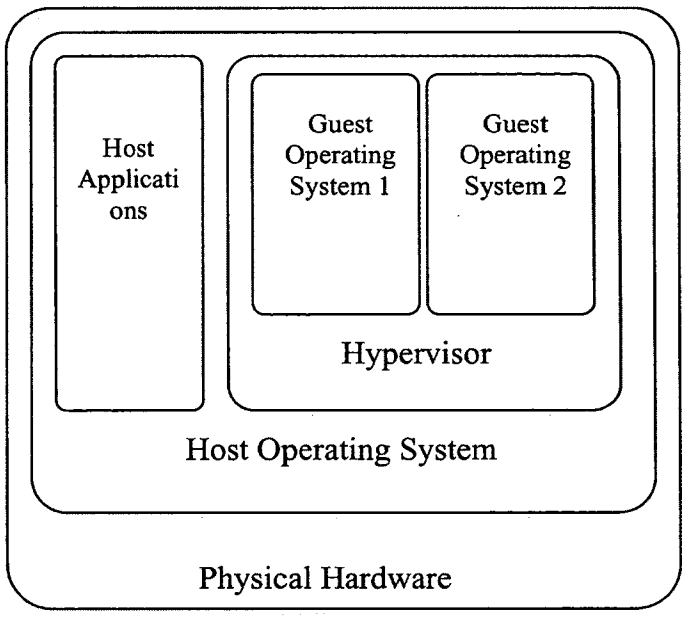

Figure 2-3: Type 2 Hypervisor [3]

Examples of type 2 hypervisors are java virtual machine (JVM), Virtual Box [18], and VMWare Workstation.

2.2 Cloud Computing

Cloud Computing offers computing resources on a pay-as-you-go basis to clients. It is defined as any computing model with a shared pool of configurable computing resources

such as Networks, servers, storage, applications, and services that can be accessed over the network in a convenient manner, and provisioned and released on-demand with minimal management or service provider effort [26]. Every resource the cloud offers is virtualized and can be accessed via a virtual machine. Cloud computing offers flexibility to customers; they can increase or decrease the amount of resources they need, and not have to worry about maintenance or configuration [26][28][33]. Customers rent resources and just pay for what they use [28]. As such, customers don't have to buy expensive hardware just because they need it for a short period of time; they can just rent it from their cloud provider.

There are five essential characteristics of a cloud Table 2-1 lists and describes them. These are the minimum characteristics that define a cloud [26].

Table 2-1: Characteristics of a cloud [26]

Characteristic	Description
On-demand self-service	Customer can provision computing services without requiring any human interaction.
Broad network access	Service is provided over the network and can be access on a wide range of devices (cellphones, PDA's, tablets, computers, etc.)
Resource pooling	Resources are virtualized and made available to multiple customers.
Rapid elasticity	Customer can provision more resource or release resources easily at any time without human intervention.
Measured Service	The services used must be measureable based on the resource and a bill made out based on the measured use.

2.2.1 Cloud services

There are three types of services offered by a cloud: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [26].

- **Infrastructure as a Service (IaaS)** provides the customer with access to fundamental computing resources such as networks, storage, and processing [26]. With this the customer is free to run any arbitrary software on the recourses including operating systems and manage them as they pleases.

- **Platform as a Service (PaaS)** provides the customer with a platform (for example an operating system) to which they can run any application created using any programing language or tool supported by the cloud provider [26]. The customer has no control over the underlying computing infrastructure.

- **Software as a Service (SaaS)** provides the customer with access to applications running on the cloud [26]. An example could be video editing software. The customer has no control over the application functionality or the underlying infrastructure.

2.2.2 Types of clouds

A cloud can be deployed in four different ways depending on access capabilities to the cloud. The different deployment models of a cloud are:

- **Private cloud:** This type of cloud is used by only one organization [26]. It could be owned and managed by third party, however only operated by one organization [26]. An example of a private cloud is Amazon VPC (Virtual Private Cloud) cloud.

- **Public cloud:** This type of cloud is open for anyone to use [26]. An example of a private cloud is Amazon EC2 cloud.

- **Community cloud:** This type of cloud is used by only by a group of organization sharing similar interests [26]. It could be owned and managed by third party, however only operated by the group [26]. An example is a scientific community cloud.

- **Hybrid cloud:** This type of cloud is the combination of two or more types of could that are separate entities but linked together by some proprietary or standardized technology [26].

2.3 Virtual Box

Virtual Box is a powerful open source virtualization tool developed by Oracle. It follows the type-2 hypervisor structure and can run on Windows, Mac, Linux, and Solaris machines [18]. Some important concepts used by Virtual Box are explained in the subsections below.

2.3.1 Medium

A medium refers to a virtual storage unit. This could either be a CD/ DVD image file, or a virtual hard disk image file [18]. When a VM is created, it is associated with a virtual disk image file that is used to represent the VM's hard disk [18]. This association in Virtual Box is known as attachment. The virtual disk can either be created or the VM can attach a preexisting disk either on the host or on the network. The size range of a disk is between 4MB to 2TB [18]. Virtual box supports six different virtual disk formats:

- VDI (Virtual Box Disk Image)
- VMDK (Virtual Machine Disk)
- VHD (Virtual Hard Disk)
- HDD (Parallels Hard Disk)
- QED (QEMU enhanced Disk)
- QCOW (QEMU Copy-On-Write)

Depending on the virtual disk format, there are options to select how the file growth should behave. VDI, VMDK, and VHD all have the option of creating either a dynamically allocated file or a fixed sized file. A dynamically allocated disk starts small, and grows, as more data is stored [18]. A fixed sized disk pre-allocates the full size of the virtual disk. VMDK formats have a further option to split the disk into several smaller files of a maximum of 2GB [18]. This was done to help with backing up onto devices that cannot handle larger files.

Mediums can be classified as being mutable or immutable, and base or differencing mediums. Mutable mediums save changes made to the storage when VM is powered off and Immutable mediums discard changes made when the VM is powered off [18]. In Virtual Box, the current state of a medium can be saved and all subsequent changes saved as deltas in a separate medium "chained" to the original medium. This medium is known as a differencing medium and the original medium known as the base medium. This strategy allows for multiple VMs share a single base medium by having separate differencing mediums of the same base medium. The left image in figure 2-4 shows this.

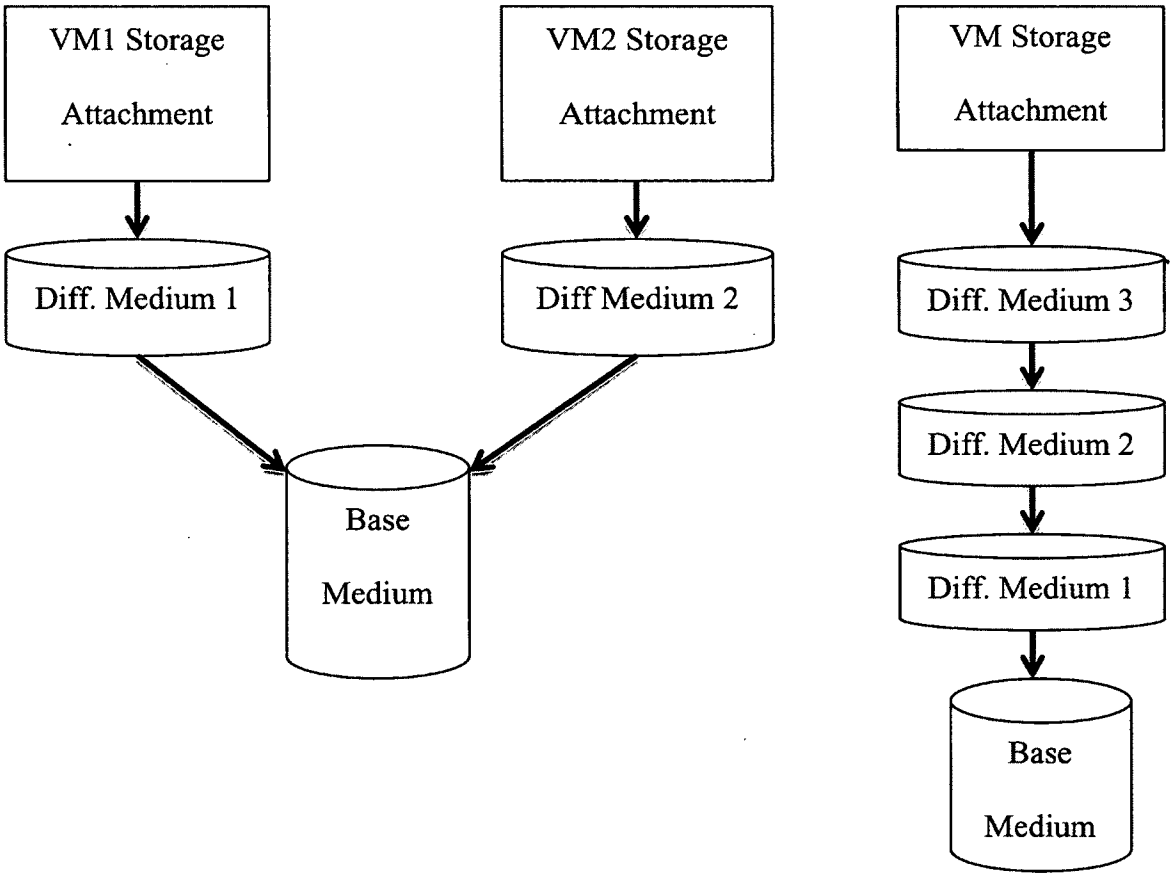

Figure 2-4: Medium Chaining

Differencing mediums play a crucial role in the creation of snapshots (section 2.3.3) In this case, different differencing mediums are created for each snapshot and chained together as shown to the right of figure 2-4.

2.3.2 Teleportation

Teleportation is a function of Virtual Box used to perform live migration of a VM between two different hosts [18]. This is a runtime migration only; the persistent state is not moved. The prerequisites for teleportation are:

- A clone (VM with all settings the same) of the source VM needs to be created at the destination.
- The source mediums must also be attached to the clone.

The second prerequisite assumes that the source's mediums are accessible from the clone. This means the mediums need to be on the LAN or shared over the WAN using iSCSI (Internet Small Computer System Interface) [18]. iSCSI is a networking standard used to access storage devices across a LAN, WAN, or over the Internet [32].

2.3.3 Taking Snapshot

A snapshot is a way of saving the current state of a VM to allow reversion back to that particular state later on. Both the runtime and persistent state are saved. Everything about the current runtime of the machine is saved to a file [18]. To save the persistent state, all mutable mediums are converted to immutable, differencing mediums are created for

them and chained to them [18]. As such the state of the medium is saved as read only and any changes henceforth are made in the differencing image chained to it and saved as change deltas. If a user wants to revert back to a snapshot, the runtime state is loaded from the saved file, and the differencing images associated with that snapshot are just deleted and the immutable medium made mutable again [18]. If a user wants to get rid of a snapshot, the runtime file is deleted and the changes in the differencing medium applied to the immutable medium it is chained to. The differencing medium is then deleted and the immutable medium made mutable.

2.4 Industry Standards

There are many companies and research bodies currently studying, publishing and advancing solutions to VM migration. Most of them use very similar approaches and all of them have their own drawbacks, such as reliability, complexity, and scalability. This section gives a brief description of some of these solutions, and discusses the solutions drawbacks in cloud computing applications.

2.4.1 Big Flat L2

In the 1980's when routing was invented, it was not feasible for routing to be done location independent, due to the amount of processing power and memory that routers would need to achieve this [6]. As such, location information needed to be encoded as part of addresses to enable the building of routers with less processing power and memory [6]. Given advancements in technology, processors and memory are much faster and cheaper

than in the 1980's, this gives rise to the ability to design and build routers that can route based on location independent MAC addresses [6].

The term Big Flat L2 refers to the flattening the Data Link Layer (Layer 2) of a network into on large flat layer 2 [13]. The main benefit of having a Big Flat L2 network is flexibility, plug and play capabilities, resources can be moved without the need for any IP reconfiguration [13]. This is possible because the routing can be done on Layer 2 using MAC addresses. Another bonus is that there won't be any need to decide routes based on the Spanning Tree Protocol (STP) because routing can be done on Layer 2 [13].

For large cloud datacenters, converting to Big Flat L2 can be very expensive [13]. This is because special routers are needed to route on layer 2 as such this will lead to a large change and reconfiguration of the datacenter's networking infrastructure.

2.4.2 Virtual Extendable LAN (VXLAN) [16]

Cloud datacenters today have many Virtual LANs (VLANs) for all the different VM network groups. Spanning Tree Protocol is used to avoid loops in the network due to duplicate paths [16]. However, datacenter operators may end up with redundant ports and links. VXLAN can be viewed as a tunneling scheme to overlay layer 2 networks on top of layer 3 networks. Each overlay is called a VXLAN segment [16]. Different VXLANs are identified with a VXLAN Network Identifier (VNI), which is unique to each VXLAN. As such MAC addresses on different VXLAN segments can overlay each other [16].

The tunnels are unrelated to any previous communication, so each frame is encapsulated according to a given set of rules [16]. VXLAN Tunnel End Point (VTEP) is the end point of the tunnel, and is located within the hypervisor of a host [16]. If two devices hosted on the same physical infrastructure have the same IP address, they would have to be configured on different VXLAN.

The main advantage of VXLAN is that different datacenters can be connected and viewed as though they were on the same LAN. Migration can be done without the need to change the IP address as the clone will still remain on the same VXLAN and packets directed through VXLAN's tunneling scheme.

2.4.3 Virtual Private LAN Service (VPLS) [12]

VPLS is a service provided by many ISPs, which allows different LANs, communicate with each other across a WAN as if they were on the same LAN [12]. In the case of VPLS, clients in a VPN are connected by a multipoint LAN, in contrast to a point-to point connection. VPLS enables MAC address learning, flooding, and forwarding functions in packet switched networks [12]. Ingress PEs (Provider Edges) can directly send a VPLS packet to egress PE without the need for intermediate entities. VPLS supports auto-discovery, in which each PE discovers which other PEs is part of a given VPLS by means of BGP protocol and exchange de-multiplexers [12]. After discovery, all PEs in a VPLS must be able to establish and tear down connections to each other in a process known as signalling. A VPLS site can be connected to multiple PEs. If a VPLS topology changes, only the affected PE's configuration changes [12].

Just as VXLAN, the main advantage of VPLS is that different VPNs can be connected and viewed as though they were on the same private network. Again this does not solve the problem of live migration with IP address change.

2.5 Related Research Work

2.5.1 Live wide area migration including persistent state [23]

The problem with migration across a Wide Area Network (WAN) is that it will require the IP address to change; changing this would break current connections if they were not somehow taken care of. Due to resource contention issues like network bandwidth, processor cycles, local disk throughput, conventional migration bears a high VM downtime. These authors' system performs live migration by transferring the system's persistent state concurrently while the VM performs its own operations in the background [23]. The way the authors achieved this is by employing a pre-copying technique for the persistent state, dynamic DNS to update the new IP address, and tunnelling between the source and target VM so as to enable the source forward any packets it receives to the target VM [23]. Write throttling is used with the pre-copying as a way of delaying future write attempts by the block driver if the VM attempts to complete more writes than a given threshold value [23]. This process repeats, with the delay and threshold doubling each time. These authors' solution was designed using Xen, a type 1 hypervisor on XenoLinux machines. For bulk transfers, Xen's XenoServers platform supports copy-on-write along with immutable template disk images, thus, transferring only the changes, rather than the

whole image [23]. Xen's live migration function is used to complete the migration by transferring the VM's runtime state over and starting the target VM [23]. At this time, the source VM builds a tunnel with the target and starts forwarding any packets it receives to the new VM. This means that the source VM has to remain running until it stops receiving packets.

This solution allows for a VM to be migrated allowing for the IP address to change. However, their tunneling technique will not allow for the source to be shut off as soon as the new VM takes over. Leaving the source VM running may not be ideal in some migration scenarios.

2.5.2 Live migration based on full system trace and reply [8]

Most migration schemes focus on pre-copying techniques where the VM's local run-time memory state is copied to the destination, before migration is completed. These techniques could increase VM downtime since in low speed LANs, the memory pages may be dirtied at a faster rate than that of pre-copy replication [8]. Ideally, a VM should have negligible downtime, low network bandwidth consumption, and low total migration time. This approach employs a check pointing/ recovery and trace/ replay mechanisms to ensure fast VM migration coupled with transparency [8]. The approach is known as CR/TR-Motion [8]. Combining a bounded, iterative log transferring phase, with a short stop and copy phase helps reduce overall downtime. The migration process involves freezing the system state, check pointing, then continuing operations forgetting the freeze phase [8]. Log files are created in the transfer rounds and copied. Concurrently, the destination host

replies with the received log files when it is recovered from the checkpoint. Log transferring is performed continuously till the size of an unconsumed log at destination reaches a threshold value [8]. The stop and copy phase then sets in and suspends the source VM and transfers the remaining log files to the destination host. At the end, a replica of the VM exists on both source and destination hosts and the migrated VM at the destination takes over [8].

This is yet another solution that does not account for IP address change. It focuses on improvements to the runtime state migration.

2.5.3 Effective wide-area live VM migration using distributed content-based addressing [20]

Current migration techniques depend on shared storage between hosts, which avoids transferring the entire file system of the VMs. As such shared storage is usually placed on the local network of a datacenter [20]. A problem arises if migration is needed across different datacenter local networks. These authors' approach addresses this concern and provides an efficient way to transfer the persistent state. This is done by determining what pages exist somewhere on the target network and only transferring those pages that are not [20]. The name of their solution is Shrinker [20]. The approach uses a distributed content based addressing to detect VM memory pages already resident at the destination network. This leverages content based addressing using cryptographic hash functions that are populated by periodic memory indexing of VMs. If a page that is requested already exists on a server at the destination site, they are transferred using the local network rather

than over the Internet [20]. In order to ensure dynamic migration by the above distributed content based addressing mechanism, there are two subsystems that are used. One is a site-wide Distributed Hash Table (DHT), used to locate nodes having a copy of the given page using its hash value. The other is periodic memory indexing, used in order to assist page location. Other functionalities involve a mechanism to ensure pages at multiple locations are not outdated, and another mechanism identifies when a page is stable enough to be copied [20].

This solution only addresses persistent state migration. It does not address network connections or runtime concerns.

2.5.4 Summary

Live migration based on full system trace and reply [8] focuses mainly on a better way to perform runtime state migration. Effective wide-area live VM migration using distributed content-based addressing [20] focuses on a more efficient way to perform persistent state migration. Live wide area migration including persistent state [23] addresses both runtime and persistent state migration. It uses a type 1 hypervisor and a tunneling mechanism to forward packets.

Chapter 3 : Live VM Migration System Design

This chapter describes the design of our VM live migration solution with IP address change. The design process was guided by two main assumptions: All connections to the VM will be made over the network, and the migration is performed to enable the host to be powered down for maintenance. The power down for maintenance assumption was made because it is the worst-case maintenance scenario and will allow for any other form of maintenance such as hardware updates to be done. The virtualization tool used is Virtual Box (Section 2.2). This decision was made due to its stability, open source license, and experience using the tool. Furthermore, we are using type 2 hypervisor in comparison to type 1 used in [23].

3.1 Overview

The activity diagram in figure 3-1 gives an overview of the whole migration process. As a first step, a suitable target host is identified for the source VM migration. The criterion is that the target host must be capable of hosting the VM. That is, there must be enough hard drive space to host the VM's virtual disk, RAM for the VM's memory requirements, processing power to support the VM's processing needs, and have all the I/O device requirements the VM's has access to on the source host. After, a clone of the VM is created having the same settings as the source VM. The creation and setting up of the clone VM at the target host is handled using Virtual Box's remote management tools.

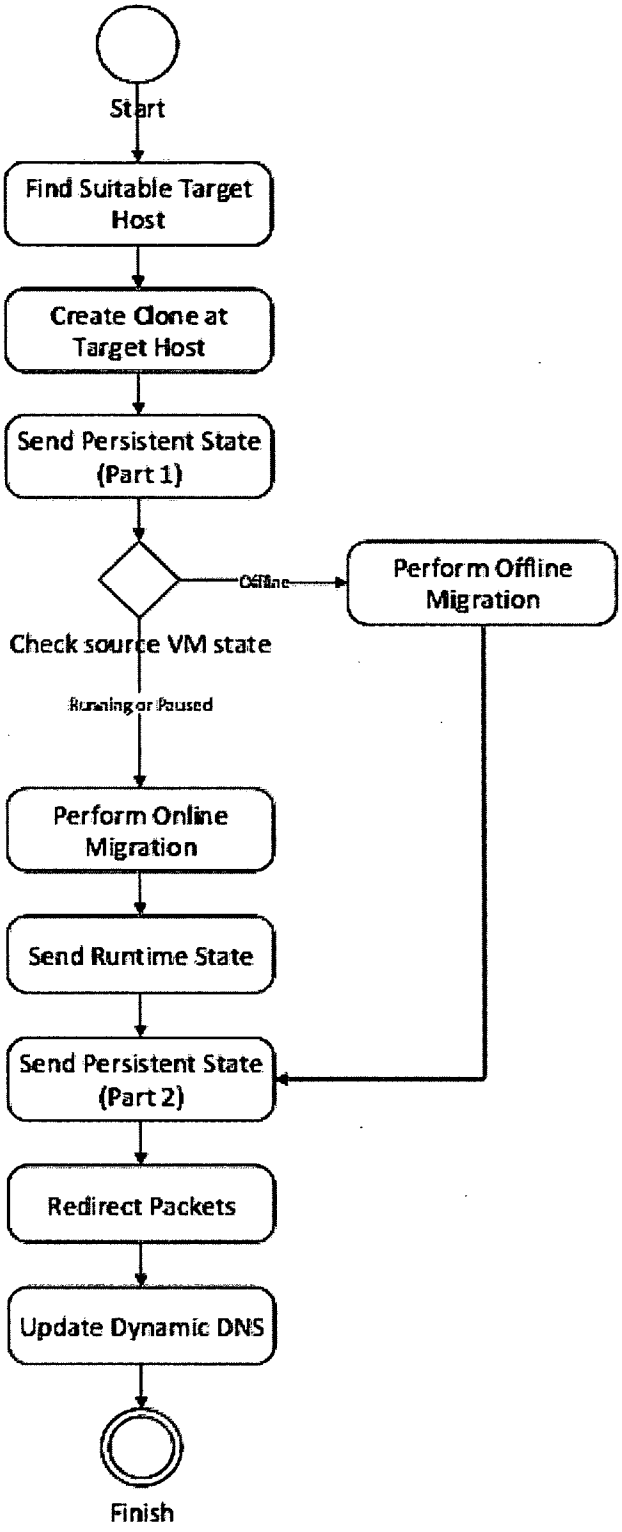

Figure 3-1: Overview of migration process

The migration process is broken up into three main parts: the runtime state, persistent state, and the network connections. The runtime state is the current processing state of the machine. This is a volatile state and consists of data currently in the physical machines volatile storage such as main memory. The persistent state refers to the saved state and consists of data stored in the file system on the virtual disk (Section 2.2.1). The migration of the runtime state is only needed if a VM is currently in a running or paused state (Section 2.2.2). This part of the migration is handled using Virtual Box's teleportation function (Section 2.2.3). Migration of the persistent state is done as a file transfer over the network.

3.2 Client - VM Connections

In designing we considered two types of network connections between a VM and a client: Current connections and new connections. Current connections are live connections between a client and a VM in which packet transfer is in progress while migration is going on, and connections made during migration. For this type of connection, we use packet redirection at the edge routers of the source host's network to reroute packets to the target VM. An alternative is redirection done via tunneling between the source and destination hosts, which would involve the source host being directly involved with the redirection. New connections are network connections trying to be built after the VM has been migrated. For this type of connection, Dynamic IP is used to change the IP address associated to the source VM's domain name to the target VM's IP address. A redirection watchdog was also developed to stop packet redirection once communications to the old IP

address ceases. This is to allow the IP address to be reused on the network without the having packets from external networks directed elsewhere.

The packet redirection is done at the network level through IP table manipulation at the gateway routers of the datacenter's network. This is done by adding a pre-routing rule to the routers IP table that changes the destination address of any packet going to the source VM to that of the destination VM. This rule is explained in more detail in section 3.3.10. This is done at the network level to enable the host to be free of any packet redirection responsibilities that may prevent it from being powered down without breaking client connections. For these same reasons, the watchdog is also run within these gateway routers. These gateway routers are known as Managers within the migration system as they are in charge of managing the traffic coming to the VMs.

3.3 Migration Process

The migration process is handled by a single thread and is broken up into stages that are in turn broken up into steps. Figure 3-2 describes the stages of the migration process as a sequence diagram and explained through the subsections of this section.

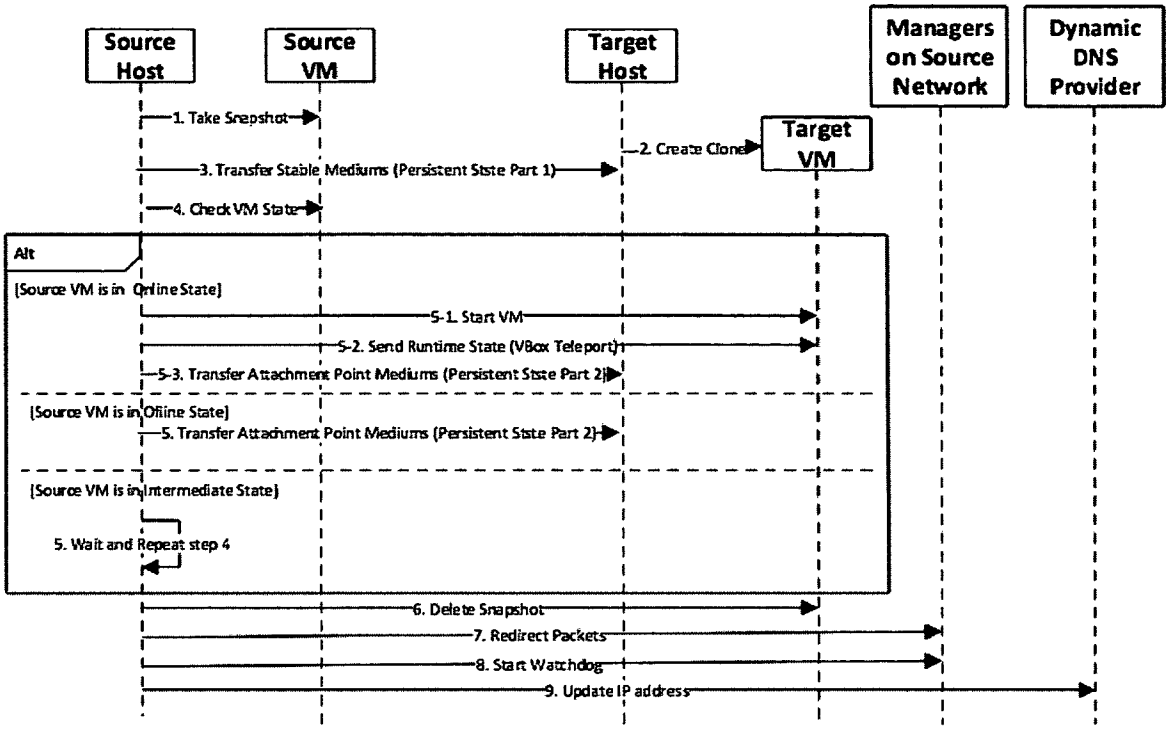

Figure 3-2: Stages of the migration process

3.3.1 Construction

This stage consists of the first two steps of the migration process and together form the Migration Thread Constructor. The first is initializing the thread instance, and the second is adding the migration to the database.

3.3.2 Initialization

This stage involves five initialization steps. The first step is connecting to the VBox Web Servers of the two Hosts. Second step is locking the source machine object for reads. Third step is creating a new Machine object at the destination with the same name as the source and locking it for writes and modifications. Note that this step only creates the VBox folders and does not reserve any memory or have any configuration yet. This is done

later. The fourth and fifth steps connect the source and destination VM command clients to the host command servers at the source and target hosts.

3.3.3 Taking Snapshot

In this stage, a snapshot is taken of the source machine. This is step one in the sequence diagram shown in figure 3-2. This is needed to ensure consistency when transferring the persistent state. The virtual disk images can be very large and so can take a lot of time to be transferred to the target host. Taking a snapshot ensures that any changes made to the virtual disk are stored in a separate differencing disk image as deltas and a stable (unchanging) part of the virtual disks is available for transfer separate from the differencing image. This allowed for all the changes to the virtual disk be transferred closer to the end of the migration process and merged with the already transferred stable part. In addition, taking a snapshot is the only way VBox allows the creation of differencing images.

Currently VBox has a bug with its snapshot mechanism when it comes to remotely taking live snapshots. The only way around it was to pause the machine before the snapshot is taken. The performance degradation caused by this workaround is negligible because taking a snapshot is very quick (less than 1ms) and almost unnoticeable, so the pause time of the machine is also very short and almost unnoticeable.

3.3.4 Cloning

The cloning process is broken up into two steps; first step reads all the settings of the source VM and applies them to the destination VM. The next step reads the medium (disk image) arrangement of the source and creates the same arrangement at the destination. This is an important pre-step to transferring the persistent state data because simply transferring the disk image will fail to attach to a VM. This is because when a disk image is created at a given host, it is given a unique identification number, and is registered to the hypervisor under which it is created before being attached to a machine. If a disk is simply transferred it will not be registered to the new host's hypervisor and as such will not be recognized. Moreover, problems will arise when accessing one or both disks if it as the same unique ID as an already existing disk at the destination. As such, these mediums being created serve as stand-ins until the actual mediums are transferred over from the source. The most important reason for cloning is because VBox's teleport function which is responsible for migrating the runtime state will not initiate if the destination machine does not have an identical medium attachment arrangement and have the same settings as the source. In the sequence diagram (figure 3-2), the two cloning steps form the second step in the sequence (create clone).

3.3.5 Transfer Stable Mediums

This is a major step in the migration process; it is the first part of the persistent state data transfer in which all the stable (immutable) mediums of the persistent state data are transferred. In the sequence diagram (figure 3-2) this is step 3 in the sequence. Each

medium contains a header portion with the details of its registration to the hypervisor such as its unique ID, type, parent medium, and geometry, as can be seen in figure 3-3 below.

```
# Disk DescriptorFile
version=1
CID=6cc1c455
parentCID=ffffffff
createType="monolithicSparse"

# Extent description
RW 20971520 SPARSE "Ubuntu1.vmdk"

# The disk Data Base
#DDB

ddb.virtualHWVersion = "4"
ddb.adapterType="ide"
ddb.uuid.image="7684db56-c007-4052-86ba-0d7c64bff30f"
ddb.uuid.parent="00000000-0000-0000-0000-000000000000"
ddb.uuid.modification="350fdc8f-0937-4d07-b4a4-9f64bf2bf8eb"
ddb.uuid.parentmodification="00000000-0000-0000-0000-000000000000"
ddb.geometry.cylinders="16383"
ddb.geometry.heads="16"
ddb.geometry.sectors="63"
ddb.geometry.biosCylinders="1024"
ddb.geometry.biosHeads="255"
ddb.geometry.biosSectors="63"
```

Figure 3-3: Part of a virtual disk header

ddb.uuid.image represents the unique ID of the disk header. **ddb.uuid.parent** represents the unique ID of this disk's parent, that is, if this disk is attached to another disk for instance a differencing disk image. The **ddb.adapterType** represents the disk adapter type, whether SATA or IDE. The **SPARSE** attribute identifies this disk as being dynamically allocated.

It is difficult to determine the actual size of the disk header; however through studying the structure of VBox's mediums, we can estimate that the first 1181B of a medium contained the important and unique portion of the header that can be skipped without jeopardizing the workings of the VM after migration. As such, when transferring the mediums, the first 1181B were always skipped.

3.3.6 Check Source Machine State

This is step 4 of the sequence in the sequence diagram (figure 3-2). Although the main concern is live (online) migration, for completeness, we also designed the system to support offline migration. This step checks the state of the source VM in order to decide what type of migration to perform. There are numerous states that a VM can be in; however we grouped them into three main categories, Online, Offline, and Intermediate. In the sequence diagram (figure 3-2), the different state categories are represented as alternative paths following step 4 of the sequence. The breakdown of the states can be seen in Table 3-1. An intermediate state is a state where by neither online or offline migration can be performed, such as Powering Off, Powering On, and Teleporting. If the source happens to be in an intermediate state, a wait cycle is injected that will either end on timeout or if the source machine state changes. This is shown as step 5 in the "Source VM is in Intermediate State" alternative path in the sequence diagram (figure 3-2). This is to give the VM a time to enter into a state it can be migrated from. If the wait state times is out, an exception is thrown that will result in an error dialog box to the user, asking if to wait longer.

Table 3-1: Virtual Machine States [31]

Machine State	Description	Category
Running	The machine is currently being executed.	Online
Paused	Execution of the machine has been paused.	Online
PoweredOff	The machine is not running and has no saved execution state; it has either never been started or been shut down successfully.	Offline
Saved	The machine is not currently running, but the execution state of the machine has been saved to an external file when it was running, from where it can be resumed.	Offline
Teleported	The machine was teleported to a different host (or process) and then powered off. Take care when powering it on again may corrupt resources it shares with the teleportation target (e.g. disk and network).	Offline
Aborted	The process running the machine has terminated abnormally. This may indicate a crash of the VM process in host execution context, or the VM process has been terminated externally.	Offline
Stuck	Execution of the machine has reached the "Guru Meditation" condition. This indicates a severe error in the hypervisor itself.	Offline
Teleporting	The machine is about to be teleported to a different host or process. It is possible to pause a machine in this state, but it will go to the TeleportingPausedVM state and it will not be possible to resume it again unless the teleportation fails.	Intermediate
LiveSnapshotting	A live snapshot is being taken. The machine is running normally, but some of the runtime configuration options are inaccessible. Also, if paused while in this state it will transition to Saving and it will not be resume the execution until the snapshot operation has completed.	Intermediate
Starting	Machine is being started after powering it on from a zero execution state.	Intermediate
Stopping	Machine is being normally stopped powering it off, or after the guest OS has initiated a shutdown sequence.	Intermediate
Saving	Machine is saving its execution state to a file, or an online snapshot of the machine is being taken.	Intermediate

Restoring	Execution state of the machine is being restored from a file after powering it on from the saved execution state.	Intermediate
TeleportingPausedVM	The machine is being teleported to another host or process, but it is not running. This is the paused variant of the state.	Intermediate
TeleportingIn	Teleporting the machine state in from another host or process.	Intermediate
FaultTolerantSyncing	The machine is being synced with a fault tolerant VM running elsewhere.	Intermediate
DeletingSnapshotOnline	Like DeletingSnapshot, but the merging of media is ongoing in the background while the machine is running.	Intermediate
DeletingSnapshotPaused	Like DeletingSnapshotOnline, but the machine was paused when the merging of differencing media was started.	Intermediate
RestoringSnapshot	A machine snapshot is being restored; this typically does not take long.	Intermediate
DeletingSnapshot	A machine snapshot is being deleted; this can take a long time since this may require merging differencing media. This value indicates that the machine is not running while the snapshot is being deleted.	Intermediate
SettingUp	Lengthy setup operation is in progress.	Intermediate
FirstOnline	Pseudo-state: first online state (for use in relational expressions).	Intermediate
LastOnline	Pseudo-state: last online state (for use in relational expressions).	Intermediate
FirstTransient	Pseudo-state: first transient state (for use in relational expressions).	Intermediate
LastTransient	Pseudo-state: last transient state (for use in relational expressions).	Intermediate
Null	Null value (never used by the API).	Intermediate

3.3.7 Online and Offline Migration

The online migration is broken up into four steps. First step is to start up the destination machine. This is step 5-1 in the sequence diagram (figure3-3). The destination machine needs to be running and set to receive teleportation in order for VBox to be able

to initiate teleportation between the source and the destination VMs. The second step just waits for the destination VM to power up. This is timed and after an reasonable amount of time (5 minutes), if the machine has yet to power up, an exception is thrown that leads to an error dialog window that asks the user if to wait longer. The third step initiates VBox's teleport function that will transfer the source VM's runtime state to the destination VM. This is step 5-2 in the sequence diagram (figure3-3). The forth step waits for the teleportation to complete. The timeout here is set to a reasonable 2 hours, after which an exception is thrown that leads to an error dialog window that asks the user if to wait longer. Both the 5 minute and the 2 hour timeouts are arbitrary numbers chosen because they were reasonable and had no impact on the overall performance.

With respect to offline migration, no special task needs to be done; as such this step just proceeds straight to the next step. This step is solely in place for logging and detailing purposes.

3.3.8 Sending Attachment Point Mediums

This forms the second part of the persistent state transfer. This is step 5-3 for the online alternative sequence, and step 5 in the offline alternative sequence in the sequence diagram (figure3-3). Given that a snapshot was taken at the beginning of the migration, the attachment point medium (the differencing medium at the start of the medium chain) will only contain change deltas made to the medium from when the snapshot was taken till this step. As such, this medium is always small in size (a few KBs at worst) and took less than one second to transfer.

3.3.9 Delete Destination Snapshot

At this point both the persistent state and runtime have been migrated over. The snapshot taken at the beginning of the migration for persistent state transfer purposes was duplicated at the destination when setting up the destination machine's mediums. As such this snapshot needs to be deleted in order to merge the changes made during the migration process onto the parent medium of the differencing medium and restore the medium chains back to its original state before migration. This is step 6 in the sequence diagram (figure3-3).

3.3.10 Switching IP Addresses

Switching the client over to the new IP address is performed in two steps. First step is redirecting packets, and secondly updating the Dynamic DNS.

The first part of redirecting packets destined for the source to the new IP address involves connecting to the managers to execute remote shell commands via Telnet. In the sequence diagram (figure 3-2), this is step 7 in the sequence. Originally our approach is designed for the connection via SSH, however, the routers being used only support Telnet, so telnet was used as an alternative. The command "iptables -t nat -I PREROUTING -d **sourceIP** -j DNAT --to **destinationIP**" is executed. Where **sourceIP** represents the source VM's IP address and **destinationIP** represents the destination VM's IP address. In the command:

-t represents the IP table. In this case the Network Address Translation table (represented by "nat").

-I represent the insert action meaning that a new rule is to be inserted at the top of the requested chain. In this case the prerouting chain (represented by "PREROUTING"). This chain contains the rules to be executed as soon as a packet is received before routing is done.

-d represents the destination IP address of the packet. In this case the address will be the source VM's IP address.

-j represents the jump which applies the specified rule/function represented by a target to this packet. In this case the target is DNAT, which alters the packets destination address.

--to forms part of the input for the DNAT and represents that the packets destination address should be changed to the specified IP address. In this case the address will be the destination VM's IP address.

The result of this command is that every packet received at this router heading towards the source VM's IP address will have its destination IP address changed to the destination VM's IP address before routing takes place.

The second part is running the watchdog passing to it the rule just added to the prerouting chain. In the sequence diagram (figure 3-2), this is step 8 in the sequence. The iptable watchdog is a simple program that every given time interval checks the counter for the number of times the specified rule has been applied. In our case 30 minutes is chosen

as the time interval. This is arbitrary but fair interval. The command "iptables -t nat -L PREROUTING -v" (-L represents list, and –v represents verbose) returns a verbose list of the prerouting rules. The count (number of times the rule was executed) and rule number (the position of the rule in the chain) are obtained by parsing the return string. The count is then compared with that of the previous interval. If the count has not changed, it is thus assumed that the clients have disconnected themselves; as such the rule is deleted from the chain with the command "iptables -t nat -D PREROUTING **ruleNum**" (-D represents delete, and **ruleNum** represents the parsed rule number) and the watchdog is terminated.

For the second step of switching the IP address involves updating the IP address with a Dynamic DNS provider. In the sequence diagram (figure 3-2), this is step 9 in the sequence. We used No-ip as the dynamic DNS provider. No-ip is a free online dynamic domain name service provider [29] [30] and was chosen because it is a free service and provides an easy to use system updating that just involved sending a HTTP request packet to **http://dynupdate.no-ip.com/nic/update** [29]. The structure of the HTTP request can be seen in figure 3-4 below.

GET /nic/update?hostname=**sourceMachineName**-vmmigrationtool.no-ip.org&myip=**newIPAddress** HTTP/1.0
Host: dynupdate.no-ip.com
Authorization: Basic **username:password**
User-Agent: VMMigrationTool/1.0 johnDoe@abc.com

Figure 3-4: No-ip HTTP request header [29]

This is a HTTP get request. **sourceMachineName** refers to the source machine name. **newIPAddress** referes to the IP address of the destination VM. **username:password** refers to the base64 encoding of the string "username:password" where username and password refer to your No-ip username and password [29]. After the request is sent, a success response from No-ip is listened to. The response sent back contains a status code. Table 3-2 below shows the different status codes that can be received back and what they mean.

Table 3-2: No-ip response Status codes [30]

Status	Type	Description
good IP_ADDRESS	Success	DNS hostname update successful. Followed by a space and the IP address it was updated to.
nochg IP_ADDRESS	Success	IP address is current, no update performed. Followed by a space and the IP address that it is currently set to.
nohost	Error	Hostname supplied does not exist under specified account, client exit and require user to enter new login credentials before performing and additional request.
badauth	Error	Invalid username password combination
badagent	Error	Client disabled. Client should exit and not perform any more updates without user intervention.
!donator	Error	An update request was sent including a feature that is not available to that particular user such as offline options.
abuse	Error	Username is blocked due to abuse. Either for not following our update specifications or disabled due to violation of the No-IP terms of service. Our terms of service can be viewed at http://www.noip.com/legal/tos. Client should stop sending updates.
911	Error	A fatal error on our side such as a database outage. Retry the update no sooner 30 minutes.

3.3.11 Finalization

This step is simply to ensure the migration process exits properly. At this point the target host is running the clone VM. This step involves deleting the snapshot taken at the source, unlocking both the source and destination VMs, killing the database update thread, and closing all connections opened by the migration tool the hosts.

3.4 Summary

This chapter has presented the full design of our live migration system that allows for IP change. We have done this by breaking the system up into three areas, runtime state, persistent state, and network connections.

The migration of the runtime state is handled by Virtual Box's teleportation function, which assumes a clone of the source VM already exists at the destination (section 3.3.4), and that the virtual storage mediums (persistent state) of is already attached to the clone VM.

We designed the migration of the persistent state to be handled in four stages. First stage involved taking a snapshot (section 3.3.3), which will save the current state of all persistent state storage mediums and create differencing mediums that will store any coming changes to the medium as change deltas. The second stage is transferring the saved

stable mediums to the destination (section 3.3.5). The third stage is to transfer the attachment point differencing mediums (section 3.3.8). The final stage is to apply the changes in the attachment point differencing mediums by deleting the snapshot (3.3.9).

To account for the change in IP address we considered two types of client VM connections, current or ongoing connections and new connections. To handle current connections, we used packet redirection at the network level. This was done by adding a rule to the prerouting chain of the NAT IP table at the gateway routers, which we refer to them as managers (section 3.3.10). We also developed a watchdog that removes the rule after a time interval that it was never executed. To handle new connections, dynamic DNS was used to change the IP address associated to the source VM's domain name to that of the clone.

As a proof of concept, we developed a migration tool that automates our migration process in a simple and user-friendly manner. This is discussed in more detail in the next chapter.

Chapter 4 : Live VM Migration Prototype

This chapter describes the implementation of the design of the live migration with IP change system. As a proof of concept, we developed a software migration tool to perform migration requests. The software is developed in java using Netbeans. Java and Netbeans were chosen because of my experience coding in Java and using as an IDE. In addition, Virtual Box provides Java libraries and APIs for the remote management functionality.

4.1 Overview

The software tool is divided into five major components: the hosts, managers, database, core, and graphical user interface (GUI). Figure 4-1 shows how all these components are linked together. The manager consists of the traffic-monitoring watchdog. Each host cosists of a VBox webserver and a Host command Server. The VBox webserver is the server Virtual box uses for its remote management functionality. The host command server executes the custom remote commands not supported by Virtual Box's webserver. The core consists of the Brain and Migration Thread. The core communicates with all other components and serves as the central control unit of the software. Communication with the hosts, managers, and database are over remote connections over a local or wide area network.

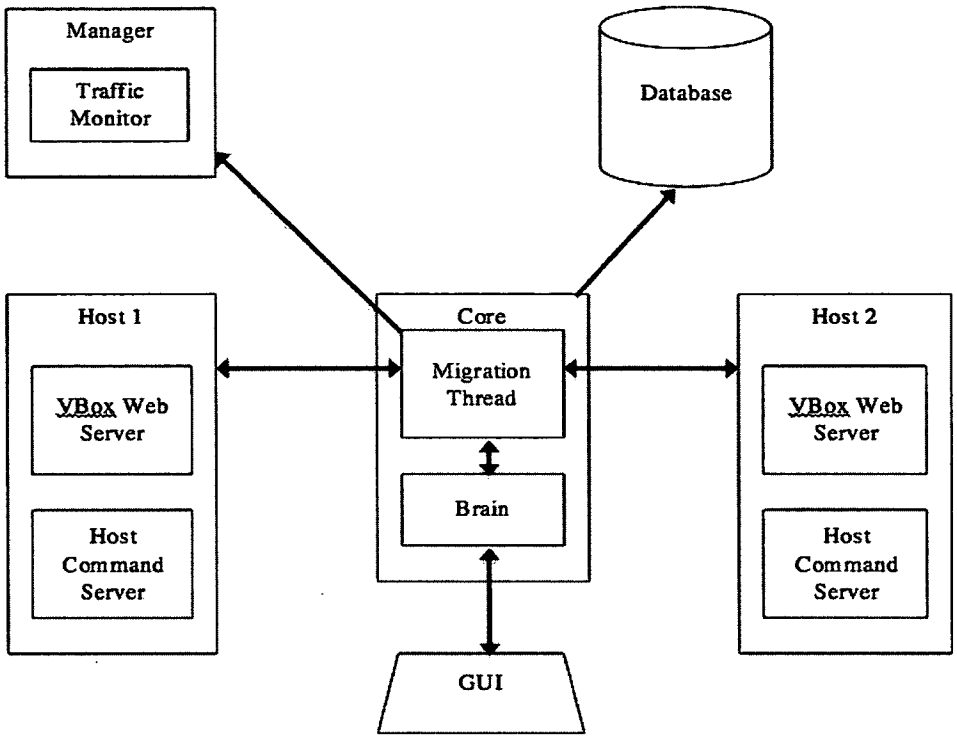

Figure 4-1: Software Component Layout

4.2 Database

The database provides a place where all the information needed by the tool to perform a migration can be kept and organized. The database also plays a key role in synchronization of the data if multiple instances of the software are running simultaneously. We used MYSQL to implement the database. Figure 4-2 consisting of six entities shows the conceptual modeling of the database. Each entity is reflected in the database as a table. The name of each table is represented in brackets in the entities in figure 4-2.

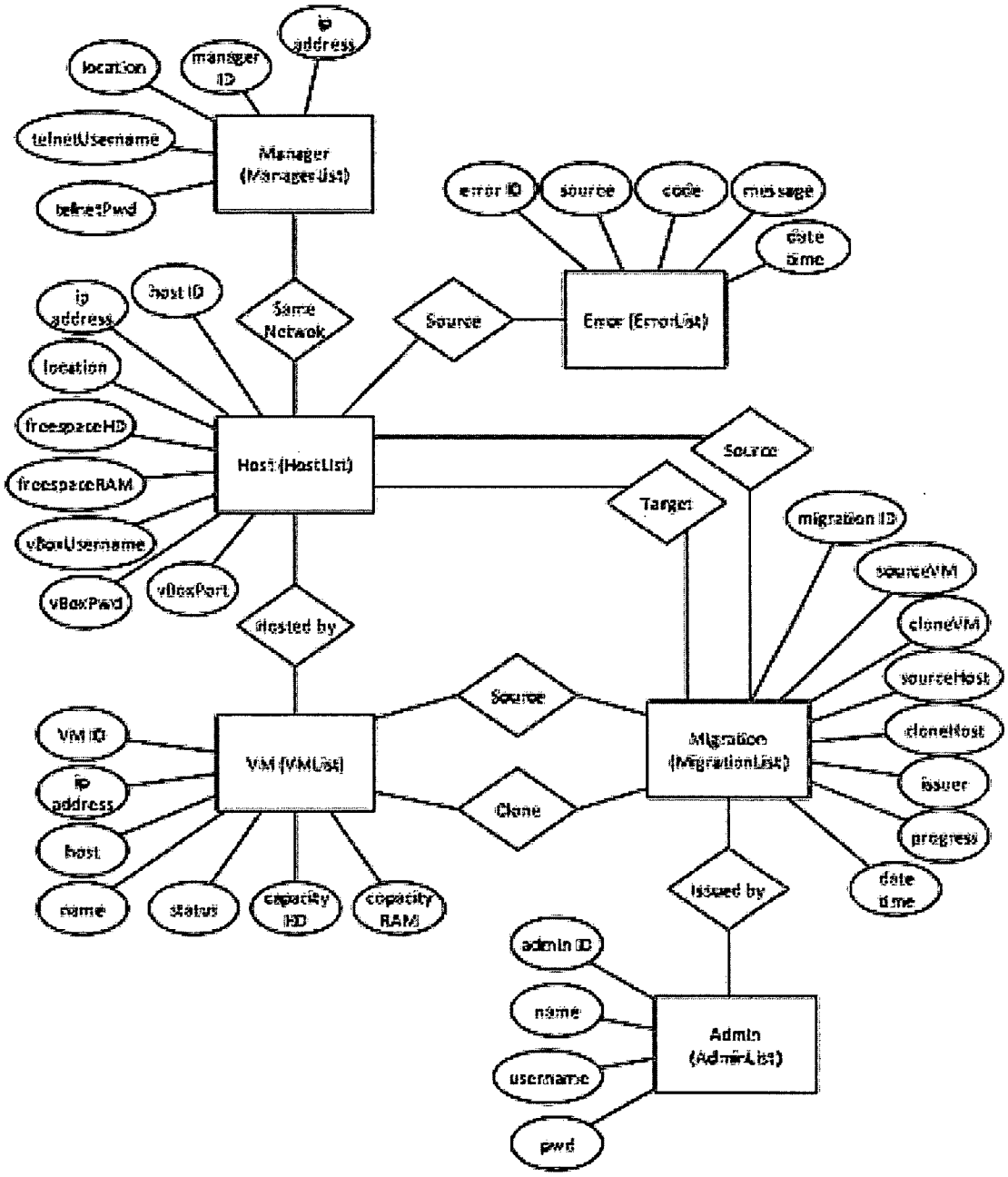

Figure 4-2: Database entity relationship

AdminList: This table holds information about authorized users of the software and is used in the login process. Its attributes are the users name, username and password (pwd).

ErrorList: This table keeps a record about any errors that have occurred during previous migration attempts. Its attributes are source, error code (code), error message (message), and date and time (date time). The source is the host at which the error occurred. The error code is a numerical representation of the type of error as shown in table 4-1.

HostList: This holds information about host servers and is used by the software to list suitable target hosts for a migration. Its attributes are the host's IP address, name of the network its located on (location), amount of free space on the hard drive (freespaceHD), amount of free space on the RAM (freespaceRAM), the host's Virtual Box web server's remote username (VBoxUsername) and password (VBoxPwd), and the port that the host's Virtual Box web server is listening on (VBoxPort). When a VM is created, the user must allocate RAM out of the total RAM on of the host. The free space RAM attribute is the amount of RAM not yet allocated.

ManagerList: This table holds information about all the managers, and is used by the software to figure out which managers to send redirection commands to. Its attributes are the manager's IP address, name of network its located on (location), the manager's Telnet username (telnetUsername) and password (telnetPwd).

VMList: This table holds information about all the VMs on all the hosts. Its attributes are the VM's IP address, ID of the host its located on (host), the name of the VM, current migration status, and the capacity of its hard drive (capacityHD) and RAM (capacityRAM). Table 4-2 lists and describes our classification of the different migration statuses of a VM.

MigrationList: This table holds a record of all current and completed migrations performed. Its attributes are the source VM (sourceVM) and target VM (cloneVM), source host (sourceHost) and target host (cloneHost), name of the admin that initiated the migration (issuer), progress of the migration (progress), and the date and time the migration started (date time).

Table 4-1: Migration errors

Error	Code	Description	Causes
Redirect Error	0	Packet redirection was unsuccessful.	Cannot access Manager's Telnet.
File Transfer Failed	1	Persistent state transfer was unsuccessful.	Problem with connection to host command server.
Machine State Error	2	Could not read machine state.	Problem with connection to Virtual Box web server.
Teleportation Error	3	Runtime state transfer was unsuccessful.	Problem occurred with Virtual Box teleportation function.
DNS Error	4	IP address update was unsuccessful.	Problem updating with Dynamic DNS provider.
VBox Failed	5	Virtual Box error.	Problem with connection to Virtual Box web server.

Table 4-2: Migration Statuses

Status	Description
Active	The VM can be accessed by a client and has not been migrated.
Inactive	The VM has been migrated, and cannot be accessed by any client.
Migrating	The VM is currently being migrated.

For ease of use, we developed java classes to assist in the setting up and populating of the database. To simplify access to the database, we developed a database utility class as a façade to the database. Figure 4-3 shows the class diagram of these classes. The DatabaseSetup class creates the database, and all its tables and attributes. The AdminSetup class takes a name, username and password as parameters and adds it to the database. The ManagerSetup class takes an IP address, location, and Telnet username and password as parameters and adds it to the database. The HostSetup class needs to be run from a host that is to be added to the database. It takes Virtual Box's webserver username, password, and port to which it should listen on as parameters and adds them to the database. It also adds the hosts IP address, location, free space on its hard drive and RAM. Finally, it gathers the name, IP address, status, RAM and hard drive capacities of each VM on the host and adds them too to the database.

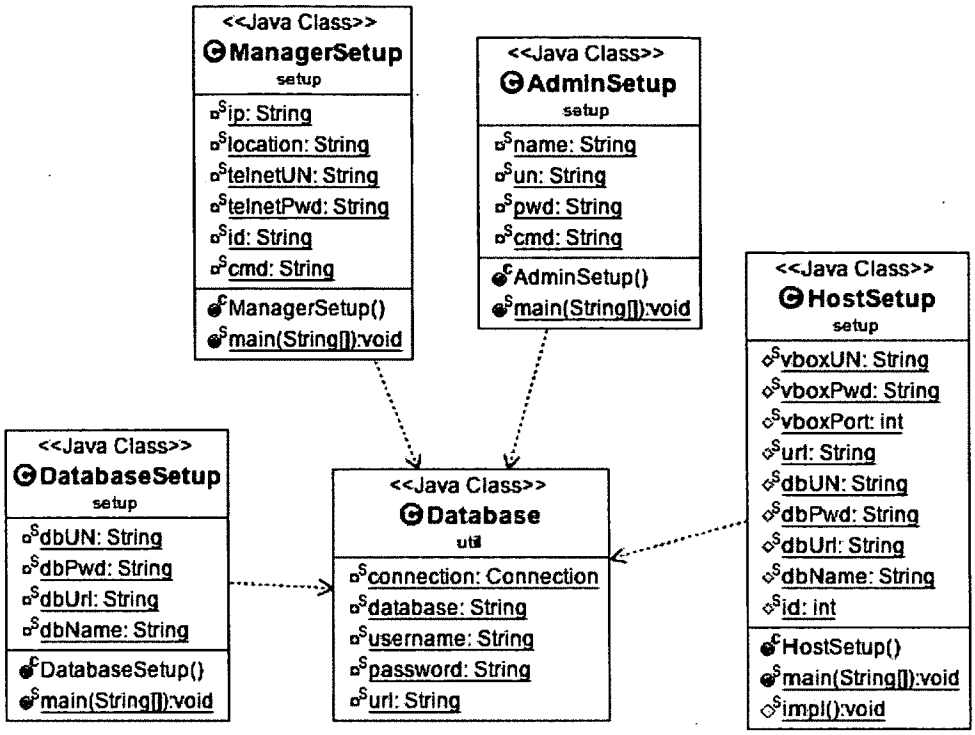

Figure 4-3: Database setup class diagram

4.3 Hosts

As shown in figure 4-1, every host comprises of two servers, the Host Command Server and the Virtual Box Web Server. The system requires both these servers be left running on every host for migration purposes. The Virtual Box Web Server enables Virtual Box's remote management capabilities. This server listens on a specified port for remote commands. The default port is 18083. The Virtual Box web server does not support commands for manipulating files on the host, as such, we developed host command server (HostCmdServer) to enable this support. We use the file manipulation commands to transfer the VM's persistent state. We also included the support to remotely ask the host to update the information on the database. This is used at the end of migration to update the

host's information on the database. Table 4-3 gives the different commands supported by the host command server.

Table 4-3: Host commands

Command	Parameters	Description
Receive File	Source, Location, Port, Skip	This command when issued spawns a FileReciever thread that listening on **Port** for a file transfer from **Source**, saves the file to **Location** and the first **Skip** bytes.
Delete File	Location	This command when issued deletes the file at **Location**.
Send File	Target, Location, Port	This command when issued spawns a FileSender thread that starts sending the file at **Location** to the **Target** IP on **Port**.
Database Update		This command when issued updates the database host information for its specific host.
Stop Server		This command when issued stops the host command server handler.
Connect Server	Port	This command when issued will spawn a handler thread listening on **Port**.

We also developed a Host Command Client (HostCmdClient) to send the commands. The host command client and server work together to form the communication (Comm) software. Figure 4-4 shows a sequence diagram illustrating how the two work together. The host command server listens on an arbitrarily chosen port (we chose 9000) for a connection request with a port number as the argument. When the request is received, the server spawns a client handler thread that will listen on the port sent as an argument. A connection is then built between the client and the server's client handler thread for sending commands. This threaded design is done to enable a host to handle multiple

simultaneous connections to command clients. When a command is sent, the client waits for a response from its handler. Table 4-4 lists the different responses and their descriptions.

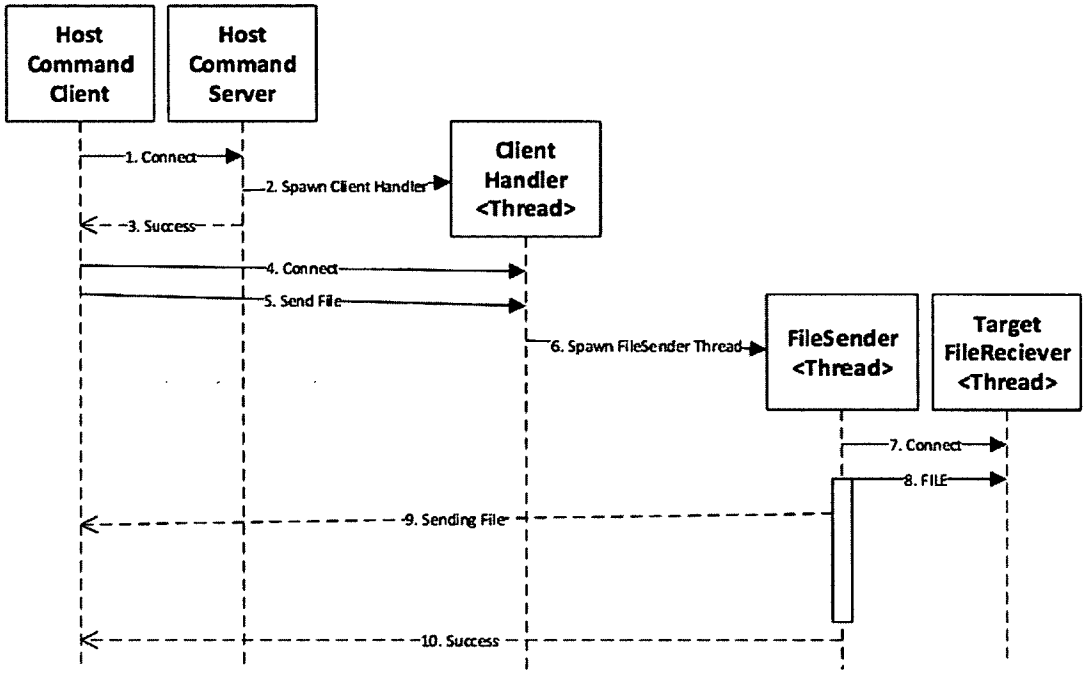

Figure 4-4: Host command client and server sequence diagram

Table 4-4: Host command server responses

Response	Description
Success	This response is issued in successful completion of the command.
Failed	This response is issued for a general failure in execution of the command.
IO Failure	This response is issued for a failure due to an IO Error. This usually means that the something went wrong with the communication channel.
Sending File	This response is issued to indicate that the file to be transferred has started being sent.
Receiving File	This response is issued to indicate that the file to be received has started being received.
File Does Not Exist	This response is issued for a failure due to a File Does Not Exist Error. This usually means that the file requested could not be found for some reason.

Unknown Host Failure	This response is issued for a failure due to an Unknown Host Error. This usually means that the specified IP address could not be accessed.

4.4 Migration Tool

The Migration tool's communication with managers is done via Telnet, and communication with the host is executed as previously described in section 4.3. The classes in the tool are grouped into 10 different packages. Figure 4-5 shows the relationships between the packages and Table 4-5 shows the breakdown of the packages and gives a description of each class within the packages.

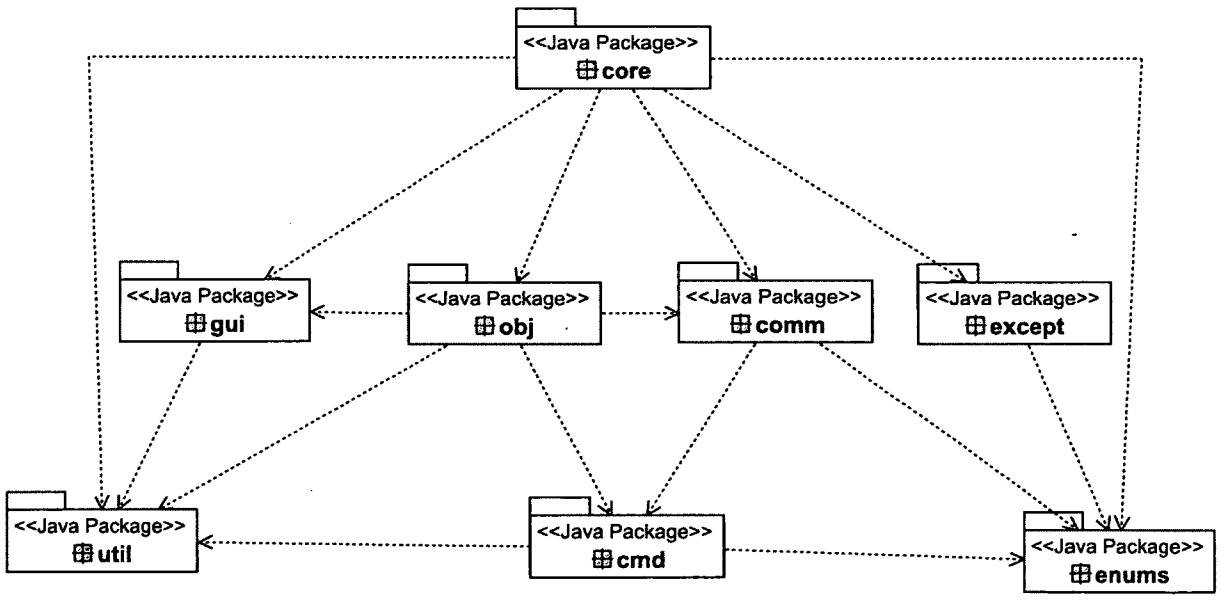

Figure 4-5: Software package relationship

Table 4-5: Class description

Class	Package	Description
FileReceiver	Command (cmd)	Command thread that receives a file.
FileSender	Command (cmd)	Command thread that sends a file
HostUpdate	Command (cmd)	Command thread that updates the host's information in the database.
HostCmdClient	Communication (comm)	Connects with **HostCmdServer** class object at host and sends commands.
HostCmdServer	Communication (comm)	Listens for **HostCmdClient** connections and spawns **Clienthandler** thread.
ClientHandler	Communication (comm)	Listens for commands from **HostCmdClient** and spawns Command threads.
Brain	Core	Control class between the GUI and the rest of the system.
Migration Thread	Core	Coordinates an entire migration process. (Section 4.4.1)
DatabaseUpdate	Core	Updates progress of migration to the database.
ErrorCode	Enumeration (enum)	Types of errors for **ErrorMessage** class. (Table 4-1)
HostCommand	Enumeration (enum)	Types of host commands used by **HostCmdClient**, **ClientHandler** and **HostCmdServer**. (Table 4-3)
HostResponse	Enumeration (enum)	Types of host responses used by **HostCmdClient**, **ClientHandler** and **HostCmdServer.** (Table 4-4)
DNSException	Exception (except)	Indicates problem updating Dynamic DNS
FileTransferException	Exception (except)	Indicates problem sending persistent state
HostException	Exception (except)	Indicates a problem originating from a host.
MachineStateException	Exception (except)	Indicates a problem accessing a VM's state.
ManagerException	Exception (except)	Indicates a problem originating from a manager.
MigrationException	Exception (except)	Indicates a problem in the migration process.
RedirectException	Exception (except)	Indicates a problem trying to redirect packets.
SnapshotException	Exception (except)	Indicates a problem with Virtual Box's snapshot system.

TeleportationException	Exception (except)	Indicates a problem with Virtual Box's teleportation System.
VmException	Exception (except)	Indicates a problem originating from a VM.
ErrorDialog	GUI	Dialog box that pops up when an error occurs during migration. (Figure 4-8)
LoginWindow	GUI	Login screen for authorized users to gain access to the migration tool's **MainWindow**.
MainWindow	GUI	Home window of the tool. (Figure 4-9)
HostInfoPanel	GUI	Part of **MainWindow** holding information about a selected host.
InitialOverviewPanel	GUI	Part of **MainWindow** holding information about the system in general.
ManagerInfoPanel	GUI	Part of **MainWindow** holding information about a selected manager.
VmInfoPanel	GUI	Part of **MainWindow** holding information about a selected VM. (Figure 4-9 Viewing Section)
MigrationInProgressPanel	GUI	Part of **MainWindow** holding information about a particular migration.
ErrorMessagePanel	GUI	Part of **MainWindow** holding information about a particular migration related errors.
MigrationWizardWindow	GUI	Main Window used to initiate and monitor a migration. (Figure 4-10)
Step1Panel	GUI	Part of **MigrationWizardWindow** where the source host of the migration is selected.
Step2Panel	GUI	Part of **MigrationWizardWindow** where the source VM of the migration is selected.
Step3Panel	GUI	Part of **MigrationWizardWindow** where the target host of the migration is selected.
ReviewPanel	GUI	Part of **MigrationWizardWindow**

		holding a summary of the selected migration criteria before starting the migration.
MigrationDetailsPanel	GUI	Part of **MigrationWizardWindow** holding detailed information about the progress of a migration. (Figure 4-10)
Admin	Object (obj)	Entity class holding information about an admin retrieved from the database.
ErrorMessage	Object (obj)	Entity class holding information about an error retrieved from the database.
Host	Object (obj)	Entity class holding information about a host retrieved from the database.
Manager	Object (obj)	Entity class holding information about a manager retrieved from the database.
Migration	Object (obj)	Entity class holding information about a migration retrieved from the database.
VM	Object (obj)	Entity class holding information about a VM retrieved from the database.
Database	Utility (util)	Boundary class to the database.
PerformanceCollector	Utility (util)	Used by the **MigrationThread** to collect performance metrics for migrations. (Section 4.4.2)
Metric	Utility (util)	Abstract performance metric class.
CountMetric	Utility (util)	Subclass of **Metric**. Used to collect count metrics.
TimeMetric	Utility (util)	Subclass of **Metric**. Used to collect time metrics.

4.4.1 Migration Thread

The Migration thread serves as the processing unit in charge of coordinating the migration process. A thread handles the migration process and it is designed to have a separation of the actual migration process from the workings of the software itself, and to allow for multiple migration processes to run simultaneously.

The migration thread handles migration in a step-by-step basis as outlined in section 3.3. This is implemented to allow a more accurate identification of the progress of the migration for both the user and error analysis. On entering each step, a message is relayed back to the user via the GUI's **MigrationDetailsPanel** (Table 4-5). When a problem with the migration is encountered, depending on the nature of the problem, the migration thread will act appropriately (Section 4.4.3).

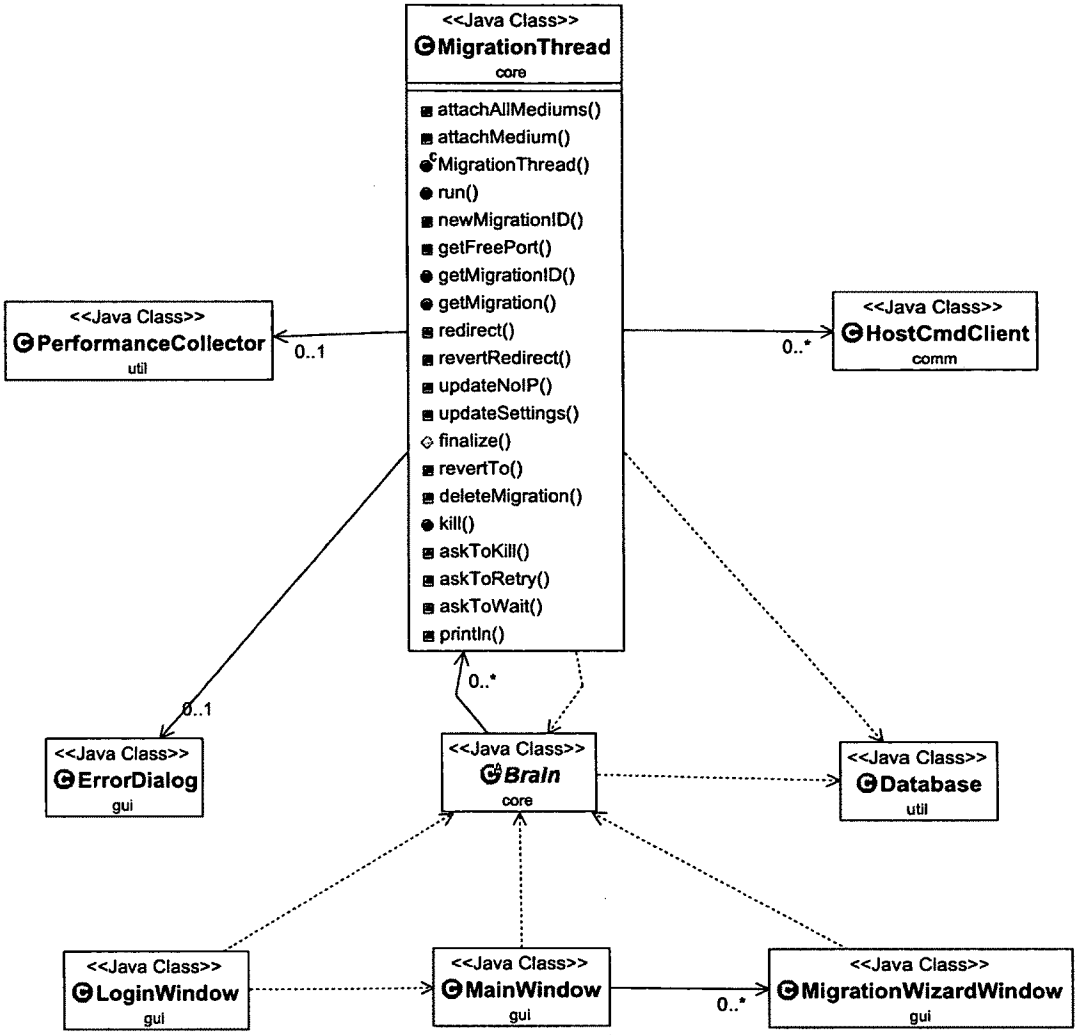

Figure 4-6: Migration Thread Class Diagram

Figure 4-6 shows the Migration Thread's class diagram and relationships with some of the other classes involved with the migration process.

4.4.2 Performance Collection

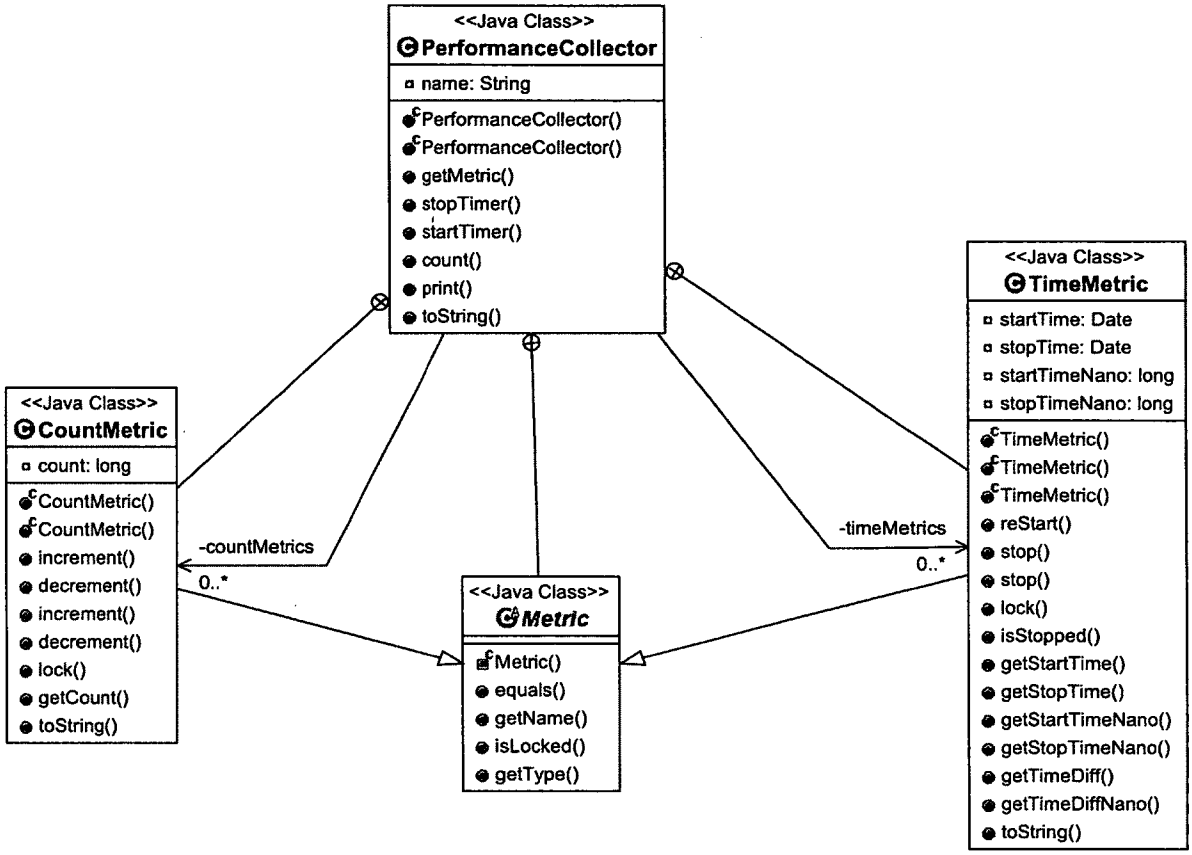

Figure 4-7:Performance Collector Class Diagram

The PerformanceCollector utility uses the CountMetric, and TimeMetric classes. Figure 4-7 shows the class diagram and shows the relationships between the classes. We implemented three methods for collecting performance metrics in the PerformanceCollector class, startTimer, stopTimer, and count. Metrics are stored in two arrays, one for count metrics and the other for time metrics. The methods take a string argument that represents a label for the metric. If the label does not exist in the array it is added as a new metric. The PerformanceCollector class is used by the Migration Thread to

collect performance metrics of a migration, and the results are displayed at the end of the migration and logged.

4.4.3 Error Handling

Error handling is implemented to improve robustness by providing failover capabilities for certain types of errors. Errors vary depending of what they are and what step they occurred in. We classify errors into three categories: fatal errors, moderate errors and small errors.

A fatal error is an error that the migration process cannot recover from. As such, fatal errors result in the termination of the migration process. An example of a fatal error is a connection break to a VBox web server.

Moderate errors are errors that require the users input to resolve it. In this case an error dialog window pops up to the user given options that will either resolve the problem or terminate (kill) the migration process. Figure 4-8 shows an image of the error dialog window. Depending on the problem the resolution option could either be to wait longer, retry the step, or skip the step entirely. An example of a problem that can be resolved by waiting longer is a timeout exception thrown if the VM is taking too long to leave an intermediate state. An example of a problem that can be resolved by retrying the step is if the transfer of a persistent state file failed. An example of a problem that can be resolved

by skipping the step is if an error occurs when trying to delete the snapshot at the destination. This is because this can always be done manually.

Small errors are errors that can be resolved by skipping the step do not impact the performance of the migration process. These types of errors are resolved automatically. An example of such an error is a problem updating the dynamic DNS. For instance, the packet redirection mechanism will act as a failsafe to make sure packets still get to the target VM.

We implemented a revert function that restores the migration process back to a specified step by working backwards to undo the changes of steps. This function is used when retrying a failed step and terminating the migration process midway.

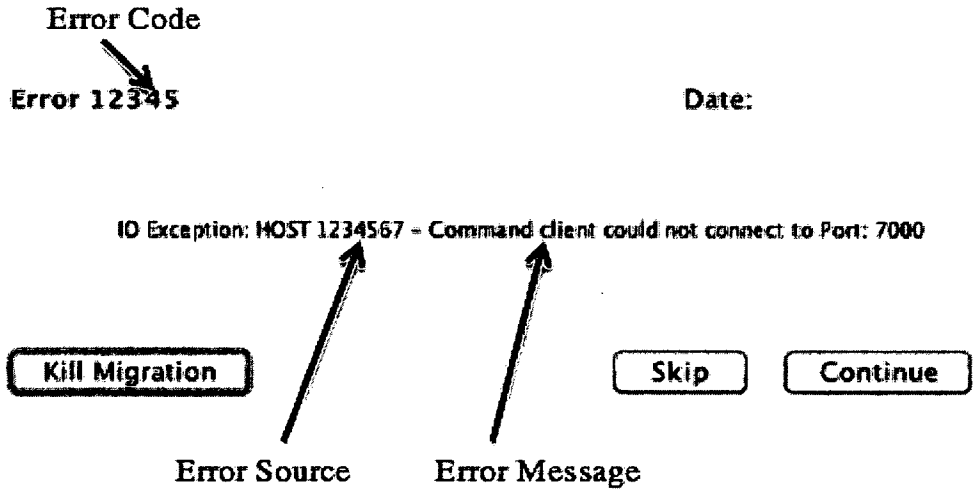

Figure 4-8: Error Dialog

4.4.4 Graphical User Interface

Launching the software will open the login window. On correctly logging in, the main window is opened. Figure 4-9 shows the Main window. The main window is broken up into two main sections, the selection section and the viewing section. In the selection section the user can select which Host, VM, or Manager's information they want to view and the information about the selection is shown in the overview panel of the viewing section. When a selection is made, the other two selection lists are modified to only contain relevant options. For instance, if a particular host is selected, the VM list is modified to contain only the VMs within that particular Host, and the Manager list is modified to only contain the managers on the same network as the selected host. The error and migration tabs in the viewing section contain detailed information about migration errors, and migrations currently in progress or recently completed respectively.

At the bottom right of the Main window is the migrate button that launches the Migration Wizard. This wizard is used to initiate and monitor migrations. We broke up the initialization of a migration into steps. The first step provides a drop down list of all the hosts for selection of a source Host. Step 2 shows a dropdown list of all the VMs in the selected host for selection of the source VM. The third step displays a dropdown list of all the suitable target hosts for selecting the destination host. The next step displays a review of all the selections. In this step a start button will be shown that will initiate the migration according to the selections. When the migration is started, a detailed display of the

migration progress is shown. Figure 4-10 shows the migration wizard detailed progress view.

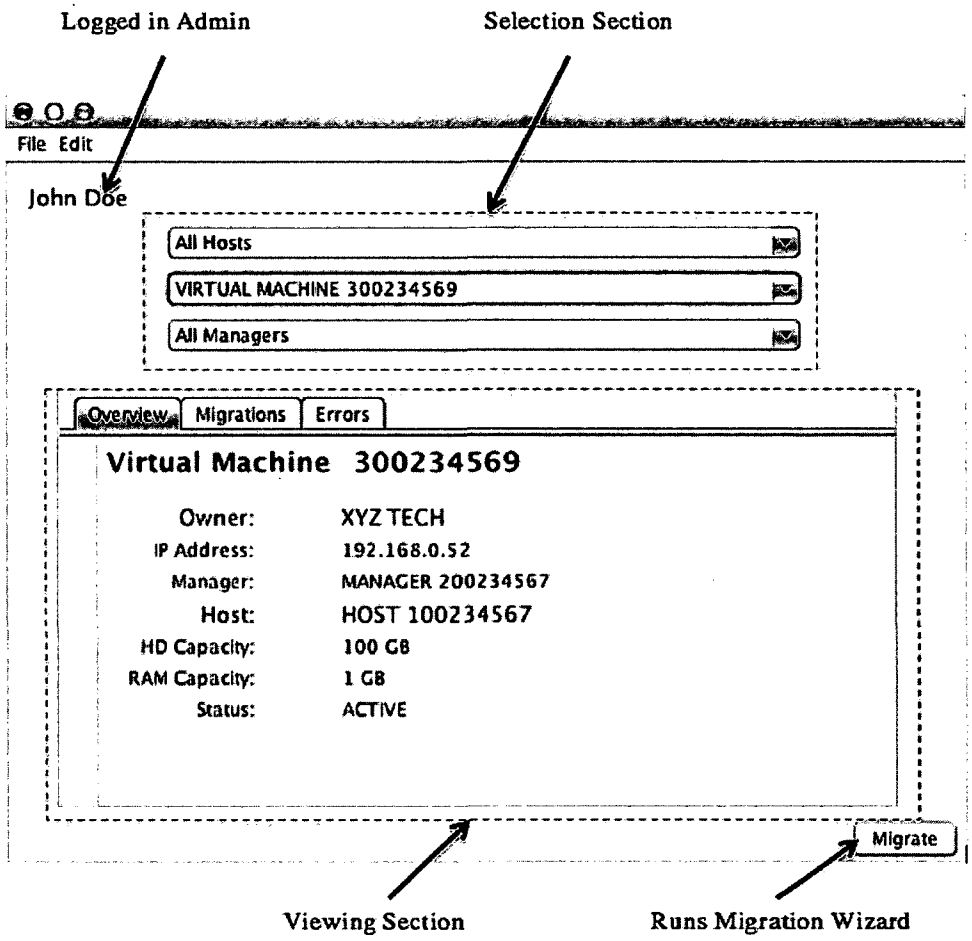

Figure 4-9: Main Window

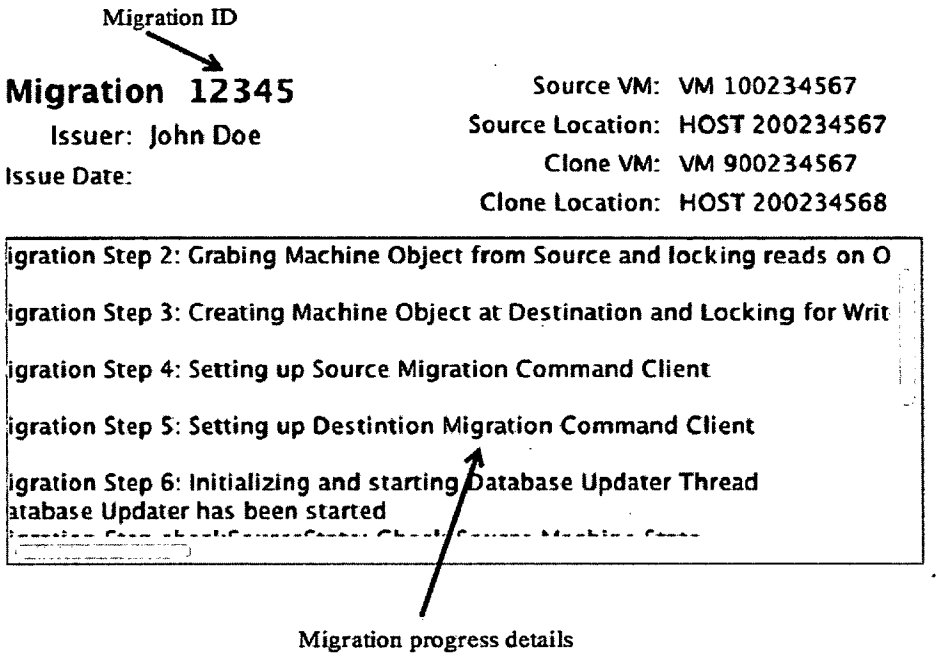

Figure 4-10: Migration Wizard

Chapter 5 : Testing and Evaluation

This chapter discusses the evaluation of the prototype. The testing and evaluation is based on a hardware upgrade maintenance scenario. This scenario will require the host to be powered down, and is chosen because of its critical nature. As such the main performance metrics that were evaluated are total completion time, and total disruption time.

The evaluation is focused on online migration (live migration) and compared to the results of freeze and copy migration. Freeze and copy is a live migration technique that "freezes" the source VM, transfers the VM to the target host, and "unfreezes" it. The way this is achieved in our case is by pausing the VM, creating the clone at the destination, transferring the persistent and runtime state to the target VM, redirect packets and update dynamic DNS, and finally resuming the VM.

5.1 Experimental Setup

Figure 5-1 shows the experimental setup for our testing. For this testing we used two hosts and one manager in the setup. The hosts are a 13" Macbook Pro running Mac OSX Mountain Lion and an Acer Aspire 5734Z running Windows 8. Both hosts ran Virtual Box 4.2.6. Table 5-1 shows the specifications of these hosts. The manager used is a DLink DIR-300 router with a DDWRT v24 firmware. DDWRT is an Open Source routing firmware that allows remote rooting of the router. The MySQL database used is set up on

the Macbook Pro. The VMs setup on each host were created with 512 MB of RAM, 10 GB VMDK virtual disk dynamically allocated running Ubuntu or Windows XP.

Although our prototype works over WANs, we conducted our tests on a LAN to simplify testing. The other reason is that we want to compare a "wired" connection to a "wireless" connection.

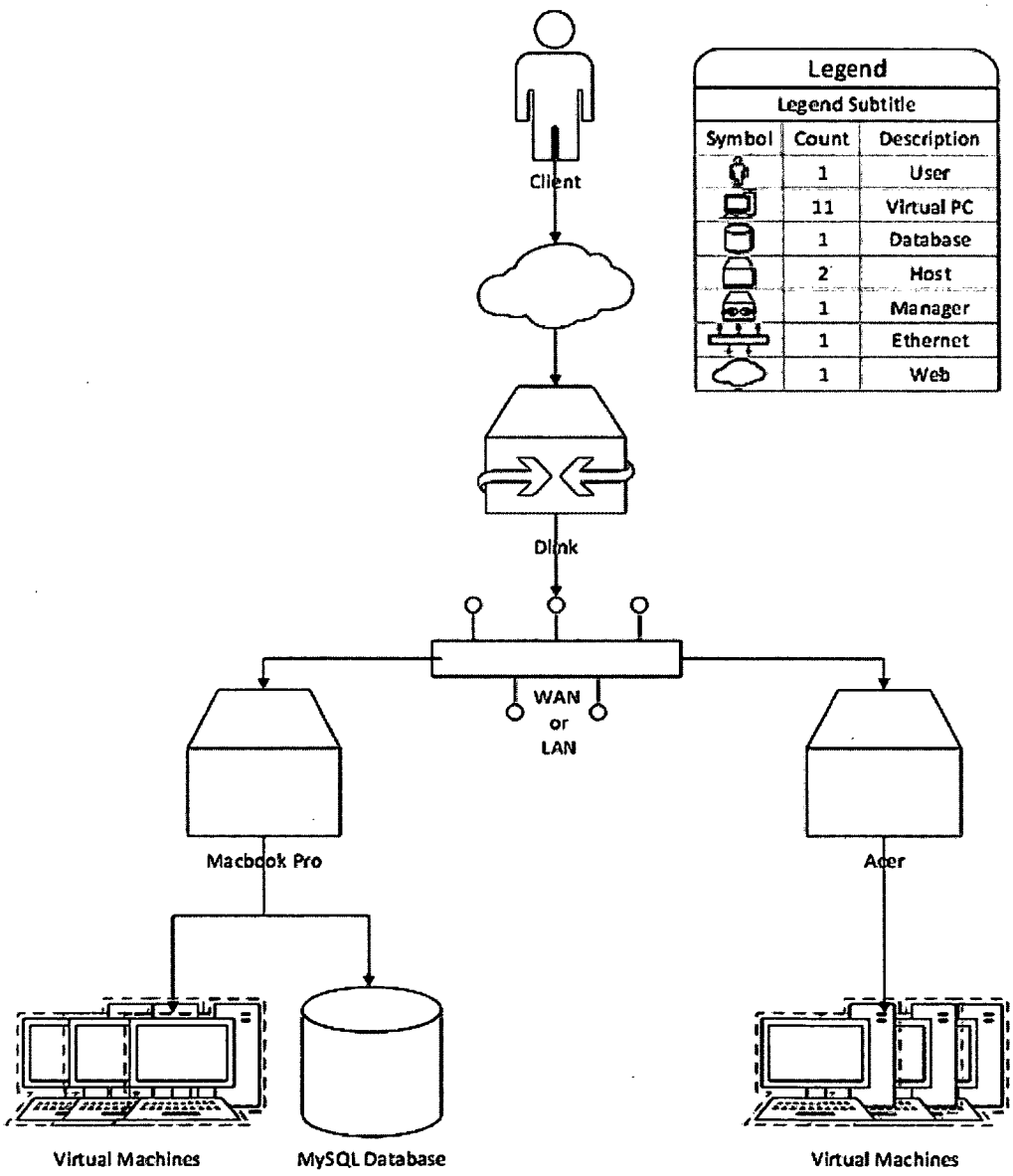

Figure 5-1: Experimental Setup

Table 5-1: Host specifications

	Acer Aspire 5734Z	Macbook Pro
Operating System	Windows 8	OSX Mountain Lion
Processor	Intel Dual Core	Intel i5
Hard Drive Capacity	250 GB	500 GB
Total RAM	4GB	8GB
Virtual Box Version	v4.2.6	v4.2.6
Number of VMs	4	7
Available Disk Space	127 GB	340 GB
Available RAM	2GB	4GB

5.2 Testing and Results

We used two types of tests to evaluate the prototype: time tests and quality of experience tests. For timing tests, we considered total migration time and total disruption time as performance metrics. Total migration time is the time from initiation of the migration to the completion. This metric is selected because it directly relates to the time in which the host can be powered down. Total disruption time is the total time the VM is paused or services are unavailable for use by the client. This metric is selected because it gives an idea of what service disruptions the client will experience. This time can be used to give more accurate billing to the client.

5.2.1 Timing Tests

The performance collector class collected all timing data. Figure 5-2 shows a snippet of a migration log showing the performance collection output for an online migration with the source VM state as running. In this figure, **timeDifferenceNano** is the time difference of the metric in nanoseconds. For testing, we ran a migration of the same

Ubuntu VM from the Macbook Pro host to the Acer host 10 times for different source VM states, and for offline, online, and freeze and copy migrations. The reason for the 10 tests is to average out the timing results. For every one of the three migration types, we observed a consistency with the timing results. Table 5-2 shows the average of the timing results for each migration type. We collected 12 different time metrics as shown in figure 5-2, however we only display 5 in table 5-2. This is because theses are the 5 key time metrics we would like to focus on. All other time metrics with the exception of the "send stable medium time" were either unchanging or changed by an order of nanoseconds. The "send stable medium time" is only affected by the size of the stable medium and distance of the transfer. We could not test offline migrations for Teleported, Aborted, and Stuck states because we could not reproduce these states.

Performance Metrics - Migration_5

SourceVM - Ubuntu1

Migration – Online

Total_Disruption_Time - timeNano= 1709367597

Count Metric - Number_of_Failed_Stable_Medium_Transfers : count = 0

Count Metric - Number_of_Failed_Attachment_Point_Medium_Transfers : count = 0

Time Metric - Total_Run_Time : startTime= Thu Feb 21 15:23:42 EST 2013 , stopTime= Thu Feb 21 15:34:57 EST 2013 , timeDifferenceNano= 675647850876

Time Metric - Initialization_Time : startTime= Thu Feb 21 15:23:42 EST 2013 , stopTime= Thu Feb 21 15:23:51 EST 2013 , timeDifferenceNano= 9238694674

Time Metric - Source_Snapshot_Time : startTime= Thu Feb 21 15:23:51 EST 2013 , stopTime= Thu Feb 21 15:23:52 EST 2013 , timeDifferenceNano= 1459367597

Time Metric - Destination_Attach_Mediums_Time : startTime= Thu Feb 21 15:23:52 EST 2013 , stopTime= Thu Feb 21 15:23:55 EST 2013 , timeDifferenceNano= 3246598567

Time Metric - Send_Stable_Medium_Time : startTime= Thu Feb 21 15:23:55 EST 2013 , stopTime= Thu Feb 21 15:28:51 EST 2013 , timeDifferenceNano= 296754889534

Time Metric - Destination_Power_Up_Time : startTime= Thu Feb 21 15:28:52 EST 2013 , stopTime= Thu Feb 21 15:28:53 EST 2013 , timeDifferenceNano= 1357521108

Time Metric - Teleportation_Time : startTime= Thu Feb 21 15:28:53 EST 2013 , stopTime= Thu Feb 21 15:33:58 EST 2013 , timeDifferenceNano= 305026785693

Time Metric - Send_Attachment_Point_Medium_Time : startTime= Thu Feb 21 15:33:58 EST 2013 , stopTime= Thu Feb 21 15:33:58 EST 2013 , timeDifferenceNano= 78456785

Time Metric - Redirect_Time : startTime= Thu Feb 21 15:33:58 EST 2013 , stopTime= Thu Feb 21 15:34:02 EST 2013 , timeDifferenceNano= 4365467311

Time Metric - DNS_Update_Time : startTime= Thu Feb 21 15:34:02 EST 2013 , stopTime= Thu Feb 21 15:34:08 EST 2013 , timeDifferenceNano= 6175857325

Time Metric - Destination_Delete_Snapshot_Time : startTime= Thu Feb 21 15:34:08 EST 2013 , stopTime= Thu Feb 21 15:34:09 EST 2013 , timeDifferenceNano= 865487397

Time Metric - Final_Step_Time : startTime= Thu Feb 21 15:34:09 EST 2013 , stopTime= Thu Feb 21 15:34:57 EST 2013 , timeDifferenceNano= 48983442854

Figure 5-2: Performance collector output

Table 5-2: Timing Test Results

Migration Type	Machine State	Snapshot Time	Telep. Time	Attachment point Medium Send Time	Total Migration Time	Total Disruption Time
Online	Running	1.53 sec	6 min	0.07 sec	11min 27sec	1.7 sec
Online	Paused	0.89 sec	5 min	0.001 sec	10min 8sec	N/A
Freeze and Copy	Running	N/A	5min	N/A	10min 8sec	10min 8sec
Freeze and Copy	Paused	N/A	5min	N/A	10min 8sec	N/A
Offline	Powered Off	0.4 sec	N/A	0.001 sec	5min 57sec	N/A
Offline	Saved	0.4 sec	N/A	0.001 sec	5 min 57sec	N/A

The metrics not shown in table 5-2 are not affected by the type of migration so are irrelevant in comparing the different migration methods. Comparing the results of online and freeze and copy for a running VM, we observed that the total migration time of freeze and copy is better than that of live online migration by 1 minute and 19 seconds on average. This means that freeze and copy is 1.13 times faster than our live approach to complete a migration. Although freeze and copy will complete migration faster, it has a 358 times worse total disruption time. The disruption time is calculated, as the total time the VM is paused or inaccessible by the client. In our case, being that the VM needs to be paused in order to take the snapshot, this time contributed to the total disruption time. The second disruption is seen during teleportation. This exact disruption time is difficult to measure, as such; we assume the full time constraint for the teleportation disruption to be the teleportation disruption. By default in Virtual Box, this constraint is 250ms [18]. The way virtual box uses this constraint is that if it calculates that it will not be able to perform a migration with a disruption time less than or equal to the constraint, it will not begin teleportation. The actual process of this calculation is not documented, and the measurement of the actual disruption is never given.

Both hosts used a wired network connection during the testing for the results shown above. This was switched to a wireless connection to explore the changes in results. It was observed that this change drastically increased total time of migration by 4 to 5 times. This result is as expected given the reduced bandwidth for wireless connection.

We tested the theory that the size of the virtual disk will affect the total migration time. To do this, we initially created and migrated five VMs with virtual disk sizes of 5GB, 20GB, 30GB, 40GB and 50GB. Surprising results were received with all five tests with a total migration time of between 5 to 6 min. After some investigation, it is discovered that the size of the total contents of the virtual disk is affecting the migration time and not just the mere size of the virtual disk. In essence, all five virtual drives only truly had about 3GB of content in them and as such took roughly the same time to transfer. To amend our tests we used video files to pad up the virtual disks to almost full with new test scenarios as 5GB, 10GB, 15GB, 20GB and 25GB. Table 5-3 shows the results of these tests.

Table 5-3: Migration Times (Different Virtual Disk Sizes)

Virtual Disk Size	5GB	10GB	15GB	20GB	25GB
Average Migration Time	11min 54sec	18min 42sec	25min 27sec	32min 20sec	39min 9sec

5.2.2 Quality of Experience Tests

The Quality of Experience (QoE) tests are performed to investigate any disruptions, inabilities, and restrictions that a client may experience during the migration process. For these tests the migration process is divided into three sections, start, middle, and end. The start section includes the construction and initialization steps. The middle section refers to the migration steps from taking the source VM snapshot till the finalization step. The end section begins at the finalization step and after migration. The results of these tests are classified as no disruption, slight disruption, and heavy disruption. No disruption refers to

either zero or unnoticeable disruptions experienced. Slight disruptions refer to small but noticeable disruptions. Heavy disruptions refer to disruptions that last for several seconds. This reporting scheme was chosen due to the inability to accurately measure these disruptions. Table 5-4 shows the results for certain activities performed during the migration process.

Table 5-4: Quality of Experience Results

Activity	Start	Middle	End
Remote Access	No Disruption	Slight Disruption	No Disruption
Video Playback	No Disruption	Slight Disruption	No Disruption
Application Launch	No Disruption	Slight Disruption	No Disruption
File Access	No Disruption	Slight Disruption	No Disruption
Web Access	No Disruption	Slight Disruption	No Disruption

The results show that a slight disruption was experienced for all activities in the middle of the migration. This disruption only happened when the snapshot is being taken. This is because the VM needed to be paused for the duration of the snapshot because of the bug in Virtual Box's remote snapshotting mechanism. We predict that without this bug, the disruption caused by taking the snapshot will not be noticeable. There were no other disruptions noticed for the remainder of the tests.

Chapter 6 : Conclusion

6.1 Summary

This thesis researched live virtual Machine migration over a wide area network allowing for the VM's IP address to change. Live VM migration is the transfer of a VM from one host to another while the VM is running. Current solutions involve making thing transfer across a flat network. This means that the transfer is done on Layer 2, as such the IP address remains unchanged [13]. Flattening and entire cloud network can be very complex and expensive because special routers and reconfiguration of the network would be needed [13].

We deigned our solution using a Virtual box (a type 2 hypervisor). In our design, we identified three key aspects: runtime state migration, persistent state migration, and network connections handling. Virtual Box's teleportation function handled the runtime state migration. We divided our persistent state migration into three stages; first a snapshot is taken of the source VM at the beginning of the migration to save the state of the virtual disks and save coming changes to a differencing medium. Second the stable mediums are transferred to the target host before the runtime state is migrated. After the runtime state is migrated, the third stage is sending the attachment point differencing mediums, and applying the changes stored in the differencing mediums by deleting the snapshot. We handle the network connections by using dynamic DNS to update the IP address of the

VM's domain name, and packet forwarding to redirect packets. This redirection was achieved via IP table manipulations on the managers in the source host's network.

Our design is validated through a software prototype. The prototype consistes of five components: host, manager, database, GUI, and core. The host represents the host server holding VMs. It consists of a VBox webserver for remote Virtual Box commands, and a Host Command Server for our custom commands. The manager represents a gateway router on the host's network. It consists of a traffic-monitoring watchdog used to monitor packet redirection. The core is the mail control unit for the prototype; it consists of a Brain that coordinates the entire system, and a Migration Thread responsible for coordinating all migration activity. The database holds vital information about the hosts, VMs, managers, migrations, errors, and admins. Access to this database does not affect the migration process as it is used by our prototype to get information about the different entities before any migration is initiated. The migration thread uses a database update thread to update the database without affecting the migration process.

The prototype is evaluated with timing and quality of experience tests. Results were compared against freeze and copy migration, which pause the VM at the beginning of the migration, transfers the VM to the target host, then resumes the VM. Results showed that freeze and copy is 1.13 times faster in completion than our live migration, but it is 358 times worse in total disruption time compared to our solution. For the quality of experience tests, the only slight disruption was experienced when the initial snapshot is being taken. We noted that because of the bug in Virtual Box's remote snapshotting mechanism, the

VM is paused before taking the snap shot. This results in a significant increase in disruption during snapshots. Without the bug, we expect that there will be negligible or no disruption experienced during snapshotting.

Comparing our solution with the approach in [23], our system excels in actual completion time, which is measured as the time from beginning of migration until when the source host is completely out of the migration process and can then powered down. This is because our approach redirects packets at the edge of the network. The approach used in [23] uses tunneling between the source and destination VMs to forward packets. This limitation prevents the source host from being powered down until the tunnel has been taken down.

6.2 Future Work and Recommendations

6.2.1 Sending attachment point mediums

At the end of our migration process when sending the attachment point mediums, if an attachment point medium is too large, or growing too rapidly, inconsistencies between the source and target VMs may exist after this medium is sent. To prevent this, an addition to the design is needed to check the size of this medium, and if it is too large, take yet another snapshot and transfer the immutable part first before the differencing part. This can be placed in a loop to constantly do this until a small enough size is reached. At the end, all these new snapshots would need to be deleted.

An alternative solution would be to pause the source VM before sending the attachment point medium to make sure no new changes are made. This will have a slight disruption to the client.

6.2.2 Cross hypervisor migration

Our migration system is designed for migration between two host servers both running Virtual Box. To improve flexibility, migration across different hypervisors, could be explored. Virtual Box already supports many different virtual disk images used by other hypervisors. Virtual Box also has an export function that packages the VM in a way that other hypervisors may understand. Exploring this function can give an insight into how cross hypervisor migration can be performed.

6.2.3 Simultaneous migrations

Our migration system currently provides functionality that allows multiple migrations to be performed simultaneously. However, the tests performed involved single migrations at a time. The reason for this is because the hosts used for testing were not capable of running more than a few VMs simultaneously. It will be advantageous to determine the limits of simultaneous migrations. As such, our prototype can be installed on much larger hosts so as to be able to perform these tests properly.

REFERENCES

[1] W. Voorsluys, J. Broberg, S.Venugopal, R.Buyya. *Cost of Virtual Machine Live Migration in Clouds: A Performace Evaluation.* Department of Computer Science and Software Engineering, The University of Melbourne, The University of New South Wales, Sydney: Cloud Com, 2009.

[2] Waldspurger, Carl. *Memory Resource Management in VMware ESX Server.* VMWare Inc, Palo Alto: Symposium on Operating Systems Design and Implementation, 2002.

[3] VanDoorn, L. *Hardware Virtualization Trends.* T.J. Watson Research Center, IBM, New York: IBM Corporation, 2006.

[4] VMWare. *Virtualization Overview.* Witepaper, Palo Alto: WMware Inc, 2006.

[5] Alam, Naveed. "Survey On Hypervisors." School Of Informatics and Computing, Indiana University, Bloomington.

[6] *Cloud Computing Basics.* http://tech.blogeous.com/tag/cloud-computing-basics/ (accessed 2012).

[7] Intel. *Intel Virtualization Technology.* 2006. http://www.intel.com/technology/itj/2006/v10i3/1- hardware/3-software.htm (accessed 2012).

[8] H. Liu, H. Jin, X. Liao, L. Hu, C. Yu. *Live Migration of Virtual Machine Based Full System Trace and Replay.* School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan: High Performance Distributed Computing, 2009.

[9] J. Szefer, R. B. Lee. *Architectural Support for Hypervisor-Secure Virtualization.* Princeton University, Princeton: Architectural Support For Programming Languages and Operating Systems ASPLOS XVII, 2012.

[10] J. R. Santos, Y. Turner, G. Janakiraman, I. Pratt. *Bridging the Gap between Software and Hardware Techniques for I/O Virtualization.* Hewlett Packard Laboratories, University of Cambridge, Palo Alto: USENIX Annual Technical Conference, 2008.

[11] K. Adams, O. Agesen. *A comparison of software and hardware techniques for x86 virtualization.* VMware, San Jose: Architectural Support For Programming Languages and Operating Systems ASPLOS'06, 2006.

[12] K. Kompella, Y. Rekhter. "Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling." Juniper Networks, 2007.

[13] Layland, Robin. *Flat Networks are the way to go.* 2011. http://www.channelworld.in/features/flat-networks-are-way-go (accessed 2012).

[14] LeVasseur, Joshua, Uhlig, V. Yang, Y. Chapman, M. Chubb, P. Leslie, B. Heiser, Gernot H. "Pre-virtualization: Soft layering for virtual machines." Technical Report, Karlsruhe University, University of New South Wales, Watson Research Center, Watson Research Center , Karlsruhe, 2006.

[15] M. Hines, K.Gopalan. *Post-Copy Based Live Virtual Machine Migration Using Adaptive Pre-Paging and Dynamic Self-Ballooning.* Binghamton University, Binghamton: Virtual Execution Enviroments VEE '09, 2009.

[16] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, C. Wright. "VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks." 2012.

[17] M.Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, O. Wasserman, B. Yassour, A. Gordon, A. Liguori. "The Turtles Project: Design and Implementation of Nested Virtualization ." Linux Technology Center, IBM.

[18] Oracle Corporation. *Oracle VM Virtual Box User Manual.* Oracle, 2012.

[19] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R.Neugebauer, I. Pratt, A. Warfield. *Xen and the Art of Virtualization.* Computer Laboratory, University of Cambridge, Cambridge: Symposium on Operating Systems Principles SOSP'03, 2003.

[20] P. Riteau, C. Morin, T. Priol. Shrinker: Efficent Wide-Area Live Vertual Machine Migration using Distributed Content-Based Addressing. Centre de recherche INRIA , Rennes, : INRIA, 2010.

[21] S. Sim, S. Han, J. Park, S. Lee. *Seamless IP Mobility for Flat Architecture Mobile WiMAX Networks.* SK Telecom, Yonsei University, Seoul: IEEE, 2009.

[22] Spector, S. *Why Xen?* Xen Org, 2010.

[23] R. Bradford, E. Kotsovinos, A. Feldmann, H. Scioberg. *Live Wide-Area Migration of Virtual Machines Including Local Persistent State.* Deutsche Telekom Laboratories, Berlin: Virtual Execution Enviroments VEE'07 , 2007.

[24] *Ring Computer Security.* http://en.wikipedia.org/wiki/Ring_%28computer_security%29 (accessed 2012).

[25] T. Maoz, A. Barak, L. Amar. *Combining Virtual Machine Migration with Process Migration for HPC on Multi-Clusters and Grids.* Department of Computer Science , The Hebrew University of Jerusalem, Jerusalem : IEEE, 2008.

[26] Peter Mell, Timothy Grance. "The NIST Definition of Cloud Computing ." Computer Security Division , National Institute of Standards and Technology , Gaithersburg , 2011.

[27] Dacentec. *What is a Cloud Datacenter.* 2011. http://www.dacentec.com/What+is+a+Cloud+Datacenter (accessed 2013).

[28] O. Brian, T. Brunschwiler, H. Christ, B. Falsafi, M. Fischer, S. G. Grivas, C. Giovanoli, R. E. Gisi, R. Gutmann, M. Kaiserswerth, M. Kündig, S. Leinen, W. Müller, D. Oesch, M. Redli, D. Rey, R. Riedl, A. Schär, A. Spichiger, U. Widmer, A. Wiggins, M. Zollinger, M.Kaiserswerth. *Cloud Computing* . White paper, Swiss Academy of Engineering Sciences , Seidengasse : Socity of American Travel Writers SATW, 2013.

[29] No-IP. *Request Method.* https://www.noip.com/integrate/request/ (accessed 2012).

[30] No-IP. *Internet Response.* https://www.noip.com/integrate/response/ (accessed 2012).

[31] Oracle Coorperation. "Programming Guide and Reference ." Oracle, Redwood City.

[32] Rouse, Margaret. *What is iSCSI?* 2011. http://searchstorage.techtarget.com/definition/iSCSI (accessed 2012).

[33] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia. *A View of Cloud Computing.* eliable Adaptive Distributed Systems Laboratory, University of California Berkeley, Berkeley: Communications of the ACM, 2010.

[34] Alan Shieh, Srikanth Kandula, Albert Greenberg, Changhoon Kim. *Seawall: Performance Isolation for Cloud Datacenter Networks* . Cornell University, Microsoft Research , HotCloud '10, 2010.