BOOTSTRAPPING A PRIVATE CLOUD

by

Deepika Kaushal

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the Degree of

Master of Science

Department of Computer and Information Technology

West Lafayette, Indiana

August 2020

THE PURDUE UNIVERSITY GRADUATE SCHOOL

STATEMENT OF COMMITTEE APPROVAL

Dr. Thomas J Hacker, Chair

Department of Computer and Information Technology

Prof. Philip T Rawles

Department of Computer and Information Technology

Dr. Jin Wei-Kocsis

Department of Computer and Information Technology

Approved by:

Dr. Eric T. Matson

Head of the Graduate Program

ACKNOWLEDGMENTS

I wish to gratefully acknowledge Dr. Thomas J Hacker for his continued guidance and support throughout my research. I would like to thank Prof. Phillip Rawles and Dr. Jin Kocsis for their insightful comments and review of my work. I am truly blessed to have my friends and family alongside me through thick and thin.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

DHCP	Dynamic Host Configuration Protocol
HaaS	Hardware as a Service
IAC	Infrastructure as Code
IP	Internet Protocol
MaaS	Metal as a Service
NFS	Network File System
PXE	Preboot Execution Environment
REST	Representational State Transfer
TFTP	Trivial File Transfer Protocol
VSFTPD	Very Secure FTP Daemon

ABSTRACT

Author: Kaushal, Deepika. M.S.
Institution: Purdue University
Degree Received: August 2020
Title: Bootstrapping a Private Cloud
Major Professor: Thomas J Hacker

Cloud computing allows on-demand provision, configuration and assignment of computing resources with minimum cost and effort for users and administrators. Managing the physical infrastructure that underlies cloud computing services relies on the need to provision and manage bare-metal computer hardware. Hence there is a need for quick loading of operating systems in bare-metal and virtual machines to service the demands of users. The focus of the study is on developing a technique to load these machines remotely, which is complicated by the fact that the machines can be present in different Ethernet broadcast domains, physically distant from the provisioning server. The use of available bare-metal provisioning frameworks require significant skills and time. Moreover, there is no easily implementable standard method of booting across separate and different Ethernet broadcast domains. This study proposes a new framework to provision bare-metal hardware remotely using layer 2 services in a secure manner. This framework is a composition of existing tools that can be assembled to build the framework.

CHAPTER 1. INTRODUCTION

In this era of Big Data and Internet of Things, Cloud Computing has become ubiquitous. Cloud Computing provides a number of services over the Internet varying from Infrastructure as a Service (IaaS) to Software as a Service (SaaS) that are offered on a metered (pay-for-use) basis. These services can be provided to cloud customers in mainly three ways: bare-metal or physical servers, virtual machines and containers. Bare-metal is a cloud solution where physical machines are assigned for any individual application and hence a single operating system is used to serve end-users. A fully dedicated hardware is configured and delivered on demand of users without any virtualization (Kominos, Seyvet, & Vandikas, 2017). A virtual machine allows a single machine to be used as multiple isolated systems. Containers enable virtualization by sharing a single kernel among different virtual machines, which serves to virtualize not the hardware level but rather at the sub-system level. Although virtualized cloud services can satisfy most of the applications' requirements, there are some applications which require physical server solutions within a cloud. Both virtual machines and containers can be used on top of bare-metal servers once the basic computing system has been installed and configured. Thus, the first step for setting up a private cloud is to provision bare-metal servers. Provisioning is the process of loading and installing an operating system and initial configuration on a bare-metal server. There are a number of situations in which bare-metal machines are used in clouds ("Introduction to Ironic", 2019 (accessed November 10, 2019)):

- High-performance computing clusters
- Database hosting for databases that may run poorly in a purely virtualized environment
- Those tasks that make direct use of hardware devices
- Applications that need hardware dedicated to application for security, dependability, performance and other regulatory requirements

- Cloud infrastructure that needs rapid deployment

Bare-metal servers are needed when ”consumers need to run high-performance algorithms for short periods of time” (Venkateswaran & Sarkar, 2019, p. 1). These servers increase power efficiency as the machines can be switched on and loaded only when needed. Additionally if any system is compromised, the bare-metal server can be reloaded to restore secure services. Hence their use is efficient for cybersecurity. However, it is difficult to predict the provisioning time for bare metal. Also, pre-provisioning would lead to idle resources running for peak demand, which will eliminate the fundamental advantage of cloud adoption - to provision on-demand. The bare metal provisioning-time can vary from minutes to hours, depending on technologies used by the cloud provider. The focus of this study is to investigate the remote installation of operating systems and initial configuration on bare metal systems as well as virtual machine platforms to provide resources for a cloud computing system.

1.1 Motivation

The motivation for this study lies in the possibility of reducing human effort and time for the loading process of bare-metal in remote locations. The focus is on automating the setting up of initial architecture for a private cloud. Cloud providers spend significant time and effort to setup the basic infrastructure for cloud environment. The aim of this study is to minimize this effort and boot any bare metal remotely using a centralized server.

1.2 Problem Statement

Provisioning bare-metal is a tedious and time consuming job as it requires many steps which includes OS installation, configuration management, performance adjustment, application installation and many other small tasks (Shiau, Sun, Tsai, Juang, Huang, 2018). Due to distributed computing nodes in cloud computing, there is a need for remote loading because a centralized server makes administration easier and reduces energy consumption (Wu, Ko, Huang, & Huang, 2013). Multi-tenancy in cloud computing demands isolation of network traffic for security. This isolation is needed at Layer 2 (data link layer) as Layer 3 (network layer) communication cannot be used for booting a system remotely —i.e., outside an Ethernet broadcast domain because the client does not have its IP address before booting. Many protocols used in booting operate at data link layer. Moreover for other services in cloud, Layer 3 cannot be used as multiple tenants can have same set of IP addresses within their network. The problem addressed by this study is whether bare metal can be booted and hence operating system can be loaded remotely and securely using Layer 2 services for cloud computing? The focus will be on security of Layer 2 communications to ensure that appropriate operating system and applications are installed.

There are several bare-metal provisioning frameworks including Emulab (Emulab, 2019), Ironic (Introduction to Ironic, 2019), Maas (MAAS, 2019), Cobbler (Cobbler, n.d.) and Razor (Puppet Razor,n.d.) which seek to help providers in setting up cloud infrastructure. But all these frameworks require significant skills and time. IT administrators need to follow long tutorials and run various commands to get the framework working. For instance, the installation documentation of Ironic is around 15 pages long (Chandrasekar Gibson,2014). There is no easily implementable standard method of booting across different Ethernet broadcast domains.

1.3 Significance

Bare-metal provisioning is the first step in setting up of any cloud infrastructure. Virtual machines and containers can be built on top of bare-metal hardware. Configuration and installation of resources is a cumbersome and tedious process. This study is significant as “there is not much guidance on building, operating, trouble-shooting, and managing a secure and scalable private cloud infrastructure, especially for public agencies” (Babar & Ramsey, 2015). The focus is on bare-metal provisioning, as existing bare-metal solutions for cloud can cause delays of tens of minutes (Turk et al., 2016). Adding, removing or upgrading servers all require some kind of hand editing which is susceptible to errors, (John et al., 2018) so automation of this process can reduce the potential errors to a great extent.

1.4 Purpose

The purpose of this study is to allow booting and loading of an operating system on virtual machines or bare-metal for a private cloud remotely. There exist various frameworks and software which can be utilized to automate this process. The aim is to build a framework using which the system administrator needs to just run a few commands to get the whole system up and running. By the end of this research project, a framework will be developed to load operating system and initial configuration on bare-metal remotely.

1.5 Hypothesis

This study aims to simplify the process of booting and loading operating system on bare-metal. To establish a private cloud infrastructure, there is a need for secure and private communication channel between server and client. Layer 2 is physically controllable as it is closely linked to Layer 1 (physical layer). This opens up Layer 2 to variety of potential attacks including unauthorized access to the network. The hypothesis for the study is that it is possible to load operating system and initial configuration remotely on bare-metal (or virtual machines) for private cloud in a secure way.

1.6 Delimitation

The study focuses on loading operating systems on bare-metal and virtual machines. It does not cover containers. This study is for the initial setup of cloud infrastructure.

1.7 Definitions

The ongoing research is focused on provisioning bare-metal servers for clouds. A number of tools and technologies are being used for provisioning. Definitions of all of these major terms/concepts central to the proposed study are discussed below:

- Bare-metal: These are devices which have no operating systems installed. “No virtualization, the hardware is fully dedicated, delivered on demand” (Kominos et al., 2017, p. 1).

- Containers: “A light-weight approach to isolating resources, where applications share a common kernel”(Kominos et al., 2017, p. 1).

- Virtualization: It refers to the abstraction of physical hardware which forms large clusters of logical units for cloud. This can be offered to users in the form of Virtual Machines (Kominos et al., 2017).

- VirtualBox: VirtualBox is an emulation of a computer system. Multiple operating systems can be run on a host using VirtualBox. For example, you can run windows or Linux inside your mac (“Virtual Box Documentation”, 2019 (accessed November 25, 2019)).

- Virtual Machine: “Traditional virtualization where the machine appears as a self-contained computer, boots a standard OS kernel, and runs an unmodified application process on top of a Hypervisor layer” (Kominos et al., 2017, p. 1).

1.8 Summary

The resources can be provided to cloud customers using virtualization or physical servers. Hence, there is a need for bare-metal hardware on which to build a cloud computing system. The provisioning of bare-metal can be tedious and time consuming. Moreover, the security of communication channels is vitally important for securely loading hardware. This study focuses on the provisioning of bare-metal for private cloud in a secure manner.

CHAPTER 2. REVIEW OF LITERATURE

This chapter provides a review of the literature relevant to the problem of bootstrapping bare-metal hardware for private clouds and briefly discusses the basic concepts needed to understand the thesis.

2.1 Methodology of Review

The problem addressed by this study is whether bare metal can be bootstrapped remotely using layer 2 services. Setting up a bare metal server is a cumbersome process and usually requires human effort. The aim of this study is to minimize the human effort and time spent in this process and hence automate the process of initial setup. Some of the terms related to this study are discussed in subsections given below.

2.1.1 Cloud Computing

Cloud computing provides a common pool of configurable computing resources with easy on-demand network access. These resources can comprise servers, applications, storage, and various other services. These resources can be rapidly provisioned and released when not required with minimal effort (Mell, Grance, et al., 2011).

Clouds can be categorized into four types on the basis of deployment: Private cloud, Public cloud, Hybrid cloud and Community cloud. In public clouds, infrastructure is operated on-premises of the cloud provider and can be used by external customers of multiple organizations. A private cloud is owned and operated as a resource to which access is restricted to authorized organizations. In case of community clouds, organizations which have common interests share the same infrastructure. A hybrid cloud is a combination of two or more unique cloud infrastructures instances.

In this study the focus will be on setting up the infrastructure for a private cloud, for which secure and private communication channels are needed for remotely booting and loading operating systems on bare-metal servers or newly created virtual machines.

2.1.2 Cloud Provision

The provisioning of cloud systems requires the composition of technologies that when put together produces an operational cloud infrastructure. Bare-metal provisioning refers to deployment of Operating System and applications on physical machines, which includes many sub-tasks including OS deployment, installing applications, configurations, preparation and restoration of images, and performance adjustments (Shiau, Sun, Tsai, Juang, & Huang, 2018). In this study, the focus is on building a secure and private cloud infrastructure, and hence using tools that allows the installation of operating systems as well as basic configuration.

The IT administrators need to ensure that systems load appropriate operating system images and system configuration. Instead of using a local hard drive, the use of a server on network to provision another server is *Network Booting*. Cloud computing requires the quick booting of machines depending on the demand (Vemula, 2016). Some of the common terms which are important to understand the process of network booting and loading operating systems are defined below:

- Dynamic Host Configuration Protocol (DHCP): Network protocol that provides IP addresses and configuration information to devices.
- Domain Name System (DNS): Allows humans to use domain names instead of using Internet Protocol (IP) address to address systems.
- Preboot Execution Environment (PXE): Process of booting a device remotely using its network card. PXE uses a DHCP server to store and serve required information such as system IP address and DNS server address.
- Intelligent Power Management Interface (IPMI): A set of computer interface specifications for hardware-based platform management systems used to assist the controlling and monitoring of servers.
- Trivial File Transfer Protocol (TFTP): A protocol used by servers for transferring files to clients usually in the early stages of booting.

2.2 Background

Various solutions have been proposed to increase the ease of provision and management of bare-metal systems. Some of the related topics are discussed in this section.

2.2.1 Network Booting

The typical process for bare-metal provision is discussed in detail. A typical bare metal provisioning process involves the steps mentioned in figure 2.1 (Chandrasekar & Gibson, 2014):

- First, registration of available hardware that is to be loaded with bare metal provisioning system. When any request is received by a provisioner for node allocation, it picks a suitable node from its store and uses a power driver like IPMI (Intelligent Power Management Interface) configured on physical machines to power on required number of nodes.
- The nodes start to boot and transmit a DHCP Discovery request on an Ethernet broadcast address to discover a DHCP (Dynamic Host Configuration Protocol) /PXE (Preboot Execution Environment) server running on the provisioner on the same Ethernet broadcast domain.
- Once this is done, a bootstrap image is downloaded on a node to be loaded from a provisioner using TFTP (Trivial File Transfer Protocol), and nodes initiates the download of actual OS image required.
- The guest node restarts and boots using the OS image on the hard disk and sends acknowledgement to the bare metal provisioner.

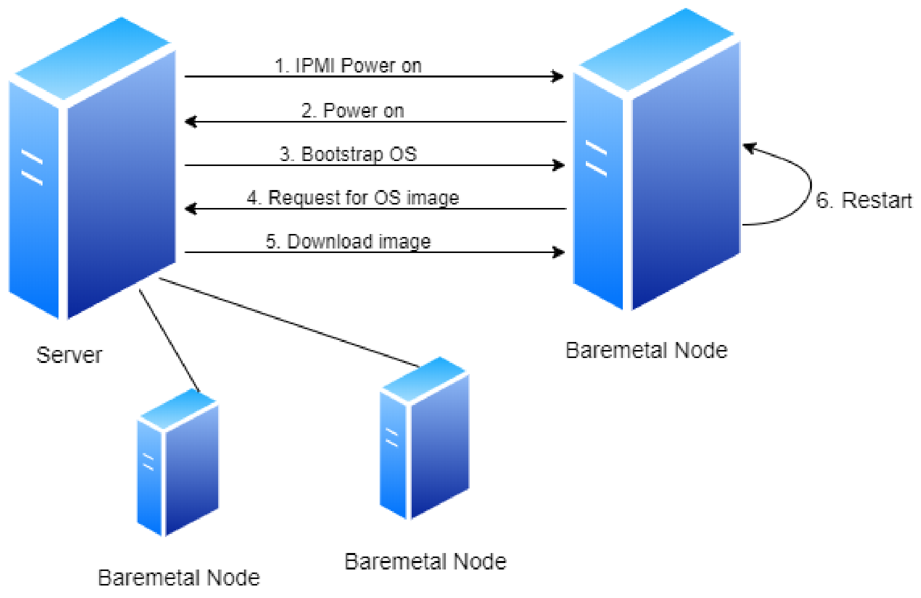


Figure 2.1. Default bare-metal provision

2.2.2 Data link Layer Communication

The data link layer is the second layer in Open Systems Interconnection (OSI) model. All the devices that run layer 2 protocol are referred as *nodes*. This layer provides a communication channel among adjacent nodes in a network. For end-to-end communication between any source and destination, data needs to be moved over these individual links. Every Ethernet or wireless device has a unique Media Access Control address (MAC address) which is a unique identifier for that device associated with the Network Interface Card (NIC) of a device. Communications at layer 2 uses MAC addresses to form and address frames carrying data. For Layer 3 IP addressing, each node has a IP/ MAC mapping address table for all nodes in the same Local Area Network (LAN) which is formed using Address Resolution Protocol (ARP) (Kurose & Ross, 2017).

A logical division of a network at the data link layer in which a broadcast address (i.e. ff:ff:ff:ff:ff:ff) can be used to reach any other node in the network is called a *broadcast domain*. A Virtual LAN (VLAN) can be used to segment one LAN into multiple broadcast domains. This can be used to facilitate the communication among various nodes which are not even physically located together. Though really useful, VLAN links are insufficient for the requirements of cloud providers. Firstly, VLAN links can create up-to 4094 networks which creates limitation in the number of networks that can be used for a cloud computing infrastructure. Cloud providers have multiple customers that are hosted by a data center. These tenants need their own isolated network domain. Layer 3 solutions cannot solve the issue because systems do not have assigned IP addresses before booting. In addition to this, multiple tenants of cloud providers can use same set of IP addresses within the network (Mahalingam et al., 2014). These issues gave rise to Virtual Extensible LAN (VXLAN). VXLAN is a protocol that can be used to create overlay networks of layer 2 broadcast domains over layer 3 network. ”In data centers, VXLAN is the most commonly used protocol to create overlay networks that sit on top of the physical network, enabling the use of a virtual network of switches, routers, firewalls, load balancers and so on. The VXLAN protocol supports the virtualization of the data center network and addresses the needs of multi-tenant data centers by providing the necessary segmentation on a large scale” (What is VXLAN?, 2019). Figure 2.2 shows VXLAN as tunneling protocol for the communication between two ethernet broadcast domains.

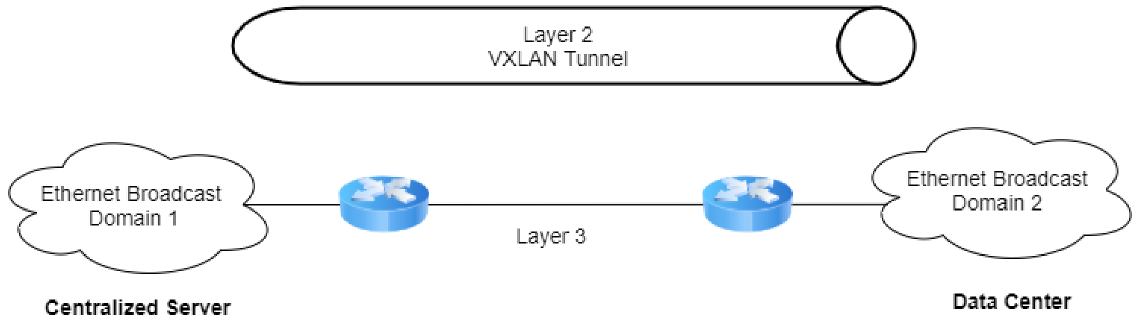


Figure 2.2. VXLAN - Tunneling protocol.

2.3 Related Work

Studies conducted in similar area are discussed briefly in this section. Venkateswaran describes an approach using broker and inventory optimization for ”time-sensitive fulfillment of bare-metal server requests” (Venkateswaran & Sarkar, 2019). Shiau proposed and developed a novel system architecture in an open source manner ”to support single machine backup and recovery, massive deployment of disks in a computer, and massive deployment in large-scale computers” (Shiau et al., 2018, p.1). ”Clonezilla live”, an on-line deployment system which doesn’t require any ”tedious installation or configuration before usage” was used by Shiau.

Bare metal solutions introduced before Turk’s work acknowledged the benefits of network booting over local installation but they rejected it assuming that there will be unacceptable overhead in the process (Turk et al., 2016). Turk used the approach of separate boot and data disks, boot devices to be stored in a centralized repository. This significantly reduced the overhead related to network boot. Hennessey used Hardware as a Service (HaaS) for allocating physical servers in isolation (Hennessey et al., 2014). He also proposed a new Exokernel-like layer called Hardware Isolation Layer (HIL), to allow sharing of resources of a data center among mutually untrusting physically deployed services (Hennessey et al., 2016).

Cloud Computing demands dynamic resource allocations. Yan uses KVM and Open vSwitch to serve the demands of network in Cloud Computing (Pu, Deng, & Nakao, 2011). He performed the migration of virtual machines using Open vSwitch.

2.4 Bare-Metal Frameworks

There are multiple bare-metal provisioning frameworks available in the market today. Chandrasekar gives a comparative study of these frameworks (Chandrasekar and Gibson (2014)). The popular frameworks discussed in the paper are Emulab, Ironic, Crowbar, Maas, Cobbler and Razor.

2.4.1 Emulab

”Emulab is a testbed for networked and distributed systems experimentation” (Burtsev, Radhakrishnan, Hibler, & Lepreau, 2009). For using this bare metal provisioning framework, Emulab needs to be pre-installed in the cluster. Emulab takes a network topology given by user in a Network Simlulator (NS) file and allocates and connects the required number of nodes using configured topology. Emulab has been in use from a long time but the installation process of Emulab is quite long (Chandrasekar & Gibson, 2014). The documentation of Emulab says that it requires significant time, expertise and specific hardware infrastructure for installation (“Installing Emulab”, 2018 (accessed November 30, 2019)).

2.4.2 MaaS

Canonical MaaS (Metal as a Service) provides self-service provisioning of Windows, Ubuntu, CentOS and ESXi, and turns your data center into bare-metal cloud. MaaS manages a pool of nodes through IPMI (or another baseboard management controller, BMC), letting you choose how you want to control power on machines. It can also overwrite the machine’s disk space with your chosen OS images, which are cached for quick deployment. Users can specify requirements for a machine when acquiring a node from API/CLI. Juju which is a service and model management tool, is designed to work with MaaS. Juju manages the services running on the machines managed by MaaS (*MAAS*, 2019).

2.4.3 Ironic

Openstack is a software for controlling large pools of networking, storage and compute resources in a datacenter through APIs with common authentication mechanisms (“Introduction to Ironic”, 2019 (accessed November 10, 2019)). Ironic is an integrated OpenStack project to provision bare-metal (not virtual machines). It is built using Nova bare-metal driver. It manages bare-metal using the common protocols like TFTP and PXE. Ironic integrates with the OpenStack Identity (keystone), Compute (nova), Network (neutron), Image (glance) and Object (swift) services. To get started with Ironic, all these services must be installed beforehand.

2.4.4 Cobbler

Cobbler is a Linux installation server for provisioning bare-metal hardware. It can also help with managing DNS and DHCP, power management, configuration management orchestration, package updates and much more. Cobbler supports Chef (Network automation tool) to automate the deployment of tools. Cobbler is designed to manage a wide variety of technologies and consequently it can be a somewhat complex system to get started with. A quick-start guide is available for Red Hat distributions but missing for other distributions (“Cobbler”, 2019 (accessed November 12, 2019)).

2.4.5 Razor

Razor is a provisioning, power control and management application to deploy both virtual and bare-metal computing resources (“Puppet Razor”, 2019 (accessed November 12, 2019)). In a Razor deployment, newly added machines will PXE-boot from a special Razor Micro-kernel image. These machines will check in, provide Razor with inventory information and wait for further instructions. Razor consults user-created policy rules to choose the task requirement for any new node and this node gives feedback to Razor as it completes various steps. Razor provides plugins for integration with third party configuration management systems such as Puppet.

2.5 Technologies

There are various tools and technologies which were studied to design the proposed framework. Some of these are discussed below:

- Infrastructure as Code: The main idea is to configure and setup our infrastructure using scripts or code instead of doing it manually. So it will minimize the human effort required in the whole process (“Infrastructure as code: What is it? Why is it important”, 2019 (accessed November 20, 2019)).Vagrant can be used for this purpose. Vagrant is a tool which puts all the dependencies and their configuration within a single, consistent and disposable environment. To use vagrant, a user creates a *Vagrantfile* and vagrant will install and configure a virtual machine using ”vagrant up” command (“Introduction to Vagrant”, 2019 (accessed November 25, 2019)).

- Internet Protocol Security (IPSec): IPSec is a suite of protocols that provides security at IP Layer. It is most commonly used to provide Virtual Private Networks (VPN) between two locations. Mostly, Internet Key Exchange (IKE) is used with IPSec to provide keying material (Frankel & Krishnan, 2011). Many IPSec implementations exist today like Libreswan, Strongswan and Openswan. Among these Strongswan has more comprehensive documentation than the others and it supports more features. Strongswan is a multi-platform IPsec solution for authentication and encryption mechanisms between server and clients. It can be used efficiently to secure connections with remote networks. It uss Internet Key Exchange protocols to establish security associations between two machines (“Introduction to StrongSwan”, (accessed November 12, 2019)).

- iPXE: It is an Open Source network boot firmware. It enables computers with PXE support to extend the functionality by adding boot from a web server via HTTP, from an iSCSI SAN and by supporting many other additional features.(“iPXE Open Source boot firmware”, 2019 (accessed November 10, 2019)).

- Jump Host: Jump Host or a Bastion Host is a critical security component in a network. It is fortified host and is exposed to public network (Norberg, 1999).

- Kickstart file: A kickstart file is a text file which can provide all the configuration information a system needs while booting ad installing process (“Kickstart Installation”, 2020 (accessed May 15, 2020)).

- Overlay Network: To pass traffice between different virtual machines in separate networks, VXLAN or GRE protocols can be used. These both have similar tasks to encapsulate the traffic and differentiate between different virttual networks. VXLAN is more recent technology than GRE. “In data centers, VXLAN is the most commonly used protocol to create overlay networks that sit on top of the physical network, enabling the use of a virtual network of switches, routers, firewalls, load balancers and so on. The VXLAN protocol supports the virtualization of the data center network and addresses the needs of multi-tenant data centers by providing the necessary segmentation on a large scale” (“What is VXLAN?”, 2019 (accessed November 15, 2019)).

- Secure Boot: Secure boot is a security standard which uses a keying mechanism to make sure that a device boots using authorized software. The host processor and secure boot device exchange encrypted messages to communicate securely (Davis, 1999).

- Switch: Switches enable communication across a network. Linux bridge can be used as a network switch. It can be used to forward packets on routers, virtual machines and on gateways. But Open vSwitch provides many additional features to a simple Linux bridge. Open vSwitch is a virtual switch for virtual machine environments in networking. It can be used to connect virtual machines within and across different networks by creating a network bridge (“Open vSwitch?”, 2020 (accessed February 2, 2020)). It is open to programmatic control and extensions. Most importantly, it is a distributed switch, several virtual machines running on different hypervisors can be connected using it.

2.6 Summary

This chapter provided a review of the literature for the proposed study. Firstly, cloud and its deployment models are discussed. A typical bare-metal provisioning process and related work is given. Communication at data link layer is discussed. There are many existing frameworks for provisioning but there is no easily implementable standard method for booting remotely.

The review really helped in constructing the methodology of this study as the different existing frameworks gave a clear view of all components that are needed to build the proposed framework. It helped in planning the overall flow of the process of provision.

CHAPTER 3. METHODOLOGY

The goal of this study is to provide a remote private cloud infrastructure that would allow network administrators to boot and load operating systems on bare-metal machines that are separated from the boot server across Ethernet broadcast domains. This chapter describes the proposed methodology for the study. It details the experimental design and steps to be followed to build the proposed framework.

3.1 Research Method

In this study, a new framework will be developed to bootstrap a bare-metal machine to be used as a part of a cloud infrastructure. The aim of this study is to answer whether a bare-metal provisioning process can be automated, and if communication across Ethernet broadcast domain at Layer 2 can be done remotely and securely. As this is a subjective approach which involves answering these research questions based on the results of analysis from our framework, the study is mainly qualitative. The main aim of the study will be to minimize the human effort required in the whole process of provisioning and to check if Layer 2 communication can be done remotely and securely across Ethernet broadcast domains. Furthermore, the effect of using squid will be analyzed on both the central server and the jump host. This collection and analysis of data makes this research quantitative as well.

3.2 Experimental Design

There is a need for more dynamism in the provision of machines so that we can quickly scale up machines during peak time and scale down otherwise. That is why the plan is to use the concept of Infrastructure as Code (IAC). The idea is to use a script or code which can be run to provision the required machines during peak time. ”The premise is that modern tooling can treat infrastructure as if it were software and data” (Morris, 2016, p.5). Cloud computing is the biggest use-case of IAC because there is a need for rapid installation, updates and management of infrastructure.

The plan is to establish a remote private cloud infrastructure that would allow a remote IT administrator to quickly create a secure endpoint for loading systems using the Vagrant and IAC approach to stand up loading points in a short time, and to direct communications from the loaded infrastructure through the secure encrypted network link to the central server. Vagrant, a tool which puts all the dependencies and their configuration within a single consistent and disposable environment will be used. Once a Vagrantfile is created, after that you just need to run ”vagrant up” command and required operating system and packages are installed and configured for you to use irrespective of the operating system of host machine you are using (“Introduction to Vagrant”, 2019 (accessed November 25, 2019)). The infrastructure components would be built and defined in a vagrant file and hence can be used to create both ends of the network. For setting up a server that can be used to boot and load clients, several packages will be used. These packages are detailed in table 3.1.

Table 3.1.: Packages to be used

Lightpd	
Role	It is a very flexible, secure, fast and compliant web-server which is optimized for high-performance environments (“Lighttpd”, 2019 (accessed November 26, 2019)).
Expected use in the study	Setup central server as a web-server for booting

	remote machines using HTTP.
Dnsmasq	
Role	It is lightweight web-server and appropriate for high-performance environments.
Expected use in the study	It will be used for providing network infrastructure. It will provide DHCP, PXE and DNS services to the server.
Squid	
Role	It will be used for caching and reusing the most frequently-requested web pages at the server and the jump host.
Expected use in the study	Squid will reduce the time required to boot machine as all the frequently used boot files will be cached at server and jump host.
iPXE	
Role	Open source network-boot firmware. Provides additional features over standard PXE.
Expected use in the study	Will be used with patches applied to support HTTP proxy connections to allow the use of Squid as a proxy server
Open vSwitch	
Role	A multilayer and distributed virtual switch to connect virtual machines within and across different networks.
Expected use in the study	For creating a network bridge between VXLAN and ethernet ports.
VXLAN	
Role	It is an overlay encapsulation protocol.

	It allows every LAN segment to extend across Layer 3 networks.
Expected use in the study	VXLAN will be used to create both sides of layer 2 endpoints. VXLAN tunnel will be created over IPSec.
StrongSwan	
Role	Strongswan is a multi-platform IPsec (IP Security) solution for authentication and encryption mechanisms between a server and the clients.
Expected use in the study	It will to used to create end-to-end encrypted tunnel between local jumphost and remote central server (“Introduction to StrongSwan”, (accessed November 12, 2019)).

The experimental design of this study will be comprised of following tasks:

- The server will be setup using Vagrant. This server will be protected by built-in system firewall. The plan is to first install all the required packages on server so that it should be able to boot and load operating systems on local machines. After it is done, remote provisioning will be handled.

- All the commands that need to be run to set up the server will be written in a vagrantfile. In this way, this file can be sent to any system, and with one command, the server can be set up anywhere. Basically, the infrastructure will be set up using the code in the vagrantfile.

- Lighttpd and Dnsmasq will be installed and configured on the server. This allows the web server to use DHCP, PXE and DNS services to boot client.

- The plan is to use iPXE, Open Source network boot firmware, on server by chainloading to obtain the features of iPXE without the hassle of reflashing ("iPXE Open Source boot firmware", 2019 (accessed November 10, 2019)). Patches will be applied to iPXE to support HTTP proxy connections which will allow the use of squid as a proxy server (Chirossel, 2016 (accessed November 15, 2019)). The bare-metal will be loaded with iPXE installed server.

- To reduce the boot and load time for the operating system, squid will be installed on the server. After squid is configured, the log files will be checked to verify that it caches most-frequently used files and hence, http traffic is reduced.

- At this point, the central server can boot and load operating system on virtual machines or bare-metal in the local network. Next step is to move towards remote loading.

- A virtual machine will be setup at remote network that will act as a jump server to remote central server. Bare-metal or virtual machine to be booted will be put in the same network.

- VXLAN interfaces will be created at both ends of the server and the jump host to build a tunnel between them. There should be a Layer 3 connection between these two ends over which VXLAN tunnel will be established.

- Strongswan will be installed and configured at both ends of the connection to ensure that the traffic flows using VXLAN tunnel over IPSec.

- Open vSwitch will be used to create network bridges between internal and external network ports and virtual machines. After this step, the central server and the jump host will be able to boot a virtual machine.

- To reduce the booting time, squid will be installed on the jump host too. So the most-frequently requested web content will be cached on the jump host too, reducing the traffic flow between the server and the jump host.

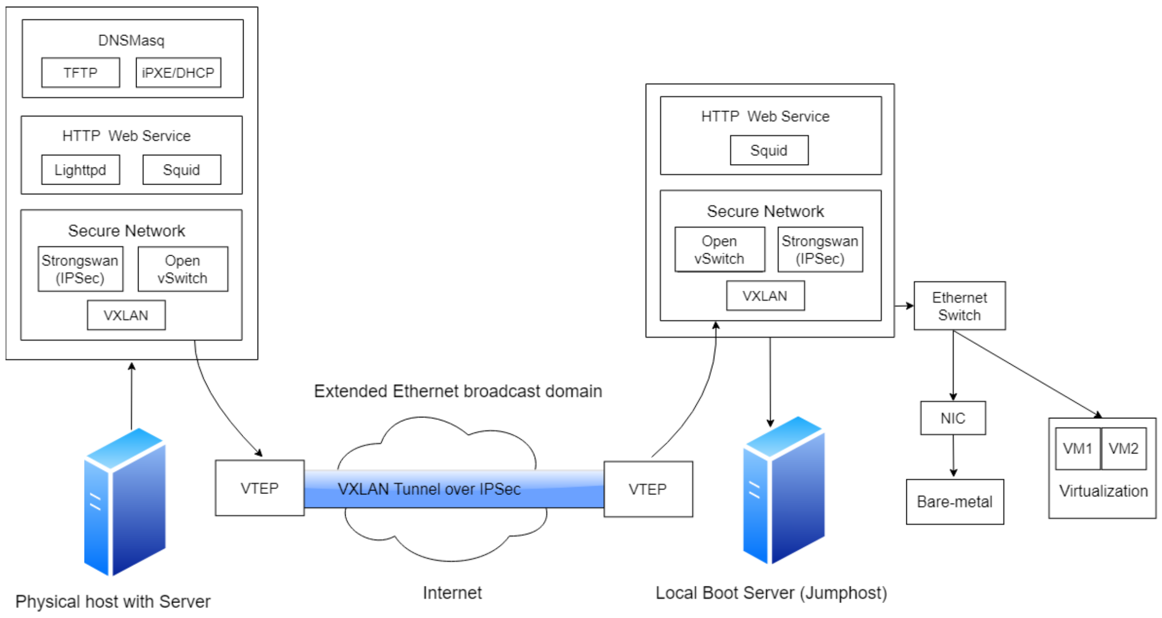


Figure 3.1. Architecture for the study.

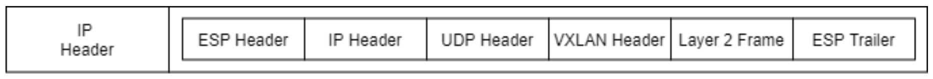


Figure 3.2. Format of encrypted packets flowing through the tunnel.

Above steps would allow a remote administrator to create endpoints in two networks using IAC approach for communications using a secure encrypted network link to the central server.

3.3 Evaluation

This study will be evaluated on the basis of various qualitative values. There will be an answer to the question if communication at Layer 2 can be established remotely. Moreover, time taken by the provisioning process will be evaluated by imposing network delays using Network Emulator ("NetEm - Network Emulator", 2020 (accessed May 25, 2020)). Time taken by the proposed framework will be compared with and without the use of Squid on the jump host.

3.4 Scope

The study applies to all the laboratories or companies that aim to provide private cloud to its customers. For building any cloud infrastructure, there is a need of servers which can be provided in the form of virtualization, physical machines or containers. For getting any one of the three mentioned options, there is a need of basic computing system. This study aims to provide this basic computing system.

3.5 Summary

This chapter gives a brief introduction of the problem and proposal. It describes the research method for the study and a thorough experimental design which will be followed to build the framework. Moreover, scope, data collection and timeline for the study is given. The results for the study will be given in the next chapter.

CHAPTER 4. IMPLEMENTATION

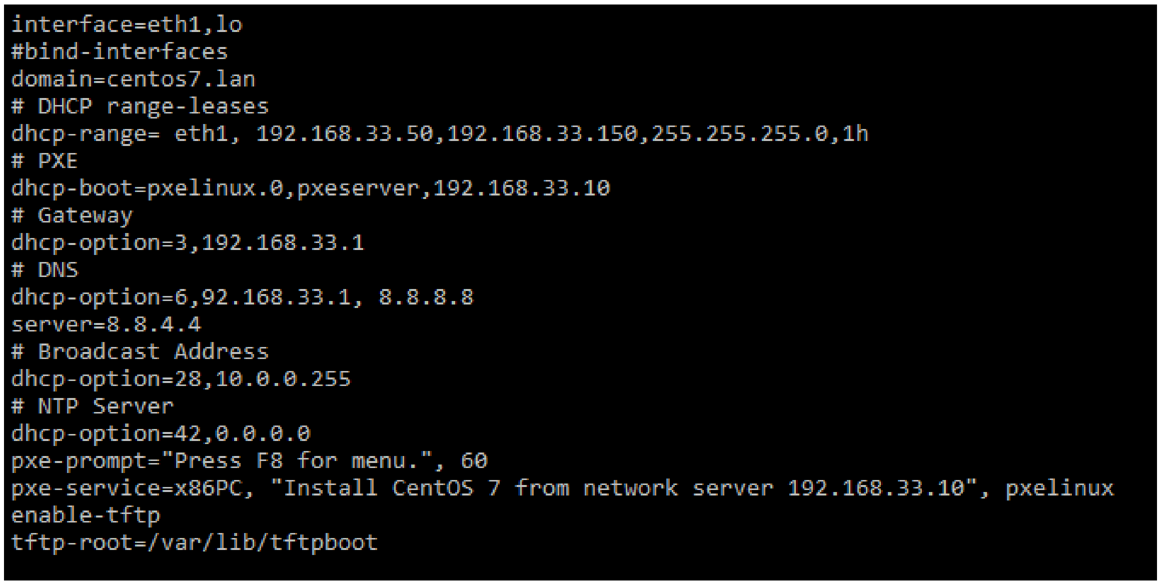
This chapter gives a detailed implementation of all the steps mentioned in the experimental design. As discussed in the last chapter, booting is done initially in a local network and then steps are taken to boot a virtual machine in a remote network. To use the IAC approach, Virtual Box and Vagrant were installed on a system where the central server needs to be setup. CentOS 7 was installed on a virtual machine with IP 192.168.33.10 using Vagrant. Command-line of the host system was used to access this virtual machine. The central server will be protected by built-in system firewall.

4.1 Dnsmasq

To install and configure a PXE server, Dnsmasq was used which provided DNS, TFTP server, VSFTPD server, DHCP services, and Syslinux package. Bootloaders for network booting were provided by the Syslinux package. After editing and saving the dnsmasq configuration file (dnsmasq.conf) according to network requirements, Syslinux PXE bootloader and TFTP server was installed. All the bootloader files were copied to the TFTP server path specified in the dnsmasq.conf file. The PXE server reads its configuration from the files saved in pxelinux.cfg which must be present in TFTP server path. The default file was added to pxelinux.cfg directory which provided the menu for the bare-metal, while loading OS.

To add CentOS kernel and initrd files to the server, CentOS ISO image was downloaded and mounted. The CentOS 7 bootable kernel and initrd images were made available to TFTP server path from the DVD mounted location. To start booting in the local network, FTP was chosen because it is really easy to set up with VSFTPD server. All the DVD mounted content was copied to VSFTPD default server path (/var/ftp/pub). Dnsmasq and VSFTPD were started with required firewall rules.

On the client side, the system was configured to boot from the network. The default menu specified in pxelinux.cfg directory appeared on screen and the system loaded OS from the PXE server.



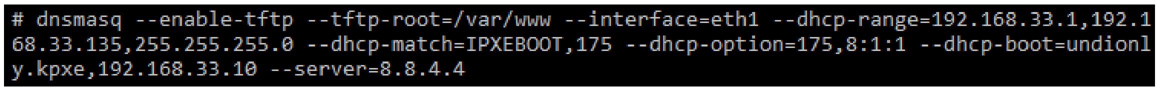
```
interface=eth1,lo
#bind-interfaces
domain=centos7.lan
# DHCP range-leases
dhcp-range= eth1, 192.168.33.50,192.168.33.150,255.255.255.0,1h
# PXE
dhcp-boot=pxelinux.0,pxeserver,192.168.33.10
# Gateway
dhcp-option=3,192.168.33.1
# DNS
dhcp-option=6,92.168.33.1, 8.8.8.8
server=8.8.4.4
# Broadcast Address
dhcp-option=28,10.0.0.255
# NTP Server
dhcp-option=42,0.0.0.0
pxe-prompt="Press F8 for menu.", 60
pxe-service=x86PC, "Install CentOS 7 from network server 192.168.33.10", pxelinux
enable-tftp
tftp-root=/var/lib/tftpboot
```

Figure 4.1. Dnsmasq configuration using FTP Server.

4.2 HTTP Web Service

Minimal installation of CentOS using TFTP was done, but TFTP is slower than HTTP. The next task was to use the HTTP for downloading all the required packages from the internet for installation. Lighttpd was installed as a web server for HTTP requests. For network boot, iPXE was chosen as it provides the feature of booting a system from a web server using HTTP and many other additional features to basic PXE. iPXE could be used by replacing PXE ROM of NIC or by chainloading. Chainloading was selected as it allows iPXE implementation without the hassle of reflashing. Implementation of iPXE using chainloading involved the following steps:

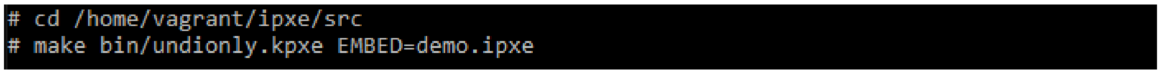
- Undionly.kpxe file was downloaded and placed in TFTP server directory.
- Dnsmasq was configured to hand out undionly.kpxe as the boot file to all the clients using the command given in figure 4.1.
- After getting undionly.kpxe, the chainloaded iPXE sends a new DHCP request to the server.



```
# dnsmasq --enable-tftp --tftp-root=/var/www --interface=eth1 --dhcp-range=192.168.33.1,192.1
68.33.135,255.255.255.0 --dhcp-match=IPXEBOOT,175 --dhcp-option=175,8:1:1 --dhcp-boot=undionl
y.kpxe,192.168.33.10 --server=8.8.4.4
```

Figure 4.2. Configure Dnsmasq to use iPXE.

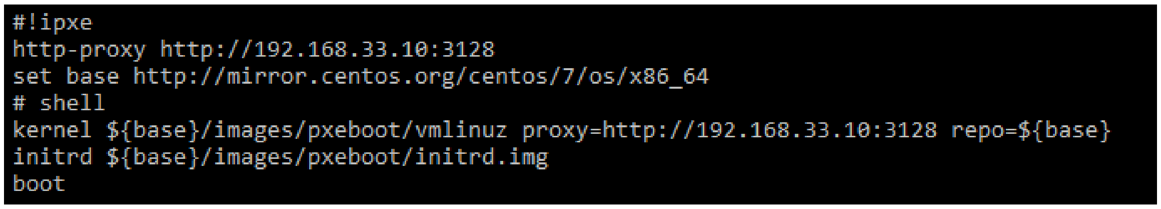
- At this point the server hands out the actual file, menu.ipxe, using which the client will boot. The information of this file was embedded while building undionly.kpxe using demo.ipxe file.



```
# cd /home/vagrant/ipxe/src
# make bin/undionly.kpxe EMBED=demo.ipxe
```

Figure 4.3. Build undionly.kpxe with Embedded demo.ipxe.

- Once the client receives menu.ipxe, it loads OS and boots using the information in menu.ipxe.



```
#!ipxe
http-proxy http://192.168.33.10:3128
set base http://mirror.centos.org/centos/7/os/x86_64
# shell
kernel ${base}/images/pxeboot/vmlinuz proxy=http://192.168.33.10:3128 repo=${base}
initrd ${base}/images/pxeboot/initrd.img
boot
```

Figure 4.4. Contents of menu.ipxe.

4.3 HTTP Proxy

Squid was installed and configured to be used as an HTTP proxy server to cache images for the remote loading of the operating system. Patches were applied to iPXE to support HTTP proxy connections which allowed the use of squid as a proxy server (Chirossel, 2016 (accessed November 15, 2019)). By default squid listens to HTTP requests on port 3128. In addition to it, ICP port number 3130 was added as it will be needed to make connections with squid on jump host for remote connection.

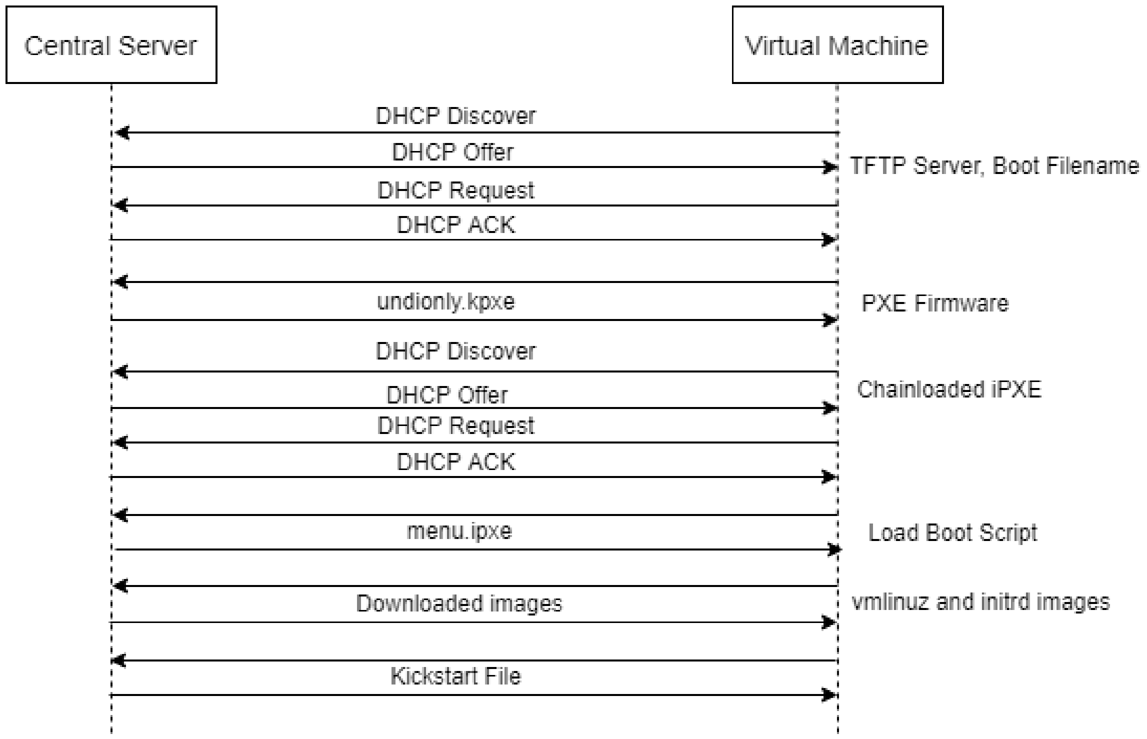


Figure 4.5. Ladder Diagram for Local Boot.

The client was configured to boot from the network. After proper configuration of squid, logs for the connection could be seen in /var/log/squid/access.log file. Squid made the communication faster between the server and the client. At this point, the server could boot a virtual machine in the local network.

4.4 Booting in Remote Network

To get started with remote booting, a system was set up which worked as a jump host with IP 192.168.43.20 for the remote network. The bare-metal or virtual machine to be loaded was set up behind this jump host. The server was on a network with subnet 192.168.33.0/24 and the jump host was set on a network with subnet 192.168.43.0/24, both the same VirtualBox. The first requirement was that there should be a Layer 3 connection between the jump host and the central server. So these two machines were put on a shared network behind 192.168.40.0/24. With the addition of routes to each other, the server and the client were able to ping each other from 192.168.33.10 to 192.168.43.20 and vice-versa.

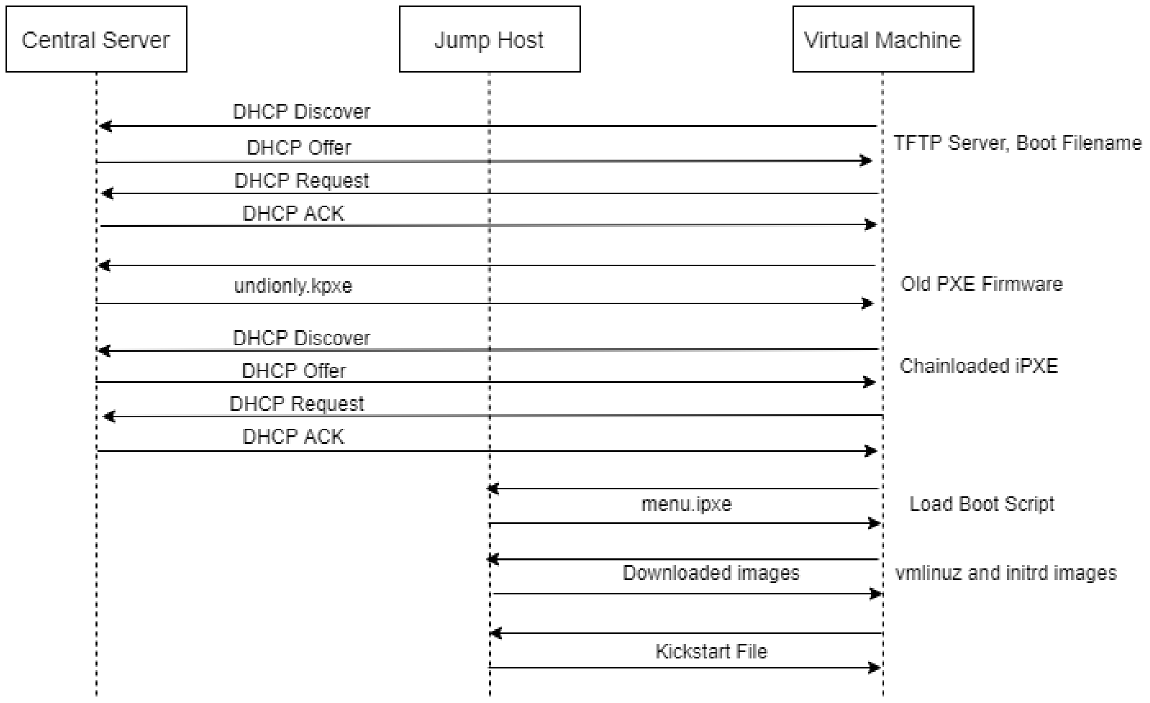
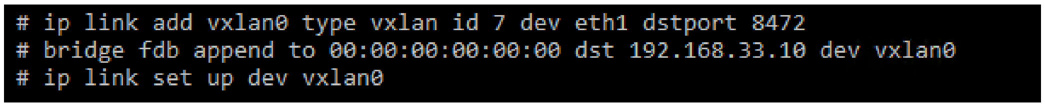


Figure 4.6. Ladder Diagram for Remote Boot.

4.4.1 VXLAN Tunnel

To carry Ethernet traffic over the existing Layer 3 IP network, VXLAN was used. VXLAN interface was built to create Layer 2 endpoints on both sides. An interface named vxlan0 was created and connections were added for jump host. The vxlan0 interface was turned on both on the server and the jump host.

To verify that the traffic is going through VXLAN tunnel, a tool named Etherate was used. Etherate was used with tcpdump to verify that the Layer 2 traffic is going through the established tunnel.



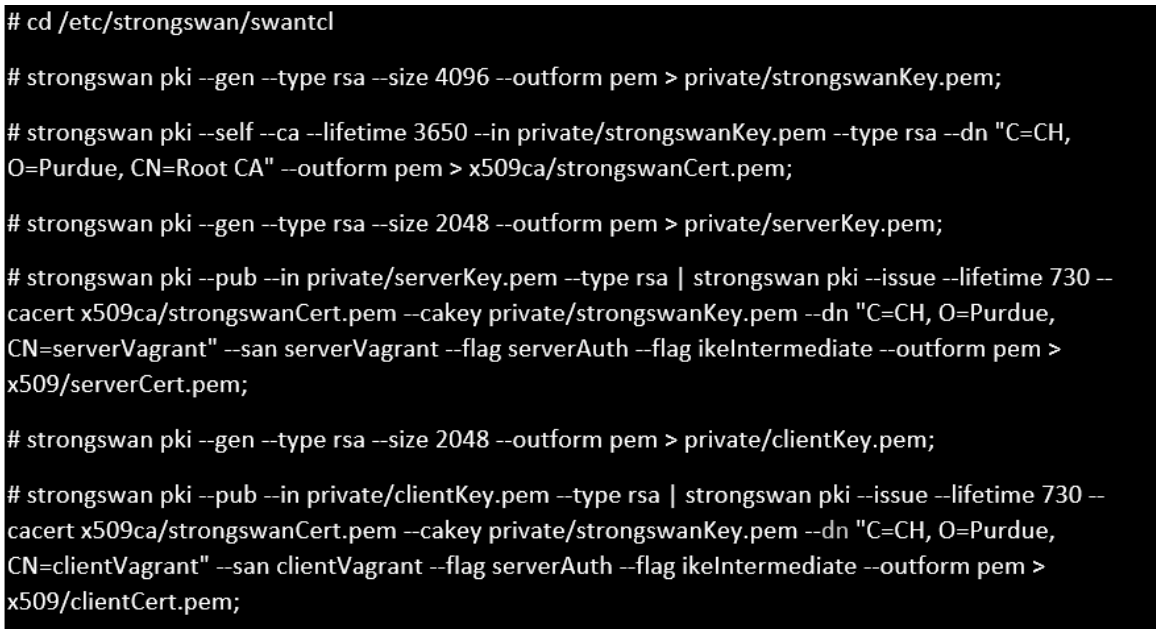
```
# ip link add vxlan0 type vxlan id 7 dev eth1 dstport 8472
# bridge fdb append to 00:00:00:00:00:00 dst 192.168.33.10 dev vxlan0
# ip link set up dev vxlan0
```

Figure 4.7. VXLAN set up on the server.

4.4.2 VXLAN over IPSec

After setting up the VXLAN tunnel, the next step was to make the communication secure between the server and the jump host. An end-to-end encrypted tunnel was created using Strongswan between the jump host and the remote central server. After installing Strongswan on both ends, a local CA certificate was generated at the server which was used to issue end-entity certificates for the communication. X.509 standard was used to generate the certificates. X.509 is a public key infrastructure (PKI) standard that links the cryptographic keys to its issuers which can be an individual or an organization (“X.509 Certificate”, 2020 (accessed May 25, 2020)). So it can be used for establishing encrypted and authenticated links between networked computers. SSL (Secure Sockets Layer) and its successor, TLS (Transport Layer Security) can be implemented using X.509 certificates. The Strongswan pki tool was used to generate all keys and certificates. Private keys and certificates were generated for both the server and the jump host. Swanctl, a command-line utility was used to control, configure and monitor the connection using Strongswan. These keys and certificates were installed in their respective swanctl directories on both ends.

In this scenario, there are two gateways on 192.168.40.0/24 connecting two subnets of the server and the jump host. The swanctl.conf file was configured accordingly. This created a tunnel between the two ends. To verify that encrypted frames were travelling between the server and the jump host, Etherate was used again with tcpdump. Encrypted frames could be seen in the traffic flowing between these two ends. Figure 4.6 shows the tcpdump output. This creates an end-to-end encrypted tunnel using Strongswan over VXLAN.

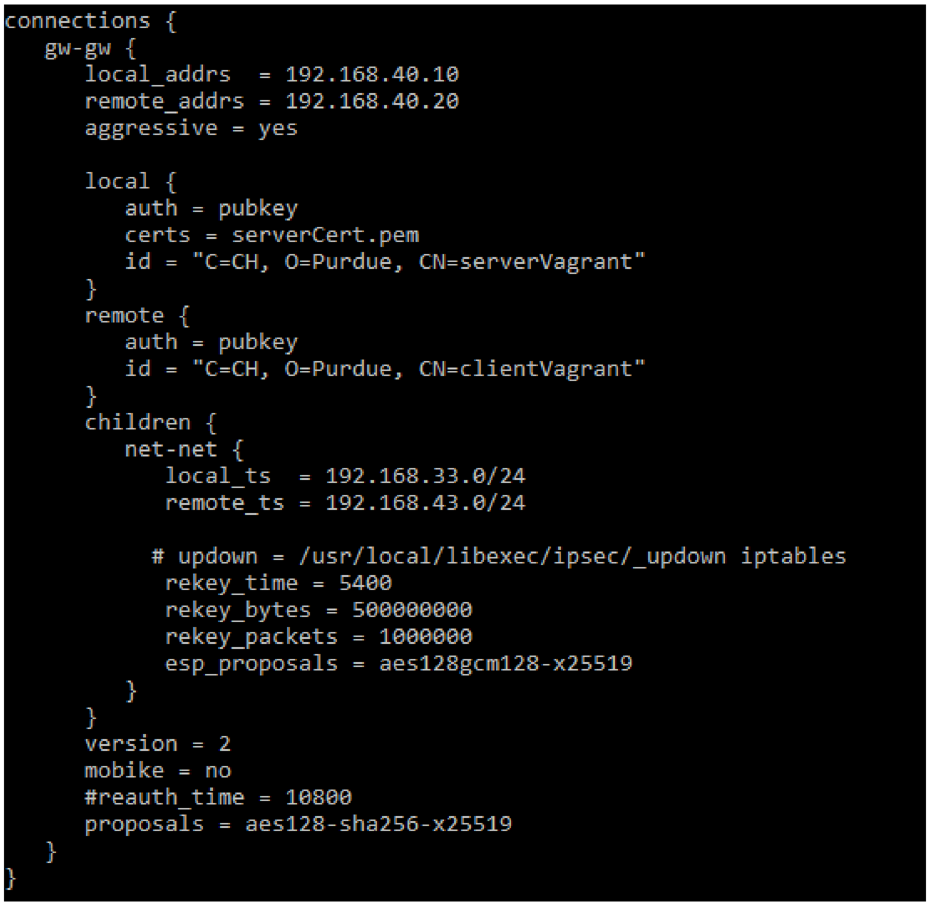


```
# cd /etc/strongswan/swantcl

# strongswan pki --gen --type rsa --size 4096 --outform pem > private/strongswanKey.pem;

# strongswan pki --self --ca --lifetime 3650 --in private/strongswanKey.pem --type rsa --dn "C=CH,
O=Purdue, CN=Root CA" --outform pem > x509ca/strongswanCert.pem;

# strongswan pki --gen --type rsa --size 2048 --outform pem > private/serverKey.pem;

# strongswan pki --pub --in private/serverKey.pem --type rsa | strongswan pki --issue --lifetime 730 --
cacert x509ca/strongswanCert.pem --cakey private/strongswanKey.pem --dn "C=CH, O=Purdue,
CN=serverVagrant" --san serverVagrant --flag serverAuth --flag ikeIntermediate --outform pem >
x509/serverCert.pem;

# strongswan pki --gen --type rsa --size 2048 --outform pem > private/clientKey.pem;

# strongswan pki --pub --in private/clientKey.pem --type rsa | strongswan pki --issue --lifetime 730 --
cacert x509ca/strongswanCert.pem --cakey private/strongswanKey.pem --dn "C=CH, O=Purdue,
CN=clientVagrant" --san clientVagrant --flag serverAuth --flag ikeIntermediate --outform pem >
x509/clientCert.pem;
```

Figure 4.8. Creation of keys and certificates for Strongswan.

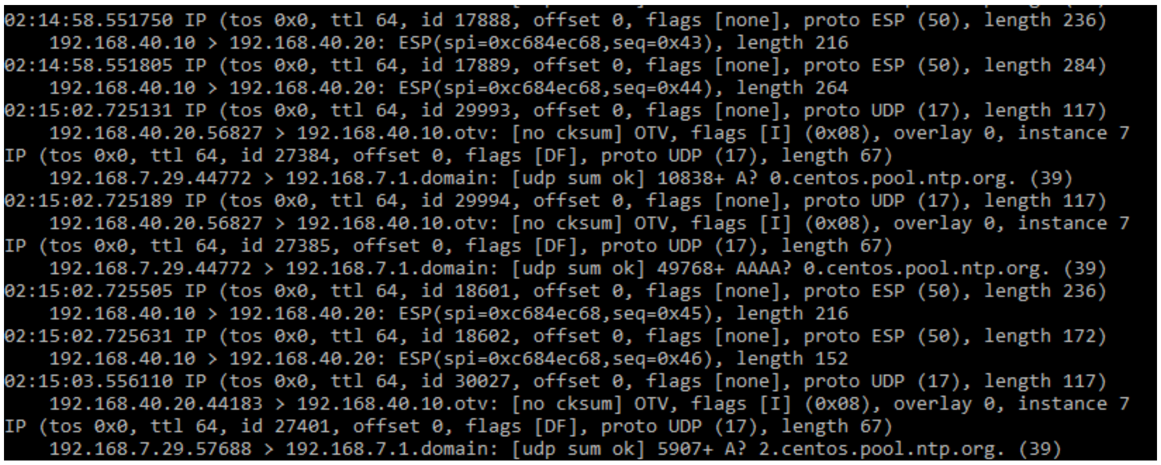
4.4.3 Open vSwitch

The next step was to create a network bridge between the two ports connecting VXLAN endpoints. In addition to it, there was a need of connecting bare-metal or virtual machine, which needs to be loaded, to the jump host. Open vSwitch was used for this purpose. Open vSwitch named ’ovsBridge’ was added on both ends with eth1 and vxlan0 interfaces. The server was given an IP of 192.168.7.1 and the jump host was given an IP of 192.168.7.2. The machines could now ping each other using ovsBridge. On the client side, one virtual machine was added on the network (192.168.43.0/24) of jump host which was configured to boot from the network. Broadcast traffic could be seen flowing between the central server and the jump host. To allow the flow of traffic from the virtual machine to the central server, all adaptors on the jump host were put into promiscuous mode. Moreover, dnsmasq was configured to use ovsBridge interface for the communication. The virtual machine could boot and load operating system at this point.



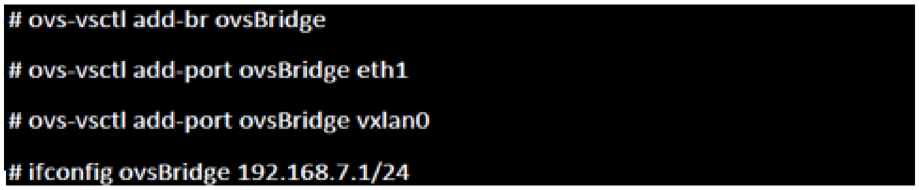
```
connections {
   gw-gw {
      local_addrs  = 192.168.40.10
      remote_addrs = 192.168.40.20
      aggressive = yes

      local {
         auth = pubkey
         certs = serverCert.pem
         id = "C=CH, O=Purdue, CN=serverVagrant"
      }
      remote {
         auth = pubkey
         id = "C=CH, O=Purdue, CN=clientVagrant"
      }
      children {
         net-net {
            local_ts  = 192.168.33.0/24
            remote_ts = 192.168.43.0/24

           # updown = /usr/local/libexec/ipsec/_updown iptables
            rekey_time = 5400
            rekey_bytes = 500000000
            rekey_packets = 1000000
            esp_proposals = aes128gcm128-x25519
         }
      }
      version = 2
      mobike = no
      #reauth_time = 10800
      proposals = aes128-sha256-x25519
   }
}
```

Figure 4.9. Swanctl.conf.



```
02:14:58.551750 IP (tos 0x0, ttl 64, id 17888, offset 0, flags [none], proto ESP (50), length 236)
    192.168.40.10 > 192.168.40.20: ESP(spi=0xc684ec68,seq=0x43), length 216
02:14:58.551805 IP (tos 0x0, ttl 64, id 17889, offset 0, flags [none], proto ESP (50), length 284)
    192.168.40.10 > 192.168.40.20: ESP(spi=0xc684ec68,seq=0x44), length 264
02:15:02.725131 IP (tos 0x0, ttl 64, id 29993, offset 0, flags [none], proto UDP (17), length 117)
    192.168.40.20.56827 > 192.168.40.10.otv: [no cksum] OTV, flags [I] (0x08), overlay 0, instance 7
IP (tos 0x0, ttl 64, id 27384, offset 0, flags [DF], proto UDP (17), length 67)
    192.168.7.29.44772 > 192.168.7.1.domain: [udp sum ok] 10838+ A? 0.centos.pool.ntp.org. (39)
02:15:02.725189 IP (tos 0x0, ttl 64, id 29994, offset 0, flags [none], proto UDP (17), length 117)
    192.168.40.20.56827 > 192.168.40.10.otv: [no cksum] OTV, flags [I] (0x08), overlay 0, instance 7
IP (tos 0x0, ttl 64, id 27385, offset 0, flags [DF], proto UDP (17), length 67)
    192.168.7.29.44772 > 192.168.7.1.domain: [udp sum ok] 49768+ AAAA? 0.centos.pool.ntp.org. (39)
02:15:02.725505 IP (tos 0x0, ttl 64, id 18601, offset 0, flags [none], proto ESP (50), length 236)
    192.168.40.10 > 192.168.40.20: ESP(spi=0xc684ec68,seq=0x45), length 216
02:15:02.725631 IP (tos 0x0, ttl 64, id 18602, offset 0, flags [none], proto ESP (50), length 172)
    192.168.40.10 > 192.168.40.20: ESP(spi=0xc684ec68,seq=0x46), length 152
02:15:03.556110 IP (tos 0x0, ttl 64, id 30027, offset 0, flags [none], proto UDP (17), length 117)
    192.168.40.20.44183 > 192.168.40.10.otv: [no cksum] OTV, flags [I] (0x08), overlay 0, instance 7
IP (tos 0x0, ttl 64, id 27401, offset 0, flags [DF], proto UDP (17), length 67)
    192.168.7.29.57688 > 192.168.7.1.domain: [udp sum ok] 5907+ A? 2.centos.pool.ntp.org. (39)
```

Figure 4.10. Encrypted frames in Etherate.

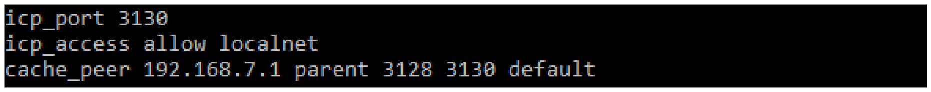


```
# ovs-vsctl add-br ovsBridge

# ovs-vsctl add-port ovsBridge eth1

# ovs-vsctl add-port ovsBridge vxlan0

# ifconfig ovsBridge 192.168.7.1/24
```

Figure 4.11. Commands to add Open vSwitch to the server.

4.4.4 Squid on jump host

To make the process of loading an operating system faster, Squid was installed on the jump host to serve as an HTTP cache. Until now caching was only done at the central server. Caching the most frequently used content on the jump host can make the communication even faster. Squid was installed on the jump host which was linked to the central server using parent links. This formed a cache hierarchy using squid. ICP port 3130 needed to be opened on both ends.



```
icp_port 3130
icp_access allow localnet
cache_peer 192.168.7.1 parent 3128 3130 default
```

Figure 4.12. Adding central server squid as parent in squid.conf file.

After configuring squid with proper access control lists (ACL), the connection was made with the server from the jump host. The cache.log file can be checked to verify that the connection had been made. Once the virtual machine starts loading the operating system, cache entries could be seen in the access.log file. In case a network administrator wants to invalidate the Squid cache, squidclient program can be used to purge the required object.

4.4.5 Kickstart File

At this point, the whole framework is ready to boot and load operating system in a remote network. To make the process more automated, a kickstart file was handed out to the virtual machine to be booted by the central server. Initially, a manual installation of operating system was done. This created a file named anaconda-ks.cfg in the root directory of the installed system. This file was copied to the central server. The path of this file was specified in the menu.ipxe file. Whenever any system booted and installed operating system using the menu.ipxe file, it uses this kickstart file. So there was no need of any human input and system could boot using the information in the kickstart file.

4.5 Summary

This chapter gave a detailed explanation of all the steps followed to build the proposed framework. A framework was successfully built that could securely boot and load an operating system on a virtual machine (or bare-metal) in a remote network. The use of Vagrant and the kickstart file reduced the human effort required for the whole process. To evaluate the total cost of booting from the remote network, various delays were imposed in the network using NetEm. NetEm is a network emulator that can be used to add delays, packet losses, and many other characteristics to the outgoing packets through a specified interface “NetEm - Network Emulator” (2020 (accessed May 25, 2020)). Using various NetEm delays, the time taken to boot a virtual machine was evaluated. Initially, the time taken to boot a virtual machine with a Squid cache on the jump host was computed. After that, the total time to boot without the Squid cache on the jump host was computed. The results obtained from these experiments are explained in the next chapter.

CHAPTER 5. RESULT AND CONCLUSION

This chapter gives the results obtained from the implementation of the proposed framework. Firstly, the implementation confirms the hypothesis that it is possible to load an operating system and initial configuration remotely on a virtual machine (or bare-metal) for private cloud in a secure way. The framework created was able to boot and load an operating system on a virtual machine, which is in a different network from the central server. To make the whole process easier and more automated, a kickstart file was handed out to the jump host while booting. The system reads this kickstart file while booting and hence no further user input was required. Moreover, the use of Vagrant for establishing both the ends of communication allowed the use of IAC approach.

After building the framework, the next step was to evaluate the effect of using Squid on the jump host. In this study, the central server and the jump host were set up on the same VirtualBox. So to evaluate actual latency in the network, Netem was used to impose some real-time delays (“NetEm - Network Emulator”, 2020 (accessed May 25, 2020)). The framework was then tested for 5 different network latencies and the results were compared. The first was the ideal one, -i.e. 0 ms delay in the network. For the actual delays, average network delays for the past 12 months were taken for the regions given in table 5.1 (“AT&T Global Latency”, 2020 (accessed June 8, 2020)). The average delay across the US, Europe, and Asia Pacific was taken. Other than this, the approximate delay between the west and the east coast of the US was taken (between Cambridge and San Diego).

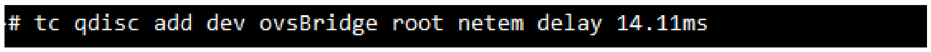

```
# tc qdisc add dev ovsBridge root netem delay 14.11ms
```

Figure 5.1. Command to add a delay of 14.11ms in network.

Table 5.1. Regions and delays.

Regions	Regional Averages (in ms)
Cambridge to San Diego (Intra-US)	67
Across US	31
Across Europe	14.11
Across Asia Pacific	55.47

For all the five latencies mentioned, the time taken to boot and load an operating system on a virtual machine was recorded. Initially, the framework was run with a Squid cache on the jump host. Before imposing a new NetEm delay, the Squid cache was cleared both on the central server and the jump host. For eleven consecutive runs (first run with a cleared Squid cache), the results were plotted and are shown in figure 5.2. It can be seen that the framework takes less time once the requested data had been cached after the first run. The time taken to boot is not affected by the NetEm delays imposed.

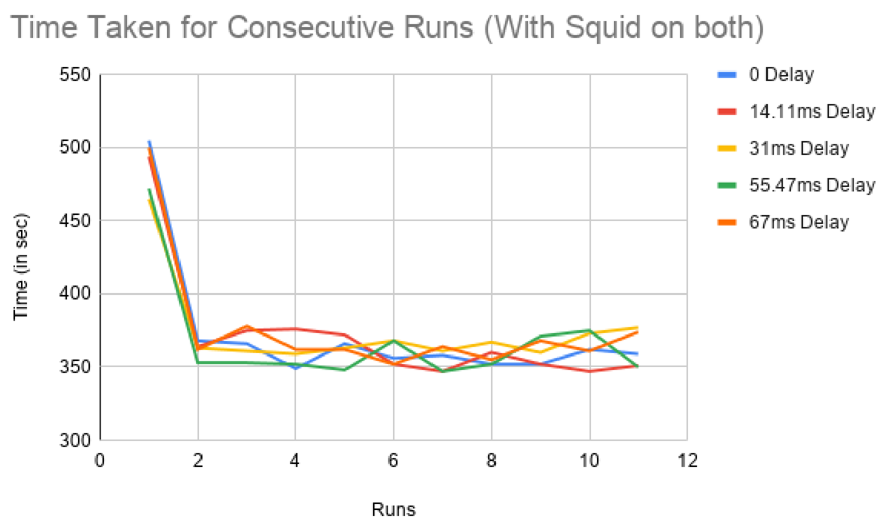


Figure 5.2. Time taken with Squid on jump host for eleven runs.

Table 5.2. Average and Standard Deviation for various Delay with Squid on jump host.

NetEm Delays (ms)	Average Time Taken	Standard Deviation (sec)
0	5 min 58 sec	6.63
14.11	5 min 59 sec	11.48
31	6 min 5 sec	5.97
55.47	5 min 56 sec	10.29
67	6 min 3 sec	7.87

For each network latency, ten consecutive boots were run to measure the average time needed for loading and the standard deviation was calculated to demonstrate the impact of the Squid cache on the jump host. With 0 ms delay, the average time taken was 5 min 54 sec. While with a NetEm delay of 67 ms, the average time taken was 6 min. As can be seen from the table 5.2, there is not a significant difference between the average time taken by the framework for various NetEm delays. The maximum standard deviation recorded is 11.48 sec with a delay of 14.11 ms which is not significant compared to the total time taken by the framework.

After recording the data with Squid on the jump host, the next step was to remove the Squid cache from the jump host. After making the necessary changes on the central server and the jump host, tests were run with the same delays as with the Squid cache on the jump host. Figure 5.3 gives the plot of eleven consecutive runs where the first run represents the time taken by the framework with a cleared cache.

The average and standard deviation of the ten runs excluding the first run are given in table 5.3. As can be seen from the figure 5.4, the average time taken by the framework to boot and load an OS increases linearly with an increase in the network delay which is in accordance with the Mathis equation. The R-squared value (0.9771) supports the linearity in the data collected. The framework takes around 6 min when there is no delay in the network and this value increases to 9 min when there is a delay of 67 ms in the network.

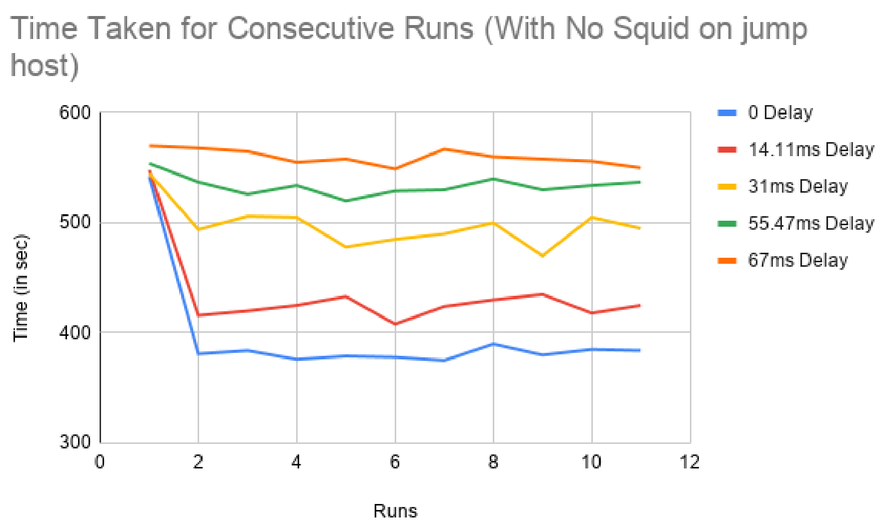


Figure 5.3. Time taken with Squid on jump host for eleven runs.

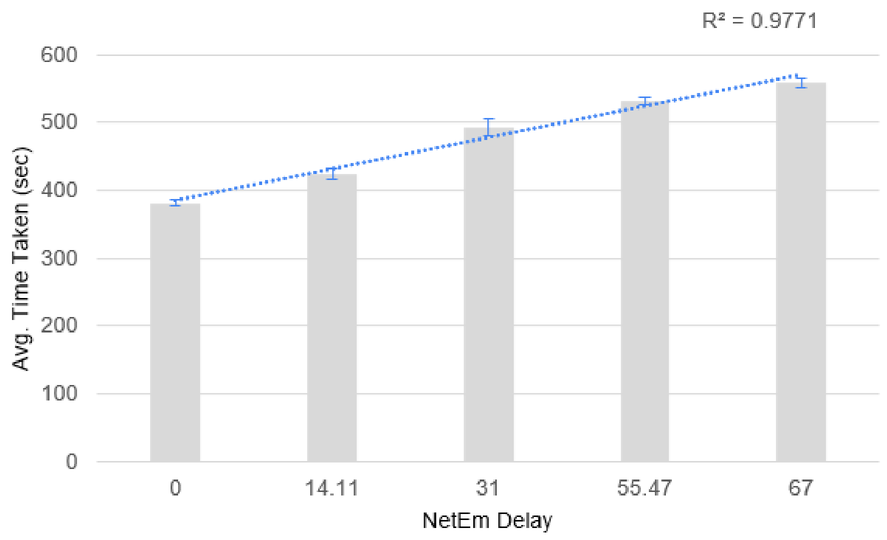


Figure 5.4. Bar Graph for runs without Squid on Jump host.

Figure 5.5 gives the comparison of average time taken by the framework with and without Squid installed on the jump host. The figure also includes an error bar with the standard deviation for all the experimental runs for each delay category. With Squid running on both the server and the jump host, the NetEm delay causes negligible change in the average time taken to boot and load an OS on the virtual machine. When there is no Squid on the jump host, the time taken increases linearly as the NetEm delay increases.

Table 5.3. Average and Standard Deviation for various Delay with Squid on jump host.

NetEm Delays (ms)	Average Time Taken	Standard Deviation (sec)
0	6 min 21 sec	4.59
14.11	7 min 3 sec	8.22
31	8 min 12 sec	12.19
55.47	8 min 51 sec	5.94
67	9 min 18 sec	6.57

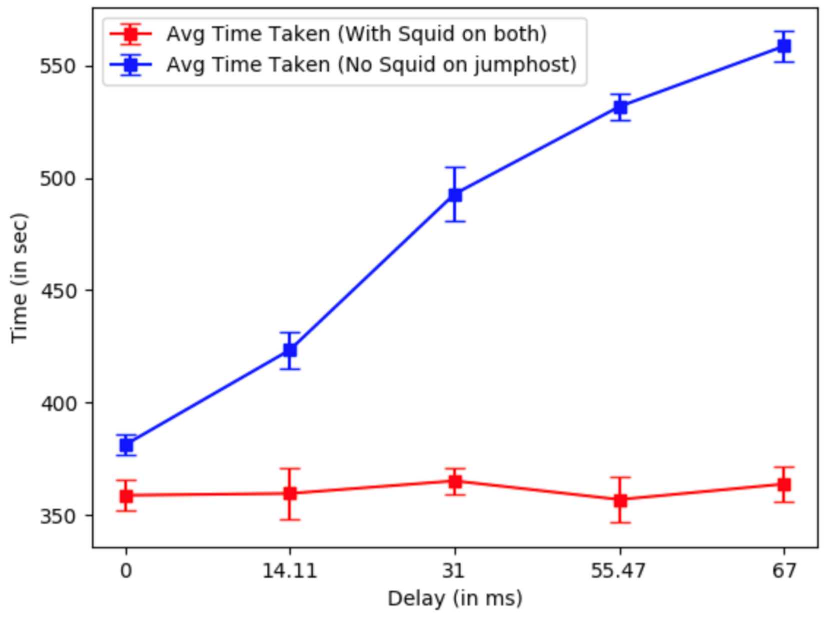


Figure 5.5. Plot of Average Time Taken (with and without Squid on jump host)

5.1 Summary

The results confirmed the hypothesis that secure booting and loading of an operating system is possible from a remote centralized server. An IAC approach was used with Vagrant that significantly reduced the human effort in the whole process for configuring and creating the server and the jump host. In addition to it, various delays were imposed in the network using NetEm to evaluate the effect of using Squid on the jump host. It could be seen from the results that the Squid cache on the jump host significantly reduced the time taken by the framework for the whole process. Thus, Squid on the jump host significantly reduces the cost of having the central server in a remote network.

REFERENCES

AT&T global latency [Computer software manual]. (2020 (accessed June 8, 2020)). Retrieved from `https://ipnetwork.bgtmo.ip.att.net/pws/global_network_avgs.html`

Babar, A., & Ramsey, B. (2015). Tutorial: Building secure and scalable private cloud infrastructure with open stack. In *2015 ieee 19th international enterprise distributed object computing workshop* (pp. 166–166).

Burtsev, A., Radhakrishnan, P., Hibler, M., & Lepreau, J. (2009). Transparent checkpoints of closed distributed systems in emulab. In *Proceedings of the 4th acm european conference on computer systems* (pp. 173–186).

Chandrasekar, A., & Gibson, G. (2014). A comparative study of baremetal provisioning frameworks. *Parallel Data Laboratory, Carnegie Mellon University, Tech. Rep. CMU-PDL-14-109*.

Chirossel, O. (2016 (accessed November 15, 2019)). Http proxy support [Computer software manual]. Retrieved from `https://lists.ipxe.org/pipermail/ipxe-devel/2016-December/005292.html`

Cobbler [Computer software manual]. (2019 (accessed November 12, 2019)). Retrieved from `http://www.cobblerd.org/`

Davis, D. L. (1999, August 10). *Secure boot.* Google Patents. (US Patent 5,937,063)

Frankel, S., & Krishnan, S. (2011). Ip security (ipsec) and internet key exchange (ike) document roadmap. *Request for Comments*, *6071*.

Hennessey, J., Hill, C., Denhardt, I., Viggnesh, V., Silvis, G., Krieger, O., & Desnoyers, P. (2014). Hardware as a service-enabling dynamic, user-level bare metal provisioning of pools of data center resources..

Hennessey, J., Tikale, S., Turk, A., Kaynar, E. U., Hill, C., Desnoyers, P., & Krieger, O. (2016). Hil: Designing an exokernel for the data center. In *Proceedings of the seventh acm symposium on cloud computing* (pp. 155–168).

Infrastructure as code: What is it? why is it important [Computer software manual]. (2019 (accessed November 20, 2019)). Retrieved from `https://www.hashicorp.com/resources/what-is-infrastructure-as-code/`

Installing emulab [Computer software manual]. (2018 (accessed November 30, 2019)). Retrieved from `https://gitlab.flux.utah.edu/emulab/emulab-devel/wikis/install/Introduction`

Introduction to ironic [Computer software manual]. (2019 (accessed November 10, 2019)). Retrieved from `https://docs.openstack.org/ironic/6.2.1/deploy/user-guide.html/`

Introduction to strongswan [Computer software manual]. ((accessed November 12, 2019)). Retrieved from `https://wiki.strongswan.org/projects/strongswan/wiki/IntroductionTostrongSwan/`

Introduction to vagrant [Computer software manual]. (2019 (accessed November 25, 2019)). Retrieved from `https://www.vagrantup.com/intro/index.html`

ipxe open source boot firmware [Computer software manual]. (2019 (accessed November 10, 2019)). Retrieved from `https://ipxe.org/`

John, W., Halén, J., Cai, X., Fu, C., Holmberg, T., Katardjiev, V., ... others (2018). Making cloud easy: design considerations and first components of a distributed operating system for cloud. In *10th {USENIX} workshop on hot topics in cloud computing (hotcloud 18).*

Kickstart installation [Computer software manual]. (2020 (accessed May 15, 2020)). Retrieved from https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/installation_guide/sect-kickstart-howto

Kominos, C. G., Seyvet, N., & Vandikas, K. (2017). Bare-metal, virtual machines and containers in openstack. In *2017 20th conference on innovations in clouds, internet and networks (icin)* (pp. 36–43).

Kurose, J. F., & Ross, K. W. (2017). *Computer networking – a top-down approach featuring the internet* (7th ed.). Addison-Wesley Professional.

Lighttpd [Computer software manual]. (2019 (accessed November 26, 2019)). Retrieved from https://www.lighttpd.net/

MAAS. (2019). https://maas.io/. ([Online; accessed 12-November-2019])

Mahalingam, M., Dutt, D. G., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., … Wright, C. (2014). Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. *RFC*, *7348*, 1–22.

Mell, P., Grance, T., et al. (2011). The nist definition of cloud computing.

Morris, K. (2016). *Infrastructure as code: managing servers in the cloud.* " O'Reilly Media, Inc.".

NetEm - Network Emulator [Computer software manual]. (2020 (accessed May 25, 2020)). Retrieved from https://www.linux.org/docs/man8/tc-netem.html

Norberg, S. (1999). *Building a windows nt bastion host in practice.* Version.

Open vswitch? [Computer software manual]. (2020 (accessed February 2, 2020)). Retrieved from http://docs.openvswitch.org/en/latest/intro/what-is-ovs/

Pu, Y., Deng, Y., & Nakao, A. (2011). Cloud rack: Enhanced virtual topology migration approach with open vswitch. In *The international conference on information networking 2011 (icoin2011)* (pp. 160–164).

Puppet razor [Computer software manual]. (2019 (accessed November 12, 2019)). Retrieved from `http://puppetlabs.com/solutions/next-generation-provisioning/`

Shiau, S., Sun, C.-K., Tsai, Y.-C., Juang, J.-N., & Huang, C.-Y. (2018). The design and implementation of a novel open source massive deployment system. *Applied Sciences*, *8*(6), 965.

Turk, A., Gudimetla, R. S., Kaynar, E. U., Hennessey, J., Tikale, S., Desnoyers, P., & Krieger, O. (2016). An experiment on bare-metal bigdata provisioning. In *8th {USENIX} workshop on hot topics in cloud computing (hotcloud 16).*

Vemula, S. (2016). *Performance evaluation of openstack deployment tools.*

Venkateswaran, S., & Sarkar, S. (2019). Time-sensitive provisioning of bare metal compute as a cloud service. In *2019 ieee 12th international conference on cloud computing (cloud)* (pp. 447–451).

Virtual box documentation [Computer software manual]. (2019 (accessed November 25, 2019)). Retrieved from `https://www.virtualbox.org/wiki/Documentation/`

What is vxlan? [Computer software manual]. (2019 (accessed November 15, 2019)). Retrieved from `https://www.juniper.net/us/en/products-services/what-is/vxlan/`

Wu, J.-N., Ko, Y.-H., Huang, K.-M., & Huang, M.-K. (2013). Heterogeneous diskless remote booting system on cloud operating system. In *International conference on grid and pervasive computing* (pp. 114–123).

X.509 certificate [Computer software manual]. (2020 (accessed May 25, 2020)). Retrieved from `https://www.sslauthority.com/x509-what-you-should-know/`

APPENDIX A. VAGRANTFILE FOR CENTRAL SERVER

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  # Box Settings
  config.vm.box = "centos/7"

  # Provider Settings
  config.vm.provider "virtualbox" do |vb|
   vb.memory = 4096
   vb.cpus = 4
  # vb.gui = true
  end

  # Network Settings
  # config.vm.network "forwarded_port", guest: 80, host: 8080
  # config.vm.network "forwarded_port", guest: 80, host: 8080,
  host_ip: "127.0.0.1"
  # config.vm.network "private_network", ip: "192.168.33.10"
  config.vm.network "private_network", ip: "192.168.33.10",
  virtualbox__intnet: "intnet"
  config.vm.network "private_network", ip: "192.168.40.10",
  virtualbox__intnet: "intshared"

  # config.vm.network "public_network"

  # Folder Settings
  # config.vm.synced_folder ".", "/var/www/htdocs"

  #
  # View the documentation for the provider you are using for more
  # information on available options.

  # Provision Settings
  config.vm.provision "shell", inline: <<-SHELL
  # Package installations
    yum -y update
    yum -y install dnsmasq dnsmasq-utils syslinux  tftp-server tftp
    vsftpd syslinux-tftpboot ipxe-bootimgs git xz-devel genisoimage
    squid mod_fcgid lighttpd-fastcgi mod_fastcgi openvswitch
    yum -y install net-tools vim wget patch epel-release tcpdump
    yum -y install lighttpd
    yum -y install strongswan

  #iPxe configuration
    cd /home/vagrant;
    git clone https://github.com/deepikakaushal39/RemoteBooting.git
    git clone git://github.com/ipxe/ipxe.git;
    cd ipxe;
    git clone https://github.com/tjhacker/ipxehttpproxy.git;
    yum -y group install "Development Tools"
    patch -p1 < ipxehttpproxy/proxypatch.p;
    cd src; touch demo.ipxe;
    echo '#!ipxe' >> demo.ipxe;
    echo 'dhcp' >> demo.ipxe;
    echo 'set http-proxy http://192.168.7.2:3128' >> demo.ipxe;
    echo 'chain http://192.168.7.1/menu.ipxe' >> demo.ipxe;
    make bin/undionly.kpxe EMBED=demo.ipxe;
    yum -y install lighttpd;
    mv /var/www/lighttpd/ /var/www/htdocs/;
    cp bin/undionly.kpxe /var/www/htdocs/;cd /home/vagrant;
    cp RemoteBooting/ks /var/www/htdocs/;
    echo '#!ipxe' >>
    /var/www/htdocs/menu.ipxe;
    echo 'set http-proxy http://192.168.7.2:3128' >>
    /var/www/htdocs/menu.ipxe;
    echo 'set base http://mirror.centos.org/centos/7/os/x86_64>>
    /var/www/htdocs/menu.ipxe;
    echo '# shell' >> /var/www/htdocs/menu.ipxe
    echo 'kernel ${base}/images/pxeboot/vmlinuz ks=http://192.168.7
    .1/ks proxy=http://192.168.7.2:3128 repo=${base}' >> /var/www/
    htdocs/menu.ipxe;
    echo 'initrd ${base}/images/pxeboot/initrd.img' >>
    /var/www/htdocs/menu.ipxe
    echo 'boot' >> /var/www/htdocs/menu.ipxe
    echo "reached 1!";
    sed -i 's|server.document-root = server_root + "/lighttpd"|
    server.document-root = server_root + "/htdocs"|g' /etc/
    lighttpd/lighttpd.conf;
    sed -i 's/server.use-ipv6 = "enable"/server.use-ipv6 =
    "disable"/' /etc/lighttpd/lighttpd.conf;
    systemctl restart lighttpd
    sudo chmod -R 775 /var/www/htdocs;
    sudo chmod 775 /var/www/htdocs/menu.ipxe;


  #squid configuration
    sed -i "s:\#cache_dir ufs /var/spool/squid 100 16 256:cache_dir
    ufs /var/spool/squid 100000 16 256:" /etc/squid/squid.conf;
    echo "maximum_object_size 30 GB" >> /etc/squid/squid.conf;
    echo "acl localnet src 192.168.33.0/16" >> /etc/squid/squid.conf;
    echo "icp_port 3130" >> /etc/squid/squid.conf;
    echo "icp_access allow localnet" >> /etc/squid/squid.conf;
    squid -z; sleep 5;
    systemctl enable squid;
    systemctl start squid;


  #VXLAN configuration
    sudo ip route add 192.168.43.0/24 via 192.168.40.20 dev eth2
    sudo ip link add vxlan0 type vxlan id 7 dev eth1 dstport 8472
    sudo bridge fdb append to 00:00:00:00:00:00 dst 192.168.43.20
    dev vxlan0
    sudo  ip link set up dev vxlan0

  #OpenvSwitch
    yum -y install https://repos.fedorapeople.org/repos/openstack/
    EOL/openstack-juno/epel-7/openvswitch-2.3.1-2.el7.x86_64.rpm
    systemctl enable openvswitch
    systemctl start openvswitch
    ovs-vsctl add-br ovsBridge
    ovs-vsctl add-port ovsBridge eth1
    ovs-vsctl add-port ovsBridge vxlan0
    ifconfig ovsBridge 192.168.7.1/24

  #dnsmasq configuration
    dnsmasq --enable-tftp --tftp-root=/var/www/htdocs --interface=
    ovsBridge --dhcp-range= 192.168.7.1,192.168.7.100,255.255.255.0
    --dhcp-match=IPXEBOOT, 175 --dhcp-option=175,8:1:1 --dhcp-boot=
    undionly.kpxe, 192.168.7.1 --server=8.8.4.4

  #Strongswan configuration
    yum -y install strongswan
    systemctl start strongswan;
    systemctl enable strongswan;
    cd /etc/strongswan/swanctl;
    cp /home/vagrant/RemoteBooting/serverKey.pem private
    cp /home/vagrant/RemoteBooting/strongswanKey.pem private
    cp /home/vagrant/RemoteBooting/strongswanCert.pem x509ca
    cp /home/vagrant/RemoteBooting/serverCert.pem x509
    cp /home/vagrant/RemoteBooting/clientCert.pem x509
    cp /home/vagrant/RemoteBooting/serverSwanctl.conf swanctl.conf
    swanctl --load-all
    swanctl -i -c net-net

  #Etherate tools
    cd /home/vagrant
    git clone https://github.com/jwbensley/Etherate.git
    cd Etherate/
    ./configure.sh && make

  #Firewall rules
    systemctl start firewalld
    firewall-cmd --add-service=dns --permanent
    firewall-cmd --add-service=dhcp --permanent
    firewall-cmd --add-service=tftp --permanent
    firewall-cmd --add-service=http --permanent
    firewall-cmd --add-service=https --permanent
    firewall-cmd --add-service=squid --permanent
    firewall-cmd --add-masquerade --permanent
    firewall-cmd --zone=public --list-ports
    firewall-cmd --get-active-zones
    firewall-cmd --add-port 3130/udp --permanent
    firewall-cmd --permanent --add-rich-rule='rule protocol
    value="esp" accept'
    firewall-cmd --permanent --add-rich-rule='rule protocol
    value="ah" accept'
    firewall-cmd --permanent --add-port=500/udp
    firewall-cmd --permanent --add-port=4500/udp
    firewall-cmd --permanent --add-service="ipsec"
    firewall-cmd --permanent --add-port=8472/udp
    firewall-cmd --permanent --add-port=8472/udp --zone=dmz
    firewall-cmd --add-interface=ovsBridge --permanent
    firewall-cmd --add-source=192.168.7.0/24 --permanent
    firewall-cmd --reload
    sudo systemctl restart network

   SHELL
  # Can use different file for all shell commands too
end
```

APPENDIX B. VAGRANTFILE FOR JUMP HOST

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
 # Box Settings
  config.vm.box = "centos/7"

  # Provider Settings
  config.vm.provider "virtualbox" do |vb|
   vb.memory = 4096
   vb.cpus = 4
   vb.customize ["modifyvm", :id, "--nicpromisc1", "allow-all"]
   vb.customize ["modifyvm", :id, "--nicpromisc2", "allow-all"]
   vb.customize ["modifyvm", :id, "--nicpromisc3", "allow-all"]
  end

  # Network Settings
  # config.vm.network "forwarded_port", guest: 80, host: 8080
  # config.vm.network "forwarded_port", guest: 80, host: 8080,
  host_ip: "127.0.0.1"
  #config.vm.network "private_network", ip: "192.168.33.10"
config.vm.network "private_network", ip: "192.168.43.20",
virtualbox__intnet: "intnet2"
config.vm.network "private_network", ip: "192.168.40.20",
virtualbox__intnet: "intshared"

  # config.vm.network "public_network"

  # Folder Settings
  # config.vm.synced_folder ".", "/var/www/html"

  #
  # View the documentation for the provider you are using for more
  # information on available options.

  # Provision Settings
  config.vm.provision "shell", inline: <<-SHELL

    sudo ip route add 192.168.33.0/24 via 192.168.40.10 dev eth2;
    sudo ip link add vxlan0 type vxlan id 7 dev eth1 dstport 8472;
    sudo bridge fdb append to 00:00:00:00:00:00 dst 192.168.33.10
    dev vxlan0;
    sudo  ip link set up dev vxlan0;
    # sudo systemctl enable strongswan
    # sudo systemctl start strongswan
    # sudo swanctl --load-all
    # sudo swanctl -i -c net-net

#OpenvSwitch
    yum -y install https://repos.fedorapeople.org/repos/openstack/
    EOL/openstack-juno/epel-7/openvswitch-2.3.1-2.el7.x86_64.rpm
    yum -y install net-tools
    systemctl enable openvswitch
    systemctl start openvswitch
    ovs-vsctl add-br ovsBridge
    ovs-vsctl add-port ovsBridge eth1
    ovs-vsctl add-port ovsBridge vxlan0
    ifconfig ovsBridge 192.168.7.2/24

#Squid configuration
    yum -y install squid;
    sed -i "s:\#cache_dir ufs /var/spool/squid 100 16 256:cache_dir
    ufs /var/spool/squid 100000 16 256:" /etc/squid/squid.conf;
    echo "maximum_object_size 30 GB" >> /etc/squid/squid.conf;
    echo "acl localnet src 192.168.7.0/16" >> /etc/squid/squid.conf;
    echo "icp_port 3130" >> /etc/squid/squid.conf;
    echo "icp_access allow localnet" >> /etc/squid/squid.conf;
    echo "cache_peer 192.168.7.1 parent 3128 3130 default" >>
    /etc/squid/squid.conf
    echo "cache_peer_domain 192.168.7.1 allow" >>
    /etc/squid/
    squid.conf
    squid -z; sleep 5;
    systemctl enable squid;
    systemctl start squid;
    export http_proxy=http://192.168.7.2:3128
    chown -R squid:squid /var/spool/squid/

#Strongswan configuration
    yum -y install git net-tools vim wget patch epel-release tcpdump;
    yum -y group install "Development Tools"
    yum -y install strongswan
    cd /home/vagrant
    git clone https://github.com/deepikakaushal39/RemoteBooting.git
    systemctl start strongswan;
    systemctl enable strongswan;
    cd /etc/strongswan/swanctl;
    cp /home/vagrant/RemoteBooting/clientKey.pem private
    cp /home/vagrant/RemoteBooting/strongswanKey.pem private
    cp /home/vagrant/RemoteBooting/strongswanCert.pem x509ca
    cp /home/vagrant/RemoteBooting/clientCert.pem x509
    cp /home/vagrant/RemoteBooting/serverCert.pem x509
    cp /home/vagrant/RemoteBooting/clientswanctl.conf swanctl.conf
    swanctl --load-all
    swanctl -i -c net-net

#Etherate tools
    cd /home/vagrant
    git clone https://github.com/jwbensley/Etherate.git
    cd Etherate/
    ./configure.sh && make

   SHELL
  # Can use different file for all shell commands too
end
```

APPENDIX C. FILES FOR IPXE

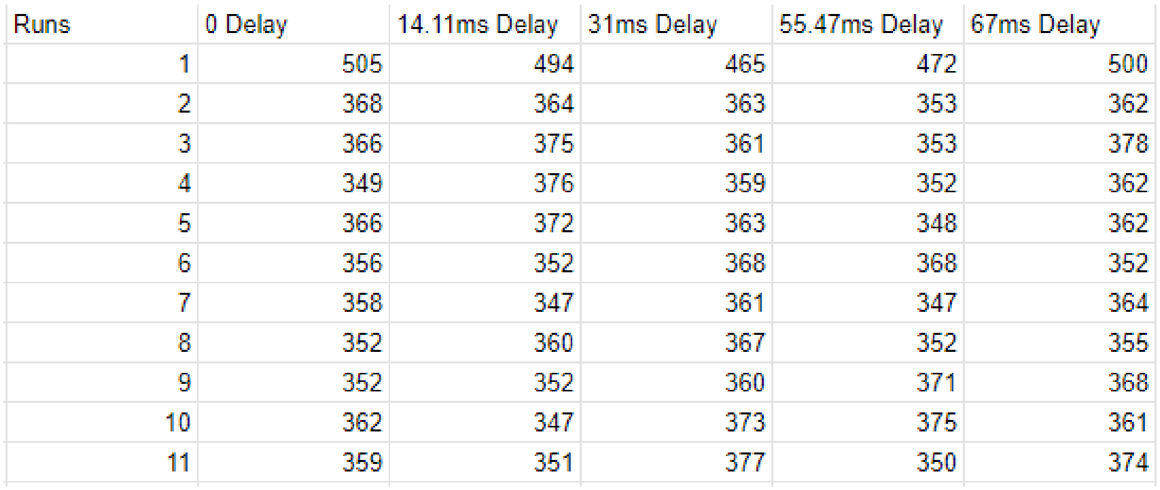
```
#!ipxe

dhcp
set http-proxy http://192.168.7.2:3128
chain http://192.168.7.1/menu.ipxe
```

Demo.ipxe

```
#!ipxe
#
set http-proxy http://192.168.7.2:3128
set base http://mirror.centos.org/centos/7/os/x86_64
# shell
kernel ${base}/images/pxeboot/vmlinuz ks=http://192.168.7.1/ks
proxy=http://192.168.7.2:3128 repo=${base}
initrd ${base}/images/pxeboot/initrd.img
boot
```

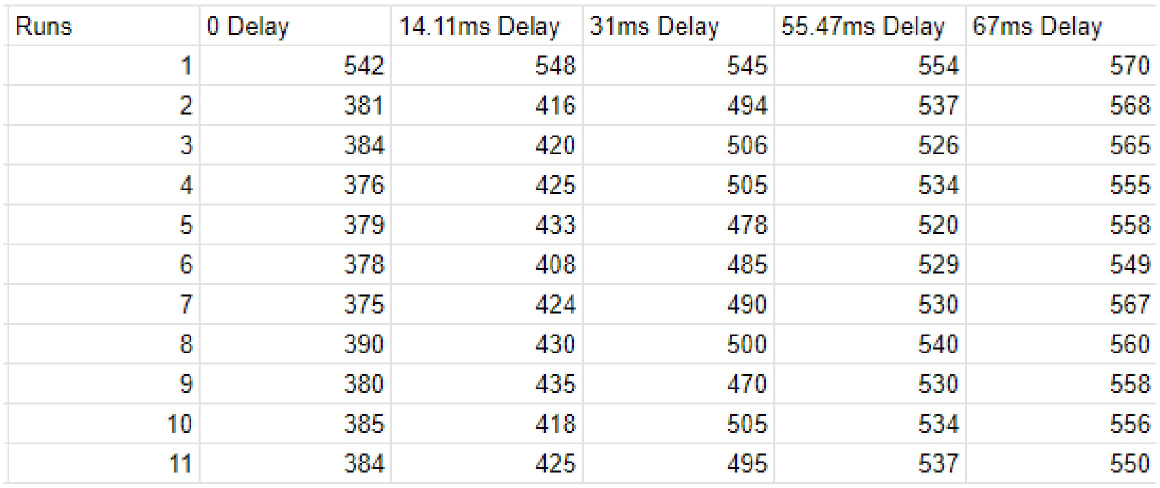
Menu.ipxe

APPENDIX D. DATA COLLECTED FOR BOOTING TIME



Runs	0 Delay	14.11ms Delay	31ms Delay	55.47ms Delay	67ms Delay
1	505	494	465	472	500
2	368	364	363	353	362
3	366	375	361	353	378
4	349	376	359	352	362
5	366	372	363	348	362
6	356	352	368	368	352
7	358	347	361	347	364
8	352	360	367	352	355
9	352	352	360	371	368
10	362	347	373	375	361
11	359	351	377	350	374

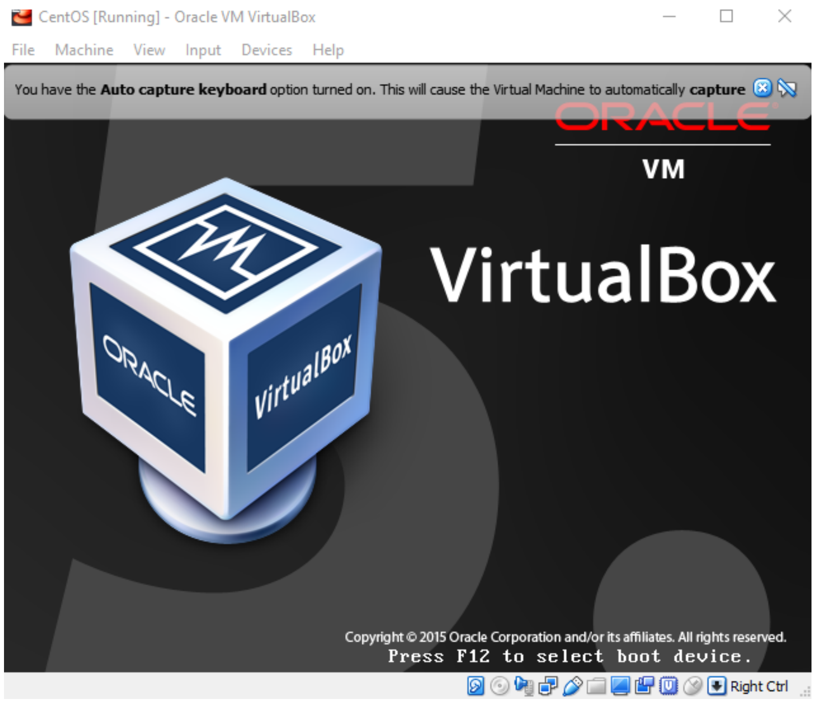
Figure 6. Time taken to boot (in sec) for 11 consecutive runs with Squid on the jump host.



Runs	0 Delay	14.11ms Delay	31ms Delay	55.47ms Delay	67ms Delay
1	542	548	545	554	570
2	381	416	494	537	568
3	384	420	506	526	565
4	376	425	505	534	555
5	379	433	478	520	558
6	378	408	485	529	549
7	375	424	490	530	567
8	390	430	500	540	560
9	380	435	470	530	558
10	385	418	505	534	556
11	384	425	495	537	550

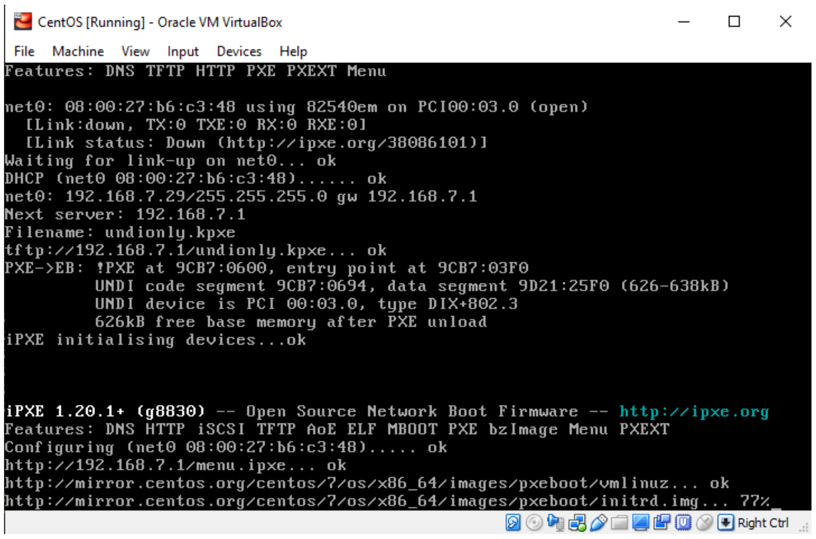
Figure 7. Time taken to boot (in sec) for 11 consecutive runs without Squid on the jump host.

APPENDIX E. SCREENSHOTS OF PROCESS


CentOS [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to automatically **capture**
ORACLE
VM
VirtualBox
Right Ctrl

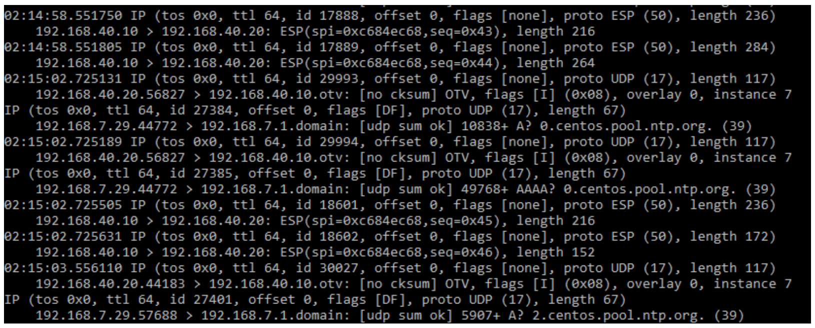
```
Copyright © 2015 Oracle Corporation and/or its affiliates. All rights reserved.
Press F12 to select boot device.
```

Figure 8. Power on virtual machine


CentOS [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

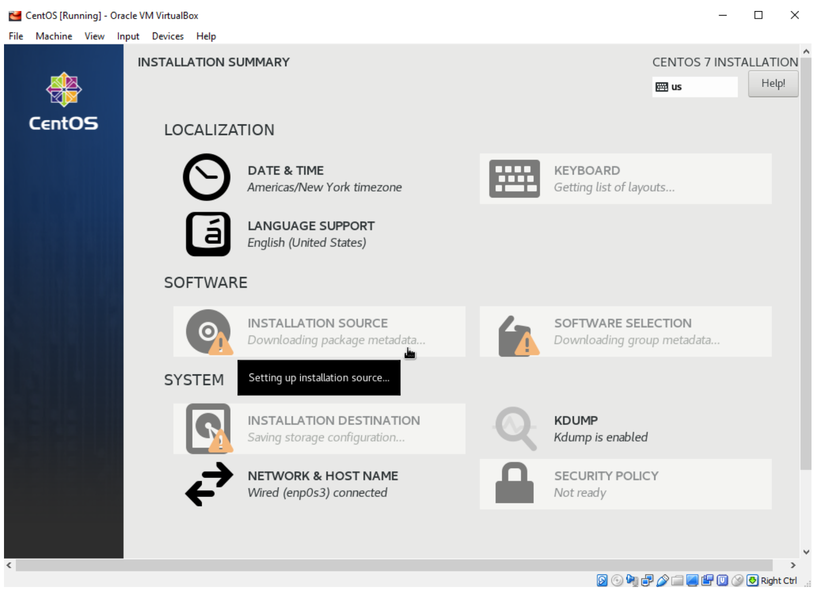
```
Features: DNS TFTP HTTP PXE PXEXT Menu

net0: 08:00:27:b6:c3:48 using 82540em on PCI00:03.0 (open)
  [Link:down, TX:0 TXE:0 RX:0 RXE:0]
  [Link status: Down (http://ipxe.org/38086101)]
Waiting for link-up on net0... ok
DHCP (net0 08:00:27:b6:c3:48)...... ok
net0: 192.168.7.29/255.255.255.0 gw 192.168.7.1
Next server: 192.168.7.1
Filename: undionly.kpxe
tftp://192.168.7.1/undionly.kpxe... ok
PXE->EB: !PXE at 9CB7:0600, entry point at 9CB7:03F0
         UNDI code segment 9CB7:0694, data segment 9D21:25F0 (626-638kB)
         UNDI device is PCI 00:03.0, type DIX+802.3
         626kB free base memory after PXE unload
iPXE initialising devices...ok



iPXE 1.20.1+ (g8830) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS HTTP iSCSI TFTP AoE ELF MBOOT PXE bzImage Menu PXEXT
Configuring (net0 08:00:27:b6:c3:48)..... ok
http://192.168.7.1/menu.ipxe... ok
http://mirror.centos.org/centos/7/os/x86_64/images/pxeboot/vmlinuz... ok
http://mirror.centos.org/centos/7/os/x86_64/images/pxeboot/initrd.img... 77%_
```

Figure 9. DHCP and iPXE in process



```
02:14:58.551750 IP (tos 0x0, ttl 64, id 17888, offset 0, flags [none], proto ESP (50), length 236)
    192.168.40.10 > 192.168.40.20: ESP(spi=0xc684ec68,seq=0x43), length 216
02:14:58.551805 IP (tos 0x0, ttl 64, id 17889, offset 0, flags [none], proto ESP (50), length 284)
    192.168.40.10 > 192.168.40.20: ESP(spi=0xc684ec68,seq=0x44), length 264
02:15:02.725131 IP (tos 0x0, ttl 64, id 29993, offset 0, flags [none], proto UDP (17), length 117)
    192.168.40.20.56827 > 192.168.40.10.otv: [no cksum] OTV, flags [I] (0x08), overlay 0, instance 7
IP (tos 0x0, ttl 64, id 27384, offset 0, flags [DF], proto UDP (17), length 67)
    192.168.7.29.44772 > 192.168.7.1.domain: [udp sum ok] 10838+ A? 0.centos.pool.ntp.org. (39)
02:15:02.725189 IP (tos 0x0, ttl 64, id 29994, offset 0, flags [none], proto UDP (17), length 117)
    192.168.40.20.56827 > 192.168.40.10.otv: [no cksum] OTV, flags [I] (0x08), overlay 0, instance 7
IP (tos 0x0, ttl 64, id 27385, offset 0, flags [DF], proto UDP (17), length 67)
    192.168.7.29.44772 > 192.168.7.1.domain: [udp sum ok] 49768+ AAAA? 0.centos.pool.ntp.org. (39)
02:15:02.725505 IP (tos 0x0, ttl 64, id 18601, offset 0, flags [none], proto ESP (50), length 236)
    192.168.40.10 > 192.168.40.20: ESP(spi=0xc684ec68,seq=0x45), length 216
02:15:02.725631 IP (tos 0x0, ttl 64, id 18602, offset 0, flags [none], proto ESP (50), length 172)
    192.168.40.10 > 192.168.40.20: ESP(spi=0xc684ec68,seq=0x46), length 152
02:15:03.556110 IP (tos 0x0, ttl 64, id 30027, offset 0, flags [none], proto UDP (17), length 117)
    192.168.40.20.44183 > 192.168.40.10.otv: [no cksum] OTV, flags [I] (0x08), overlay 0, instance 7
IP (tos 0x0, ttl 64, id 27401, offset 0, flags [DF], proto UDP (17), length 67)
    192.168.7.29.57688 > 192.168.7.1.domain: [udp sum ok] 5907+ A? 2.centos.pool.ntp.org. (39)
```

Figure 10. Encrypted packets in tcpdump


CentOS [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
CentOS
Help!
Right Ctrl

INSTALLATION SUMMARY

CENTOS 7 INSTALLATION

us

LOCALIZATION

DATE & TIME
Americas/New York timezone

KEYBOARD
Getting list of layouts...

LANGUAGE SUPPORT
English (United States)

SOFTWARE

INSTALLATION SOURCE
Downloading package metadata...

SOFTWARE SELECTION
Downloading group metadata...

SYSTEM

Setting up installation source...

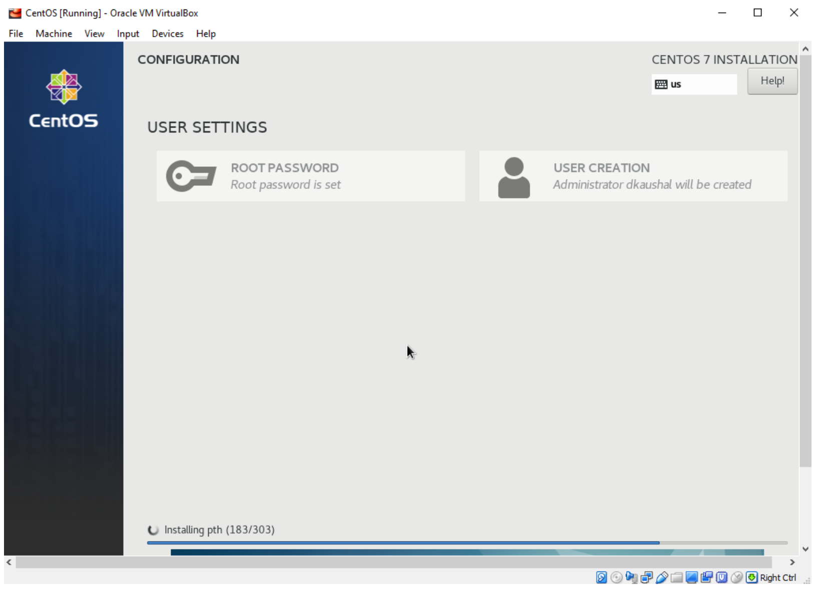
INSTALLATION DESTINATION
Saving storage configuration...

KDUMP
Kdump is enabled

NETWORK & HOST NAME
Wired (enp0s3) connected

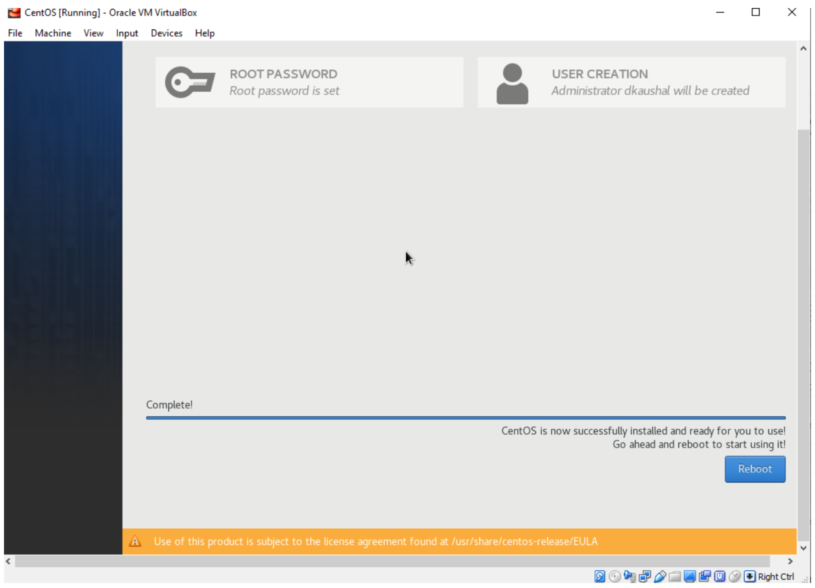
SECURITY POLICY
Not ready

Figure 11. Initial configuration using kickstart file



Installing pth (183/303)

Figure 12. Installation of packages


CentOS [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
ROOT PASSWORD
Root password is set
USER CREATION
Administrator dkaushal will be created
Reboot
Right Ctrl

Complete!

CentOS is now successfully installed and ready for you to use!
Go ahead and reboot to start using it!

Use of this product is subject to the license agreement found at /usr/share/centos-release/EULA

Figure 13. Loading completed