V-NetLab: A Test-bed for Security Experiments

by

Varun Katta

to

The Graduate School
in partial fulfillment of the
Requirements
for the degree of

Master of Science
in
Computer Science

Stony Brook University
May 2006

Stony Brook University

The Graduate School

Varun Katta

We, the thesis committee for the above candidate for the

Master of Science degree,

hereby recommend acceptance of this thesis.

Dr. R. C. Sekar, Thesis Advisor,
Computer Science Department

Dr. Radu Sion, Chairman of Thesis Committee,
Computer Science Department

Dr. Rob Johnson, Comittee Member,
Computer Science Department

This thesis is accepted by the Graduate School.

Dean of the Graduate School

Abstract of the Thesis

V-NetLab: A Test-bed for Security Experiments

by

Varun Katta

Master of Science

in

Computer Science

Stony Brook University

2006

Network security experiments usually require root privileges to a set of networked machines as they typically involve tasks like network surveillance, intrusion detection, configuring firewalls, installing and running malware etc. Some of these experiments require regular software and OS (re)installations, re-configuration of networks (addition or removal of machines and networking components). Due to obvious security concerns these test-bed networks should be totally isolated from regular external networks. Building and managing such dedicated physical networks for security experiments on a per-user basis is cumbersome and expensive.

We propose to address the aforementioned problems through V-NetLab which overlays virtual machines (VM's) over a set of physical nodes, emulates networking of these machines and provides access to them as virtual networks. V-NetLab allows users to easily configure, start and shutdown virtual networks by automating these tasks. It allows administrators to monitor, configure networks for users, manage user groups as teams through an easy-to-use management interface. Virtualization is achieved at data-link layer, allowing identical networks to run simultaneously on the same underlying hardware but isolated from each other. Users can run multiple virtual networks, share access to networks through the notion of teams, access networks remotely via SSH. The framework is realized on regular commodity PCs with VMware Workstations as VM's.

Table of Contents

List of Figures

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and writing of this thesis.

First and foremost, Dr. R C Sekar for his guidance, patience and support throughout my research and writing of this thesis.

I would like to thank Weiqing Sun and Kumar Krishna for their contributions to this work.

I would also like to thank Weiqing Sun and Alok Tongaonkar for the valuable time they spent with me discussing issues.

I would additionally like to thank my lab colleagues and friends for their assistance and valuable inputs at various times during my research.

Chapter 1

Introduction

Providing access to a group of machines networked together for security experiments has its own challenges.

Security experiments usually require users to have administrative privileges to the machines in their networks. When users have root access and make any fatal errors while configuring the machines or running the experiments, then machines can get into an unrecoverable state and re-installing the software completely including OS might be the only option to use the machine again for experiments.

Some class of experiments need frequent re-configuration of network definitions. During re-configuration some physical nodes or networking components like hubs, switches might have to be added or deleted. If the network size is sufficiently large then it becomes really cumbersome to modify the current topology into the desired topology. Thus, the latency in switching network configurations for some experiments might be high and is not desirable.

Since administrative privileges are provided to users to run potentially malicious programs as part of security experiments, it is important to isolate test-bed networks from regular networks and prevent users from having unauthorized and illegal access to other users data.

The challenge of scaling up the network to support a large base of users is prohibitively expensive in terms of human-effort needed to setup, maintain the networks (wire machines and other networking components, configure machines with the right OS and software), and hardware resources needed to support the networks.

Given the above challenges, virtualization of physical resources looks like a promising solution to the problem of providing networked machines for security experiments. V-NetLab is one such effort. We replace the physical machines by virtual machines, there by replacing physical networks by virtual networks. We use VMware Workstation as VM's for V-NetLab currently. Using VMware, we avoid the problem of reinstalling OS every time something goes wrong by simply reverting

back to initial VM image, if we reach an unrecoverable state. The number of VM's which can run on a physical machine is bounded only by hardware constraints.

V-NetLab allows a user to specify a network definition file, and register her network. Once a network is registered she can start and shutdown the network with little effort. Thus, V-NetLab addresses the issues of scalability and reconfigurability in the physical world as a new virtual network of required size can be realized by providing a new network definition file, registering it and starting the network. The size of virtual networks one can realize is limited only by resource constraints.

Virtualization is achieved at the data-link layer and virtual networks are isolated from each other and from the external world (Internet). Thus, V-NetLab addresses the security problems which arise with users gaining administrative privileges to machines on the network. User's root privileges on virtual machines are confined to the virtual networks alone. Data-link layer virtualization provides the benefit to run identical networks simultaneously, isolated from the each other. The framework allows users to run multiple networks, and multiple users to share networks. Sharing allows users to work together on experiments, which is essential when collaborating. The framework also eases monitoring and managing virtual networks and users through an administrator interface. Users can be grouped to teams and teams in turn can be managed by the administrator of V-NetLab.

The work presented in this thesis describes V-NetLab and talks about the challenges in coming up with such a system.

1.1 Thesis Overview

In the next chapter we talk about virtual machines and the advantages of virtualization of underlying hardware. Chapter 3 talks about the design of V-NetLab, its architecture and gives a sufficiently good overview of the system. Chapter 4 talks about some of the technical challenges, implementation issues and is followed by the Summary chapter.

Chapter 2

Background

2.1 Virtualization

Virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments. Virtualization techniques create multiple isolated partitions (Virtual Machines (VM) or Virtual Private Servers (VPS)) on a single physical server. It is an abstraction that decouples the physical hardware from the operating system to deliver greater IT resource utilization and flexibility.

Virtualization allows multiple virtual machines, with heterogeneous operating systems to run in isolation, side-by-side on the same physical machine. Each virtual machine has its own set of virtual hardware (e.g., RAM, CPU, NIC, etc.) upon which an operating system and applications are loaded. The operating system sees a consistent, normalized set of hardware regardless of the actual physical hardware components.

2.1.1 Types of Virtualization

There are several kinds of virtualization techniques which provide similar features but differ in the degree of abstraction and the methods used for virtualization.

- **Virtual Machines (VMs):** Virtual Machines emulate some real or fictional hardware, which in turn requires real resources from the Host (the machine running the VMs). This approach, used by most System Emulators, allows the emulator to run an arbitrary Guest Operating System without modifications because OS isn't aware that its not running on real hardware. The main issue with this approach is that some CPU instructions require additional privileges and may not be executed in user space thus requiring a Virtual Machines Monitor (VMM) to analyze executed code and make it safe on-the-

fly. Hardware Emulation approach is used by VMware products and Microsoft Virtual Server.

- **Para-Virtualized Machines:** This technique also requires a VMM, but most of its work is performed in the Guest OS code, which in turn is modified to support this VMM and avoid unnecessary use of privileged instructions. The paravirtualization technique also enables running different OS' on a single server, but requires them to be ported. The paravirtualization approach is used by Xen, UML.

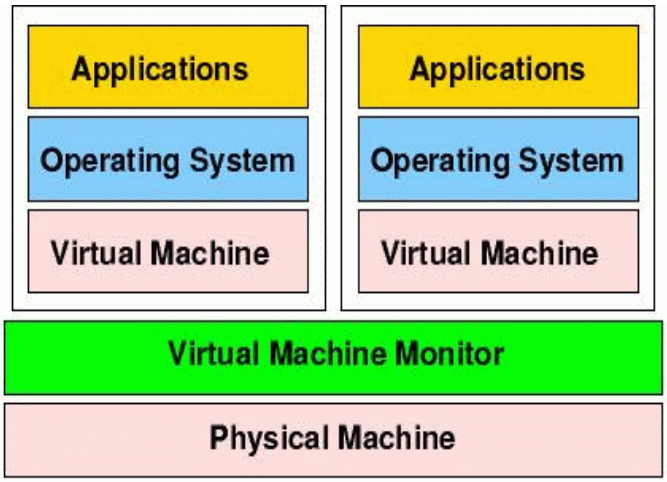
- **Virtualization on the OS Level:** Most applications running on a server can easily share a machine with others, if they could be isolated and secured. Further, in most situations, different operating systems are not required on the same server, merely multiple instances of a single Operating System. OS Virtualization systems have been designed to provide the required isolation and security to run multiple applications or copies of the same (or similar i.e different Linuxes) OS on the same server. OpenVZ, Linux VServer are examples of OS virtualization.

The three techniques differ in complexity of implementation, breadth of OS support, performance in comparison with standalone server, and level of access to common resources. For example, VMs have wide scope of usage, but poor performance. Para-VMs have better performance, but can support fewer OSs because of the need to port them.

2.1.2 Benefits of Virtualization

- **Partitioning**
  - Multiple applications and operating systems can be supported within a single physical system
  - Servers can be consolidated into virtual machines on either a scale-up or scale-out architecture
  - Computing resources are treated as a uniform pool to be allocated to virtual machines in a controlled manner

- **Isolation**
  - Virtual machines are completely isolated from the host machine and other virtual machines. If a virtual machine crashes, all others are unaffected
  - Data does not leak across virtual machines and applications can only communicate over configured network connections



A virtual machine monitor provides a virtual machine abstraction in which standard operating systems and applications may run.

Figure 2.1: Virtual Machine Monitor

- **Encapsulation**
  - Complete virtual machine environment is saved as a single file; easy to back up, move and copy
  - Standardized virtualized hardware is presented to the application - guaranteeing compatibility

2.1.3 Virtualization at Hardware Abstraction Layer (HAL)

Virtualization at the HAL [14] exploits the similarity in architectures of the guest and host platforms to cut down the interpretation latency. Most of the today's world's commercial PC emulators use this virtualization technique on popular x86 platforms to make it efficient and its use, viable and practical. Virtualization technique helps map the virtual resources to physical resources and use the native hardware for computations in the virtual machine. When the emulated machine needs to talk to critical physical resources, the simulator takes over and multiplexes appropriately. For such a virtualization technology to work correctly, the VM must be able to trap every privileged instruction execution and pass it to the underlying Virtual Machine Moniter (VMM) to be taken care of. This is because, in a VMM environment, multiple VMs may each have an OS running that wants to issue privileged instructions and get the CPU's attention. When a trap occurs during

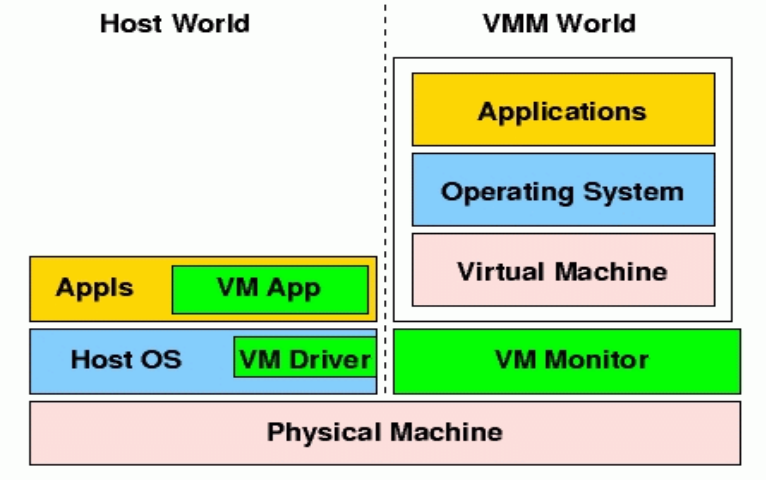
privileged instruction execution, rather than generating an exception and crashing, the instruction is sent to the VMM. This allows the VMM to take complete control of the machine and keep each VM isolated. The VMM then either executes the instruction on the processor, or emulates the results and returns them to the VM. However, the most popular platform, x86, is not fully-virtualizable, i.e. certain supervisor (privileged) instructions fail silently rather than causing a convenient trap when executed with insufficient privileges. Thus, the virtualization technique must have some workaround to pass control to the VMM when a faulting instruction executes. Most commercial emulators use techniques like code scanning and dynamic instruction rewriting to overcome such issues.

2.1.4 VMware Workstation

VMware Workstation provides virtualization at hardware abstraction layer. It works by creating fully isolated, secure virtual machines that encapsulate an operating system and its applications. The VMware virtualization layer maps the physical hardware resources to the virtual machine's resources, so each virtual machine has its own CPU, memory, disks, and I/O devices, and is the full equivalent of a standard x86 machine. VMware Workstation installs onto the host operating system and provides broad hardware support by inheriting device support from the host.

VMware's VMMs can be standalone or hosted. A standalone VMM is basically a software layer on the base hardware that lets users create one or more VMs (Figure 2.1). These are similar to operating systems, require device drivers for each hardware device, and are typically limited in hardware support. Such VMMs are typically used in servers, VMware ESX [3] server being a prime example of such an architecture. A hosted VMM, however, runs as an application on an existing host operating system. It can take advantage of the host operating system for memory management, processor scheduling, hardware drivers, and resource management. VMware Workstation group of products use this hosted virtual machine architecture. VMware products are targeted towards x86-based workstations and servers. Thus, it has to deal with the complications that arise as x86 is not a fully-virtualizable architecture. VMware deals with this problem by using a patent-pending technology that dynamically rewrites portions of the hosted machine code to insert traps wherever VMM intervention is required. Although it solves the problem, it adds some overhead due to the translation and execution costs. VMware tries to reduce the cost by caching the results and reusing them wherever possible. Nevertheless, it again adds some caching cost that is hard to avoid.

To understand how VMware workstation is installed and run, it helps to look at the way IA32 platform works. On the intel architecture, the protection mechanism provides four privilege levels, 0 through 3. Levels are also called rings. These protection rings exist only in Protected Mode. According to Intel, ring 0 is meant



VMware's hosted virtual machine model splits the virtualization software between a virtual machine monitor that virtualizes the CPU, an application that uses a host operating system for device support, and an operating system driver for transitioning between them.

Figure 2.2: VMware Virtualization

for operating systems and kernel services, ring 1 and 2 for device drivers, and ring 3 for applications. However, in practice, most operating systems along with the device drivers run completely in ring 0 and applications in ring 3. Privileged instructions are allowed only in ring 0, and cause protection violation if executed anywhere else.

VMware Workstation has three components: the VMX driver and VMM installed in ring 0, and the VMware application (VMApp) in ring 3 (See Figure 2.2). The VMX driver is installed within the operating system to gain the high privilege levels required by the virtual machine monitor. When executed, the VMApp loads the VMM into kernel memory with the help of VMX driver, giving it the highest privilege (ring 0). The host OS, at this point, knows about the VMX driver and the VMApp, but does not know about the VMM. The machine now has two worlds: the host world and the VMM world. The VMM world can communicate directly with the processor hardware or through the VMX driver to the host world. However, every switch to the host world would require all the hardware states to be saved and restored on return, which makes switching hit the performance.

When the guest OS or any of its applications run purely computational programs, they are executed directly through the VMM in the CPU. I/O instructions, being privileged ones, are trapped by the VMM and are executed in the host world by a world switch. The I/O operations requested in the VM are translated to high-level

I/O related calls and are invoked through the VMApp in the host world, and the results are communicated back to the VMM world. An example is illustrated in Figure 2.3. for the network send and recv operation [16].

The newer versions of VMware Workstation come with some of the striking features. Pointer integration with the host desktop allows the use to move the mouse pointer seamlessly in and out of the VMware Application's display window like it happens with any other window-based application. File sharing allows the user to share files and folders between the host and the guest machines to help easy transfer of data from and to the virtual machines for backing up and other purposes. Dynamic display resizing lets the user dynamically resize the VMware Application's display window like any other window. This is not so trivial realizing the fact that every resize operation changes the screen resolution for the virtual machine.

2.2 Introduction to VMware Networking

2.2.1 VMware Networking Components

There are three main VMware networking components:

- **Virtual Switch:**

  Like a physical switch, a virtual switch lets one connect other networking components together. Virtual switches are created as needed by the VMware Workstation software, up to a total of nine switches. One can connect one or more virtual machines to a switch. A few of the switches and the networks associated with them are, by default, used for special named configurations. The bridged network normally uses VMnet0. The host-only network uses VMnet1 by default. And the NAT network uses VMnet8 by default. The others available networks are simply named VMnet2, VMnet3, VMnet4, and so on.

- **Bridge:**

  The bridge lets one connect a virtual machine to the LAN of the host computer. It connects the virtual network adapter in the virtual machine to the physical Ethernet adapter in your host computer.

- **Host Virtual Adapter:**

  It is a virtual ethernet adapter that appears to the host operating system as a Host-Only Interface on a Linux host. It allows communication between the host computer and the virtual machines on that host computer. The host virtual adapter is used in host-only and NAT configurations.

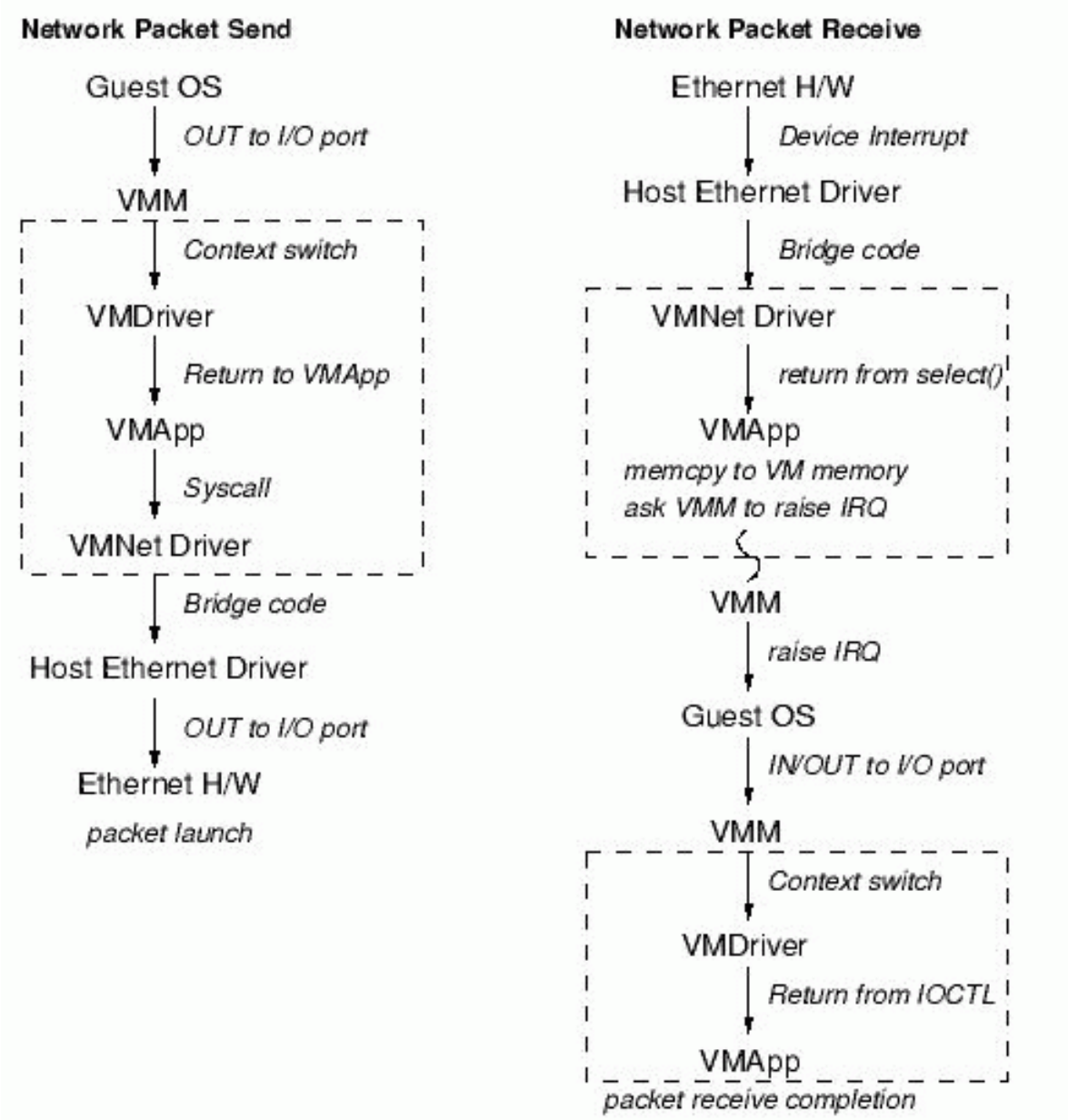
2.2.2 VMware Networking Basics

VMware Workstation supports a variety of networking setups to help user connect to the network according to her convenience.

- **Bridged:** If you use bridged networking, the virtual machine is a full participant in the network. It has access to other machines on the network and can be contacted by other machines on the network as if it were a physical computer on the network. In this mode the virtual machine's network interface is bridged to physical host's interface and sets the physical host's network interface in promiscuous mode.

- **NAT:** In NAT mode, the virtual machine does not have its own IP address on the external network. Instead, a separate private network is set up on the host computer. The virtual machine gets an address on that network from the VMware virtual DHCP server. The VMware NAT device passes network data between one or more virtual machines and the external network. It identifies incoming data packets intended for each virtual machine and sends them to the correct destination.

- **Host-Only:** In this mode, VMware installs a virtual Ethernet adapter in the host OS that communicates with the VMnet1 switch. The host OS believes this is just another Ethernet adapter. Workstation also runs a virtual DHCP server connected to the VMnet1 switch. In this mode, the DHCP service will assign addresses to the VM's virtual Ethernet adapters (and actually the host OSs virtual Ethernet adapter as well) that are connected to the VMnet1 switch. This allows communication between a virtual machine and the host operating system, but it is not routed to the outside world. Multiple virtual machines can talk to each other as well. Host-Only networking also allows connection of virtual machines to outside networks through the host OS.



Components involved in a virtual machine network packet send and receive. Boxed components delineate components that are due to the hosted nature of the network device virtualization.

Figure 2.3: VMware Networking

Chapter 3

System Design

3.1 Design Goals

Given the kind of scenarios we propose V-NetLab to support, we believe V-NetLab should support the following design goals and features.

- **Isolation**

  V-NetLab should ensure that the operation of one virtual network does not affect the operation of any other networks, expect possibly due to resource contention issues that arise as a result of sharing the same underlying hardware.

- **Faithful Network Emulation**

  - *Virtualized Ethernet*: Since network security projects often involve tasks such as packet sniffing, it is necessary that networks be emulated accurately after they are overlaid on physical workstations, so that all and only those packets that are supposed to be visible at any network node become visible.
  - *Virtualized network devices*: To provide support for the user to define his own network topology, network devices like switch and hub should be supported.

- **Cost-effective and Scalable**

  The software design should provide good performance and scale well.

- **Automation**

  - *Automatic network configuration*: Network configuration for virtual networks such as setting up IP addresses, DNS servers, host names etc should

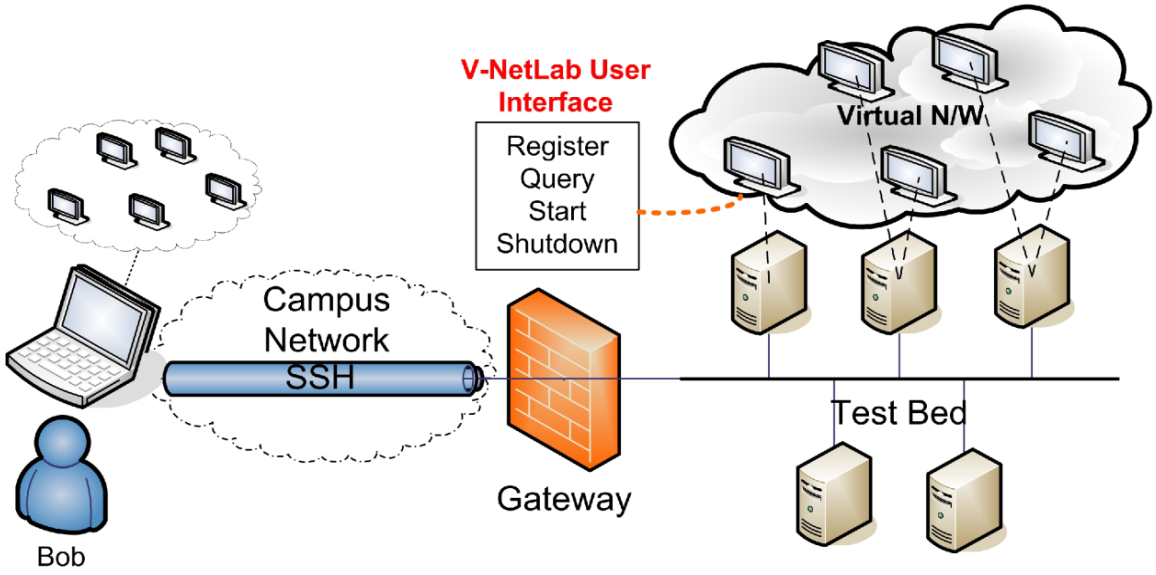


Figure 3.1: Logical View for V-NetLab Laboratory

highly automated. Users should be able to accomplish this by providing this information through a configuration file.

- *Location transparency*: Since our approach is based on overlay of virtual machines over a set of physical machines, to take full advantage of available computing resources, the generation of overlay scheme should be automated and transparent to the user.
- *Migration*: Since, overlay scheme of a particular network can change with every session based on the available computing resources, V-NetLab should have an efficient scheme to migrate the dormant virtual machines images of virtual networks.

- **Sharing**
  - *Groups*: Sometimes, users would want to work in groups and share virtual networks. One scenario where this is relevant is where students of security course have to work together as teams on course assignments/projects. V-NetLab should allow this notion of grouping users into teams. Users of the same team should be allowed to share networks i.e each of the users of a particular team should have access to the team's virtual network simultaneously. This is currently supported by allowing users to `ssh` to any of the virtual machines from their workstations.

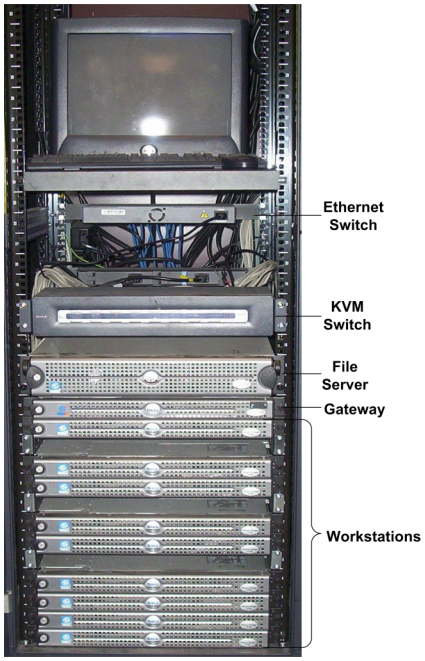


Figure 3.2: V-NetLab Hardware Platform

  - *Data Transfer*: It is possible to move data from inside virtual networks to outside and vice-versa. This is currently supported by allowing users to `sftp` from user's workstation to virtual machine that is part of their network.

- **Manageability**
  - *Easy to use*: The software should allow virtual networks to be configured and operated easily. The input specification format for virtual network topologies should allow users to define sufficiently complicated topologies with relative ease. The effort to start, shutdown virtual networks should be fairly minimal.
  - *Easy to Administer*: V-NetLab should allow an administrator to manage and monitor virtual networks, users, user groups and the usage of computing resources at any point. The framework should provide an Administrator Interface to this effect. This interface currently allows to manage tasks like adding/deleting authorized users, grouping of users into teams, add/delete teams, querying the V-NetLab to view resource allocation statistics.

- *Multiplicity*: V-NetLab should allow a single user to run multiple networks with varied topologies simultaneously, each of them isolated from each other. At the same time, it should allow different users to run identical network topologies each isolated from others.

- **Remote Access**

  For users who are on dial-up, X-startup of virtual networks and subsequent usage might be slow. Thus, there is need to support, starting of networks either in *console* (X enabled) mode or *non-console* mode. Irrespective of the start mode, users should have the option of remotely logging into running networks via SSH.

3.2 System Overview

V-NetLab consists of multiprocessor servers configured to run virtual machines. These virtual machines become the basis to run virtual networks. A virtual network is defined as a set of interconnected virtual machines. The interconnections between these virtual machines is established through a set of emulated hubs and switches. The emulated hubs and switches in our environment are referred as v-hubs and v-switches. The v-hubs and v-switches implement the virtualization of the network at the data-link layer. The benefits of implementing at data-link layer is that an IP address can be reused in different virtual networks. The re-usability of IP addresses helps to realize multiple instances of a single virtual network topology and each instance is isolated from other instances. These network configurations can be replicated across all the users in the network without any traffic interference between their corresponding virtual networks. The virtual networks emulate the exact behavior of the underlying physical network.

We currently support VMware Workstation [3] and VMPlayer [3] as virtual machines. The operating environment in the V-NetLab, as viewed by users, is illustrated in Figure 3.1.

3.2.1 Hardware Platform

The selection of a hardware platform is critical to the implementation of a V-NetLab platform. In general, two classes of hardware platforms support in implementing a virtual network. A single powerful multiprocessor appears an ideal choice for supporting V-NetLab with its wide array of commercial software to support virtualization at a management level. However, the cost of the hardware is usually very high, in addition to limitations with regard to disk space, processor speed and performance of virtual networks. These limitations do not make this class of hardware platform an ideal choice. The second class of hardware, a collection of commercial

PC's are cost effective and not constrained by space and speed limitations. However these systems have minimum or zero scalability. We have built a hardware platform consisting of network workstation-class PC's to overcome the drawbacks of the above discussed classes of hardware platforms. The solution is cost-effective and yet scalable.

V-NetLab hardware platform shown in Figure 3.2 consists of dual-processor workstation class PCs connected together by a switched gigabit network. The host OS on the system is Linux. However, the virtual machines can run a Linux, Windows or any other OS image provided it is supported by the VM software. Linux has been chosen as the host OS to enable relatively easy development of virtual network control and software management. The three main components of our hardware setup are:

- *A Single NFS Server (PC-based).* All the virtual machine disk images are stored on this server. The storage capacity of the server is 600GB realized through a 6-way RAID set up which ensures an excellent I/O performance. The storage space can be utilized to store 200 to 500 distinct virtual machine images.

- *9 Workstations (PC-based).* The PC based workstations run Linux OS with a virtual machine software from VMware Inc. [3]. Each of these workstations has 3 to 4 GB memory and 2.8 to 3.0 GHz Intel dual processor. The workstations have been partitioned into 6 "production" PC's and 3 "development" PC's. The production PC's support course related projects while the development PC's are used for continued development and testing of the V-NetLab software.

- *A gateway host.* The gateway host acts as an access and login machine for the users. Every user has to login to the gateway host inorder to access their virtual networks. The gateway host provides a management interface that enable V-NetLab access to users. The gateway host does not forward any packets and hence ensures isolation between V-NetLab and the Internet.

The testbed also includes gigabit Ethernet switches and a management console. Overall, this hardware platform can host between 130 to 300 virtual machines simultaneously. The entire setup shown in Figure 3.2.

3.2.2 User's View

The hardware setup is not totally transparent to the users. The operating system environment in the V-NetLab, as viewed by users, is illustrated in Figure 3.1. The laboratory contains a single Gateway host and is hosted in an isolated network. The gateway host acts as an access and login machine for the users. The gateway host

does not forward any packets and hence ensures isolation between V-NetLab and the Internet.

Every user has to login to the gateway host inorder to access her virtual networks. The gateway host provides a management interface that enables V-NetLab access to users. No user can directly access any of the machines in the test-bed or hardware setup. A user however, can login to the gateway (i.e V-NetLab) from any computer that runs X-Windows Server and Secure Shell (SSH) software with Internet connectivity.

The user has to login to the gateway using SSH and then start up the virtual network. VM's consoles are displayed on the user's computer. Figure 3.3 shows screen-shots of VM consoles for a network topology. Each VM console represents a virtual machine (VM). The user may have administrative or non-privileged access on these machines. The privileges are specified in the configuration file. If a user has an administrative privilege, she can move data into the virtual network from outside or vice-versa by performing a NFS mount of her home directory (on the NFS server) onto any of the virtual machines.

The details of the interface provided to the users are described below. The interface is implemented using scripts that run on the gateway machine and it provides the following functions:

- *Registration.* A user can register her network by submitting a configuration file that contains the virtual network definition. A virtual network definition contains the virtual machines and the interconnecting network devices (v-hubs and v-switches). The V-NetLab system validates the network definition by performing lexical, syntactic and semantic checks and analysis.

  - *Network Definition*

    The course staff needs to define the configuration of each virtual machine and the overall network topology apriori for any virtualization project or experiment. An excerpt of the original configuration file (of 50 lines) for the network shown in Figure 3.3 is listed below. The VM images are referenced through the `src` field in the VM definition section. Some definitions, such as hubs and switches have been abbreviated with "`...`" to conserve space.

```
vm ExtFW {
  os : LINUX
  ver : "7.3"
  src : "/mnt/qinopt/vmnetwork/vmSrc/assgn1"
  eth0 : "200.200.100.162"
  eth1 : "200.200.100.65"
  eth2 : "200.200.100.130"
```

```
}
vm Nfs {
  os : LINUX
  ver : "7.3"
  src : "/mnt/qinopt/vmnetwork/vmSrc/assgn1"
  eth0 : "200.200.100.195"
}
vm IntFW { ... }
vm DefaultGW { ... }
vm Dmz { ...  }
vm Gemini { ... }

hub hub1 {
  inf : DefaultGW.eth0, ExtFW.eth0
  subnet : "200.200.100.160"
  netmask : "255.255.255.224"
}
hub hub2 { ... }
hub hub3 { ... }

switch s1 {
          IntFW.eth1, Gemini.eth0, NFS.eth0
}
```

Each virtual machine contains an IP address, hostname and the file containing the disk image. A v-hub or a v-switch is specified in terms of the host interface or other connected v-hubs or v-switches along with a range of network addresses associated with it.

– *VM Images*

A virtual network configuration file can reference more than one VM disk image. These images store the contents for the virtual machine's hard drive and need to be prepared before starting the network. Each image must contain all the software required for a user's experiment so as to minimize the administrative efforts by a user.

Typically, VMs in a virtual network differ in many ways from each other. For instance, they will have distinct IP addresses and host names, and run different services. To overcome this problem, we typically use the exact same image (master image) for all VMs, but modify the boot scripts (which are part of this image) so that different sets of services can be started up on different VMs. Note, however, that this approach is specific to a given guest OS. In our implementation, this feature has been implemented for Linux and Windows. The modified boot scripts also

take care of routine network configuration aspects such as the set up of IP address, host name, DNS server, and default gateway, based on the contents of the virtual network configuration file.

After validation, this network is associated with the user. The registration step may not always be done by a user, administrator may preregister the network that is supposed to be used by each user.

- *Startup.* A user is permitted to start his network only if it has been previously registered. If the network is started for the first time then a mapping of virtual machines onto physical network hosts is computed. This mapping attempts to (a) minimize copying of large VM disk images (of the order of GBs), and (b) balance the load across physical machines. Optimal allocation algorithms turn out to be computationally expensive, so simple heuristics are employed in our current implementation. After computing the mappings, any VM disk images that may need copying are copied from the file server to the workstations, the VMs are started up and their console displays are tunneled to the user's desktop.

- *Shutdown.* This command shuts down the user's network. Depending upon the configuration of the virtual machines, their states (disk images) may be saved or simply discarded.

- *Query.* This command displays user's virtual network status (running, registered or unregistered) and available capacity of hardware platform (estimated) in terms of number of VMs.

3.3 System Architecture

The following are the important components of V-NetLab as seen in Figure 3.4.

- *VNetMgr*

  VNetMgr is the user interface program for users to register, de-register, start, query, remote-login and shutdown networks.

- *GrpAdmin*

  GrpAdmin is the user interface program for administrators. This interface allows administrators to add/delete users to his domain, register/de-register networks for the users under his domain, create/delete teams, add/remove users to/from teams, query usage statistics of hardware resources for running networks etc.

- *GrpMgr*

  GrpMgr is a central daemon which listens to user interface programs, process these requests and relay these requests to HostMgr. It talks back to user interface programs after hearing back from all the HostMgr's it contacted for a particular user request. There is exactly one instance of a GrpMgr for every domain. GrpMgr is the nucleus of V-NetLab and maintains information about all the virtual networks which are currently running, the current resource usage, residual computing resources available etc. It overlays virtual network topologies over physical machines which host VM's and generate all the configuration files needed by each of these hosts to setup, configure, start VM's of virtual networks and reliably relay the packets which belong to VM's of a particular virtual network.

- *Compiler*

  Its utility is two fold.

  - Verify the validity of network topology definitions given by users. This is done during registration phase. VNetMgr and GrpAdmin invoke the Compiler during network registration phase.
  - Parse the network topology file and fill up the necessary data structures which are later used by GrpMgr to generate all the necessary configuration files for successfully running an overlay virtual network. GrpMgr invokes the Compiler during the start-up phase of a virtual network.

- *HostMgr*

  HostMgr is a deamon which runs on each of the physical nodes which host VM's for V-NetLab. HostMgr's set up, configure and start virtual networks on the request of GrpMgr which in turn are initiated by GrpMgr on behalf of the user.

- *Loadable Kernel Module (Kernel Mac Translator)*

  Its the core component of the run-time system and responsible for selectively forwarding virtual network traffic that involves packet translation and restoration and as well as providing a mechanism for connecting to a virtual network from an external network through SSH.

3.4 Technical Challenges

V-NetLab enables overlay of virtual networks over a set of physical machines and this poses a lot of interesting problems to be solved. Some of them have been

addressed. The following sections talk about the challenges to be solved to realize the design goals mentioned in the Section 3.1

3.5 Network Virtualization

The problem of simulating the networking phenomena and virtualization of networking devices like hubs and switches is not straight forward as the virtualization scheme would need to address the challenges of

- Isolation of multiple networks running simultaneously on the same underlying hardware oblivious to each other's presence
- Support remote logging in of users into Virtual Machines via SSH
- Transparent and reliable transmission of packets across VM's (belonging to a network) running on different physical machines

The issues above have been addressed by our virtualization scheme, discussed in brief detail in the implementation section of Chapter 4

3.6 Resource Management

As we overlay networks, we have to solve the classic network overlay problem of efficient mapping of virtual machines on physical resources. In simpler terms, when we overlay a virtual network onto physical machines we would like to minimize the traffic induced by the overlay network on the physical network and at the same time balance load of VM's on the physical nodes. The objective is to use the current resources efficiently. Resources encompass network bandwidth of physical links, virtual network interfaces, CPU, RAM and hard drives of physical nodes. If the overlay scheme is not fair then some of the physical nodes might be overloaded when compared to other machines, This will affect the performance of virtual networks whose VM's are hosted on the affected physical machines. Since V-NetLab is a distributed system, its only fair to assume, that the overlay scheme should judiciously try to balance the load on the test-bed machines.

At the same time, if a particular network is spread across too many physical nodes, the traffic induced by the overlay due to the communication between VM's might be high. This might lead to congestion on the physical link sooner than expected. Thus the overlay scheme should ideally try to reduce the traffic induced on the physical network as much as possible.

The objective of the overlay scheme is to maximize the efficiency of usage of computing resources and hosting of virtual networks.

3.6.1 Assignment Problem

Input: A description of virtual (logical) topology along with link costs. Though link costs cannot be specified currently in the network definition file, we assume to have this information separately. We don't assume any specific traffic characteristics among virtual machines and deal only with link costs.

- A list of virtual nodes
- A list of links with link costs associated with them
- A list of switches and hubs
- A list of physical nodes with their computing capacities.

Output: A mapping of virtual nodes to physical nodes.

Optimal Assignment

Consider n physical nodes, M_i, $i = 1, n$ and let L_i, $i = 1, n$ be the load on these machines. These loads are normalized. That is, $0 < L_i < 1,\ i = 1, n$. We define L_M to be mean load on the testbed. Also, we define $CommOvHead$ as the communication overhead induced by the overlay. This is equal to the sum of link costs of all the links, which induce a cost on the physical links due to the overlay scheme. Thus, if VM's of a network are hosted on the same machine, then the contribution of the virtual link costs between them towards $CommOvHead$ is zero. An optimal assignment should minimize $CommOvHead$ and $variance$ defined as

$$variance = \Sigma_{i=1,n}(L_M - L_i)^2$$

Topology Partitioning

Graph partitioning is a difficult, long-standing computational problem. It has applications to VLSI design [12], sparse matrix-vector multiplication [6], and parallelizing scientific algorithms [5, 8]. The general way partitioning problem is described a graph, $G(V, E, W_V, W_E)$, where W_V and W_E represent vertex and edge weights respectively. output of partitioning G consists of subsets of $vertices, V_1, V_2, ..., V_K$, where $V_i \bigcap V_j = \phi$ for $j \neq i$ and $\bigcup_i V_i = V$. The goal is to balance the sum of vertex weights for each vertex V_i , and minimize the sum of edge weights whose incident vertices belong to different partitions. Though this problem is NP-hard [7], heuristics yield good partitions in practice [10, 11, 13].

Simplified Assignment Problem

Consider a graph $G'(V', E', V'_W, E'_W)$, where

- V' is the set of virtual machines.
- V'_W is the set of loads induced by corresponding virtual machines in V' on the physical nodes if they are hosted. This set represents the vertex weights.
- E' is the set of edges among virtual machines. There wont be an edge between two VM's if they are connected to the same hub or a switch. The link cost associated with the edge becomes its weight.
- E'_W is set of edge weights.

If we consider, a special case of our assignment problem where $L_i = l$ for $i = 1, k$, then, the general graph partitioning problem on G' is equivalent to finding out an optimal solution for the simplified assignment case. This equivalence proves that our Assignment problem is hard. So, we propose various heuristics to solve the problem at hand.

Offline Assignment Problem

In the offline assignment problem, we know before hand all network topologies we would like to overlay. We consider two cases here

- We never allow splitting of networks (overlay of a single network topology on more than one physical machine) and try to assign them to physical hosts. If at any point there is a network which cannot be accommodated onto a physical node, then we deem it as failure and abort.

  If $W = \{W_i | i = 1, k\}$ are network capacities, $P = \{P_j | j = 1, n\}$ are the physical nodes with $C = \{C_j | j = 1, n\}$ as their respective physical capacities, we can obtain a reasonable assignment scheme by allocating elements in set W to each of $P_j \in P$ such that the $\Sigma_{i \in allocated\_set} W_i$ is maximized but is less than $C_j \in C$. We are interested in allocations where the utilization of physical node is maximum. We accecpt this allocation into our solution set iterate recursively over the remaining networks and nodes till we allocate all the networks. The best way of allocating elements of W to P_j such that the utilization of P_j is maximum can be solved using the 0-1 knapsack problem.

- If we allow splitting of networks, then arbitrary splitting scheme becomes sufficiently hard as we will be solving many instances of the NP hard allocation problem. So, we look at some heuristics, mostly greedy to arrive at solutions.

- *Largest network with largest available capacity*

  We try to match the largest available network with largest available capacity any point and try to make an assignment. In case, there is no such physical node then we split the network and consider the connected components for allocation. We iterate over all the input networks set in this manner. In case, one needs to split the network then we suggest a scheme to split the networks (see below).

- *Largest network with large enough available capacity*

  We try to match the largest available network with a machine whose capacity is large enough (not necessarily largest) but is smallest among all the available hosts which can host the network of interest without splitting. In case there is no large enough network then we split the network and allocate the connected components. We iterate over all the input networks set in this fashion.

- *Maximization of load of physical machines*

  In this scheme, we start with the machine with maximum capacity and fill it up starting from the largest network and subsequently all the other networks, driving up the utilization of the physical node close to 100% before we move to the next largest physical node.

In all the above approaches, in case, we need to split the network then we split into connected components whose number is proportional to the switches/hubs. Each hub gets the machines to which it is connected to. Cases where a single machine is connected to multiple hubs, the machine belongs to the hub with maximum degree. If, we have an upper bound on the number of pieces into which a network be split, then we can reduce the number of connected components by grouping them together based on some preferred logic. For example, grouping them into similar sized groups.

Online Assignment Problem

In the Online case we wont know the networks we might have to allocate in the future. So, doing a best allocation every time need not necessary lead to a optimal allocation after n allocations. The heuristics mentioned in the offline along with other new heuristics might be good to test and simulate.

Ultimately, all the approaches mentioned above should be simulated to choose how they perform for different types of inputs.

3.7 Virtual Image Management

The size of VMware images are typically greater than 700 MB. Currently, as we create virtual networks, a image is created for each VM from a source VM image. Thus, if a virtual network has 25 machines, then 25 separate copies of such VM images have to copied from NFS and maintained. Also, if virtual machine images have to be moved due to changes in the overlay scheme, then it gets quite cumbersome and is detrimental to both efficiency and usage of V-NetLab. We propose a method to tackle this problem. This is currently not integrated into V-NetLab but has been developed and tested independently.

A *vmdk* file forms the primary disk image for a VMware Workstation. As we run virtual networks, these *vmdk* images get modified. This *vmdk* image, say *new.vmdk*, is first copied from a master *vmdk* image, say *orig.vmdk*. Depending on the amount of modifications, the difference between the two files will vary.

Observation: For typical usage scenarios the amount of modifications are only a small fraction of the size of the original file. So, it makes sense to create the difference of them (*part.vmdk*), and send it instead of moving around large *new.vmdk* file.

Mechanism:

- A hashfile is created corresponding to *org.vmdk* file. Md5 hash is computed against each block (e.g. 4K, can be customizable) of the file, and it is dumped into a file (*org.vmdk.hash*) that accompanies the *org.vmdk* file, this step can be done before hand.
- Similarly, a hash for each block of *new.vmdk* is computed and is looked up in *org.vmdk.hash*.
  - If the lookup fails then then the block is written into the *part.vmdk* in the same location as that in the *new.vmdk*. It should be noted that *part.vmdk* file created as a sparse file,
  - If the lookup is successful that means the current block in *new.vmdk* and the block corresponding to lookup entry in *org.vmdk.hash* are identical. A map file, *diff.map* maintains a mapping of this information. *part.vmdk* is untouched in this case.

  From the above logic, it is obvious that *new.vmdk* can be created given *old.vmdk*, *diff.map* and *part.vmdk*.

Patching: *org.vmdk* is s present on each physical host involved in test-bed. If *part.vmdk*, *diff.map* are available, then patching can be done in two ways:

- **Static Patching**

  Each record in map file is read and the corresponding block from *org.vmdk* is picked and written into *part.vmdk* at the block location specified mapfile. After processing all the records, *part.vmdk* will be the identical to *new.vmdk*. Static patching takes a fraction of the vmdk copying time, and it can be slow.

- **Dynamic Patching**

  This is to minimize the *startup* time involved in static patching approach. A PRE_LOADED library is prepared for intercepting the vmware process, and for each read, a lookup for the map file is incurred to check whether the portion needs to be picked up from *org.vmdk* or *part.vmdk*, and the correct portion is read from them accordingly and return to vmware process.

  When a write is occurs, its more involved. It leads to a look up, first from the map file, if the write portion only involves *part.vmdk*, then write it in *part.vmdk* directly, if not, migrate the blocks of interest from *org.vmdk* to *part.vmdk*, then perform the write to *part.vmdk*. The map file needs to be updated accordingly to indicate the blocks in question can now be directly picked up in *part.vmdk*.

3.8 Usability

Centralized and efficient administrative management of V-NetLab can be daunting, if there are no proper management tools. Currently,

- Specifying arbitrary topology specifications is relatively simple through the rich network specification language.
- VNetMgr already provides functionality for users to register/start/shutdown/query and remote access to virtual machines.
- GrpAdmin provides the following functionality for administrators
  - *register:*
    - Register a network for a user/all users in the group.
    - Register a network for a team/all teams in the group.
    - Register a network for a set of users (input from a file).
  - *query:*
    - query the GrpMgr for the virtual networks its running currently.

    - query the usage statistics of a particular user.
    - query for usage of specified resources (vmnets currently being used, host(s) on which the virtual network is mapped to) by a particular user network.
    - query for usage of resources on a particular physical host/All physical hosts.
- *teams:*
    - Create/Delete a team.
    - Add/Delete members to/from a team.
- *deregister:* Deregister networks for a team/user/all users for a group.

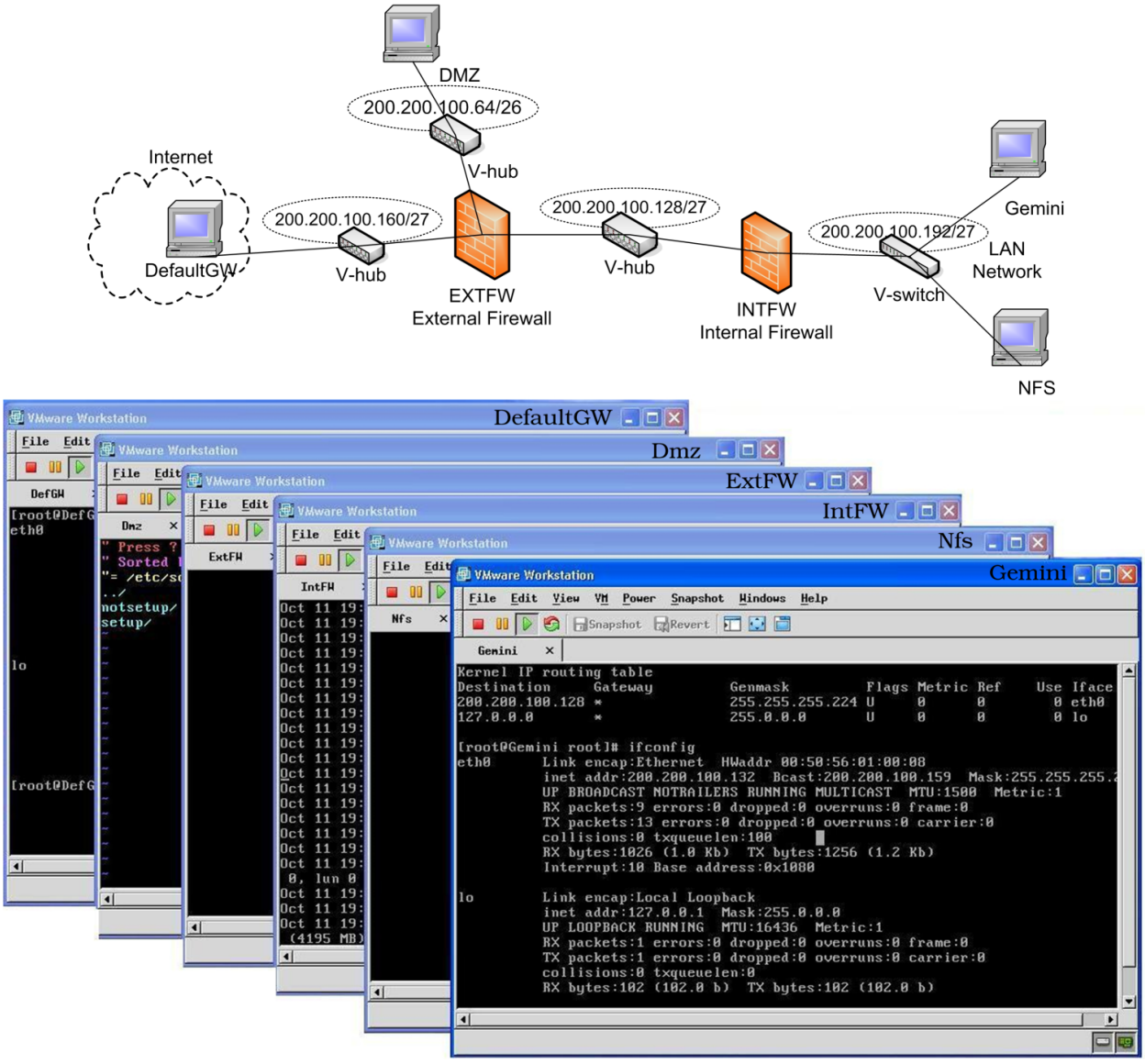


Figure 3.3: Sample Virtual Network Topology and its Screen Shot

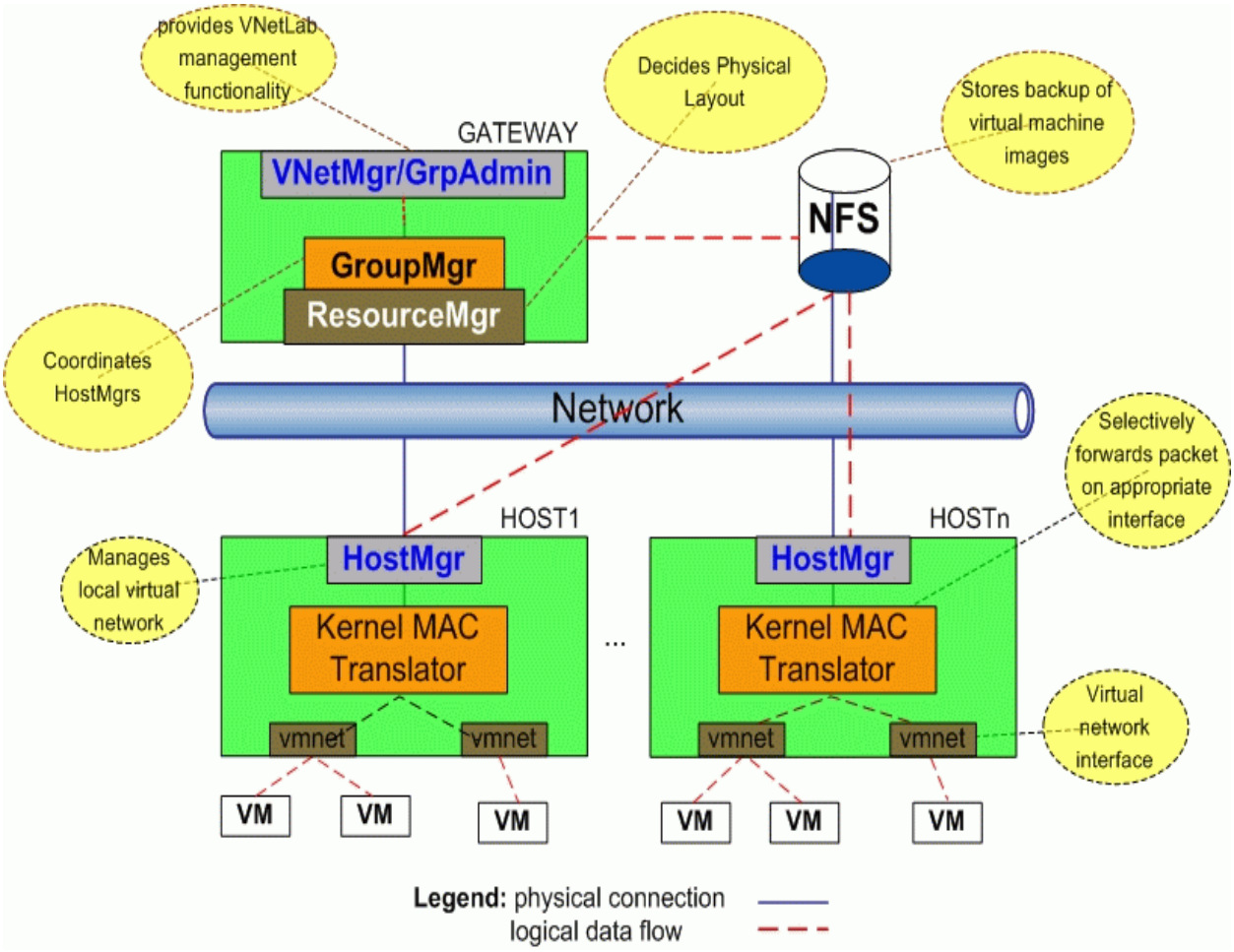


Figure 3.4: V-NetLab Architecture

Chapter 4

Implementation

This section talks about implementation details.

Most of the implementation has been done in C++. Compiler was written using Lex/Yacc and it generates a C++ compiler. Kernel module is approximately 3000 lines of code (LOC) in C. No parts of the Kernel module were directly dealt with during the work for this thesis. It was mostly used as a black-box and modified only to fix bugs while building the V-NetLab framework. The entire V-NetLab framework was designed and implemented from ground-up and forms a major part of the effort towards thesis. This included building the following major components.

- Compiler
- VNetMgr
- GrpAdmin
- GrpMgr
- HostMgr

V-NetLab framework is currently around 18000 LOC in C++/C. There are also various shell scripts which are used at various stages of execution of V-NetLab.

4.1 Implementation

4.2 Compiler

We have developed a simple language that allows one to define ethernet based networks by writing a *topology file*. A Network is specified in terms of virtual machines

and interconnecting devices such as hub and switches. Compiler takes as input the user-defined topology specified in the language. It checks whether the given topology is valid (and populates a set of internal data structures). Following is a brief description of the language and format of the topology file:

- **Virtual Machine Definition**

  User must begin by deciding the number of virtual machines the in the virtual network. She must then decide the number of network interfaces in each machine and their IP addresses, operating system type etc. Each virtual machine in the network topology has an entry in the topology file with the following properties

  - Hostname
  - Operating system and its version
  - Location of the virtual disk image for that virtual machine.
  - Interface IP addresses

  eg.

```
vm Machine1 {
  os : LINUX
  ver : "9.0"
  src : "/mnt/qinopt/vmnetwork/vmSrc/linux9"
  eth0 : "192.168.2.2"
  eth1 : "192.168.3.2"
}
```

  The above entry specifies that Machine1 has 2 interfaces having IP addresses as mentioned and the default gateway is 192.168.2.2 on eth0. Thus when the virtual machine comes up, it is already assigned these IP address and router information. This virtual machine is to run Linux (Red Hat 9.0) as the operating system and the location of the appropriate virtual disk image is "/mnt/qinopt/vmnetwork/vmSrc/linux9".

- **Connecting components(Hubs/Switches) definition**

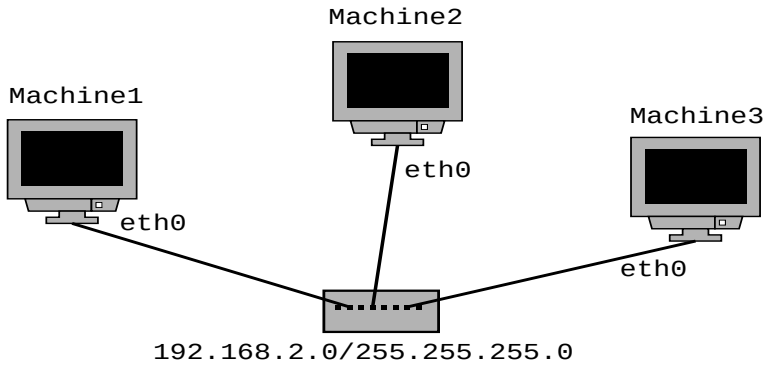
  After all the virtual machines have their definition in the topology file, we need to define the connecting components - *hubs* and/or *switches* by specifying,

  - Machine names and their respective interfaces connected by the hub/switch

- Subnet address of the network to which these interfaces belong
- Netmask of the network to which these interfaces belong
- Domain name of the network

eg.

To connect virtual machines by a hub

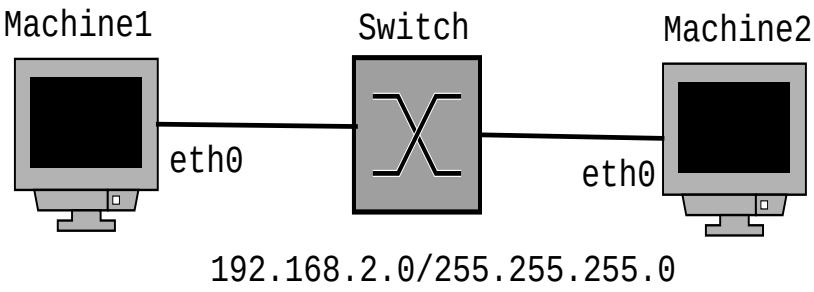


Hub

```
hub hub1 {
  inf : Machine1.eth0, Machine2.eth0, Machine3.eth0
  subnet : "192.168.2.0"
  netmask : "255.255.255.0"
}
```

This entry specifies that Machine1's, Machine2's and Machine3's eth0 interfaces are connected to a hub, and the network they belong to is "192.168.2.0/255.255.255.0", with domain name of seclab.cs.sunysb.edu and dns server IP "192.168.2.2" .

To connect virtual machines by a switch



Switch

```
      hub hub1 {
inf : Machine1.eth0
subnet : "192.168.2.0"
netmask : "255.255.255.0"
      }

      hub hub2 {
inf : Machine2.eth0
subnet : "192.168.2.0"
netmask : "255.255.255.0"
      }

      switch s1 {
hub1, hub2
      }
```

This entry specifies that Machine1's and Machine2's eth0 interfaces are connected together by a switch and the network they belong to is "192.168.2.0/255.255.255.0".

4.3 GrpMgr

GrpMgr provides a centralized control over the virtual network. It is responsible for responding to VNetMgr and, GrpAdmin requests by communicating with HostMgr's on users behalf, maintain current state of the system and serve requests. It handles start, shutdown and query command for all virtual networks.

4.3.1 Network Virtualization Configuration Information

Based on the information generated by the compiler, GrpMgr first decides the physical distribution of the virtual machines over physical machines. In doing that it

takes into consideration factors such as current load on the physical machine, location of the virtual machines in the previous run, minimization of network traffic on the physical LAN, etc. Physical distribution step is carried out each time the virtual network is started, thus it is dynamic. After deciding the placement of the virtual machines, various layout files are written that represent the network topology and the physical distribution in order to simulate hubs, switches, etc. The engine also creates migration scripts for copying of VM images from NFS to the local disks of the involved physical machines.

4.4 HostMgr

Each involved physical host machine has a HostMgr daemon running on it. HostMgr takes commands from GrpMgr and performs appropriate functions. eg. When it receives start command, it copies the VM disk images, layout files and other configuration files from NFS to local disk using the migration scripts generated by the virtualization engine. It then loads the layout files by communicating with the LKM using netlink sockets. It then brings up virtual machines. HostMgr also consists of a local CacheManager. When the VMClient wishes to start VMs belonging to a virtual network it first checks if the corresponding disk image is there in local cache. If it is, then the image of the VM in the cache is used to start-up the VMs; if it is not, then the master copy is restored for the latest VM image before bringing up the virtual machines.

4.5 Loadable Kernel Module (LKM): A Host-Only solution

The layout files created by the virtualization engine are populated as kernel data structures (hash tables) by the LKM through netlink sockets. LKM performs packet translation and restoration function to ensure that all the virtual machines that are supposed to receive network packets belonging to some conversation, are able to do so, even if they are distributed over different physical machines, thus transparently transmitting and receiving packets. To decide which virtual machines or physical machines should receive the packet, it looks up the populated hash tables. Using this LKM feature we emulate the entire network topology. The details of LKM is followed in the next section.

4.6 Networking

4.6.1 Technical Background

The Ethernet card is hard-wired with a particular link layer (or MAC) address and is always listening for packets on its interface. When it sees a packet whose MAC address matches either its own address or the link layer broadcast address (i.e., FF:FF:FF:FF:FF:FF for Ethernet) it starts reading it into memory. Upon completion of packet reception, the network card generates an interrupt request. The interrupt service routine that handles the request is the card driver itself, which runs with interrupts disabled and typically performs the following operations:

- Allocates a new sk_buff structure which represents the kernel's view of a packet.
- Fetches packet data from the card buffer into the freshly allocated sk_buff, possibly using DMA.
- Invokes netif_rx(), the generic network reception handler.
- When netif_rx() returns, re-enables interrupts and terminates the service routine.

The netif_rx() function prepares the kernel for the next reception step. It puts the sk_buff into the incoming packets queue for the current CPU and marks the NET_RX softirq (softirq is explained below) for execution via the cpu raise softirq() call. This processing is resumed by a call to the net_rx_action() function. It dequeues the first packet (sk_buff) from the current CPU's queue and runs through the two lists of packet handlers, calling the relevant processing functions. The two lists are called ptype_all and ptype_base and contain, respectively, protocol handlers for generic packets and for specific packet types. Protocols which wish to receive all incoming packets are linked into a list pointed to by ptype_all. These protocols register the have type ETH_P_ALL and are processed before considering the protocols that consume only a specific packet type. Protocol handlers register themselves, either at kernel startup time or when a particular socket type is created, declaring which protocol type they can handle; the involved function is dev_add_pack() in netcoredev.c, which adds a packet type structure (see includelinuxnetdevice.h) containing a pointer to the function that will be called when a packet of that type is received. Upon registration, each handler's structure is either put in the ptype_all list (for the ETH_P_ALL type) or hashed into the ptype_base list (for other ETH_P_* types).So, what the NET_RX softirq does is call in sequence each protocol handler function registered to handle the packet's protocol type. Generic handlers (that is, ptype_all protocols) are called first, regardless of the packet's protocol; specific handlers follow.

4.6.2 Virtualization at Datalink Layer

Adoption of Host-Only VMware networking

Since host only mode provides communication between VMware and the host OS, we can insert a module in the host OS which can process packets coming from virtual machines (VM's) through virtual interfaces to achieve virtualization of network components. Also, this mode does not put physical host's interface in promiscuous mode unlike Bridged mode, which hinders performance. Thus Host-only mode is the best choice for creating virtual networks using VMware Workstations. Also, Bridged mode automatically extends the virtual machine network interface onto the LAN of physical machine on which it is hosted. This might be in direct violation with our *Isolation* principle. NAT mode is not feasible either because, we would like to support arbitrary user network topologies with specific IP and network addresses.

A virtual network typically consists of a number of VM's distributed over a set of physical hosts. VM's belonging to the same subnet (eg. connected to the same *Hub*) in the logical topology might be distributed over multiple physical hosts. In that case we need a mechanism for transparent and lossless transmission of packets among virtual hosts via physical Hosts. This calls for the need of a packet processing module within each physical host's kernel. In other words, we need a bridge to connect *vmnet* interfaces with the physical interface.

To achieve virtualisation of network components – Hub and Switch. we perform two tasks:

1. Introduce a packet processing logic that decides where to forward or on which interface to send the packet to, based upon the network topology (logical topology) specified by the user.

2. Transparently transmit and receive packets on the physical wire and make sure that each physical host receives virtual network packets, and are handed over to the packet processing logic in the physical Host OS.

Concept of Sibling Closure

Virtual Hosts' interfaces that are connected together by one or more hubs without any switch or bridge involved, form a Sibling closure or SC. A virtual hosts' interface in an SC should be able to listen to conversation involving any of the virtual host interface in the SC. The Logical topology provided by the user can be divided into SC's. Each SC can be given a unique Id. Each SC can have a number of virtual hosts interface, each of which is numbered uniquely within that SC.

MAC-based Identification

A MAC address is a six-byte number. Each network adapter manufacturer gets a unique three-byte prefix called an OUI - Organizationally Unique Identifier - that it can use to generate unique MAC addresses. VMWare has two OUIs - one for automatically generated MAC addresses and one for manually set addresses.The VMWare OUI for automatically generated MAC addresses is 00:0C:29. Thus the first three bytes of the MAC address that is automatically generated for each virtual network adapter have this value. The MAC addresses can be assigned manually by system administrators. VMWare uses a different OUI for manually generated addresses: 00:50:56.

Thus last 3 bytes of the MAC addresses of the interfaces on the virtual hosts can be used to uniquely identify them in the logical topology. SCID is made part of 6 byte MAC address. Thus Mac address is of the form 00:50:56:01:02:03
where, first 3 bytes are vmware OUI prefix,
4th byte is network id, that is unique for all the networks,
5th byte is SCID,
6th byte is derived from a counter.

Participation Table

Another important data structure introduced is **ParticipationTable** or **PT**. This is generated by GrpMgr during the start-up of the network after validating the user's network topology. By definition, each virtual host's interface in a Sibling closure is able to listen to conversations involving any other virtual host's interface of that SC. Virtual Host interfaces belonging to a SC can map to more than one Host Virtual Adapter (vmnet device) that may reside on a single or multiple physical hosts. Participation table describes the group of Host Virtual Adapters that are associated with a SC. So a packet generated in a SC will need to be forwarded on all the Host Virtual Adapters which are participating in that SC. This table is used in making forwarding decision. The format of each entry in Participation Table is as follows:
⟨Src SCID⟩ ⟨Dst SCID⟩ ⟨no. of local vmnetx interface⟩ ⟨list of local vmnetx inteferface⟩ ⟨no. of remote macs⟩ ⟨list of remote macs⟩

where, Src SCID is source SCID,
Dst SCID is destination SCID,
third field specifies the number of local interfaces,
fourth field lists the local interfaces,
fifth field specifies the number of remote hosts,
sixth field lists the list of remote mac addresses.

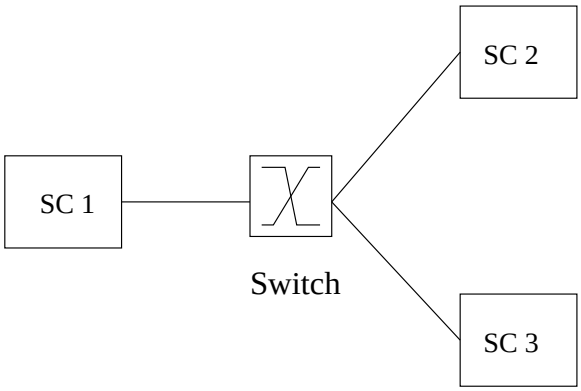


Figure 4.1: Three SCs connected by a Switch

1 0	3 1 2 3 0
1 1	1 1 0
2 0	3 1 2 3 0
2 2	1 2 0
3 0	3 1 2 3 0
3 3	1 3 0

Table 4.1: Partcipation Table for Figure 4.1

Each entry specifies the lists of local vmnet interfaces and list of remote hosts where a virtual network packet needs to be forwarded so as to emulate the virtualized ethernet environment. For instance, if a SC is mapped to vmnet device number 2 and 3 (vmnet2 and vmnet3) then any packet from this SC should be visible on both vmnet2 and vmnet3 devices. We further take up a case where SCs are interconnected by a switch.

Figure 4.1 shows three SCs conected by a switch, their ids are 1, 2 and 3. Assuming that SC1 has been mapped to vmnet1, SC2 to vmnet2 and SC3 to vmnet3. The participation table for it is shown in Table 4.1. The entry where destination SC id is 0 refers to a broadcast entry. It means that if a broadcast packet is generated in any of the SCs then according to switch semantics it should be visible in all three SCs. First entry of the table says that a broadcast packet generated in SC1, should be visible on three local vmnet interfaces – vmnet1, vmnet2 and vmnet3. Second entry depicts that any non-broadcast packet generated in SC 1 needs to be visible only on device vmnet1. In both the cases, its not required that packets be forwarded on remote interfaces as the last field has an entry 0. The rest of the entries are similar to first two entries.

Transparent transmission and reception of packets

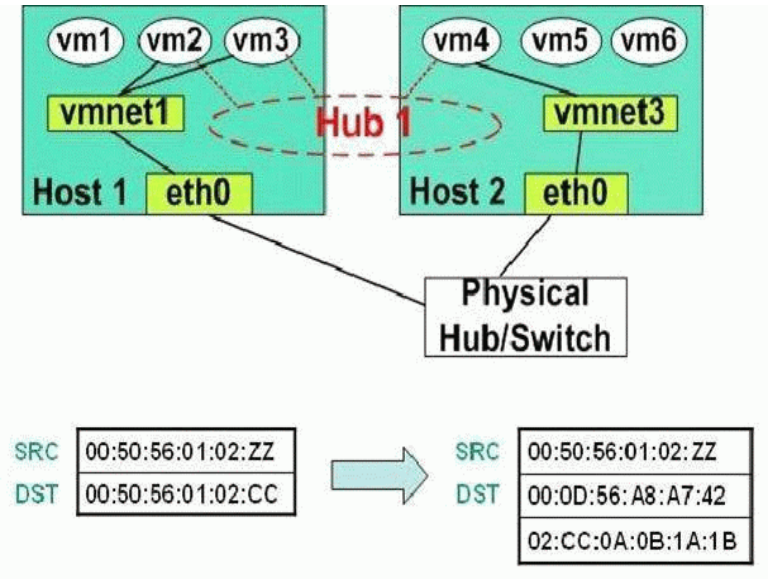


Figure 4.2: Network Virtualization in V-NetLab

Virtual network packets that need to be forwarded on a remote *vmnet* interface has to be transmitted on physical wire through host physical interface (eth0). To transparently transmit packets generated by a VMware on to the physical wire, we need a mechanism to grab each and every packet arriving at *vmnet* (virtual interface) and forwarding it on to physical host's interface (eth0) as per Participation Table. Thus the packet will then be visible to all the physical hosts on the network. Also, the packet needs to be captured at eth0 interface of the receiving physical host and transferred to the packet processing logic. To make sure that the packet is not dropped at the data link layer of the receiving end, the packet's destination MAC address needs to be changed to receiving host MAC address (according to entry in participation table). Also to make this modification transparent for the receiving VMware, we will need to restore the original packet at the receiving end. Thus we need to encode the lost information in the new packet i.e the orginal destination MAC address. This will require to send some meta information with each packet so that destination mac address can be reconstructed after it has been received at the other end. Therefore, every ethernet packet contains meta information and authentication information at its tail so as to allow the reconstruction of actual destination mac address. Altogether, 6 bytes of data are appended at the tail, out of which first 2 bytes are meta information and remaining 4 bytes serves the purpose

of signature. In this way the receiving physical host retrieves the last 6 bytes from a ethernet packet. First, it does the authentication by comparing the last 4 bytes with the signature data. After authentication, destination mac address is reconstructed with the help of first 2 bytes in the following way. This approach is devoid of source mac address spoofing and works with smart ethernet switches that are able to figure out source mac address spoofing.

Thus, original source and destination mac will be of the form

```
00:50:56:01:02:ZZ (SourceMAC)
00:50:56:01:02:CC (DestMac)
```

The first 3 bytes are unique to VMware (vmware OUI) and remaining three bytes encode following information:

- virtual network id: 4th byte
- SC id: 5th byte
- unique counter value: 6th byte

Now when a virtual network packet is transferred on physical wire then destination mac address is changed to as per PT table. The source, destination MAC address and 6 byte trailer at end are as follows:

```
00:50:56:01:02:ZZ (source mac address is not mangled)
00:0D:56:A8:A7:4A (destination host mac address, obtained from PT table)
02:CC:0A:0B:1A:1B (6 byte at tail of ethernet frame)
```

As shown above, the last 2 bytes of original destination mac address is appended at the tail followed by 4 bytes of authentication data. The first 3 bytes of virtual machine mac address is vmware OUI and last 4 bytes of authentication data at the tail helps identify virtual network packets at receiving host. The destination mac address is restored by taking 4 bytes prefix from source mac address and appending to first 2 bytes of the 6 byte trailer at the end of the packet. In case if the packet is a broadcast packet then its destination mac address is restored to "FF:FF:FF:FF:FF:FF". For a broadcast packet the first 2 bytes of the 6 byte trailer is "00:00". Please refer to Figure 4.2.

Chapter 5

Summary

5.1 Related Work

There was work done earlier for providing administrative access to physical machines in a safe and secure way. Jean *et al.* built an advanced systems laboratory which allowed root access to machines. Though users had root access to lab machines they couldn't jeopardize the security of the network. Their approach was to store the disk images on NFS, copy them on demand and boot for users to login.This was no more novel when adoption of VM's had become mainstream. Also, one couldn't have access to a network of machines.

Planetlab [2] is distributed networking laboratory which allows users to choose an overlay scheme of this distributed set of nodes which run identical software. Emulab [1] is a time and space-shared network emulator which achieves new levels of ease of use. It allows users to specify a network topology with user defined packet loss characteristics, latency and band-width characteristics. It supports virtualization based on FreeBSD Jails. This mode of usage allows a single type of OS to be hosted where as in our case we can run different OSes on the same physical machine. Though dedicating nodes of specific OSes solves the problem, it still doesn't allow multiple' OSes to share single machine like in V-NetLab.

In contrast to these approaches, our work provides an approach that combines flexibility and versatility with low hardware and administration/management costs. VNET [17] tries to solve the grid-computing problem, by extending the LAN to include VM's. They use VMware Workstation as VM's and network virtualization is achieved by using bridged mode networking of VMware. VNET tunnels ethernet packets over TCP/IP to achieve VM visibility as a node on the local LAN. VIOLIN [9] uses an application-level virtual network architecture built on top of an overlay infrastructure such as Planetlab. They use UDP tunneling in the Internet domain to emulate the physical layer in the VIOLIN domain. Though they are more

similar than others (as they use VM's to realize virtual networking) they are clearly less efficient than V-NetLab since we achieve network virtualization at data-link layer.

Network assignment problem has been studied previously by many researchers. Anderson *et al.* [4] discusses theoretical approaches and general techniques to solve a class of node assignment problems. Robert *et al.* [15] define a general network test-bed mapping problem and present a general purpose simulated annealing approach to solve the problem. Ken *et al.* [18] also discuss about methods to create partitions based on expected communication across the network topology. They use METIS [10] to perform fast and efficient topology partitioning. Ananth *et al.* [17] have a formulation of the problem of mapping VM assignments to physical nodes based on bandwidth, latency and computing rate as primary characteristics.

5.2 Conclusion and Future Work

V-NetLab provides a framework to conduct security experiments by taking full advantage of virtualization technologies. It effectively addresses the problems faced in the physical world, in order to provide networks for security experiments. It allows users to create arbitrary network topologies with minimal effort by providing a rich specification language and solves practical problems like cost escalation when we try to support bigger virtual networks. V-NetLab provides complete isolation of networks from other virtual networks and also the physical network on which these virtual networks run, thereby solving the security problem which arises due to administrative access of users. There are still some challenges to be solved in the areas of resource allocation and virtual image management in order to make V-NetLab more reliable and efficient.

Currently, only VMware Workstation and Player are being supported as VM's. Looking at other virtualization technologies will be good direction for experimentation and hopefully improvement to the current framework.

Bibliography

[1] Emulab. http://www.emulab.net/.

[2] Planetlab. http://www.planet-lab.org/.

[3] Vmware. http://www.vmware.com/.

[4] David G. Anderson. Theoretical approaches to node assignment. *Unpublished Manuscript. http://nms.lcs.mit.edu/papers/anderson-assign.ps*, December, 2002.

[5] A. Boukerche and C. Tropper. A static partitioning and mapping algorithm for conservative parallel simulations. In *In Proc. of the Eighth Workshop on Parallel and Distributed Simulation*. ACM Press, 1994.

[6] Umit V. Catalyurek and Cevdet Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):673–693, 1999.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[8] J. R. Gilbert and E. Zmijewski. A parallel graph partitioning algorithm for a message-passing multiprocessor. In *Proceedings of the 1st International Conference on Supercomputing*, pages 498–513, New York, NY, USA, 1988. Springer-Verlag New York, Inc.

[9] Xuxian Jiang and Dongyan Xu. Violin: Virtual internetworking on overlay infrastructure. In *International Symposium on Parallel and Distributed Processing and Applications (ISPA) 2004*, 2004.

[10] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[11] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Sys. Tech. J.*, 49(2):291–308, 1970.

[12] T. Leighton and S. Tragoudas F. Makedon. Approximation algorithms for vlsi partition problems. In *In Proc. of IEEE International Symposium on Circuits and Systems*, pages 2865–2878, 1990.

[13] Joseph W. H. Liu. A graph partitioning algorithm by node separators. *ACM Trans. Math. Softw.*, 15(3):198–219, 1989.

[14] Susanta Nanda. A survey on virtualization technologies. *RPE Report. http://www.ecsl.cs.sunysb.edu/tr/TR179.pdf*, February, 2005.

[15] Robert Ricci, Chris Alfeld, and Jay Lepreau. A solver for the network testbed mapping problem. *SIGCOMM Comput. Commun. Rev.*, 33(2):65–81, 2003.

[16] Jeremy Sugerman, Ganesh Venkitachalam, and Beng-Hong Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, USA, 2001. USENIX Association.

[17] Ananth I. Sundaraj and Peter A. Dinda. Towards virtual networks for virtual machine grid computing. In *USENIX-VM '04: 3rd Virtual Machine Research and Technology Symposium*, pages 177–190, 2004.

[18] K. Yocum, E. Eade, J. Degesys, D. Becker, J. Chase, and A. Vahdat. Toward scaling network emulation using topology partitioning. In *Proc. of MASCOTS*, 2003.

Appendix A

Grammar Definition

A.1 Host Definition in CFG

host_list → *host*
  | *host_list host*
host → **host** : *id* { *host_def* }
host_def → *cpu_def mem_def disk_def root_dir ip_def*
cpu_def → **cpu** : *floatnum*
mem_def → **mem** : *intnum*
disk_def → **disk** : *floatnum*
root_dir → **root** : *string*
ip_def → **ip** : *string*

A.2 User Definition in CFG

user_def → *user_def_wo_par_sol*
  | *user_def_wo_par_sol par_sol*
user_def_wo_par_sol → *vm_list hub_list*
  | *vm_list hub_list switch_list*
vm_list → *vm*
  | *vm_list vm*
vm → **vm** *id* { *vm_def* }
vm_def → *os_def ver_def src_def eths*
os_def → **os** : **LINUX**
  | **os** : **WIN**
  | **os** : **BSD**
ver_def → **ver** : *string*

$src\_def \rightarrow \textbf{src} : string$
$eths \rightarrow ethIP$
$\quad\quad\quad\quad\quad\quad | \; eths \; ethIP$
$ethIP \rightarrow eth\_def{:}string$
$eth\_def \rightarrow ETH$
$hub\_list \rightarrow hub\_def$
$\quad\quad\quad\quad\quad\quad | \; hub\_list \; hub\_def$
$hub\_def \rightarrow \textbf{hub} \; id \; \{ \; hub\_decl \; \}$
$hub\_decl \rightarrow if\_list \; subnet\_def \; netmask\_def$
$if\_list \rightarrow \textbf{inf} : peer\_if$
$\quad\quad\quad\quad\quad\quad | \; if\_list \; , \; peer\_if$
$peer\_if \rightarrow id.eth\_def$
$subnet\_def \rightarrow \textbf{subnet} : string$
$netmask\_def \rightarrow \textbf{netmask} : string$
$switch\_def \rightarrow \textbf{switch} \; id \; \{ \; switch\_entry \; \}$
$switch\_entry \rightarrow id$
$\quad\quad\quad\quad\quad\quad | \; inflist\_el$
$\quad\quad\quad\quad\quad\quad | \; switch\_entry{,}id$
$\quad\quad\quad\quad\quad\quad | \; switch\_entry \; , \; inflist\_el$
$inflist\_el \rightarrow id.eth\_def$
$par\_sol \rightarrow \textbf{partialSolution} \; par\_sol\_defn$
$par\_sol\_defn \rightarrow par\_sol\_pair$
$\quad\quad\quad\quad\quad\quad | \; par\_sol\_defn \; par\_sol\_pair$
$par\_sol\_pair \rightarrow (id.eth\_def \; id.VINF)$
$floatnum \rightarrow floatnumber$
$intnum \rightarrow integer$
$id \rightarrow [A - Za - z0 - 9.]+$
$string \rightarrow "id"$
$ETH \rightarrow \textbf{eth}[0 - 9]$
$VINF \rightarrow \textbf{vinf}[0 - 9]+$