PLANETLAB

Isolation of Shared Network Resources in XenoServers

Andrew Warfield
Cambridge University

Steve Hand
Cambridge University

Timothy Harris
Cambridge University

Ian Pratt
Cambridge University

PDN-02-006
November 2002

Status: Final.

Isolation of Shared Network Resources in XenoServers

Andrew Warfield, Steve Hand, Timothy L. Harris, Ian Pratt
Computer Laboratory, University of Cambridge

1 Introduction

This document presents some issues involved in virtualizing network resources so that they may be shared across a set of isolated virtual machines (VMs). After discussing the issues in design, general details of *Xen*, the XenoServers hypervisor[1] are presented as a specific implementation example. We hope that this presentation will encourage a discussion regarding the best approach to these issues with other researchers involved with similar projects, in particular, designers of isolation architectures for PlanetLab.

The contributions that may be most relevant to efforts to establish a general interface description for network resource isolation are as follows:

1. The presentation of the hypervisor's network system as being a virtualization of a local area network.
2. The explicit use of a packet classifier with the hypervisor that may be configured to appropriately manage traffic across virtual hosts.
3. Efforts to move closer to a description of functionalities that may exist below the virtual network device and the interface to those services being an extended API available to guest VMs.

Throughout the discussion of this design it is important to consider the major trade-offs involved. Primary among these is the balance between the utility provided to virtual machines and the performance overhead imposed on them. Parallel to this performance overhead, and perhaps more important is the complexity imposed on the hypervisor by any additional functionality. As a major design goal of the hypervisor is reliability through simplicity, it seems prudent to give each additional feature careful consideration.

[1]A note on terminology: hypervisors are also described as Virtual Machine Managers (VMM) in other literature

2 Overall System Architecture

Figure 1 presents a generic hypervisor/virtual machine architecture. The hypervisor layer serves to virtualize resources and multiplex access from a set of overlying virtual machines. Within the single host, there are now two levels of interface to a given resource: at the bottom level is the raw physical interface between the hypervisor and the device, and above this is the virtual interface that is presented to the virtual machines.

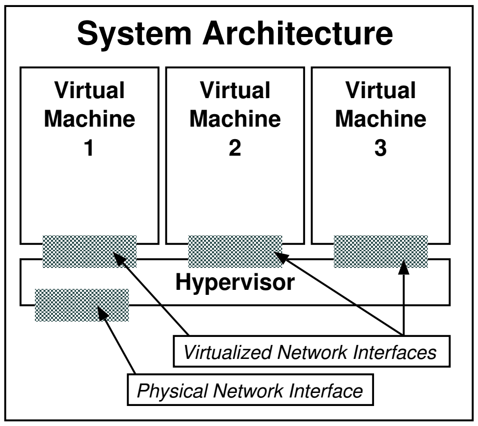

Figure 1: Network Interfaces in a XenoServer

In considering the virtual network interfaces that are provided to a set of operating system instances, there are many properties that may be desirable. As the hypervisor is multiplexing network resources, the network subsystem may be best understood as being a virtual network switching element. A simple hypervisor implementation might act as a link-layer hub, forwarding all inbound traffic to all virtual machines and multiplex outbound traffic to the network. Alternatively, the hypervisor may act as a switch or router, servicing each VM's traffic differently and possibly providing additional services.

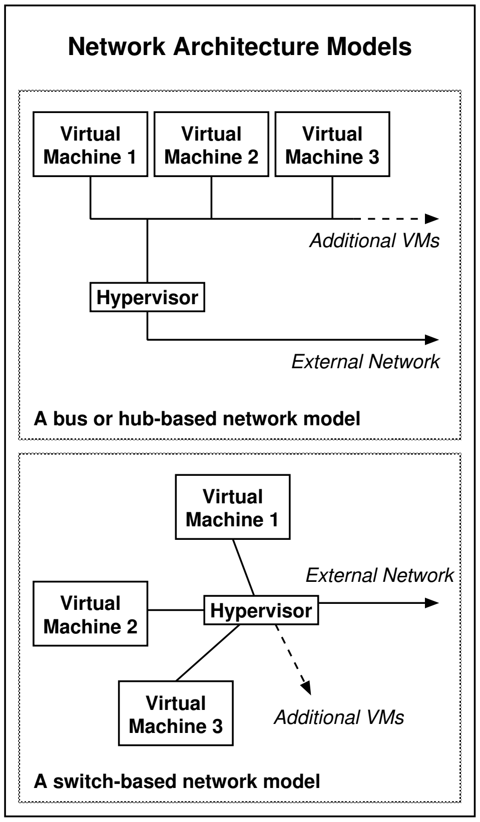

Figure 2: Forwarding Models for 'Isolated' Interfaces.

These two models are illustrated in Figure 2. In the first example, the hypervisor presents all traffic to all virtual machines. Although this may be much closer to a pure virtualization of the physical resources, it presents several problems. First, there is a security concern in that each node can see each other node's traffic. This may have management implications as to whether or not the virtual Ethernet device should allow a promiscuous mode. Second, this model presents a more complicated system structure from a performance standpoint. Incoming packets must be either copied to each VM's receive queue, incurring an overhead, or alternatively the hypervisor must provide the ability to deliver a common piece of memory as either read-only or copy-on-write to a set of VMs. Although this model's abstraction is fairly simple – a plain broadcast Ethernet between VMs, it may prove to be the case that providing what is essentially a pure virtualization of the underlying resource imposes an unreasonable cost, both in terms of complexity and performance, on the system.

In the second model, the hypervisor acts as a network switch. A packet classifier is incorporated to "route" individual packets to the appropriate virtual machine. In this model, the hypervisor acts as an IP router, and may provide additional IP-specific services. In addition, promiscuous mode can be implemented by allowing each virtual interface to see only those packets bound for the associated VM; this behaviour is be identical to what would be expected if all of the VMs were separate physical machines on a common IP router.

As the vast majority of traffic is likely be TCP/IP-based, there is an understandable benefit to providing additional IP services, and the switch model appears to be a reasonable design direction. However, a limitation of this approach is that it does not account for non-IP (and non-ARP) traffic. Developers wishing to explore alternate protocols may prefer the hub model, as it does not modify Ethernet frames before they are forwarded. A hybrid solution to this issue is to act as a router for all regular (IP/ARP) traffic, as described in the second model and as a hub for all other traffic. Developers wishing to bypass the IP routing facilities provided within the hypervisor would be left to use existing IP overlay protocols such as IP over IP.

3 Hypervisor Packet Handling

The essential network concerns within the hypervisor can be characterised according to three broad activities: scheduling, multiplexing/demultiplexing, and protection. In the case of PlanetLab, as clients have a vested interest in being able to understand how their hosts are interacting with the network, it seems wise that there be a common design philosophy applied by all isolation system implementors.

3.1 Scheduling

In order to meet the service requirements of specific VMs, the hypervisor must ensure that network traffic is scheduled to meet specified limits. Varying implementations may choose to employ drastically different approaches to scheduling inbound and outbound traffic.

An idealistic goal here might be that all implemented isolation models have identical behaviours in scheduling packets. This is obviously unrealistic and the specification of a uniform model for packet scheduling may inhibit research in this area. As such, an initial goal here should be that the network system guarantee some degree of resiliency against misuse. As much as possible, the hypervisor should prevent individual hosts from monopolizing resources in such a manner as to impair the functionality of other clients. Additionally, the system may attempt to protect VMs from being flooded by inbound traffic.

3.2 Multiplexing/Demultiplexing

Received packets should be delivered only to their target host. Transmitted packets may need to be translated to share a single external IP address.

One approach to this problem is to use a table-based packet classifier/forwarder within the hypervisor to route packets appropriately. The netfilter and IPTables modules within Linux serve as a good example, and form the basis of our implementation, which is described in the next section.

An open issue that stems from the use of a rule based classifier is exactly what operations may be performed as packets are routed. Specifically, we are concerned with how specific matching rules should be, what transformations are allowed, and so on.

3.3 Protection

Individual VMs need to be protected from one another and from malicious external traffic. Among other things, this demands that VMs only be allowed to generate IP packets that are valid, and that they not be able to spoof the identity of other hosts. Given the packet classifier approach described above, this behaviour can be achieved fairly easily by either dropping invalid packets, or overwriting the source address and port fields of all outbound packets.

3.4 Additional Considerations

In addition to the packet classification functionalities described above, services such as the following may be desirable:

- **Packet Filtering** – The hypervisor may act as a firewall, filtering traffic bound for each virtual machine.
- **Address Translation** – By performing network address translation (NAT) and port forwarding, many virtual machines may share a common external IP address.
- **Traffic Logging** – Details regarding connections may be logged to allow forensic auditing in the case of a specific virtual machine acting maliciously.
- **VM-based Packet Sniffing** – Clients may wish to have some interface approximating promiscuous mode. We suggest above that this is possible, allowing a VM to see all traffic bound for it as if all the VMs were on a router. A packet classifier could be configured to deliver a larger (or even a complete) version of the traffic visible to the external interface to individual VMs. The resolution to this issue it partially a performance concern, as it would likely necessitate multiple copies of inbound message buffers, and partially a political/administrative one, for obvious reasons.

Given the proposed system structure, it should be completely reasonable to provide all of these services. As mentioned previously, the issue that must be considered in each case is the impact that their implementation will have on the efficient processing of VM traffic.

4 Network Virtualization in *Xen*

This section describes the design approach that has been taken for the first public release of the XenoServers hypervisor, *Xen*. Individual virtual machines may have one or more virtual interfaces, each of which appears as a point-to-point Ethernet link to an IP router. A diagram of the system appears in Figure 3.

The network system within Xen consists of a virtual firewall router, which is a rule-based packet classification/forwarding engine (based on the Linux netfilter/IPTables code) responsible for simple, fast packet handling. Additionally, Xen's network system incorporates a network address translation (NAT) module that provides functions such as address translation and port forwarding[2].

Packet scheduling in Xen is at the granularity of virtual interfaces. A soft real-time scheduler moves transmit packets from virtual interface send queues through Xen's routing tables. Received packets are delivered on arrival and appropriate RX scheduling is deferred on to the CPU scheduler as VMs are responsible for emptying their own inbound message buffers. VMs which do not empty their receive queues at the inbound packet rate will have extraneous packets dropped.

Rules may be installed into classification engine through an interface provided within a privileged VM (known as domain zero). These rules are tuples of the form (*pattern*, *action*). Note that rules may be prioritized and a particular packet may match multiple rules upon classification. This means that, for instance, an arriving packet bound for a VM may be routed to that VM *and* trigger the generation of a logging event to domain zero.

As an example, the following rules are installed prior to instantiating the Windows XP VM in the diagram. The rules forward all traffic bound for the static address, but bar it's access to privileged ports. The final rules map it's outbound traffic ensuring that it is not attempting to spoof the identity of another host[3].

```
(dstAddr='128.232.103.201'
dstPort=`1-1024', DROP)
(dstAddr='128.232.103.201', FORWARD ds-
tIf=pp2)
(srcAddr='128.232.103.201' srcPort=`1-
1024', DROP)
(srcAddr='128.232.103.201' srcIf=pp2,
FORWARD dstIf=eth0)
(srcAddr='128.232.103.201', DROP)
```

Additionally, the following rule is used to log TCP SYN messages. Message headers are sent to the reporting and monitoring interface of domain zero.

```
(proto=TCP flags=SYN, LOG
fmt=LOG_PKT_HEADER)
```

[2]The NAT module does not presently attempt to provide heavier functionalities such as per-flow connection tracking and application-specific (e.g. FTP) translations.

[3]Rules for local delivery to other interfaces have been omitted for simplicity.

5 Conclusion

This paper has presented a discussion of the issues involved in the sharing of network resources for a set of isolated virtual machines. By considering the network system implemented in the hypervisor as a virtualization of a local area network, we feel that its role becomes much more understandable. This approach presents a model in which VMs appear as isolated as they would be were they separate physical machines on a shared switching element.

We hope that this discussion and the overview of our own approach serve to promote discussion on these issues with other researchers.

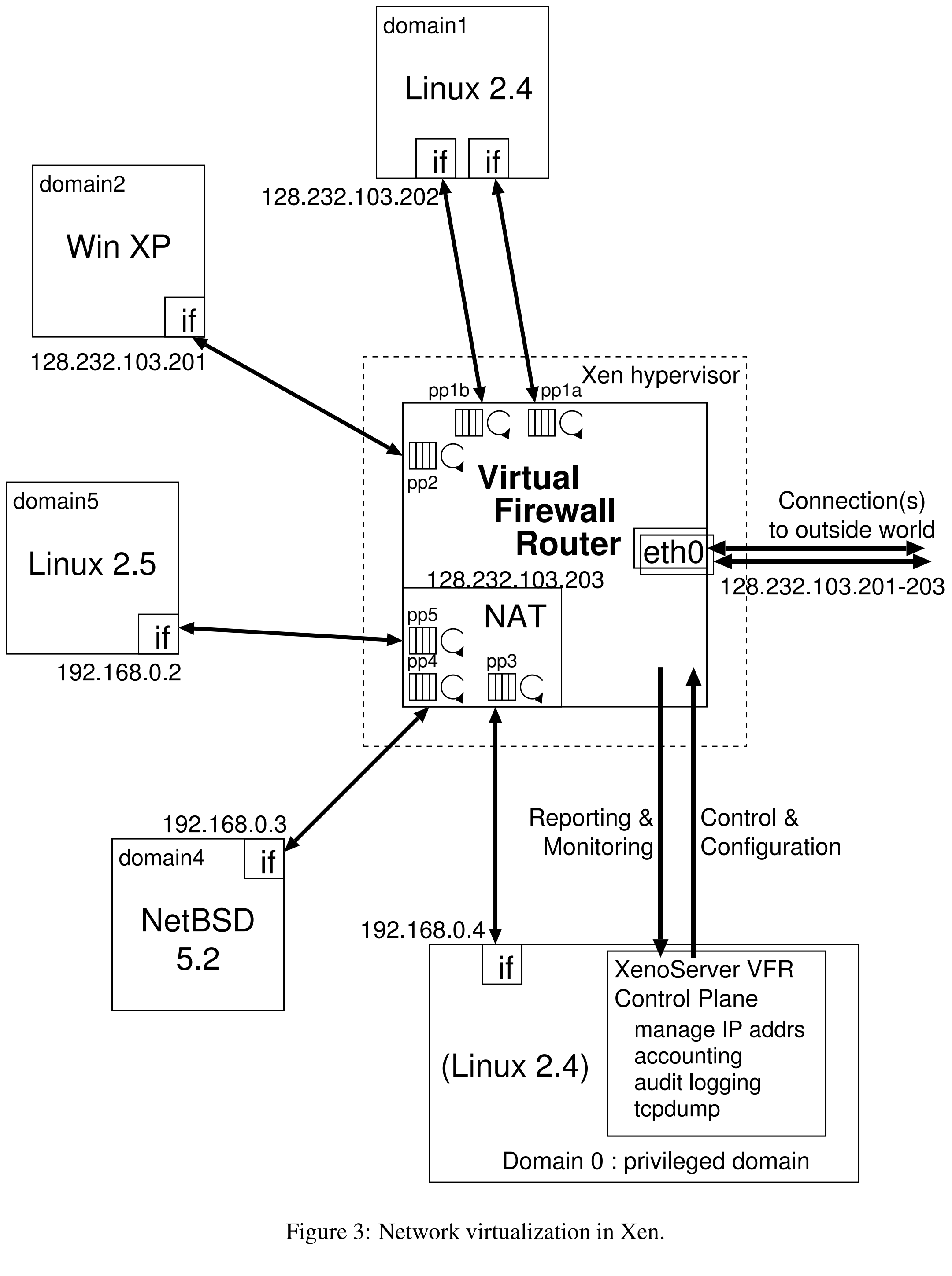

Figure 3: Network virtualization in Xen.